Proofs of Sequential Work

Meena Jagadeesan Alec Sun Alex Wei

May 25, 2018

Abstract

We begin with a survey of constructions of *proofs of sequential work* based on the work of Mahmoody, Moran, and Vadhan [MMV13] and Cohen and Pietrzak [CP18]. Proofs of sequential work are now relevant to blockchain design, but were originally motivated by the study of time-lock puzzles. We discuss one application of time-lock puzzles, the construction of fair coin flipping protocols. We also detail existing constructions by Boneh and Naor [BN00] as well our simplification of this protocol based on ideas in a work of Jershow and Mauve [JM10]. Unfortunately, these constructions use non-standard assumptions. If proofs of sequential work had unique proofs, then they would provide a construction of a fair coin-flipping protocol with standard assumptions. An open problem posed in [CP18] is to construct proofs of sequential work with unique proofs. We outline our high-level ideas on extending the constructions of [CP18] and [MMV13] to give the guarantee of unique proofs.

1 Introduction

In [MMV13], Mahmoody, Moran, and Vadhan introduce *proofs of sequential work* (PoSW), which is a protocol between a prover and a verifier, where the prover must prove that they have spent at least N *sequential* computational time for a security parameter N. In these schemes, the sequential condition is derived from wanting a proof that cannot be generated in time much less than N even when the adversary has a large number of processors to perform parallel computations at their disposal. Proofs of sequential work have also been discussed in the context of *time-lock puzzles*, which are puzzles that can be "unlocked" after a certain amount of sequential computational time has been spent.

The first known protocol for proofs of sequential work relying on standard assumptions is due to Mahmoody, Moran, and Vadhan [MMV13], who provide a publicly verifiable protocol in the random oracle model using depth-robust "hash graphs." Alwen et al. [ABP18] improved the construction of depth-robust graphs, improving the bounds in [MMV13]. In recent work, Cohen and Pietrzak [CP18] give a simpler and more efficient construction of proofs of sequential work in the random oracle model using a similar idea of hash graphs, but without the depth-robust requirement. In this survey, we will approach proofs of sequential work by surveying the constructions given by the first two papers [MMV13, CP18] in detail.

The study of proofs of sequential work was originally motivated by the study of time-lock puzzles in [CLSY93, RSW96, BN00, JM10], which are puzzles that require a certain amount of time to "unlock" while being secure against parallel attack. Cai, Lipton, Sedgewick, and Yao [CLSY93] apply this idea to CPU benchmarks that cannot be cheated, since the time required to execute the sequence of computations must be proportional to CPU speed. Rivest, Shamir, and Wagner [RSW96] consider time-lock puzzles in the context "of send[ing] messages to the future" and give a construction relying on an assumption of inherent sequentiality of exponentiation modulo an RSA modulus. Boneh and Naor [BN00] show how the same time-lock scheme in [RSW96] can also be applied to constructing a fair coin flipping scheme, which we will discuss in greater detail below. Jerschow and Mauve [JM10] discuss how offline submission of articles can also be achieved with the correct implementation of time-lock puzzle. Certain forms of time-lock puzzles can be obtained from proofs of sequential work as described in [MMV13, CP18], but others, e.g., those requiring unique proofs, will need a different construction. Proofs of sequential work now have even greater relevance today since they can be applied to blockchain designs, as is discussed by Cohen and Pietrzak [CP18].

Right now, the proof of work for blockchain is dependent on the world's current parallelizable computational power. The difficulty of finding a conforming hash to continue the evolution of the blockchain is a function of the number of participants and the speed of the equipment used to calculate the hash. Hence the bitcoin difficulty must adjust dynamically so that a proper hash is found on average once every ten minutes. One could imagine that using proofs of sequential work that the time between blocks would be independent of the total number of computers in the world since finding a solution is now limited by a sequential queries by one machine. This modification would make blockchain time more closely correspond to real world time without the need to dynamically adjust the bitcoin difficulty.

