Simple Proofs of Sequential Work

Bram Cohen[1] and Krzysztof Pietrzak[2]

[1] http://bramcohen.com/
[2] IST Austria

Abstract. Mahmoody, Moran and Vadhan [ITCS'13] introduce and construct *publicly verifiable proofs of sequential work*, which is a protocol for proving that one spend sequential computational work related to some statement. The original motivation for such proofs included non-interactive time-stamping and universally verifiable CPU benchmarks. A more recent application, and our main motivation, are blockchain designs, where proofs of sequential work can be used – in combination with proofs of space – as a more ecological and economical substitute for proofs of work which are currently used to secure Bitcoin and other cryptocurrencies.

The construction proposed by [MMV'13] is based on a hash function and can be proven secure in the random oracle model, or assuming *inherently sequential* hash-functions, a complexity assumption also introduced in their work.

In a proof of sequential work, a prover gets a "statement" χ, a time parameter N and access to a hash-function H, which for the security proof is modelled as a random oracle. Correctness requires that an honest prover can make a verifier accept making only N queries to H, while soundness requires that any prover who makes the verifier accept must have made (almost) N *sequential* queries to H. Thus a solution constitutes a proof that N time passed since χ was received. Solutions must be publicly verifiable in time at most polylogarithmic in N.

The construction of [MMV'13] is based on "depth-robust" graphs, and as a consequence has rather poor concrete parameters. But the major drawback is that the prover needs not just N time, but also N space to compute a proof.

In this work we propose a proof of sequential work which is much simpler, more efficient and achieves much better concrete bounds. Most importantly, in our construction the prover requires as little as $O(\log(N))$ space, not $O(N)$ as in [MMV'13].

1 Introduction

1.1 Proofs of Sequential Work (PoSW)

Mahmoody, Moran and Vadhan [MMV13] introduce the notion of proofs of sequential work (PoSW), and give a construction in the random oracle model (ROM), their construction can be made non-interactive using the

Fiat-Shamir methodology [FS87]. Informally, with such a non-interactive PoSW one can generate an efficiently verifiable proof showing that some computation was going on for N time steps since some statement χ was received. Soundness requires than one cannot generate such a proof in time much less than N even considering powerful adversaries that have a large number of processors they can use in parallel.

[MMV13] introduce a new standard model assumption called "inherently sequential" hash functions, and show that the random oracle in their construction can be securely instantiated with such hash functions.

Random Oracle Model (ROM). PoSW are easiest to define and prove secure in the ROM, as here we can identify a (potentially parallel) query to the RO as one time step. Throughout this paper we'll work in the ROM, but let us remark that everything can be lifted to the same standard model assumption used in [MMV13].

A proof of sequential work in the ROM is a protocol between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$, both having access to a random oracle $\mathsf{H} : \{0,1\}^* \to \{0,1\}^w$. Figure 1 illustrates PoSW as constructed in [MMV13] and also here. We'll give a more formal definition in §1.2.

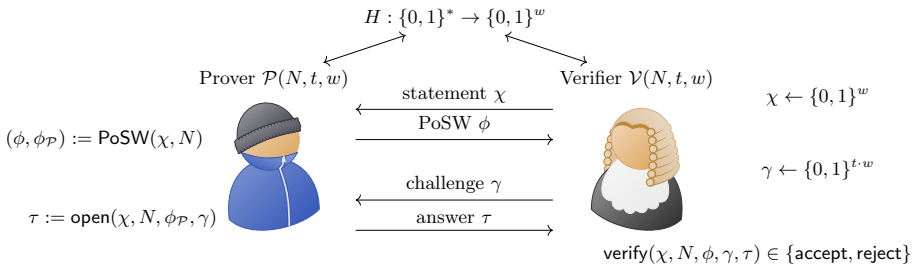

Fig. 1. Proofs of Sequential Work in the ROM as constructed in [MMV13] and this paper. w, t are statistical security parameters. N is the time parameter, i.e., $\mathsf{PoSW}(\chi, N)$ makes N queries to H computing ϕ, and any cheating prover $\tilde{\mathcal{P}}$ that makes $\mathcal{V}$ accept must make almost N sequential queries to H computing ϕ.

Non-Interactive PoSW. The first message is sent from $\mathcal{V}$ to $\mathcal{P}$, and is just a uniformly random w bit string χ. In applications this first message is a "statement" for which we want to prove that N time has passed since it was received. The distribution and domain of this statement is not important, as long as it has sufficiently high min-entropy, because we can always first hash it down to a uniform w bit string using the RO.

As the prover is public-coin, we can make the protocol non-interactive using the Fiat-Shamir heuristic [FS87]: A non-interactive PoSW for statement χ and time parameter N is a tuple (χ, N, ϕ, τ) where the challenge $\gamma = (\mathsf{H}(\phi, 1), \ldots, \mathsf{H}(\phi, t))$ is derived from the proof ϕ by hashing with the RO.

1.2 PoSW Definition

The PoSW we consider are defined by a triple of oracle aided algorithms PoSW, open and verify as defined below.

Common Inputs $\mathcal{P}$ and $\mathcal{V}$ get a as common input two statistical security parameters $w, t \in \mathbb{N}$ and time parameter $N \in \mathbb{N}$. All parties have access to a random oracle $\mathsf{H} : \{0,1\}^* \to \{0,1\}^w$.

