A Note on the (Im)possibility of Verifiable Delay Functions in the Random Oracle Model

Mohammad Mahmoody* Caleb Smith* David J. Wu*

Abstract

Boneh, Bonneau, Bünz, and Fisch (CRYPTO 2018) recently introduced the notion of a *verifiable delay function* (VDF). VDFs are functions that take a long *sequential* time T to compute, but whose outputs $y := \mathsf{Eval}(x)$ can be quickly verified (possibly given a proof π that is also computed along $\mathsf{Eval}(x)$) in time $t \ll T$ (e.g., $t = \mathrm{poly}(\lambda, \log T)$ where λ is the security parameter). The first security requirement on a VDF asks that no polynomial-time algorithm can find a convincing proof π' that verifies for an input x and a different output $y' \neq y$. The second security requirement is that that no polynomial-time algorithm running in *sequential* time $T' < T$ (e.g., $T' = T^{1/10}$) can compute y. Starting from the work of Boneh et al., there are now multiple constructions of VDFs from various algebraic assumptions.

In this work, we study whether VDFs can be constructed from ideal hash functions as modeled in the random oracle model (ROM). In the ROM, we measure the running time by the number of oracle queries and the sequentiality by the number of *rounds* of oracle queries it makes. We show that *statistically-unique* VDFs (i.e., where no algorithm can find a convincing different solution $y' \neq y$) cannot be constructed in the ROM. More formally, we give an attacker that finds the solution y in $\approx t$ *rounds* of queries and asking only $\mathrm{poly}(T)$ queries in total.

1 Introduction

A verifiable delay function (VDF) [BBBF18] $f\colon \mathcal{X} \to \mathcal{Y}$ is a function that takes long *sequential* time T to compute, but whose output can be efficiently verified in time $t \ll T$ (e.g., $t = \mathrm{poly}(\lambda, \log T)$ where λ is a security parameter). More precisely, there exists an evaluation algorithm Eval that on input $x \in \mathcal{X}$ computes $f(x) \in \mathcal{Y}$ and a proof π in time T. In addition, there is a verification algorithm Verify that takes as input a domain element $x \in \mathcal{X}$, a value $y \in \mathcal{Y}$, and a proof π and either accepts or rejects in time t. In some cases, a VDF might also have a setup algorithm Setup which generates a set of public parameters pp that is provided as input to Eval and Verify.[1] We can distinguish between VDFs with "slow setup" where Setup can run in time $\mathrm{poly}(T)$ and ones with "fast setup" where Setup runs in time $\mathrm{poly}(t)$. The two main security requirements for a VDF are (1) *uniqueness* which says that for all inputs $x \in \mathcal{X}$, no adversary running in time $\mathrm{poly}(\lambda, T)$ can find $y' \neq f(x)$ and a proof π such that $\mathsf{Verify}(x, y', \pi) = 1$; and (2) *sequentiality* which says that no adversary running in *sequential time* $T' < T$ can compute $y = f(x)$.

Verifiable delay functions have received extensive study in the last year, and have found numerous applications to building randomness beacons [BBBF18,EFKP19] or cryptographic timestamping schemes [LSS19].

*University of Virginia. Emails: {mohammad, caleb, dwu4}@virginia.edu

[1] Ideally, the public parameters can be sampled by a public-coin process [BBBF18, Wes19, Pie19]. Otherwise, we require a trusted setup to generate the public parameters [FMPS19, Sha19].

Driven by these exciting applications, a sequence of recent works have developed constructions of verifiable delay functions from various algebraic assumptions [Wes19, Pie19, FMPS19, Sha19]. However, existing constructions still leave much to be desired in terms of concrete efficiency, and today, there are significant community-driven initiatives to construct, implement, and optimize more concretely-efficient VDFs [Chi19]. One of the bottlenecks in existing constructions of VDFs is their reliance on structured algebraic assumptions (e.g., groups of unknown order [RSA78, BBHM02]).

A natural question to ask is whether we can construct VDFs generically from *unstructured* primitives, such as collision-resistant hash functions or one-way functions. In this work, we study whether black-box constructions of VDFs are possible starting from hash functions or other symmetric primitives. Specifically, we consider black-box constructions of VDFs from ideal hash functions (modeled as a random oracle). Similarly to previous work (e.g., see [MMV11, AS15]) in the random oracle model (ROM), we measure the running time of the adversary by the number of oracle queries the adversary makes and the sequentiality of the adversary by the number of *rounds* of oracle queries it makes.