One application of time-lock puzzles [BN00, JM10], that we will explore in greater detail, is to the construction of fair coin flipping protocols, in which two parties (without a third party) want to obtain a shared random bit, with the added caveat that if one party aborts, the other party can "force open" a commitment to complete the protocol. For a time-lock puzzle to be used in such a protocol requires it to have a unique unlocking, or at least be secure against "second preimage attacks." However, existing constructions in [BN00] use a non-standard assumption, that exponentiation mod a large prime, is "inherently sequential." In contrast, the proof of sequential work constructions in [MMV13] and in [CP18] use standard assumptions but have the caveat that each puzzle has many possible solutions. The construction of a proof of sequential work protocol with unique proofs remains an open problem posed in [MMV13]. Such a construction would lead to a fair coin-flipping protocol based on standard assumptions.

In Section 2, we present a survey of the proof of sequential work protocols in [CP18] and [MMV13]. We also discuss improvements on the protocol in [MMV13] if the depth-robust graphs construction is replaced by an improved construction from a recent work by Alwen, Blocki, and Pietrzak [ABP18]. In Section 3, we discuss the construction in [BN00] that uses a number-theoretic assumption to construct a coin-flipping protocol. We also present our simplification of this protocol using the ideas in [JM10] that relies on the same non-standard assumption. In Section 4, we discuss our high-level ideas to adapt the construction of [MMV13] to have unique proofs, which could potentially provide a coin flipping protocol based on standard assumptions.

2 Survey of Approaches of [CP18] and [MMV13]

We consider constructions of *proofs of sequential work* by Mahmoody, Moran, and Vadhan [MMV13] and by Cohen and Pietrzak [CP18]. One issue with the construction in [MMV13] is that the prover in addition to needing N time steps is also forced to have $\Theta(N)$ space to generate a proof. The results in [CP18] solve this issue by producing a simpler and more efficient scheme that reduces the required space to $O(\log N)$.

2.1 Assumptions

In [MMV13], Mahmoody et al. introduce “inherently sequential” hash functions and show that these hash functions, coupled with collision-resistant hash functions, can be used to instantiate the random oracle in their construction. Although [CP18] focuses on the random oracle model (in which PoSW are easiest to define), they claim that their results can be converted to the same assumptions as in [MMV13]. For completeness, we first sketch a proof for why a random oracle is collision-resistant and sequential.

Lemma 1. *Consider any adversary A which is given access to a random function $\mathsf{H}\colon \{0,1\}^* \to \{0,1\}^w$. If A makes at most queries the probability that 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 queries is at most $\frac{i-1}{2^w}$ because H is a random oracle. By the union bound, the probability that any i collides with a previous output is at most

$$\sum_{i=1}^{q} \frac{i-1}{2^w} \leq \frac{q^2}{2^{w+1}}.$$

□

Definition 2. *An H sequence is a sequence $x_0, \ldots, x_s$ such that $\mathsf{H}(x_i)$ equals x_{i+1} for every i. Such a sequence is called a hash chain of length s.*

Lemma 3. *Any adversary A who makes at most $s-1$ sequential queries to a random oracle cannot produce a hash chain of length s with more than negligible probability. This is equivalent to the random oracle model being sequential.*

Proof. There are essentially two ways A can output a hash chain $x_0, \ldots, x_s$ making only $s-1$ sequential queries.

Case 1: Lucky Guess. Then for some i we have $\mathsf{H}(x_i) = x_{i+1}$ and A did not make the query $\mathsf{H}(x_i)$. Because H is a random oracle, the probability is roughly bounded by a union bound over all i, which is negligible.

Case 2: Collision. In this case the x_i are not computed sequentially. Then it follows that for some $1 \leq i < j \leq s-1$ a query say q_i is made in round i and q_j is made in round j such that $\mathsf{H}(q_j) = q_i$. Again since the query q_j has never been made before and H is a random oracle the probability of such an event is union bounded over all such previous queries q_i and is hence negligible. □

We emphasize that all of the results in [CP18] and [MMV13] do not need the full power of the random oracle model, only the collision-resistant and sequential properties. This will become relevant in our discussion of unique proofs.

