Simple Proofs of Sequential Work

Bram Cohen[1] and Krzysztof Pietrzak[2⋆]

[1] Chia Network, bram@chia.network
[2] IST Austria, pietrzak@ist.ac.at

Abstract. At ITCS 2013, Mahmoody, Moran and Vadhan [MMV'13] introduce and construct *publicly verifiable proofs of sequential work*, which is a protocol for proving that one spent 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, which is a new standard model assumption 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, the space required can be as small as $\log(N)$ (but we get better soundness using slightly more memory than that).

An open problem stated by [MMV'13] that our construction does not solve either is achieving a "unique" proof, where even a cheating prover can only generate a single accepting proof. This property would be extremely useful for applications to blockchains.

⋆ Supported by the European Research Council (ERC), Horizon 2020, consolidator grant (682815 - TOCNeT).

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 assumptions (collision resistant and sequential hash functions) 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 $\mathsf{H} : \{0,1\}^* \rightarrow \{0,1\}^w$. Figure 1 illustrates PoSW as constructed in [MMV13] and also here. We'll give a more formal definition in §1.2.

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.

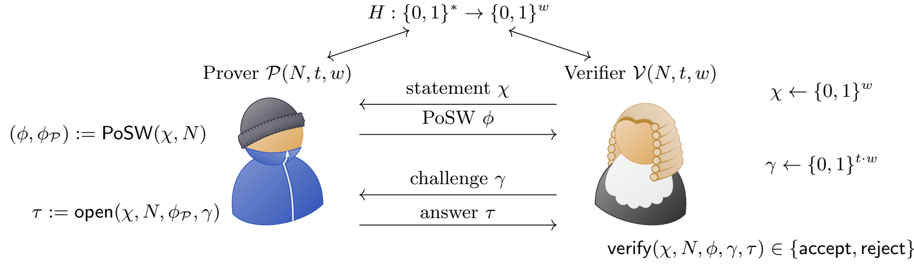

Fig. 1. Proofs of Sequential Work in the ROM as constructed in [MMV13] and this paper. N is the time parameter, i.e., $\mathsf{PoSW}(\chi, N)$ makes N queries to H computing ϕ, and any cheating prover $\tilde{\mathcal{P}}$ that makes $\mathcal{V}$ accept must make almost N sequential queries to H computing ϕ. t is a statistical security parameter, the larger t the better the soundness: any $\tilde{\mathcal{P}}$ making only $(1-\alpha)N$ sequential queries for some $\alpha > 0$, will succeed with probability at most $(1-\alpha)^t$ (e.g. with $t = 21$, a cheating prover making only $0.8 \cdot N$ sequential queries succeed with probability $< 1\%$). w is the output range of our hash function, which we need to be collision resistant and sequential, $w = 256$ is a typical value.

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 as common input two statistical security parameters $w, t \in \mathbb{N}$ and a 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 H "almost" N times sequentially. This holds even if in every round $\widetilde{\mathcal{P}}$ can query H 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 directed acyclic graph (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 $G = (V, E)$ 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 must make a large number of *sequential* queries: if he cheats on labels of nodes $S \subseteq V$, then the number of sequential queries must be at least as large as the length of the longest path in the subgraph on $V - S$. As G is depth-robust and S is not too large, this path is long.

Our construction is conceptually similar, but our underlying graph is much simpler. We use the nodes in the tree underlying the Merkle commitment not just for the commitment, but also to enforce sequential computation. For this it suffices to add some edges as illustrated in Figure 3.

Our graph has some convenient properties, for example the parents of a leaf node v are always a subset of the nodes whose 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 of a leaf label can be done simultaneously without increasing communication complexity and with only a little bit of extra computation.

But most importantly, the labels in our graph can be computed in topological order[3] while keeping only logarithmically many labels in memory at any point, whereas computing the labelling of a depth-robust graph is 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 $T \cdot S \in \Omega(N^2)$. In particular, if one wants to compute the labels in time N, or even just some $O(N)$, then linear $\Omega(N)$ space is required.

[3] A topological ordering of the vertices of a DAG is an ordering $v_1, v_2, \ldots, v_{|V|}$ such that there's no path from v_j to v_i whenever $j > i$.