Our results. In this work, we rule out the existence of *statistically-sound* VDFs (i.e., VDFs where for any $x \in \mathcal{X}$, no algorithm can find (y', π) such that $\mathsf{Verify}(x, y', \pi) = 1$ and $y' \neq f(x)$) in the random oracle model. Specifically, we construct an adversary that breaks the uniqueness of any statistically-sound VDF that asks $O(t)$ rounds of queries and a total number of $\mathrm{poly}(T)$ queries. We also observe that in the tight regime of sequentiality (e.g., requiring an adversary to need sequential time $T' \gg T \cdot (1 - 1/t)$), even *proofs of sequential work* (PoSW) [MMV13] cannot be based on random oracles. PoSW is a relaxation of VDF in which the uniqueness property is not needed. Therefore, the lower bound also applies to VDFs as well. We note, however, that since (even publicly verifiable) PoSW with more relaxed sequentially (e.g., $T' = T/2$) are known [MMV13], it is not clear whether this lower bound for PoSW can be extended to VDFs as well or not.

At a technical level, the proof of our first lower bound relies on the the techniques of Mahmoody, Moran, and Vadhan [MMV11] for ruling out time-lock puzzles in the random oracle model. In fact, for a special case of statistically-unique VDFs where the VDF function is a permutation on its input domain, which we refer to as permutation-VDF (see also [KJG+16, AKK+19]), we use the proof of [MMV11] as a black-box by reducing the task of constructing time-lock puzzles in ROM to constructing permutation-VDFs. For the more general case of statistically-unique VDFs, we still use ideas from [MMV11] that are reminiscent of similar techniques also used in [Rud88, BKSY11, MM11]. Namely, our attacker will sample full executions of the evaluation function in its head, while respecting answers to queries that it has already learned from the oracle, and then it will ask all such queries in *one round* from the oracle. Using few $O(t)$ rounds of this form, we can argue that in most of these rounds, the adversary has not hit any "new query" in the verification process. Consequently, in most of the executions it is *consistent* with the verification procedure with respect to *some* oracle O', and thus by the statistical-uniqueness property, the answer in those executions should be the correct one. Finally, by taking the majority, we obtain the correct answer with high probability.

1.1 Related Work

Verifiable delay functions are closely related to the notion of (publicly-verifiable) proofs of sequential work (PoSW) [MMV13, CP18, AKK+19, DLM19]. The main difference between VDFs and PoSWs is *uniqueness*. More specifically, a VDF ensures that for every input x, an adversary running in time $\mathrm{poly}(\lambda, T)$ can only find at most one output y and proof π that the verifier would accept (and if it does, the verifier is also convinced that the prover performed T sequential work). In contrast, a PoSW does not provide any guarantees on uniqueness. In particular, every input x, there are many possible pairs (y, π) that the verifier would accept, and indeed, in this setting, there is no need to distinguish between the output y and the proof π. Even more generally, proofs of work need not be necessarily publicly-verifiable [DN93]. In this setting, the verification key is secret, and we only require sequentiality against adversaries who do not know the secret verification key. We emphasize that the uniqueness property in VDFs is important both for applications as well as constructions. Indeed, publicly-verifiable proofs of sequential work can be constructed in the random oracle model [CP18, DLM19], while our work rules out a broad class of VDFs in the same model.

Another closely-related primitive is the notion of a time-lock puzzle [RSW96]. In a time-lock puzzle, a puzzle generator can generate a puzzle x *together* with a solution y in time $t \ll T$, but computing y from x still requires sequential time T. The main difference between VDFs and time-lock puzzles is that time-lock puzzles require knowledge of a *secret key* for efficient verification (in time t). In contrast, VDFs are publicly-verifiable (in time t). However, similar to VDFs, the output of a time-lock puzzle is *unique*. Mahmoody et al. [MMV11] leverage this very uniqueness property *and* the fact that the solution is known ahead of the time to the verifier (because it is sampled during the puzzle generation) to show an impossibility result for time-lock puzzles in the random oracle model. While VDFs also require unique solutions, these solutions might not be known when we directly sample an input.

Concurrent work. Our second result about the limits of proofs of sequential work in ROM were independently discovered in a concurrent work by Döttling et al. [DGMV19], where the authors study *tight* verifiable delay functions. Indeed, this lower bound is more natural for the tight range of security parameters in which the sequentiality guarantee T' for the adversary is very close to $T' \approx (1 - o(1)) \cdot T$. However, as we mentioned above, this lower bound also applies to (even privately-verifiable) proofs of sequential work, while (even publicly-verifiable) proofs of sequential work do exist in the non-tight regime (e.g., $T' = T/2$) in ROM [MMV13]. Thus, whether or not this lower bound in ROM can be extended to arbitrary VDFs or not still remains as an intriguing open question.