2.2 Proofs of Sequential Work

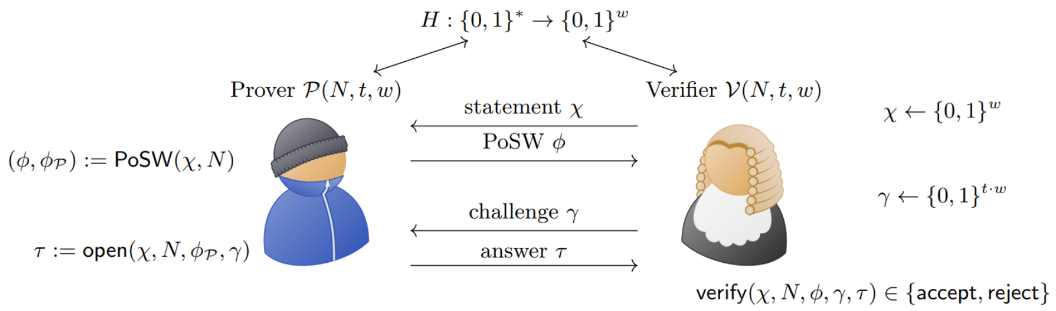
Proofs of sequential work in the random oracle model can be defined more or less as follows: The prover P and the verifier V get as common inputs statistical security parameters w and t and a time parameter N. Both parties have access to the random oracle H. The interaction between P and V is defined by the following game, and is depicted in Figure 1 (borrowed from [CP18]):

- V samples a random w-bit string χ and sends it to P. (Our goal is to force P to work for N time steps from this point forward.)
- P computes a proof $(\phi, \phi_P) := PoSW^H(\chi, N)$, sends ϕ to V and stores ϕ_P.
- V gets to challenge the prover, by sampling a random tw-bit string and sending it to P.
- P computes some function open that may require using ϕ_P and the random string and sends the output to V.
- V verifies that this ϕ and this output are consistent with χ, N, and the challenge string.

The requirements are:

1. **Correctness:** V reports accept with probability 1 if interacting with an honest prover.
2. **Soundness:** Any P who is accepted by V must have made "close" to N queries to H.

The constructions achieve the following guarantees given in the next two theorems.



$H : \{0,1\}^* \to \{0,1\}^w$
Prover $\mathcal{P}(N,t,w)$
Verifier $\mathcal{V}(N,t,w)$
statement χ
PoSW ϕ
challenge γ
answer τ
$(\phi, \phi_{\mathcal{P}}) := \mathsf{PoSW}(\chi, N)$
$\tau := \mathsf{open}(\chi, N, \phi_{\mathcal{P}}, \gamma)$
$\chi \leftarrow \{0,1\}^w$
$\gamma \leftarrow \{0,1\}^{t \cdot w}$
$\mathsf{verify}(\chi, N, \phi, \gamma, \tau) \in \{\mathsf{accept}, \mathsf{reject}\}$

Figure 1: Prover-Verifier Game

Theorem 4. *[MMV13] Let d be the in-degree of the depth-robust graph G. (This construction achieves $d = \log^3 n$ but a new construction achieves $d = \log n$.) The prover uses N queries to H and the verifier uses at most $(d+1)\log_d(N)$ queries, excluding the time needed to construct G.*

Theorem 5. *[CP18] For any $0 \leq m \leq n$, there is a correct and sound PoSW such that the prover uses space $n+1+nt+2^{m+1}w$ and can be done making at most $t(2^{n-m+1}-1)$ queries to H during the open step. The verifier must only sample a random challenge of length tw and verification can be done making tn queries to H.*

2.3 Constructions

Both the construction in [MMV13] and in [CP18] instantiate the private proof ϕ_P as a vertex-labeling of a directed acyclic graph G and the communicated proof ϕ as the label of the top vertex. Let $V = \{0, 1, \ldots, N-1\}$ be the set of vertices. The vertex labels $\{l_i\}_{i \in V}$ satisfy the following invariant. We say that vertex u is a parent of vertex v is a direct edge from u to v. If $(p_1, \ldots, p_d)$ are the parents of vertex i, then