Statement $\mathcal{V}$ samples a random $\chi \leftarrow \{0,1\}^w$ and sends it to $\mathcal{P}$.

Compute PoSW $\mathcal{P}$ computes (ideally, making N queries to H sequentially) a proof $(\phi, \phi_{\mathcal{P}}) := \mathsf{PoSW}^{\mathsf{H}}(\chi, N)$. $\mathcal{P}$ sends ϕ to $\mathcal{V}$ and locally stores $\phi_{\mathcal{P}}$.

Opening Challenge $\mathcal{V}$ samples a random challenge $\gamma \leftarrow \{0,1\}^{t \cdot w}$ and sends it to $\mathcal{P}$.

Open $\mathcal{P}$ computes $\tau := \mathsf{open}^{\mathsf{H}}(\chi, N, \phi_{\mathcal{P}}, \gamma)$ and sends it to $\mathcal{V}$.

Verify $\mathcal{V}$ computes and outputs $\mathsf{verify}^{\mathsf{H}}(\chi, N, \phi, \gamma, \tau) \in \{\mathsf{accept}, \mathsf{reject}\}$.

We require perfect **correctness**: if $\mathcal{V}$ interacts with an honest $\mathcal{P}$, then it will output accept with probability 1. The **soundness** property requires that any potentially malicious prover $\widetilde{\mathcal{P}}$ who makes $\mathcal{V}$ accept with good probability must have queried $\mathsf{H}(\cdot)$ "almost" N times sequentially. This holds even if in every round $\widetilde{\mathcal{P}}$ can query $\mathsf{H}(\cdot)$ on many inputs in parallel, whereas the honest $\mathcal{P}$ just needs to make a small (in our construction 1, in [MMV13] 2) number of queries per round.

1.3 The [MMV13] and our Construction in a Nutshell

In the construction from [MMV13], the statement χ is used to sample a fresh random oracle H. Then $\mathcal{P}$ uses H to compute "labels" of a DAG G, where the label of a node is the hash of the labels of its parents. Next, $\mathcal{P}$ computes a Merkle tree commitment of those labels, sends it to $\mathcal{V}$, who then challenges $\mathcal{P}$ to open some of the labels together with its parents.

If G is "depth-robust", which means it has a long path even after removing many verticies, a cheating prover can either (1) try to cheat and make up many of the labels, or (2) compute most of the labels correctly. The security proof now shows that in case (1) the prover will almost

certainly not be able to correctly open the Merkle tree commitments, and in case (2) he actually must make almost the same amount of sequential queries as an honest prover.

Our construction is conceptually similar, but our underlying graph is much simpler. We use the tree underlying the Merkle commitment also for enforcing honest behaviour. For this, it suffices to add some edges as illustrated in Figure 3.

Our graph has some convenient properties, for example the labels of the parents of a leaf node v are always a subset of the labels one needs to provide for the opening of the Merkle tree commitment of the label of v, so checking that the labels are correctly computed and verifying the opening can be done simultaneously without increasing communication complexity.

But most importantly, the labels in our graph can be computed in topological order while keeping only logarithmically many labels in memory at any point, whereas computing the labelling of a depth-robust graph is provably much more expensive. In fact, because of this property depth-robust graphs are used to build so called memory-hard functions. Concretely, [ABP17] show that if the labelling of a depth-robust graph on N nodes is done in time T using space S, then $T \cdot S$ must be in $\Omega(N^2)$

1.4 More Related Work

Time Release Cryptography. Rivest, Shamir and Wagner [RLRW00] introduce so called time-lock puzzles, and give a construction based on the assumption that exponentiation modulo an RSA integer is an "inherently sequential" computation, more recent constructions include [BGJ+16]. Ideas for "time-release" cryptography in general have been floating around before [CLSY93,**?**].

Time-lock puzzles allow a puzzle generator to generate a puzzle with a message of its choice encoded into it, such that this message can only be redeemed by a solver after t steps of sequential work. Such a scheme can be used as a PoSW as the decoded message constitutes a proof. In the other direction, time-lock puzzles seem more powerful than PoSW. Unlike for PoSW, we have no constructions based on random oracles, and [MMV11] give black-box separations showing this might be inherent (we refer to their paper for the exact statements).

Proofs of work – introduced by Dwork and Naor [DN93] – are defined similarly to proof of *sequential* work, but as the name suggests, here one does not require that the work has been done sequentially. Proofs of work are very easy to construct in the random oracle model, the original

construction [DN93] goes as follows: given a statement χ and a work parameter t, find a nonce x s.t. $\mathsf{H}(\chi, x)$ starts with t zeros. If $\mathsf{H}(\cdot)$ is modelled as a random oracle, finding such an x requires an expected 2^t number of queries, while verifying that x is a valid solution just requires a single query. Proofs of work are used in several decentralised cryptocurrencies, most notably Bitcoin.

1.5 Basic Notation

We denote with $\{0,1\}^{\leq n} \stackrel{\mathsf{def}}{=} \bigcup_{i=0}^{n}\{0,1\}^i$ the set of all binary strings of length at most n, including the empty string ϵ. Concatenation of bitstrings is denoted with $\|$. For $x \in \{0,1\}^x$, $x[i]$ denotes its ith bit, and $x[i \dots j] = x[i]\| \dots \|x[j]$.