2 Preliminaries

Throughout this work, we use λ to denote the security parameter. For an integer $n \in \mathbb{N}$, we write, $[n]$ to denote the set $\{1, 2, \ldots, n\}$. We write $\mathrm{poly}(\lambda)$ to denote a quantity that is bounded by a fixed polynomial in λ and $\mathrm{negl}(\lambda)$ to denote a function that is $o(1/\lambda^c)$ for all $c \in \mathbb{N}$. For a distribution $\mathcal{D}$, we write $x \leftarrow \mathcal{D}$ to denote that x is a uniform draw from $\mathcal{D}$. For a finite set S, we write $x \xleftarrow{\$} S$ to denote that x is sampled uniformly at random from S. We say that an algorithm is efficient if it runs in probabilistic polynomial time in the length of its input. We now review the definition of a verifiable delay function (VDF):

Definition 2.1 (Verifiable Delay Function [BBBF18])**.** A *verifiable delay function* with domain $\mathcal{X}$ and range $\mathcal{Y}$ is a tuple of algorithms $\Pi_{\mathsf{VDF}} = (\mathsf{Setup}, \mathsf{Eval}, \mathsf{Verify})$ with the following properties:

- $\mathsf{Setup}(1^\lambda, 1^T) \to \mathsf{pp}$: On input the the security parameter λ, and the time bound T, the setup algorithm outputs the public parameters pp.
- $\mathsf{Eval}(\mathsf{pp}, x) \to (y, \pi)$: On input the public parameters pp and an element $x \in \mathcal{X}$, the evaluation algorithm outputs a value $y \in \mathcal{Y}$ and a (possibly empty) proof π. We will typically refer to y as the "output" of the VDF on x. When the context is clear, we simply write $y \leftarrow \mathsf{Eval}(\mathsf{pp}, x)$ to denote the output of the VDF on x.

- $\mathsf{Verify}(\mathsf{pp}, x, y, \pi) \to \{0,1\}$: On input the public parameters pp, an element $x \in \mathcal{X}$, a value $y \in \mathcal{Y}$, and a proof string $\pi \in \{0,1\}^*$, the verification algorithm outputs a bit.

Moreover, the algorithms must satisfy the following efficiency requirements:

- The setup algorithm Setup runs in time $\mathrm{poly}(\lambda)$. (For simplicity, in the following sections, we sometimes write s to denote the running time of Setup.)
- The evaluation algorithm Eval runs in time T.
- The verification algorithm Verify runs in time $t = \mathrm{poly}(\lambda, \log T)$.

Correctness. Next, we define the correctness requirement on a VDF:

Definition 2.2 (Completeness)**.** A VDF $\Pi_{\mathsf{VDF}} = (\mathsf{Setup}, \mathsf{Eval}, \mathsf{Verify})$ with domain $\mathcal{X}$ and range $\mathcal{Y}$ is complete if for all $\lambda \in \mathbb{N}$, $T \in \mathbb{N}$, $x \in \mathcal{X}$ and sampling $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^T)$, we have that

$$\Pr[\mathsf{Verify}(\mathsf{pp}, x, \mathsf{Eval}(\mathsf{pp}, x), \pi)] = 1.$$

Security. There are two main security requirements we require on a VDF: *uniqueness* and *sequentiality*. We define these below:

Definition 2.3 (Uniqueness)**.** A VDF $\Pi_{\mathsf{VDF}} = (\mathsf{Setup}, \mathsf{Eval}, \mathsf{Verify})$ with domain $\mathcal{X}$ and range $\mathcal{Y}$ satisfies *statistical uniqueness* if for all adversaries $\mathcal{A}$, and sampling $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^T)$, $(x, y, \pi) \leftarrow \mathcal{A}(1^\lambda, 1^T, \mathsf{pp})$,

$$\Pr[y \neq \mathsf{Eval}(\mathsf{ek}, x) \wedge \mathsf{Verify}(\mathsf{pp}, x, y, \pi) = 1] = 0.$$

We say that Π_{VDF} satisfies *statistical uniqueness* if

$$\Pr[y \neq \mathsf{Eval}(\mathsf{ek}, x) \wedge \mathsf{Verify}(\mathsf{pp}, x, y, \pi) = 1] = \mathrm{negl}(\lambda), \tag{1}$$

and we say that Π_{VDF} satisfies *computational uniqueness* if Eq. (1) holds only for computationally-bounded adversaries.