$$l_i = H(\chi, i, p_1, p_2, \ldots, p_d).$$

Here, χ is essentially used to salt the random oracle to prevent precomputation from the prover. Observe that if the vertices are processed in topological order, then the labels can be computed in N sequential queries to the hash function. Roughly speaking, we also want to guarantee that the labels can't be computed in fewer than N sequential queries. This poses some conditions of the structure of the graph G. For example, if G were the full binary tree, then this labeling process could be made highly parallelizable through a divide-and-conquer type strategy. Both the constructions in [MMV13] and [CP18] start with the full binary tree in a Merkle-tree-based commitment scheme with n levels (so that $N = 2^{n+1} - 1$) as the base graph, but take different approaches to augmenting the graph with additional edges. The collision-resistance of H is used to force the prover to commit to unique labels.

Both constructions require as an assumption the existence of a collision-resistant and inherently sequential hash function, hence they do not need the full power of a random oracle model, although one can note a random oracle does satisfy the two properties. Moreover, in [MMV13] it is shown how to construct a collision-resistant and inherently sequential has function H from a collision-resistant hash function H_1 and an inherently sequential hash function H_2. This means that essentially the only non-standard assumption being used in these graph constructions is the existence of an inherently sequential hash function.

The tw-bit string corresponding to the verifier's challenge to the prover corresponds to t challenge leaves $\gamma_1, \ldots, \gamma_t$ in V. In the open function, for each $1 \leq i \leq t$, the prover sends the label l_{γ_i}, the labels of the parents of γ_i, as well as the labels of all siblings of the vertices on the path from γ_i to the root. For each $1 \leq i \leq t$, the verifier has enough information to confirm that the labels of the parents of γ_i satisfy the desired labeling property, as well as to confirm that the labeling is consistent with the labeling ϕ of the root.

In both constructions the verification time is very small. If N is the number of sequential queries to the hash function and n is the security parameter, then both graphs can be verified publicly in time $poly(n) \cdot polylog(N)$.

2.3.1 Construction in [MMV13]

The key ingredient in [MMV13] is a depth-robust graph. At a high-level, depth robust graphs satisfy the property that even if many vertices are removed, the graph still has a long path. This property is useful since even if the prover cheats on a small number of labels, he will still require a number of sequential queries due to the long path. More formally, a directed acyclic graph G is (e, d) depth-robust if for any subset $S \subseteq V$ such that $|S| \leq e$, the vertex-induced subgraph on $V \backslash S$ has a path of length at least d. While the complete DAG is $(e, |V| - e)$ depth-robust for any e, it is insufficient for this setting where we want the graph to have low in-degree so that the verifier can perform its check in $polylog(n)$ time for each challenge. Erdos et. al [EGS75] showed the existence of $(\Theta(N), \Theta(N))$ depth-robust DAGs with in-degree $O(\log N)$. In [MMV13], Mahmoody et al. give an explicit construction with in-degree $O(\log^3 N)$ with better constants on the guarantees on the values of e and d. However, a recent paper by Alwen et. al [ABP18] constructed explicit depth-robust graphs with in-degree $O(\log N)$.

The depth-robust graph is used to add edges to the full binary tree. The vertex set of the depth-robust graph is taken to be the leaves of the full binary tree. Figure 2 (borrowed from [CP18]) shows an example of this. Intuitively, the depth-robust graph serves to add inter-dependencies between the branches which prevent the divide-and-conquer strategy from being realizable. The security proof splits into two cases depending on the number of labels that are inconsistent. Here, inconsistent means that the label is not correctly computed from the labels of its parent according to H. If $\beta \leq e$ (where e is the parameter of the depth-robust graph), then the prover will be caught in the verification step with high probability $1 - \left(\frac{N-\beta}{N}\right)^t$. If $\beta < e$, then since H is sequential,