2 Building Blocks

In §2.1 below we summarize the properties of the random oracle model [BR93] used in our security proof, and in §2.2 we define the basic properties of graphs used.

2.1 Random Oracles Basics

Salting the RO. In [MMV13] and also our construction, all three algorithms PoSW, open and verify described in §1.2 use the input χ only to sample a random oracle H_χ, for example by using χ as prefix to every input

$$\mathsf{H}_\chi(\cdot) \stackrel{\mathsf{def}}{=} \mathsf{H}(\chi, \cdot) \ .$$

We will sometimes write e.g., $\mathsf{PoSW}^{\mathsf{H}_\chi}(N)$ instead $\mathsf{PoSW}^{\mathsf{H}}(\chi, N)$. Using the uniform χ like this implies that in the proof we can assume that to a cheating prover, the random oracle $\mathsf{H}_\chi(\cdot)$ just looks like a "fresh" random oracle on which it has no auxiliary information [DGK17].

Random Oracles are Collision Resistant.

Lemma 1 (RO is Collision Resistant). *Consider any adversary $\mathcal{A}^{\mathsf{H}}$ given access to a random function $\mathsf{H} : \{0,1\}^* \to \{0,1\}^w$. If $\mathcal{A}$ makes at most q queries, the probability it will make two colliding queries $x \neq x', \mathsf{H}(x) = \mathsf{H}(x')$ is at most $q^2/2^{w+1}$.*

Proof. The probability that the i'th query hits any of the $i-1$ previous outputs is at most $\frac{i-1}{2^w}$, by the union bound, we get that the probability that any i hits a previous output is at most $\sum_{i=1}^{q} \frac{i-1}{2^w} < \frac{q^2}{2^{w+1}}$. □

Random Oracles are Sequential.

Definition 1 (H-sequence). *An* H *sequence of length* s *is a sequence* $x_0, \ldots, x_s \in \{0,1\}^*$ *where for each* $i, 1 \leq i < s$, $\mathsf{H}(x_i)$ *is contained in* x_{i+1} *as continuous substring, i.e.,* $x_{i+1} = a \| \mathsf{H}(x_i) \| b$ *for some* $a, b \in \{0,1\}^*$.

Lemma 2 (RO is Sequential). *Consider any adversary* $\mathcal{A}^{\mathsf{H}}$ *given access to a random function* $\mathsf{H} : \{0,1\}^* \to \{0,1\}^w$ *for at most* $s-1$ *rounds, where in each round it can make arbitrary many parallel queries. If* $\mathcal{A}$ *makes at most* q *queries of total length* Q *bits, then the probability that it outputs an* H*-sequence of length* s *(as defined above) is at most*

$$q \cdot \frac{Q + \sum_{i=0}^{s} |x_i|}{2^w}$$

Proof. There are two ways $\mathcal{A}$ can output an H sequence $x_0, \ldots, x_s$ making only $s-1$ sequential queries.

1. It holds that for some i, $\mathsf{H}(x_i)$ is a substring of x_{i+1} and the adversary did not make the H query x_i. As H is uniform, the probability of this event can be upper bounded by
$$q \cdot \frac{\sum_{i=0}^{s} |x_i|}{2^w} \ .$$
2. It holds that for some $1 \leq i < j \leq s-1$, a query a_i is made in round i and query a_j in round j where $\mathsf{H}(a_j)$ is a substring of a_i. Again using that H is uniformly random, the probability of this event can be upper bounded by
$$q \cdot \frac{Q}{2^w} \ .$$

The claimed bound follows by a union bound over the two cases analysed above. □

Thus, whenever an adversary outputs an H-sequence of length s and size $\sum_{i=0}^{s} |x_i| = L$, and $q \cdot (Q + L) \ll 2^{w/2}$ – which in practice will certainly be the case if we use a standard block length like $w = 256$ – we can assume that it made at least s *sequential* queries to H.

2.2 Graphs Basics

To describe the [MMV13] and our construction we'll need the following definitions

Definition 2 (Graph Labelling). *Given a directed acyclic graph (DAG) $G = (V, E)$ on vertex set $V = \{0, \ldots, N-1\}$ and a hash function $\mathsf{H} : \{0,1\}^* \to \{0,1\}^w$, the label $\ell_i \in \{0,1\}^w$ of $i \in V$ is recursively computed as (u is a parent of v if there's a directed edge from u to v)*

$$\ell_i = \mathsf{H}(i, \ell_{p_1}, \ldots, \ell_{p_d}) \text{ where } (p_1, \ldots, p_d) = \mathsf{parents}(i) \ . \tag{1}$$

Note that for any DAG the labels can be computed making N sequential queries to H by computing them in an arbitrary topological order. If the maximum indegree of G is δ, then the inputs will have length at most $\lceil \log(N) \rceil + \delta \cdot w$.

The PoSW by Mahmoody et al. [MMV13] is based on depth-robust graphs, a notion introduced by Erdős et al. in [EGS75].

Definition 3 (Depth-Robust DAG). *A DAG $G = (V, E)$ is (e, d) depth-robust if for any subset $S \subset V$ of at most $|S| \leq e$ vertices, the subgraph on $V - S$ has a path of length at least d.*

For example, the complete DAG $G = (V, E), |V| = N, E = \{(i, j) : 0 \leq i < j \leq N-1\}$ is $(e, N-e)$ depth robust for any e, but for PoSW we need a DAG with small indegree. Already [EGS75] showed that $(\Theta(N), \Theta(N))$ depth-robust DAGs with indegree $O(\log(N))$ exist. Mahmoody et al. give an explicit construction with concrete constants, albeit with larger indegree $O(\log^2(N)\text{polyloglog}(N)) \in O(\log^3(N))$.