Definition 2.4 (Sequentiality)**.** A VDF $\Pi_{\mathsf{VDF}} = (\mathsf{Setup}, \mathsf{Eval}, \mathsf{Verify})$ with domain $\mathcal{X}$ and range $\mathcal{Y}$ is σ-sequential (where σ may be a function of λ, T and t) if for all adversaries $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$, where $\mathcal{A}_0$ runs in time $\mathrm{poly}(\lambda, t)$ and $\mathcal{A}_2$ runs in time σ, and sampling $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^T)$, $\mathsf{st}_{\mathcal{A}} \leftarrow \mathcal{A}_0(1^\lambda, 1^T, \mathsf{pp})$, $x \overset{\$}{\leftarrow} \mathcal{X}$, $y \leftarrow \mathcal{A}_1(\mathsf{st}_{\mathcal{A}}, x)$,

$$\Pr[y = \mathsf{Eval}(\mathsf{pp}, x)] = \mathrm{negl}(\lambda).$$

We can view $\mathcal{A}_0$ as a "preprocessing" algorithm that precomputes some initial state $\mathsf{st}_{\mathcal{A}}$ based on the public parameters and $\mathcal{A}_1$ as the "online" adversarial evaluation algorithm.

Definition 2.5 (Decodable VDF [BBBF18])**.** Let t be a function of λ and T. A VDF $\Pi_{\mathsf{VDF}} = (\mathsf{Setup}, \mathsf{Eval}, \mathsf{Verify})$ with domain $\mathcal{X}$ and range $\mathcal{Y}$ is t*-decodable* if there is no extra proof (i.e., $\pi = \perp$) and there is a *decoder* Dec with the following properties:

- Dec runs in time t.
- For all $x \in \mathcal{X}$, if $y = \mathsf{Eval}(\mathsf{pp}, x)$, then $\mathsf{Dec}(\mathsf{pp}, y) = x$.

Moreover, for decoable VDFs, the verification algorithm $\mathsf{Verify}(\mathsf{pp}, x, y)$ works as follows: on input (pp, x, y), compute $x' \leftarrow \mathsf{Dec}(\mathsf{vk}, y)$ and output 1 only if $x = x'$. We call a VDF *efficiently* decodable, if it is t-decodable for $t = \mathrm{poly}(\lambda, \log T)$.

Remark 2.6 (Decodable VDFs and Perfect Uniqueness)**.** By construction, the combination of completeness and decodability implies *statistical* uniqueness (Definition 2.3).

Definition 2.7 (Random Oracle Model (ROM))**.** A random oracle $\mathcal{O}$ implements a truly random function from $\{0,1\}^*$ to range $\mathcal{R}$.[2] Equivalently, one can use "lazy evaluation" for any such random oracle as follows:

- If the oracle has not been queried on $x \in \{0,1\}^*$, uniformly randomly select $y \in \mathcal{R}$, remember the mapping (x, y), and return y.
- If the oracle was previously queried on $x \in \{0,1\}^*$, return the previously-chosen value of y (associated with x).

Remark 2.8 (VDFs in the ROM)**.** We define uniqueness and sequentiality of a VDF in the ROM by extending the corresponding definitions (Definition 2.3 and 2.4). For uniqueness, we note that the probability of the adversary succeeding is taken over the random coins of Setup and of the adversary, but *not* over the choice of oracle. For sequentiality, we measure the running time of the adversary by the number of *rounds* of oracle queries the adversary makes (this is to model the capabilities of a *parallel* adversary).

3 Lower Bounds for VDFs in the Random Oracle Model

In this section, we show that statistically unique VDFs (Definition 2.3) are impossible in the random oracle model. In particular, if a VDF in ROM is statistically unique, it means that for *every* sampled random oracle $O \leftarrow \mathcal{O}$, statistical uniqueness holds.