There is a caveat. If using only logarithmic memory in our construction, the prover needs to recompute all the labels in the opening phase, whereas one wouldn't need any computation (just some lookups) in the opening phase if everything was stored. This is unfortunate, as it means we get a factor 2 difference in the sequential computation that is claimed, versus what has to actually be done, but some applications need this factor to be close to 1. Fortunately there is a simple trade-off, where using slightly more memory one can make the opening phase much more efficient. The basic idea, which we describe in detail in §5.4, is to store all the 2^m nodes at some level m of the tree. With this, one can compute any other node making just 2^{n-m} queries.

1.4 More Related Work

Time Release Cryptography. The idea of "time-release" cryptography goes back to [CLSY93,May93].

Most related to PoSW are time-lock puzzles, which were introduced by Rivest, Shamir and Wagner [RSW00]. They give a construction of such puzzles based on the assumption that exponentiation modulo an RSA integer is an "inherently sequential" computation. A recent treatment with new constructions is [BGJ+16].

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 of sequential work, but as the puzzle generator has a trapdoor, this proof will not be convincing to anyone else and as it's not public-coin, it can't be made non-interactive by the Fiat-Shamir methodology. Although incomparable, time-lock puzzles seem to be more sophisticated objects 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). Existing PoSW (including ours) have another drawback, namely, that the proofs are not unique. We'll discuss this in more detail at the end of §6.

Proofs of Work. Proofs of work (PoW) – 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 simplest construction of a PoW 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 H 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 to secure several decentralised cryptocurrencies and other blockchain applications, 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 $x \in \{0,1\}^*$, $x[i]$ denotes its ith bit, $x[i \dots j] = x[i]\| \dots \|x[j]$ and $|x|$ denotes the bitlength of x.

2 Building Blocks

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

2.1 Graphs Basics

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

Definition 1 (Graph Labelling). *Given a directed acyclic graph (DAG) $G = (V, E)$ on vertex set $V = \{0, \dots, N-1\}$ 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 (where u is a parent of v if there's a directed edge from u to v)*

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

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 2 (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 $O(\log^2(N)\text{polyloglog}(N)) \in O(\log^3(N))$.

2.2 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 H_χ 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 $\mathcal{A}^{\mathsf{H}}$ which is 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 output of the i'th query collides with 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. Below we show that ROs are "sequential", this is already shown in [MMV13], except that we use concrete parameters instead of asymptotic notations.

Definition 3 (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 $\mathcal{A}^{\mathsf{H}}$ which is given access to a random function $\mathsf{H} : \{0,1\}^* \to \{0,1\}^w$ that it can query 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 $x_0, \ldots, x_s$ (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. Lucky guess: It holds that for some i, $\mathsf{H}(x_i)$ is a substring of x_{i+1} and the adversary did not make the query $\mathsf{H}(x_i)$. 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. Collision: The x_i's were not computed sequentially. That is, it holds that for some $1 \leq i \leq j \leq s-1$, 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 where $q \cdot (Q + \sum_{i=0}^{s} |x_i|)$ is much smaller than 2^w – 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.

Merkle-Damgård. The inputs to our hash function H are of length up to $(\lceil \log N \rceil + 1)w$ bits (assuming $N \leq 2^w$, so the index of a node can be encoded into $\{0,1\}^w$). We can build a function for abitrary input lenghts from a compression function $\mathsf{h} : \{0,1\}^{2w} \to \{0,1\}^w$ using the classical Merkle-Damgård construction [Dam90,Mer90]. Concretely, let y_0 be the statement χ (used for salting as outlined above) and then recursively define

$$\mathsf{H}(x_1, \ldots, x_z) = y_z \text{ where } y_i = \mathsf{h}(x_i, y_{i-1}) \text{ for } i \geq 1 \ .$$

One must be careful with this approach in our construction. As it's possible to compute y_i using only the prefix $x_1, \ldots, x_i$, an adversary might get an advantage by computing such intermediate y_i's before the entire input is known, and thus exploit paralellism to speed up the computation. This can be avoided by requiring that x_1 is always the label of the node that was computed right before the current node.