3 The [MMV13] Construction

In this section we informally describe the PoSW from [MMV13] using the high-level protocol layout from §1.2.

For any $N = 2^n$, the scheme is specified by a depth robust DAG $G_n^{\mathsf{DR}} = (V, E)$ on $|V| = N$ vertices. Let $B_n = (V', E')$ denote the full binary tree with $N = 2^n$ leaves (and thus $2N - 1$ nodes) where the edges are directed towards the root. Let $G^{\mathsf{MMV}}n$ be the DAG we get from B_n, by identifying the N leaves of this tree with the N nodes of G_n^{DR} as illustrated in Figure 2

Now $(\phi, \phi_{\mathcal{P}}) := \mathsf{PoSW}^{\mathsf{H}_\chi}(N)$ (as introduced in §1.2 and §2.1) computes and stores the labels $\phi_{\mathcal{P}} = \{\ell_v\}_{v \in \{0,1\}^{\leq n}}$ (cf. Definition 2) of G_n^{MMV} using H_χ as hash function, and sends the label $\phi = \ell_\epsilon$ of the root to $\mathcal{V}$. We remark that in [MMV13] this is described as a two step process, where one first computes a labeling of G_n^{DR} (using a sequential hash function), and then a Merkle tree commitment of the N labels (using a collision resistant hash function).

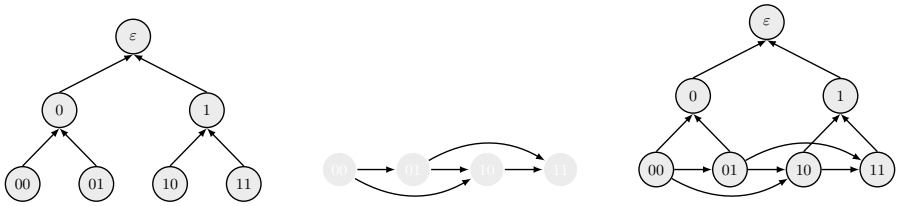

Fig. 2. Illustration of B_2 (left), a (toy example of a) depth-robust graph G_2^{DR} (middle) and the corresponding G_2^{MMV} graph.

After receiving the challenge $\gamma = (\gamma_1, \ldots, \gamma_t)$ from $\mathcal{V}$, the prover $\mathcal{P}$ computes the answer $\tau := \mathsf{open}^{\mathsf{H}_\chi}(N, \phi_{\mathcal{P}}, \gamma)$ as follows: For any $i, 1 \leq i \leq t$, τ contains the label ℓ_{γ_i} and the labels required for the opening of the Merkle commitment of this label.[3]

Upon receiving the answer τ to its challenge τ, $\mathcal{V}$ invokes $\mathsf{verify}^{\mathsf{H}_\chi}(N, \phi, \gamma, \tau)$ to check if the labels ℓ_{γ_i} were correctly computed as in Eq.(1), and if the Merkle openings of those the labels ℓ_{γ_i} are correct.

To argue soundness, one uses the fact that G_n^{DR} is (e, d)-depth robust with $e, d = \Theta(N)$. As H_χ is collision resistant, a cheating prover $\tilde{\mathcal{P}}$ must commit to unique labels $\{\ell'_v\}_{v \in \{0,1\}^n}$ of the leaves (that it can later open to). We say that a vertex i is inconsistent if it is not correctly computed from the other labels, i.e.,

$$\ell'_i \neq \mathsf{H}(i, \ell'_{p_1}, \ldots, \ell'_{p_d}) \text{ where } (p_1, \ldots, p_d) = \mathsf{parents}(i)$$

Let β be the number of inconsistent verticies. We make a case distinction:

- If $\beta \geq e$, then one uses the fact that the probability that a cheating prover $\tilde{\mathcal{P}}$ will be asked to open an inconsistent vertex is exponentially (in t) close to 1, namely $1 - \left(\frac{N-\beta}{N}\right)^t$, and thus $\tilde{P}$ will fail to make $\mathcal{V}$ accept except with exponentially small probability.
- if $\beta < e$, then there's a path of length $d = \Theta(N)$ of consistent verticies, which means the labels $\ell'_{i_0}, \ldots, \ell'_{i_{d-1}}$ on this path constitute an H_χ sequence (cf. Def.1) of length $d - 1$, and as H_χ is sequential, $\tilde{\mathcal{P}}$ must almost certainly have made $d - 1 = \Theta(N)$ *sequential* queries to H_χ.