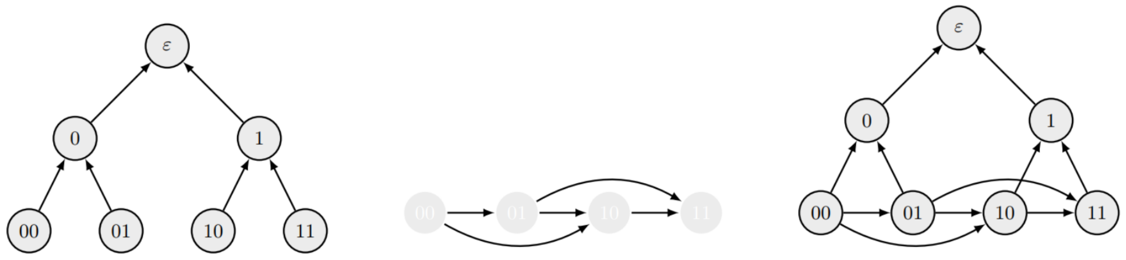


Figure 2: Construction in [MMV13]

the number of sequential queries that the prover has to make is at least the length of the longest path in $V \setminus D$, which is at least d by the depth-robust guarantees.

One drawback of this construction is the space expense the prover must necessarily face to label the depth-robust graph. The same paper by [ABP18] shows that if a depth-robust graph on N vertices is labeled in time T using space S, then $TS = \Omega(N^2)$. This trade-off stems from the fact that depth-robust graphs are used in a construction of memory-hard functions, which are functions that require a large amount of memory in many steps even if the adversary can make parallel queries to the random oracle. The main technique used in showing the lower bound is the construction of a pebbling game on the graph which takes advantage of depth-robustness of the graph. The lower bound implies that the prover operates in $O(N)$ time, linear space is required. Although the proof of sequential work seeks to give guarantees on the amount of sequential time spent by the prover, it may not be the case we want to force the prover to use a large amount of space.

2.3.2 Construction in [CP18]

The construction in [CP18] achieves better space guarantees for the prover. The main idea is to move away from depth-robust graphs and use a differently structured graph to give sequential guarantees. This construction adds the following edges to the full binary tree: edges (v, u) such that u is a leaf vertex and v is a left sibling of a vertex on the path from the u to the root. Figure 3 (borrowed from [CP18]) shows an example of this construction. While the graph is not depth-robust, it contains a path of length $2N - 1$, if N is the number of leaves.

In fact, the fact that the graph is not depth-robust enables the label processing to circumvent the lower bounds in [ABP18]. The nodes can be computed in topological order while keeping only logarithmically many labels in memory at any given point. This stems from the recursive nature of the construction of the graph. More formally, it takes $w(n + 1)$ bits of memory to compute the labels. We prove this by induction. It takes wn bits of memory to compute the left subtree. Then all of the memory is deleted except the label of the root of the left subtree which takes w bits of memory,

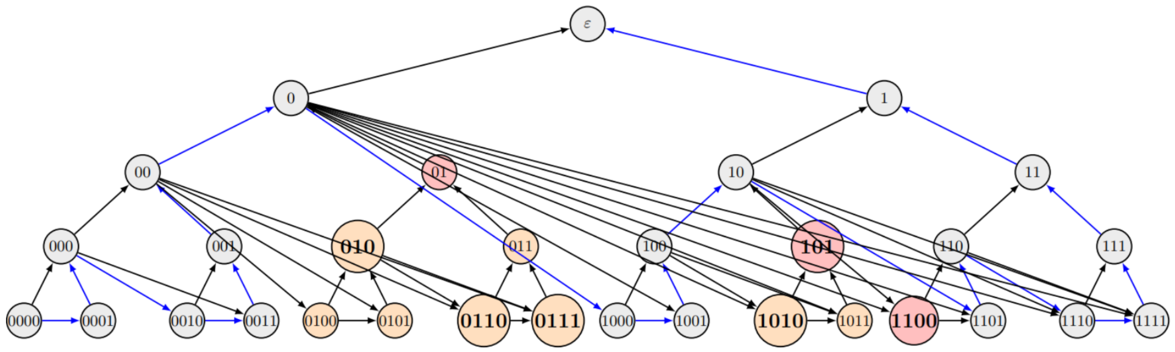


Figure 3: Construction in [CP18]

and the right subtree labels are computed using wn more bits of memory, which adds up to $w(n+1)$ bits in total.