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_n^{MMV} 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

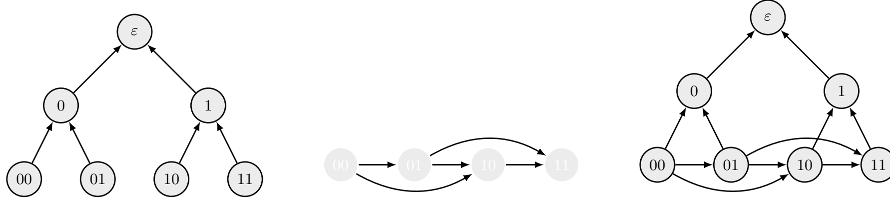

Fig. 2. Illustration of B_2 (left), a (toy example of a) depth-robust graph G_2^{DR} (middle) and the corresponding G_2^{MMV} graph.

Now $(\phi, \phi_{\mathcal{P}}) := \mathsf{PoSW}^{\mathsf{H}_\chi}(N)$ computes and stores the labels $\phi_{\mathcal{P}} = \{\ell_v\}_{v \in \{0,1\}^{\leq n}}$ (cf. Definition 1) 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).

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 opening of the Merkle commitments of the label ℓ_{γ_i}, and the labels of the parents of i, and moreover the labels labels required for the opening of the Merkle commitment of this label.[4]

Upon receiving the answer 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 the labels ℓ_{γ_i} are all 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

[4] 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 $\mathsf{H}_\chi(0, \ell_{00}, \ell_{01}) = \ell_0$ and $\mathsf{H}_\chi(\varepsilon, \ell_0, \ell_1) = \ell_\varepsilon = \phi$.

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.3) 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.

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

In this section we describe the simple DAG underlying our construction, and prove state some simple combinatorial properties about it which we'll later need in the security proof and to argue efficiency.

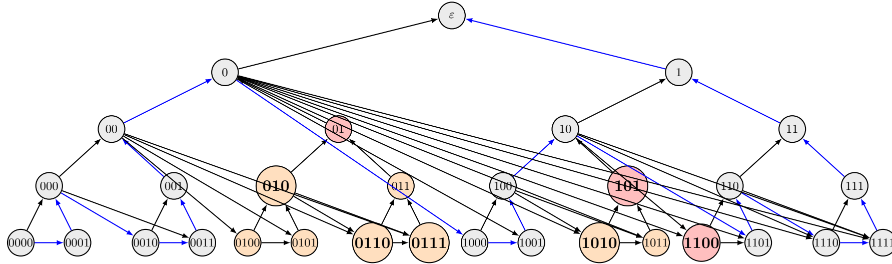

Fig. 3. Illustration of G_4^{PoSW}. The set $S^* = \{01, 101, 1100\}$ – which e.g. could be derived from $S = \{010, 0110, 0111, 101, 1010, 1100\}$ – is shown in red. D_{S^*} is the union of red and orange nodes. $\hat{S} = \hat{S}^*$ are the orange or red leaves. The path of length $2N - 1 - |B_S| = 32 - 1 - 11 = 19$ (as constructed in the proof of Lemma 4) is shown in blue.

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 *above* u if $u = v\|a$ for some a (then u is *below* v). 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

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

Lemma 3. *The labels of G_n^{PoSW} can be computed in topological order using only $w \cdot (n+1)$ bits of memory.*

Proof. The proof of the lemma follows by induction: to compute the labels of G_n^{PoSW}, start by computing the labels of the left subtree L, which is isomorphic to G_{n-1}^{PoSW}. Once the last label ℓ_0 of L is computed, keep only this one label. Now compute the labels of the right subtree R, which is also isomorphic to G_{n-1}^{PoSW}, except that it has some incoming edges from the left subtree. As all these edges start at ℓ_0, one can compute the labeling of this graph with just w extra bits of space. Once the last label ℓ_1 of R is computed, delete all labels except ℓ_0, ℓ_1, and compute the label of the root $\ell_\varepsilon = \mathsf{H}(\varepsilon, \ell_0, \ell_1)$. The memory required to compute the labels of G_n^{PoSW} is thus the memory required for G_{n-1}^{PoSW} plus w bits (to store ℓ_0 while computing the right subtree). G_1^{PoSW} just has 3 nodes and so can trivially be computed with $3 \cdot w$ bits. Solving this simple recursion gives the claimed bound. □