Theorem 3.1 (Ruling out Perfectly Unique VDFs in ROM)**.** *Suppose* $\Pi_{\mathsf{VDF}} = (\mathsf{Setup}, \mathsf{Eval}, \mathsf{Verify})$ *be a VDF in the ROM with statistical uniqueness in which (for a concrete choice of* λ*),* Setup *runs in time* s*,* Eval *runs in time* T*, and* Verify *runs in time* t*. Then, there is an adversary* $\mathcal{A}$ *that breaks sequentiality (Definition 2.4) with probability* $1 - \mathrm{negl}(\lambda)$ *and asks a total of* $O(T \cdot (t+s))$ *queries in* $2(s+t)+1$ *rounds.*

Before proving Theorem 3.1, we observe that this result already rules out the possibility of constructing decodable VDFs (which are statistically unique; see Remark 2.6) in the ROM. In fact, a special case of this theorem for the class of "permutation VDFs" is implied by the impossibility result of [MMV11] for time-lock puzzles [RSW96].[3] We define this class of special VDFs below:

Permutation-VDFs. As a special case of decodable VDFs, one can further restrict the mapping form $\mathcal{X}$ to $\mathcal{Y}$ to be a *permutation* (instead of just being an injective function). Indeed, the recent construction of [AKK$^+$19] has this property.

Proposition 3.2. *Let* Π_{VDF} *be a permutation-VDF in the ROM with a decoder* Dec *that runs in time* t*, and a setup algorithm* Setup *that runs in time* s*. Then, there is an adversary that breaks sequentiality (Definition 2.4) in* $O(s+t)$ *rounds of queries and a total of* $O(T \cdot (s+t))$ *queries.*

[2] In the literature, there are multiple ways to model the range set: sometimes range $\mathcal{R}$ is $\{0,1\}^\lambda$ for security parameter λ, sometimes it is simply $\{0,1\}$, and sometimes it is a "length preserving" by mapping any x to a string of the same length.

[3] In a time-lock puzzle, there is a puzzle-generation algorithm that runs in time t and samples a puzzle x together with a solution y, and an evaluation algorithm that runs in sequential time T that takes an input x and outputs the solution y.

Proof of Proposition 3.2. The proof follows from the impossibility result of [MMV11] after a reduction from permutation VDFs to time-lock puzzles. To generate a time-lock puzzle, the generator would first run the setup algorithm Setup of VDF to get pp. Then, it picks $y \xleftarrow{\$} \mathcal{X} = \mathcal{Y}$ (note that we need $\mathcal{X}$ to be efficiently samplable). Then, it lets $x = \mathsf{Dec}(\mathsf{pp}, y)$, and it outputs x as the puzzle (and keeps y as the solution). By definitions of VDF and time-lock puzzles, it can be shown that this way of generating puzzles would be T-sequential in the ROM, as long as the original VDF is a permutation-VDF.

Having the reduction above, we can use the result of [MMV11] showing that any time-lock puzzles in ROM with k queries during the puzzle generation and T queries during the solving of the puzzle, can be broken by $O(k)$ rounds of queries and a total of $O(k \cdot T)$ queries. We finally note that $k = s + t$ since the puzzle generation involves running both the setup and the decoding process of the VDF. $\square$

We now give the proof of Theorem 3.1. It still follows the ideas from [MMV11] for ruling out time-lock puzzles in the ROM, but this time, we cannot simply reduce the problem to the setting of time-lock puzzles, and we need to go into the proof and extend it to our setting.

Proof of Theorem 3.1. Without loss of generality, assume that Eval asks no repeated queries in a single execution. The attacker's algorithm $\mathcal{A}$ is as follows.

1. Let $Q_{\mathcal{A}} = \varnothing$ (as a set of queries) and $P_{\mathcal{A}} = \varnothing$ (as a set of query-answer pairs).
2. Let $d = 2(s + t) + 1$.
3. For $i \in [d]$ do the following:
   (a) Let $P_{\mathcal{A}}^{(i)} = Q_{\mathcal{A}}^{(i)} = \varnothing$.
   (b) Execute $(y_i, \pi_i) \leftarrow \mathsf{Eval}(\mathsf{ek}, x)$ where the random oracle queries (made by Eval) are answered using the following procedure. On every oracle query q:
      - If $q \in Q_{\mathcal{A}}$, then reply with the value r where $(q, r) \in P_{\mathcal{A}}$.
      - Otherwise, choose a uniformly random value $r \xleftarrow{\$} \mathcal{R}$ and add (q, r) to $P_{\mathcal{A}}^{(i)}$ and add q to $Q_{\mathcal{A}}^{(i)}$.
   (c) In *one round*, for all $(q, r) \in P_{\mathcal{A}}^{(i)}$, query the real oracle $\mathcal{O}$ and get $r \leftarrow \mathcal{O}(q)$ as the answer. Then for all such queries, add (q, r) to $P_{\mathcal{A}}$ and q to $Q_{\mathcal{A}}$.
4. Output $\mathrm{majority}(y_1, ..., y_d)$ where majority denotes the majority operation (that outputs $\perp$ if no majority exists).