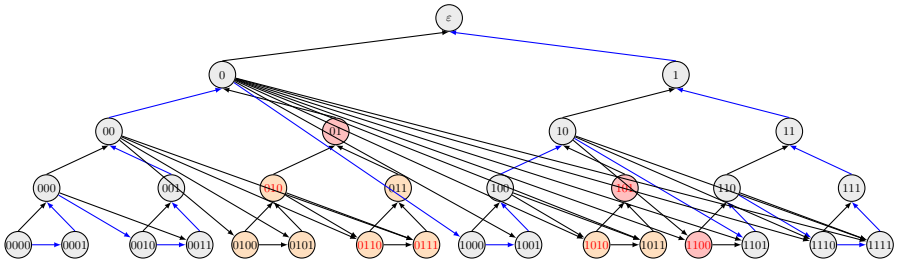

Fig. 3. Illustration of G_4^{PoSW}. For a set $S = \{010, 0110, 0111, 101, 1010, 1100\}$, its extended set S^+ is $S \cup \{01\}$, and the purified set $S^* = \{01, 101, 1100\}$ (the red nodes). The extended set S^+ is $S \cup \{01\}$, and D_{S^*} is the union of red and orange nodes. The path of length $2N - 1 - |B_S| = 32 - 1 - 11 = 19$ is shown in blue.

4 Definition and Properties of the DAG G_n^{PoSW}

For $n \in \mathbb{N}$ let $N = 2^{n+1} - 1$ and $B_n = (V, E')$ be a complete binary tree of depth n. We identify the N nodes $V = \{0,1\}^{\leq n}$ with the binary strings of length at most n, the empty string ε being the root. We say vertex v is *below* u if $u = v\|a$ for some a. The directed edges go from the leaves towards the root

$$E' = \{(x\|b, x) \ : \ b \in \{0,1\}, x \in \{0,1\}^i, i < n\}$$

We define the DAG $G_n^{\mathsf{PoSW}} = (V, E)$ by starting with B_n, and then adding some edges. Concretely $E = E' \cup E''$ where E'' contains, for all leaves $u \in \{0,1\}^n$, an edge (v, u) for any v that is a left sibling of a node on the path from u to the root ε. E.g., for $v = 1101$ we add the edges $(1100, 1101), (10, 1101), (0, 1101)$, formally

$$E'' = \{(v, u) \ : \ u \in \{0,1\}^n, u = a\|1\|a', v = a\|0\}$$

Definition 4 (purified/extended set S^*/S^+). *For a subset $S \subset V$, its* extended *set $S^+ \supseteq S$ is derived from S by adding all $u \in V$ for which theres no leaf $v \in \{0,1\}^n$ such that no vertex on the path from u to v is in S.*

The purified *set $S^* \subseteq S^+$ of S is derived from S^+ by removing all vertices u for which there's already another $v \in S^+$ on the path from u to the root (equivalently, remove u if it's below some $v \in S^+$).*

[3] That is, the labels of all siblings of the nodes on the path from this vertex to the root. E.g., for label ℓ_{01} (as in Figure 2) that would be ℓ_{00} and ℓ_1. To verify, one checks if $\mathsf{H}_\chi(0, \ell_0 0, \ell_0 1) = \ell_0$ and $\mathsf{H}_\chi(\varepsilon, \ell_0, \ell_1) = \ell_\varepsilon = \phi$.

E.g. the purified set S^* in Figure 3 could be derived from a set S like

$$\begin{aligned} S &= \{010, 0110, 0111, 101, 1010, 1100\} \\ S^+ &= \{010, 0110, 0111, 01, 101, 1010, 1100\} \\ S^* &= \{01, 101, 1011\} \end{aligned}$$

For a subset $S \subset V$, let D_S denote S plus all all the vertices which are below some vertex in S

$$D_S = \{v \| v' \ : \ v \in S, v' \in \{0,1\}^{\leq n-|v|}\}$$

Note that for any $S \subset V$, $D_{S^+} \equiv D_{S^*}$ and $D_{S^+} \cap \{0,1\}^n \equiv D_S \cap \{0,1\}^n$.

Lemma 3. *For any extended set $S^+, S \subset V$, the subgraph of G_n^{PoSW} on vertex set $V - D_{S^+}$ has a (directed) path going through all the $|V| - |D_{S^+}| = N - |D_{S^+}|$ vertices.*

Proof. The proof is by induction on n, an example path is illustrated in Figure 3. The lemma is trivially true for G_1^{PoSW}, which just contains a single node. Assume it's true for G_i^{PoSW}, now G_{i+1}^{PoSW} consists of a root ε, with a left and right subgraph L and R identical to G_i^{PoSW}. If $\varepsilon \in S^*$ the lemma is trivially true as $|V| - |D_{S^*}| = 0$. If $0 \in S^*$, then all of L is in D_{S^*}, in this case just apply the Lemma to $R \equiv G_i^{\mathsf{PoSW}}$ (with an extra last edge $1 \to \varepsilon$ if $1 \notin S^*$). If $0 \notin S^*$, apply the Lemma first to $L \equiv G_i^{\mathsf{PoSW}}$, then to $R \equiv G_i^{\mathsf{PoSW}}$, and let v be the starting vertex here, connect the two paths with the edge $0 \to v$. □

Lemma 4 (trivial). *For any purified $S^*, S \subset V$, D_{S^*} contains*

$$|\{0,1\}^n \cap D_{S^*}| = \frac{|D_{S^*}| + |S^*|}{2}$$

many leaves.