However, there is a caveat. If using only logarithmic memory in our construction, the prover needs to recompute all the labels in the challenge phase, whereas one would not need any computation (just some lookups) in this phase if all of the labels were stored as ϕ_P. 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, in [CP18], some trade-offs between memory and challenge phase time are shown. The basic idea to pick some some $0 \leq m \leq n$ where n is the total number of levels of the tree. For ϕ_P, all of the labels for the $2^{m+1}-1$ vertices in the m up-most levels are stored.With this, one only needs to compute $\leq 2^{n-m+1}-1$ labels per challenge. We consider various parameter settings for m. If $m = 0$, then all N labels need to be computed, so an honest prover will require $2N$ queries, which is unfortunate. If $m = n$, then the whole tree is stored and no additional labels need to be computed; however, this means that $O(N)$ memory is required. In the middle case where $m = n/2$, observe that about $\sqrt{N}w$ bits of memory are used and about $\sqrt{N}w$ additional queries are needed in the challenge phase.

The proof of security in [CP18] makes use of the long paths present in the graph construction.

Proof. Let S be the set of vertices that are labeled inconsistently (that is these labels do not match the labels of their parents according to H). Then, we split into two cases based on the size of a set of vertices closely related to S.

Let D_S be the set of vertices in S or below some vertex in S. Let S^* be the minimum set of vertices such that the set of leaves below S^* is the same as the set of leaves below S. The set that we will use to gauge the "size" of S in our proof of security is D_{S^*}. First, it follows from the recursive structure of the graph that the vertex-induced subgraph on $V \setminus D_{S^*}$ contains a directed path through all of its vertices. Next, we show that D_{S^*} contains $\frac{|D_{S^*}|+|S^*|}{2}$ leaves. Let $S^* = \{v_1, \ldots, v_{|S|}\}$. Observe that since S^* is minimal, we know that the D_{v_i} are distinct. Moreover, each D_{v_i} has $\frac{|D_{v_i}|+1}{2}$ leaves, giving the desired result.

Now, we use these properties to prove security. Observe that all of the vertices in $V \setminus D_{S^*}$ are consistent. If $|D_{S^*}| \leq \alpha N$, then since H is sequential, the prover must have made $(1-\alpha)N$ sequential queries. If $|D_{S^*}| > \alpha N$, then we use the fact that $|D_{S^*}|$ contains a significant number of leaves so the prover will fail the challenge phase with high probability. This relies on the fact that if a leaf in in D_{S^*}, then it will hit a vertex in S (an inconsistent vertex) on the path to the root. If this happens, then since H is sequential, with high probability, the last inconsistent vertex on the path will not be consistent with respect to the "correct values" of the previous vertices which were not queried. Thus, this inconsistency will be discovered by the verifier. This implies that since there are $\frac{|D_{S^*}|+|S^*|}{2} > \alpha 2^n$, each challenge succeeds with probability $< 1-\alpha$. Thus, the total failure probability is at least $1-(1-\alpha)^t$. $\square$

3 Coin-Flipping Protocol

We now focus on one application of time-lock puzzles, namely constructing a fair coin flipping protocol between two parties. The problem we consider [BN00] is that of defining a coin flipping protocol two parties, Alice and Bob, satisfying:

(i) the value of the coin flip has negligible bias;

(ii) even if one of the parties does not follow the protocol, the value of the coin flip is still well-defined.

A protocol satisfying these properties improves on the standard commitment scheme-based protocol for fair coin flipping in that in such a protocol, neither party can abort if they realize the result of the coin flip will come out unfavorably. In [BN00], a construction of this protocol using a commitment variant of the time-lock puzzle described in [RSW96] is given. In Section 3.1, we describe our simplified variant of their construction based on the work of [JM10].

For this construction, we use a version of a time-lock puzzle known as a *timed commitment*. A timed commitment is a commitment from Alice to Bob for a string $S \in \{0,1\}^n$, such that Alice sends a commitment string C to Bob, and then Alice can prove later to Bob that she had indeed committed the string S before sending C. Furthermore, Bob also has the option of a *forced opening*, where after spending at least T sequential computational time, Bob is able to force open the commitment and obtain S. In particular, this protocol should be secure against (i) Alice cheating and providing a proof for a different string that was not her original commitment and (ii) Bob cheating by forcing the commitment open in time less than T. That is, both of these attacks should succeed with negligible probability. Assuming the existence of a timed commitment scheme, consider the following protocol:

- Alice chooses a uniformly random bit b_A and sends a timed commitment c_{b_A} to Bob.