We now show that $\mathcal{A}$ satisfies the properties needed in Theorem 3.1. Let Q_S be the queries asked by the setup algorithm and Q_V the queries asked by the verifier for the specific challenge x and its true solution y.

For $i \in [d]$, we define H_i to be the event where there is a query $q \in Q_{\mathcal{A}}^{(i)} \cap (Q_S \cup Q_V)$ during the i^{th} round of emulation that was *not* previously asked by the adversary: $q \notin Q_P$ *at that moment*. Equivalently, when q is asked, it holds that $q \in (Q_{\mathcal{A}}^{(i)} \cap (Q_S \cup Q_V)) \setminus Q_P$.

The following claim shows that H_i cannot happen for too many i's.

Claim 3.3. *If $I = \{i \mid H_i \ \text{holds}\}$, then $|I| \le s + t$.*

Proof. The reason is that every time that H_i happens for a query q, at the end of round i, $\mathcal{A}$ asks q from the oracle $\mathcal{O}$, $\mathcal{A}$ asks a *new query* that was asked previously by either of Setup or Verify algorithms. Since Setup and Verify together ask a total of $s + t$ queries, this cannot happen more than $s + t$ times. $\square$

Claim 3.4. *If H_i does not happen, then $y_i = y$.*

Proof. Let $y_i \neq y$ for a round i in which H_i has not happened. This means that the set of oracle query-answer pairs used during Setup, and the i^{th} emulation of Eval by $\mathcal{A}$ are *consistent*. Namely, there is an oracle O', relative to which, we have $\mathsf{pp} \leftarrow \mathsf{Setup}^{O'}$, $(y', \pi') \leftarrow \mathsf{Eval}^{O'}(\mathsf{pp}, x)$, and $\mathsf{Verify}^{O'}(\mathsf{pp}, x, y, \pi) = 1$. However, this shows that the statistical-uniqueness property is violated relative to O', because for input x, there is a "wrong" solution y (i.e., $y \neq y' = \mathsf{Eval}^{O'}(\mathsf{pp}, x)$) together with some proof π for y such that the verification passes $\mathsf{Verify}^{O'}(\mathsf{pp}, x, y, \pi) = 1$. $\square$

By the above two claims, it holds that $y_i = y$ for at least $s + t + 1$ values of $i \in [2(s + t) + 1]$, and thus the majority gives the right answer y for $\mathcal{A}$. $\square$

Lower bound for *tight* proofs of sequential work. We can apply similar techniques to rule out *tight* proofs of sequential work [MMV13] in the random oracle model. At a high level, a (publicly-verifiable) proof of sequential work is a VDF without *uniqueness*. Namely, for an input x, there can be many pairs (y, π) that passes verification. In this setting, there is no need to distinguish y and π. While we have constructions of (publicly-verifiable) proofs of sequential work in the ROM, our results show that *tight* proofs of sequential work are impossible in this setting. In particular, the following barrier applies to settings where the sequentiality parameter σ is *very* close to T (e.g., this does not apply to $\sigma = T/2$).

Theorem 3.5 (Ruling out Tight Proofs of Sequential Work in ROM)**.** *Suppose $\Pi_{\mathsf{VDF}} = (\mathsf{Setup}, \mathsf{Eval}, \mathsf{Verify})$ is a proof of work in the ROM in which (for a concrete choice of λ),* Setup *runs in time s,* Eval *runs in time T, and* Verify *runs in time t. Then, for any $T' < T$ there is an adversary $\mathcal{A}$ that asks a total of at most T' queries and breaks sequentiality (Definition 2.4) with probability* $1 - (s + t) \cdot T'/T - \operatorname{negl}(\lambda)$.

Proof. Again, without loss of generality, we assume that Eval asks no repeated queries in a single execution. The attacker's algorithm A is as follows.

1. Pick a random set $\mathcal{S} \subseteq [T]$ of size T'.
2. Execute $(y, \pi) \leftarrow \mathsf{Eval}(\mathsf{pp}, x)$ while any oracle query q is answered as follows.
    - If $q \in \mathcal{S}$, ask q from the true oracle $\mathcal{O}$,
    - Otherwise choose a uniformly random value, $r \leftarrow \mathcal{R}$ for q.
3. Output (y, π).

To analyze the above attack, we compare the attacker's experiment with an "ideal" experiment. Before doing so, we first define the following experiment.