Proof. Let $S^* = \{v_1, \ldots, v_k\}$, then

$$|\{0,1\}^n \cap D_{S^*}| = \sum_{i=1}^{k} |\{0,1\}^n \cap D_{v_i}|$$

as every D_{v_i} is just a balanced binary tree, and thus has $(|D_{v_i}| + 1)/2$ many leaves

$$\sum_{i=1}^{k} |\{0,1\}^n \cap D_{v_i}| = \sum_{i=1}^{k} \frac{|D_{v_i}| + 1}{2} = \frac{|D_{S^*}| + |S^*|}{2}$$

□

5 Our Construction

Our construction is conceptually similar to [MMV13], but uses a simpler graph as illustrated in Figure 3. We describe our PoSW in detail, except for the exact specification of the underlying graphs G_n^{PoSW} which will be discussed in §4.

5.1 Parameters

We have the following parameters:

N The time parameter which we assume is of the form $N = 2^{n+1} - 1$ for an integer $n \in \mathbb{N}$.

$\mathsf{H} : \{0,1\}^{\leq w(n+1)} \to \{0,1\}^w$ the hash function, which for the security proof is modelled as a random oracle, and which takes as inputs strings of length up to $w(n+1)$ bits.

t A statistical security parameter.

M Memory available to $\mathcal{P}$, we assume it's of the form $M = (n \cdot t + 2^{m+1})w$ for some integer $m, 0 \leq m \leq n$.

5.2 The PoSW, open and verify algorithms.

Our PoSW follows the outline given in §1.2 using three algorithms PoSW, open and verify. Note that n, m are basically the logarithms of the time parameter N and the memory we allow $\mathcal{P}$ to use.

$(\phi, \phi_{\mathcal{P}}) := \mathsf{PoSW}^{\mathsf{H}_\chi}(N)$: computes the labels $\{\ell_i\}_{i \in \{0,1\}^{\leq n}}$ (cf. Def. 2) of the graph G_n^{PoSW} (to be defined in §4) using H_χ. It stores the labels $\phi_{\mathcal{P}} = \{\ell_i\}_{i \in \{0,1\}^{\leq m}}$, and sends $\phi = \ell_\varepsilon$ to $\mathcal{V}$.

$\tau := \mathsf{open}^{\mathsf{H}_\chi}(N, \phi_{\mathcal{P}}, \gamma)$: on challenge $\gamma = (\gamma_1 \ldots, \gamma_t)$, for every $i, 1 \leq i \leq t$ contains the label ℓ_{γ_i} of node $\gamma_i \in \{0,1\}^n$ and the labels of all siblings of the nodes on the path from γ_i to the root (as in an opening of a Merkle tree commitment), i.e.,

$$\{\ell_j\}_{j \in \mathcal{S}} \text{ where } \mathcal{S}_{\gamma_j} \stackrel{\mathsf{def}}{=} \{\gamma_i[1 \ldots j-1] \| (1 - \gamma[j])\}_{j=1 \ldots n}$$

So

$$\tau \stackrel{\mathsf{def}}{=} \{\ell_{\gamma_i}, \{\ell_k\}_{k \in \mathcal{S}_{\gamma_i}}\}_{i=1 \ldots t}$$

E.g., for $\gamma_i = 0101$ (cf. Figure 3) τ contains the labels of 0101,0100,011,00 and 1.

If $m = n$, $\mathcal{P}$ stored all labels in $\phi_{\mathcal{P}}$ and thus this needs no queries to H_χ. We'll discuss the case $0 < m < n$ in §5.4.

$\mathsf{verify}^{\mathsf{H}_\chi}(N, \phi, \gamma, \tau)$ Using that the graphs G_n^{PoSW} have the property that all the parents of a leaf γ_i are in $\mathcal{S}_{\gamma_i}$, for every $i, 1 \leq i \leq t$, one first checks that ℓ_{γ_i} was correctly computed from its parent labels (i.e., as in Eq.1)

$$\ell_{\gamma_i} \stackrel{?}{=} \mathsf{H}(i, \ell_{p_1}, \ldots, \ell_{p_d}) \text{ where } (p_1, \ldots, p_d) = \mathsf{parents}(\gamma_i)$$

Then we verify the "Merkle tree like" commitment of ℓ_{γ_i}, by using the labels in τ to recursively compute, for $i = n-1, n-2, \ldots 0$

$$\ell_{\gamma_i[0\ldots i]} := \mathsf{H}_\chi(\gamma_i[0\ldots i], \ell_{\gamma_i[0\ldots i]\|0}, \ell_{\gamma_i[0\ldots i]\|1})$$

and then verifying that the computed root $\ell_{\gamma_i[0\ldots 0]} = \ell_\varepsilon$ is equal to ϕ received before.

5.3 Security

Theorem 1. *Consider the PoSW from §5.2, with parameters t, w, N and a "soundness gap" $\alpha > 0$. If $\tilde{\mathcal{P}}$ makes at most $(1-\alpha)N$ sequential queries to H after receiving χ, and at most q queries in total, then $\mathcal{V}$ will output* reject *with probability*

$$1 - (1-\alpha)^t - n \cdot w \cdot q^2/2^w$$

So, for example setting the statistical security parameter $t = 20$, means a $\tilde{\mathcal{P}}$ who makes only $0.8N$ *sequential* queries will be able to make $\mathcal{V}$ accept with $\leq 1\%$ probability. This is sufficient for some applications, but if we want to use Fiat-Shamir to make the proof non-interactive, the error should be much smaller, say 2^{-50} which we get with $t = 150$.

Proof. This proof will use some notation and results from §4, the reader can skip the proof for now and only come back after reading that section.