Definition 4 **($\hat{S}, S^*, D_S$).** *For a subset $S \subseteq V$ of nodes, we denote with $\hat{S}$ the set of leaves below S*

$$\hat{S} = \{v\|u \in \{0,1\}^n \;:\; v \in S, u \in \{0,1\}^{n-|v|}\}$$

We denote with S^* the minimal set of nodes with exactly the same set of leaves as S, i.e., $S^* \subseteq V$ is the smallest set satisfying $\hat{S}^* = \hat{S}$.

We denote with D_S all the nodes which are in S or below some node in S

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

Lemma 4. *The subgraph of $G_n^{\mathsf{PoSW}} = (V, E)$ on vertex set $V - D_{S^*}$ (for any $S \subseteq V$) has a directed path going through all the $|V| - |D_{S^*}| = N - |D_{S^*}|$ nodes.*

Proof. The proof is by induction on n, an example path is illustrated in Figure 3. The lemma is trivially true for G_0^{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 isomorphic to G_i^{PoSW}, with extra edges going from the root of L – which is 0 – to all the leaves in R. 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 $R \equiv G_i^{\mathsf{PoSW}}$, and add an extra last edge $1 \to \varepsilon$. If $0 \notin S^*$, apply the Lemma first to $L \equiv G_i^{\mathsf{PoSW}}$ to get a path that ends in its root 0, then – if $1 \notin S^*$ – apply the lemma to $R \equiv G_i^{\mathsf{PoSW}}$, to get a path that starts at a leaf v. Now add the edges $0 \to v$ and $1 \to \varepsilon$. If $1 \in S^*$ we just add the edge $0 \to \varepsilon$. □

Lemma 5 (trivial). *For any $S^*, S \subset V$, D_{S^*} contains*

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

many leaves.

Proof. Let $S^* = \{v_1, \ldots, v_k\}$, using that $D_{v_i} \cap D_{v_j} = \emptyset$ for all $i \neq j$ (as otherwise S^* would not be minimal), we can write

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

As each D_{v_i} is a full binary tree it has $(|D_{v_i}| + 1)/2$ many leaves, so

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

□

5 Our Construction

In this section we specify our PoSW based on the graphs G_n^{PoSW}.

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

$$M = (t + n \cdot t + 1 + 2^{m+1})w$$

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 $n \approx \log N$ and $m \approx \log M$ are basically the logarithms of the time parameter N and the memory M (measured in w bit blocks) 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. 1) of the graph G_n^{PoSW} (as defined in §4) using H_χ. It stores the labels $\phi_\mathcal{P} = \{\ell_i\}_{i\in\{0,1\}^{\leq m}}$ of the m highest layers, and sends the root label $\phi = \ell_\varepsilon$ to $\mathcal{V}$.

$\tau := \mathsf{open}^{\mathsf{H}_\chi}(N, \phi_\mathcal{P}, \gamma)$: on challenge $\gamma = (\gamma_1 \ldots, \gamma_t)$, τ contains – for every $i, 1 \leq i \leq t$ – 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.,

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

and

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

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

If $m = n$, $\mathcal{P}$ stores 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)

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

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*

$$1 - (1-\alpha)^t - \frac{2 \cdot n \cdot w \cdot q^2}{2^w}$$

So, for example setting the statistical security parameter to $t = 21$, 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. The exponentially small $2 \cdot n \cdot w \cdot q^2/2^w$ 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. The concrete bound follows from Lemmas 1 and 2 (recall that H only takes inputs of length $\leq (n+1)w$).

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 $S \subseteq V = \{0,1\}^{\leq n}$ denote all inconsistent nodes. Then by Lemma 4 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 an 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

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

By Lemma 5 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 produce a valid proof given 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, i.e., $\gamma \cap \hat{S} \neq \emptyset$, or equivalently

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

By Eq.(2), and using that every γ_i is uniform

$$\Pr[\gamma_i \notin D_{S^*}] = 1 - |\{0,1\}^n \cap D_{S^*}|/2^n < 1 - \alpha$$

and as the γ_i are also independent

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

so $\tilde{\mathcal{P}}$ will fail to generate a valid proof with probability $> 1 - (1-\alpha)^t$ as claimed. $\square$

5.4 Efficiency

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

Proof Size The exchanged messages χ, ϕ, γ, τ are of length (we need w bits to specify a label and n bits to specify a node)

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

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 less than 200KB.