- Bob chooses a uniformly random bit b_B and sends it to Alice.
- Alice verifies that b_B arrived quickly enough (before T time has passed), and if so, she reveals her commitment b_A to Bob.
- The players' output is $b_A \oplus b_B$.

The timed commitment is useful in that if Alice aborts after receiving b_B, Bob can force open her commitment to obtain b_A and finish the protocol. Now, observe that Bob cannot bias the output of the coin flip without forcing the commitment open early, which succeeds with at most negligible probability. Similarly, Alice cannot bias the output of the commitment without either (i) cheating in the opening step or (ii) aborting before the opening the step, the latter of which can be averted by Bob's forced opening of the commitment.

3.1 A Timed Commitment Scheme

In addition to describing this protocol, [BN00] also give a construction of a timed commitment based off of the time-lock puzzle construction in [RSW96], which relies on the assumption of exponentiation modulo a RSA modulus being inherently sequential. However, this construction is somewhat complicated. We describe our similar, but simpler, construction relying on the same assumption that is based on [JM10]. The timed commitment scheme for a bit b that Alice knows proceeds as follows:

- First, Alice generates two large primes p and q of equal bit-length. Next, Alice computes the modulus $N = pq$ and $\phi(N) = (p-1)(q-1)$. Alice also draws a function f from a pseudorandom function family mapping strings of $|p|$ to $\{0, 1\}$. Finally, Alice fixes a time parameter t that is proportional to the minimum number of operations Bob must take to force open her commitment.
- After the generation step, Alice chooses a random e such that $f(2^e) = b$. Let r be the remainder of 2^t modulo N. Set $\tilde{e} = 2^t + (\phi(N) - r) + e$. Alice's commitment is then the ordered tuple $(N, f, t, \tilde{e})$.
- To unlock the commitment, Alice simply tells Bob the factorization $N = pq$. With this information, Bob can compute $\phi(N)$ and therefore efficiently compute $f(2^{\tilde{e}}) = f(2^e)$.
- If instead Bob wants to force open the commitment, he has the option to compute $2^{\tilde{e}}$ in $\Theta(\tilde{e})$ rounds of repeated squaring, which takes time $\Theta(t)$.

For the proof of correctness, note first that this commitment is binding because $f(2^{\tilde{e}})$ is well-defined, and this must be exactly the bit that Alice committed. To see that Bob cannot break this scheme efficiently, we reduce the assumption of [RSW96] that computing 2^{2^t} modulo an RSA prime is inherently sequential. Bob is given $(N, f, t, \tilde{e})$. Since e is effectively random (since f is indistinguishable from a random function), Bob should only be able to deduce information about $\phi(N) - r$ from $\tilde{e}$. However, knowledge of $\phi(N) - r$ would immediately break the assumption in [RSW96], since $2^{\phi(N)-r} = 2^{2^t}$. Thus, we have shown that this timed commitment scheme is secure assuming the original time-lock puzzle of [RSW96] is secure.

Similar to the PoSW constructions described earlier, the time for Bob to unlock the commitment given that Alice tells him the correct factorization is fast, only $O(\log t)$. If Alice aborts, Bob still takes a reasonable amount of time $\Theta(t)$ to unlock Alice's commitment. This separation between unlock times in the non-abort and abort cases is essential for constructing a time-lock puzzle.

3.2 Relationship to Constructions of [MMV13] and [CP18]

Timed commitments highlight one way in which the recent constructions of proofs of sequential work do not cover all use cases of time-lock puzzles. An important aspect of timed commitments is that they are binding, i.e., for a commitment S, Alice cannot give a proof that her commitment is actually $S' \neq S$. The same binding property does not hold for proofs of sequential work—for these hash graph constructions, it is possible to "cheat" on a small number of vertices and not have these be detected in the verification step. This potential for cheating, however, allows the commitment holder Alice to change her commitment after the fact. This motivates the question of asking whether it is possible to have proofs of shared work where the proof is unique, or at least it is difficult to find a second proof for the same puzzle.