The exponentially small $n \cdot w \cdot q^2/2^w$ loss accounts for the assumption we'll make, that $\tilde{\mathcal{P}}$, after receiving χ (1) won't find a collision in H_χ, and (2) whenever it outputs an H_χ-sequence of length s it must have made s sequential queries to H (see discussion in 2.1).

After sending ϕ, $\tilde{\mathcal{P}}$ is committed to the labels $\{\ell'_i\}_{i \in \{0,1\}^{\leq n}}$ it can open. We say a node i is inconsistent if its label ℓ'_i was not correctly computed, i.e.,

$$\ell'_i \neq \mathsf{H}(i, \ell'_{p_1}, \ldots, \ell'_{p_d}) \text{ where } (p_1, \ldots, p_d) = \mathsf{parents}(i)$$

Let us mention that i can be consistent even though $\ell'_i \neq \ell_i$ (ℓ_i denoting the label the honest $\mathcal{P}$ would compute), so being consistent is not the

same as being correct. We can also determine these ℓ'_i from just looking at $\tilde{\mathcal{P}}$'s oracle queries, but for the proof we just need that they are unique.

Let $S \subset V = \{0,1\}^{\leq n}$ denote all inconsistent nodes, and $S^+ \supseteq S$ be its extended set (cf. Definition 4). Then by Lemma 3 there's a path going through all the nodes in $V - D_{S^+}$, as all these nodes are consistent, the labels ℓ'_i on this path constitute a H_χ-sequence of length $N - |D_{S^+}|$. If $|D_{S^+}| \leq \alpha N$, $\tilde{\mathcal{P}}$ must have made at least $(1-\alpha)N$ sequential queries (recall we assume $\tilde{\mathcal{P}}$ did not break sequentiality of H_χ), so we now assume

$$|D_{S^+}| = |D_{S^*}| > \alpha N = \alpha(2^{n+1} - 1)$$

By Lemma 4 and the above equation

$$|\{0,1\}^n \cap D_{S^*}| = \frac{|D_{S^*}| + |S^*|}{2} > \alpha 2^n \tag{2}$$

$\tilde{\mathcal{P}}$ will fail to make $\mathcal{V}$ accept given the t random challenges $\gamma = \{\gamma_1, \ldots, \gamma_t\}$ if there's at least one γ_i such that a node on the path from γ_i to the root is in S, or equivalently

$$\gamma \cap D_{S^*} \neq \emptyset$$

By Eq.(2) for any $i, 1 \leq i \leq t$, $\Pr[\gamma_i \in D_{S^*}] > \alpha$, and as the γ_i are uniform and independent

$$\Pr[\gamma \cap D_{S^*} = \emptyset] < (1-\alpha)^t$$

so $\tilde{\mathcal{P}}$ will fail to open the commitment with probability $> 1 - (1-\alpha)^t$ as claimed. □

5.4 Efficiency

We'll now discuss the efficiency of the scheme from §5.2 terms of proof size, computation and memory requirements.

5.5 Proof Size

The exchanged messages χ, ϕ, γ, τ are of length

$$|\chi| = w \qquad |\phi| = w \qquad |\gamma| = t \cdot w \qquad |\tau| \leq t \cdot w \cdot n$$

When we make the proof non-interactive using Fiat-Shamir (where γ is derived from ϕ) the length of a proof for a given statement χ becomes

$$|\phi| + |\tau| \leq w(t \cdot n + 1)$$

With $w = 256$ bit blocks, $t = 150$, which is sufficient to get 2^{-50} security for soundness gap $\alpha = 0.2$ (i.e., a cheating prover must make $0.8N$ sequential queries) and $n = 40$ (i.e., over a trillion steps) the size of the proof is of size less than 200KB.

5.6 Prover Efficiency

$\mathcal{P}$'s efficiency is dominated by queries to H_χ for computing PoSW and open, so below we just count these.

$\mathsf{PoSW}^{\mathsf{H}_\chi}(N)$ can be computed making N (sequential) queries to H_χ, each input being of length at most $n \cdot w$ bits, and on average about $1/4$ of that (for comparison, the construction from [MMV13] had inputs of length $n^3 \cdot w$).

$\mathsf{open}^{\mathsf{H}_\chi}(N, \phi, \gamma)$: Here the efficiency depends on m, which specifies the size of the memory $M = (n \cdot (t+1) + 2^{m+1})w$ we allow $\mathcal{P}$ to use. Here $n \cdot t$ bits are used to store the values in τ to send back, $n \cdot w$ bits are used to compute the labels (it's indeed possible to compute all labels keeping just n other labels around!), and $2^{m+1}w$ labels are used to store $\phi_{\mathcal{P}}$, which contains the labels of the m upmost levels $\{\ell_i\}_{i \in \{0,1\}^{\leq m}}$.