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 $(n+1)\cdot w$ bits, and on average about $1/4$ of that (for comparison, the construction from [MMV13] has inputs of length $n^2 \cdot \text{polylog}(n) \cdot w$).

$\mathsf{open}^{\mathsf{H}_\chi}(N, \phi, \gamma)$: Here the efficiency depends on m, which specifies the size of the memory $M = (n + 1 + n \cdot t + 2^{m+1})w$ we allow $\mathcal{P}$ to use. Here $w \cdot n \cdot t$ bits are used to store the values in τ to send back, $(n+1) \cdot w$ bits are used to compute the label (cf. Lemma 3), 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.

5.5 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 and Open Problems

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 is only $O(\log(N))$. 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 $2^{n/2} \approx \sqrt{N}$ 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 $(1-\alpha)2^n$.

Apart from [MMV13], depth-robust graphs have been used for cryptographic applications in at least one other case, namely to construct memory-hard functions [ABP17]. Moreover the *proofs of space* protocol from [DFKP15] is quite similar to the PoSW from [MMV13], the main difference being that the underlying graph does not have to be depth-robust, but needs to have high space complexity. Due to this similarities, it seems conceivable that using ideas from this work one can get improved constructions for memory-hard functions and proofs of space.

Let us also mention the third open question asked by [MMV13]. It asks whether a PoSW based only on random oracles can be used to achieve fairness in protocols like coin tossing. We refer to their paper for the details, and just mention that to achieve this, it's sufficient to construct a PoSW with a "unique" proof (note that we already mention this problem in the related work section §1.4). That is, we not only require that to generate a proof one needs to spend sequential time, but for every input (statement and time parameter), it should be hard to come up with two different valid proofs. Such a property would also be very useful in other contexts, like for constructing blockchains, which was the main motivation for this work.

Unfortunately, our construction also does not have unique proofs. It's an intriguing open problem to construct a PoSW with unique proofs and an exponential gap between proof generation and proof verification. Currently, the publicly verifiable function with the largest gap between computation and verification is the sloth function [LW17], which is based on the assumption that computing square roots in a field of size p takes $\log(p)$ times longer than the inverse operation, i.e., squaring. Under this assumption, the gap is $\log(p)$, in practice one would probably use something like $\log(p) \approx 1000$. Sloth is not a time-lock puzzle (as discussed in §1.4), as one can't sample an input together with its output. It's also not a good PoSW as there's no further speedup if we only want to verify that a lot of sequential time has been spend on the computation, not correctness. Let us also mention that sloth, as well as our PoSW (but not [MMV13]) allow for a speedup of q in verification time if parallelism q is allowed and the proof can be of size linear in q. Basically one adds q "checkpoints" to the proof. These are intermediate states that appear during the computation, and one can verify that each two consecutive checkpoints are consistent independently.

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.

[Dam90] Ivan Damgård. A design principle for hash functions. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 416–427. Springer, Heidelberg, August 1990.

[DFKP15] Stefan Dziembowski, Sebastian Faust, Vladimir Kolmogorov, and Krzysztof Pietrzak. Proofs of space. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 585–605. Springer, Heidelberg, August 2015.

[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.

[LW17] Arjen K. Lenstra and Benjamin Wesolowski. Trustworthy public randomness with sloth, unicorn, and trx. *IJACT*, 3(4):330–343, 2017.

[May93] Timothy C. May. Timed-release crypto. http://www.hks.net/cpunks/cpunks-0/1460.html, 1993.

[Mer90] Ralph C. Merkle. A certified digital signature. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 218–238. Springer, Heidelberg, August 1990.

[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.

[RSW00] Ronald L. Rivest, Adi Shamir, and David A. Wagner. Time-lock puzzles and timedrelease crypto. Technical report, 2000.