4 Open Problems

4.1 Assumptions

One further exploration could be to see if the assumptions for PoSW and time-lock puzzles can be relaxed. Currently the instantiation of PoSW uses a random oracle and our construction of a coin-flipping protocol using timed commitment schemes uses the number theoretic assumption that the computation of 2^{2^l} modulo an RSA modulus is inherently sequential.

4.2 Sloth Function

The "sloth" function, as discussed in [CP18], is based on the assumption that computing square roots in a field with p elements takes around $\log p$ longer than squaring. This is currently the publicly verifiable function with the largest known gap between computation and verification, and it is an open problem to find a PoSW with both unique proofs and an exponential gap between proof generation and proof verification. However, one of the downsides is that because one cannot sample an input together with an output, the sloth function does not constitute a time-lock puzzle.

4.3 Unique Proofs

In order to make proofs of sequential proof applicable to coin-flipping protocols, it would be useful to construct proofs of sequential work with unique proofs, as is posed as an open problem in [CP18]. This would guarantee that not only does constructing the proof require sequential time, but for every statement and time parameter, it should be hard to come up with two valid proofs. This property could be useful in blockchain design and also would be useful in providing another instantiation of the coin-flipping protocol.

The barrier prohibiting the constructions in [MMV13] and [CP18] from achieving unique proofs is that due to the recursive labeling protocol, labeling one (or a small number) of vertices incorrectly will not be detected by the verifier, but will change the label of the top vertex and thus the proof ϕ sent to the verifier. If the verifier could also perform N sequential queries, then this problem could be rectified if the verifier also computed ϕ_P and ensured the top label matched with the prover's proof ϕ. However, we do not want to force the verifier to also query the random oracle N times.

One possible approach is for the verifier to have a "trapdoor" method of computing the ϕ which does not require N sequential queries. This would require moving away from the random oracle to some sort of structured hash family. Perhaps rather than the prover using 0^n as a seed label in ϕ_P, the verifier gives a specific binary string s to the prover for which he knows the corresponding top label ϕ through a trapdoor. Such trapdoor constructions might also be shortcuts to generating the solutions of time-lock puzzles.

One potential family of hash functions that may be relevant are chameleon hash functions [KR97]. These hash functions have a public key and private key. With the public key, the hash functions can be evaluated and are collision-resistant. With the private key, however, collisions can be found. One type of hash family that may be relevant to constructing unique proofs is a hash family that is sequential given a public key but not sequential given the private key. This could give the verifier the necessary trapdoor to avoid having to do N sequential work.

References

[ABP18] Joël Alwen, Jeremiah Blocki, and Krzysztof Pietrzak. Sustained space complexity. In *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II*, pages 99–130, 2018.

[BN00] Dan Boneh and Moni Naor. Timed commitments. In *Advances in Cryptology - CRYPTO 2000, 20th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2000, Proceedings*, pages 236–254, 2000.

[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.

[CP18] Bram Cohen and Krzysztof Pietrzak. Simple proofs of sequential work. In *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II*, pages 451–467, 2018.

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

[JM10] Yves Igor Jerschow and Martin Mauve. Offline submission with RSA time-lock puzzles. In *10th IEEE International Conference on Computer and Information Technology, CIT 2010, Bradford, West Yorkshire, UK, June 29-July 1, 2010*, pages 1058–1064, 2010.

[KR97] Hugo Krawczyk and Tal Rabin. Chameleon hashing and signatures, 1997.

[MMV13] Mohammad Mahmoody, Tal Moran, and Salil P. Vadhan. Publicly verifiable proofs of sequential work. In *Innovations in Theoretical Computer Science, ITCS '13, Berkeley, CA, USA, January 9-12, 2013*, pages 373–388, 2013.

[RSW96] R. L. Rivest, A. Shamir, and D. A. Wagner. Time-lock puzzles and timed-release crypto. Technical report, Cambridge, MA, USA, 1996.