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^T)$
- $x \xleftarrow{\$} \mathcal{X}$
- Run the adversary $\mathcal{A}$ (described above).
- Let $b \leftarrow \mathsf{Verify}(\mathsf{pp}, x, y, \pi)$.
- The output of the experiment is 1 if $b = 1$ (and 0 otherwise).

Real vs ideal experiments. Let the above experiment be the "real" experiment, and let "ideal" experiment be a similar game in which the true oracle $\mathcal{O}$ is used in all the queries. We use the notation $\Pr_{\mathsf{real}}[\cdot]$ (resp., $\Pr_{\mathsf{ideal}}[\cdot]$) to denote a probability of an event E in the Real (resp., Ideal) experiment.

Events. Let W be the event that $\mathsf{Verify}(\mathsf{pp}, x, y, \pi) = 1$ when (y, π) is the output of the adversary (i.e., W is the event that the adversary wins and the experiment outputs 1). Also, let Q_V be the oracle queries made by Verify, Q_S be the oracle queries made by Setup, and $Q_{\mathcal{A}}$ be the adversary's queries q_i during the emulation of the Eval whose where $i \notin \mathcal{S}$ (i.e., the adversary chooses the answer to q_i at random). Define the "bad" event B to be the event that $(Q_V \cup Q_S) \cap Q_A \neq \varnothing$; namely, the event that adversary makes up an answer to a query that is asked either by the setup algorithm or the verification algorithm. With these definitions, the following claim trivially holds in the ideal experiment, as there is no attack involved.

Claim 3.6. $\Pr_{\mathsf{ideal}}[W] = 1 - \mathrm{negl}(n)$.

Next, the following lemma states that until event B happens, the two experiments proceed identically.

Lemma 3.7. $\Pr_{\mathsf{real}}[B] = \Pr_{\mathsf{ideal}}[B]$, *and conditioned on the event B not happening, the two experiments have the same distribution. In particular, for any event like W, it hold that* $\Pr_{\mathsf{real}}[W \vee B] = \Pr_{\mathsf{ideal}}[W \vee B]$.

Proof. Here, we make a crucial use of the fact that oracle $\mathcal{O}$ is random. To prove the lemma, we run the two games *in parallel* using the *same* randomness for any query that is asked by any party, step by step. Namely, we start by executing the setup algorithm identically as much as possible until event B happens. More formally, we run both experiments by using fresh randomness to answer any new query asked during the execution, and we will *stop* the execution as soon as event B happens. Since until the event B happens both games proceed *identically* (in a statistical sense) and *consistently* according to their own distribution, it means that until event B happens, the two games have the same statistical distribution. $\square$

We now observe that the probability of B is small in the ideal game.

Claim 3.8. $\Pr_{\mathsf{ideal}}[B] \leq (s + t) \cdot T'/T$.

Proof. In this game, the set $\mathcal{S}$ is chosen independently of other components of the experiment. So, we can choose $\mathcal{S}$ *at the end*. By doing so, any query in in $Q_V \cup Q_S$ that is also asked by $Q_{\mathcal{A}}$ will be chosen by the adversary with probability at most T'/T. Thus, the claim follows by a union bound. $\square$

The above claims finish the proof of Theorem 3.5, as we now can conclude that the probability of W in both experiments is "close":

$$\left| \Pr_{\mathsf{real}}[W] - \Pr_{\mathsf{ideal}}[W] \right| \leq \Pr_{\mathsf{ideal}}[B].$$

We already know that $\Pr_{\mathsf{ideal}}[W] = 1 - \mathrm{negl}(n)$, therefore, we conclude that $\Pr_{\mathsf{real}}[W] \geq \Pr_{\mathsf{ideal}}[W] - \Pr_{\mathsf{ideal}}[B] \geq 1 - (s + t)T'/T - \mathrm{negl}(\lambda)$. $\square$

References

[AKK+19] Hamza Abusalah, Chethan Kamath, Karen Klein, Krzysztof Pietrzak, and Michael Walter. Reversible proofs of sequential work. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019, Part II*, volume 11477 of *Lecture Notes in Computer Science*, pages 277–291, Darmstadt, Germany, May 19–23, 2019. Springer, Heidelberg, Germany.

[AS15] Joël Alwen and Vladimir Serbinenko. High parallel complexity graphs and memory-hard functions. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th Annual ACM Symposium on Theory of Computing*, pages 595–603, Portland, OR, USA, June 14–17, 2015. ACM Press.