- If $m = n$, $\mathcal{P}$ stored all the labels computed by $\mathsf{PoSW}^{\mathsf{H}_\chi}(N)$, and thus needs no more queries.
- If $m = 0$, $\mathcal{P}$ needs to recompute all N labels. This is not very satisfying, as it means that we'll always have a soundness gap of at least 2: the honest prover needs a total of $2N$ sequential queries (N for each, PoSW and open), whereas (even an honest) prover with $m = n$ space will only require N sequential queries. Fortunately there is a nice trade-off, where already using a small memory means $\mathcal{P}$ just needs to make slightly more than N queries, as described next.
- In the general case $0 \leq m \leq n$, $\mathcal{P}$ needs to compute $2^{n-m+1} - 1$ labels for each of the t challenges, thus at most

$$t \cdot (2^{n-m+1} - 1)$$

  in total (moreover this can be done making $2^{n-m+1} - 1$ queries sequentially, each with t inputs). E.g. if $m = n/2$, this means $\mathcal{P}$ uses around $\sqrt{N} \cdot w$ bits memory, and $\sqrt{N} \cdot t$ queries on top of the N for computing PoSW. For typical parameters $\sqrt{N} \cdot t$ will be marginal compared to N. More generally, for any $0 \leq \beta \leq 1$, given $N^{1-\beta} \cdot w$ memory means $\mathcal{P}$ needs $N^\beta \cdot t$ queries to compute open (or N^β sequential queries with parallelism t).
  For our example with $w = 256, n = 40, t = 150$, setting, say $m = 20$, means $\mathcal{P}$ uses 70MB of memory, and the number of queries made by open is less than $N/1000$, which is marginal compared to the N queries made by PoSW.

5.7 Verifier Efficiency.

The verifier is extremely efficient, it must only sample a random challenge γ (of length $t \cdot w$) and computing $\mathsf{verify}(\chi, N, \phi, \gamma, \tau)$ can be done making $t \cdot n$ queries to H_χ, each of length at most $n \cdot w$ bits. This is also basically the cost of verifying a non-interactive proof.

6 Conclusions

We constructed a proof of sequential work which is much simpler and enjoys much better parameters than the original construction from [MMV13]. They also state three open questions, two of which we answer in this work. Their first question is:

> **Space Complexity of the Solver.** In our construction of time stamping and time-lock puzzles for time N, the solver keeps the hash labels of a graph of N vertices. Is there any other solution that uses $o(N)$ storage? Or is there any inherent reason that $\Omega(N)$ storage is necessary?

We give a strong negative answer to this question, in our construction the storage of the prover can be as tiny as $O(\log(N))$. Their second question is:

> **Necessity of Depth-Robust Graphs.** The efficiency and security of our construction is tightly tied to the parameters of depth-robust graph constructions: graphs with lower degree give more efficient solutions, while graphs with higher robustness (the lower bound on the length of the longest path remaining after some of the vertices are removed) give us puzzles with smaller adversarial advantage. An interesting open question is whether the converse also holds: do time-lock puzzles with better parameters also imply the existence of depth-robust graphs with better parameters?

Also here the answer is no. The graphs G_n^{PoSW} we use, as illustrated in Figure 3, are basically as terrible in terms of depth robustness as a simple path. For example just removing the vertex 0 cuts the depth in half. Or just removing the $2^{n/2} = \sqrt{N}$ verticies in the middle layer, will leave no paths of length more than $\sqrt{N}$. Maybe depth-robustness is the wrong notion to look at here, our graphs satisfy a notion of "weighted" depth-robustness: assign each leaf weight 1, the nodes one layer up weight 2, then 4 etc., doubling with every layer. The total weight of all nodes will be $n2^n$ (2^n for every layer), and one can show hat for any $0 \leq \alpha \leq 1$, removing nodes of weight $\alpha 2^n$, will leave a path of length $(1 - \alpha)2^n$.

References

[ABP17] Joël Alwen, Jeremiah Blocki, and Krzysztof Pietrzak. Depth-robust graphs and their cumulative memory complexity. In *EUROCRYPT*, LNCS, 2017. https://eprint.iacr.org/2016/875.

[BGJ+16] Nir Bitansky, Shafi Goldwasser, Abhishek Jain, Omer Paneth, Vinod Vaikuntanathan, and Brent Waters. Time-lock puzzles from randomized encodings. In Madhu Sudan, editor, *ITCS 2016*, pages 345–356. ACM, January 2016.

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

[CLSY93] Jin-yi Cai, Richard J. Lipton, Robert Sedgewick, and Andrew Chi-Chih Yao. Towards uncheatable benchmarks. In *Proceedings of the Eigth Annual Structure in Complexity Theory Conference, San Diego, CA, USA, May 18-21, 1993*, pages 2–11, 1993.

[DGK17] Yevgeniy Dodis, Siyao Guo, and Jonathan Katz. Fixing cracks in the concrete: Random oracles with auxiliary input, revisited. In *EUROCRYPT 2017*, pages 473–495, 2017.

[DN93] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In Ernest F. Brickell, editor, *CRYPTO'92*, volume 740 of *LNCS*, pages 139–147. Springer, Heidelberg, August 1993.

[EGS75] Paul Erdoes, Ronald L. Graham, and Endre Szemeredi. On sparse graphs with dense long paths. Technical report, Stanford, CA, USA, 1975.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Heidelberg, August 1987.

[MMV11] Mohammad Mahmoody, Tal Moran, and Salil P. Vadhan. Time-lock puzzles in the random oracle model. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 39–50. Springer, Heidelberg, August 2011.

[MMV13] Mohammad Mahmoody, Tal Moran, and Salil P. Vadhan. Publicly verifiable proofs of sequential work. In Robert D. Kleinberg, editor, *ITCS 2013*, pages 373–388. ACM, January 2013.

[RLRW00] Adi Shamir Ronald L. Rivest and David A. Wagner. Time-lock puzzles and timedrelease crypto. Technical report, 2000.