[BBBF18] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part I*, volume 10991 of *Lecture Notes in Computer Science*, pages 757–788, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Heidelberg, Germany.

[BBHM02] Ingrid Biehl, Johannes A. Buchmann, Safuat Hamdy, and Andreas Meyer. A signature scheme based on the intractability of computing roots. *Des. Codes Cryptography*, 25(3):223–236, 2002.

[BKSY11] Zvika Brakerski, Jonathan Katz, Gil Segev, and Arkady Yerukhimovich. Limits on the power of zero-knowledge proofs in cryptographic constructions. In Yuval Ishai, editor, *TCC 2011: 8th Theory of Cryptography Conference*, volume 6597 of *Lecture Notes in Computer Science*, pages 559–578, Providence, RI, USA, March 28–30, 2011. Springer, Heidelberg, Germany.

[Chi19] Chia. Chia network announces 2nd VDF competition with $100,000 in total prize money. https://www.chia.net/2019/04/04/chia-network-announces-second-vdf-competition-with-in-total-prize-money.en.html, 2019.

[CP18] Bram Cohen and Krzysztof Pietrzak. Simple proofs of sequential work. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part II*, volume 10821 of *Lecture Notes in Computer Science*, pages 451–467, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Heidelberg, Germany.

[DGMV19] Nico Dttling, Sanjam Garg, Giulio Malavolta, and Prashant Nalini Vasudevan. Tight verifiable delay functions. Cryptology ePrint Archive, Report 2019/659, 2019. https://eprint.iacr.org/2019/659.

[DLM19] Nico Döttling, Russell W. F. Lai, and Giulio Malavolta. Incremental proofs of sequential work. In *EUROCRYPT*, pages 292–323, 2019.

[DN93] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In Ernest F. Brickell, editor, *Advances in Cryptology – CRYPTO'92*, volume 740 of *Lecture Notes in Computer Science*, pages 139–147, Santa Barbara, CA, USA, August 16–20, 1993. Springer, Heidelberg, Germany.

[EFKP19] Naomi Ephraim, Cody Freitag, Ilan Komargodski, and Rafael Pass. Continuous verifiable delay functions. *IACR Cryptology ePrint Archive*, 2019:619, 2019.

[FMPS19] Luca De Feo, Simon Masson, Christophe Petit, and Antonio Sanso. Verifiable delay functions from supersingular isogenies and pairings. *IACR Cryptology ePrint Archive*, 2019:166, 2019.

[KJG$^+$16] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. Enhancing bitcoin security and performance with strong consistency via collective signing. In Thorsten Holz and Stefan Savage, editors, *USENIX Security 2016:*

25th USENIX Security Symposium, pages 279–296, Austin, TX, USA, August 10–12, 2016. USENIX Association.

[LSS19] Esteban Landerreche, Marc Stevens, and Christian Schaffner. Non-interactive cryptographic timestamping based on verifiable delay functions. *IACR Cryptology ePrint Archive*, 2019:197, 2019.

[MM11] Takahiro Matsuda and Kanta Matsuura. On black-box separations among injective one-way functions. In Yuval Ishai, editor, *TCC 2011: 8th Theory of Cryptography Conference*, volume 6597 of *Lecture Notes in Computer Science*, pages 597–614, Providence, RI, USA, March 28–30, 2011. Springer, Heidelberg, Germany.

[MMV11] Mohammad Mahmoody, Tal Moran, and Salil P. Vadhan. Time-lock puzzles in the random oracle model. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 39–50, Santa Barbara, CA, USA, August 14–18, 2011. Springer, Heidelberg, Germany.

[MMV13] Mohammad Mahmoody, Tal Moran, and Salil P. Vadhan. Publicly verifiable proofs of sequential work. In Robert D. Kleinberg, editor, *ITCS 2013: 4th Innovations in Theoretical Computer Science*, pages 373–388, Berkeley, CA, USA, January 9–12, 2013. Association for Computing Machinery.

[Pie19] Krzysztof Pietrzak. Simple verifiable delay functions. In *ITCS*, pages 60:1–60:15, 2019.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

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

[Rud88] Steven Rudich. *Limits on the Provable Consequences of One-way Functions*. PhD thesis, EECS Department, University of California, Berkeley, Dec 1988.

[Sha19] Barak Shani. A note on isogeny-based hybrid verifiable delay functions. *IACR Cryptology ePrint Archive*, 2019:205, 2019.

[Wes19] Benjamin Wesolowski. Efficient verifiable delay functions. In *EUROCRYPT*, pages 379–407, 2019.