Can Verifiable Delay Functions be Based on Random Oracles?

Mohammad Mahmoody,* Caleb Smith, and David J. Wu†

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

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 efficiently verified (possibly given a proof π) in time $t \ll T$ (e.g., $t = \text{poly}(\lambda, \log T)$ where λ is the security parameter). The first security requirement on a VDF, called *uniqueness*, is 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, called *sequentiality*, is that no polynomial-time algorithm running in time $\sigma < T$ for some parameter σ (e.g., $\sigma = T^{1/10}$) can compute y, even with $\text{poly}(T, \lambda)$ many parallel processors. 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 in a black-box way, 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. We rule out two classes of constructions of VDFs in the ROM:

- We show that VDFs satisfying *perfect* uniqueness (i.e., no different convincing solution $y' \neq y$ exists) cannot be constructed in the ROM. More formally, we give an attacker that finds the solution y in $\approx t$ *rounds* of queries, asking only $\text{poly}(T)$ queries in total.
- We also rule out *tight* VDFs in the ROM. Tight VDFs were recently studied by Döttling, Garg, Malavolta, and Vasudevan (ePrint Report 2019) and require sequentiality $\sigma \approx T - T^{\rho}$ for some constant $0 < \rho < 1$. More generally, our lower bound also applies to proofs of sequential work (i.e., VDFs without the uniqueness property), even in the private verification setting, and sequentiality $\sigma > T - T/2t$ for a concrete verification time t.

This is the full version of a paper published in ICALP 2020 [MSW20]. This version subsumes an earlier version of the work titled "A Note on the (Im)possibility of Verifiable Delay Functions in the Random Oracle Model" [MSW19].
*Supported by NSF CCF-1910681, CNS-1936799 and a University of Virginia SEAS Research Innovation Award.
†Supported by NSF CNS-1917414 and a University of Virginia SEAS Research Innovation Award.

Contents

1 Introduction

A verifiable delay function (VDF) [BBBF18] with domain $\mathcal{X}$ and range $\mathcal{Y}$ 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., $t = \text{poly}(\lambda, \log T)$ where λ is a security parameter). More precisely, there exists an evaluation algorithm Eval that on input $x \in \mathcal{X}$ computes a value $y \in \mathcal{Y}$ 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] Typically, we require that the setup is also fast: namely, Setup runs in time $s = \text{poly}(\lambda, \log T)$ as well. 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 $\text{poly}(\lambda, T)$ can find $y' \neq \mathsf{Eval}(x)$ and a proof π' such that $\mathsf{Verify}(x, y', \pi') = 1$; and (2) *sequentiality* with some parameter $\sigma < T$, which says that no adversary running in *sequential time* σ can compute $y = \mathsf{Eval}(x)$. The sequential time σ allows the adversary to have up to $\text{poly}(\lambda, T)$ parallel processors.

Verifiable delay functions have recently received extensive study, and have found numerous applications to building randomness beacons [BBBF18, EFKP19] and cryptographic timestamping schemes [LSS19]. 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 [HC19]. One of the bottlenecks in existing constructions of VDFs is

[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].

their reliance on structured algebraic assumptions (e.g., groups of unknown order [RSA78, BBHM02] or isogenies over pairing groups [FMPS19]).

A natural question to ask is whether we can construct VDFs generically from *unstructured* primitives, such as one-way functions, collision-resistant hash functions, or stronger forms of hash 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). Similar to previous work (cf. [MMV11, AS15]) in the *parallel* random oracle model (ROM), we measure the running time of the adversary by the number of oracle queries it makes and the sequentiality of the adversary by the number of *rounds* of oracle queries it makes. However, for sake of simplicity, in this work we simply refer to the parallel ROM as the ROM.

1.1 Our Results

In this work, we rule out the existence of VDFs with *perfect* uniqueness (i.e., VDFs where for any $x \in \mathcal{X}$, there does not exist any (y', π) such that $\mathsf{Verify}(x, y', \pi) = 1$ and $y' \neq \mathsf{Eval}(x)$) in the random oracle model. Specifically, we construct an adversary that asks $O(t)$ rounds of queries and a total number of $\mathrm{poly}(T)$ queries and breaks the uniqueness of VDFs with respect to *some* oracle.

A natural class of VDFs with perfect uniqueness is the class of *permutation* VDFs where the function $\mathsf{Eval}(\cdot)$ implements a permutation on the domain (specifically, $\mathcal{X} = \mathcal{Y}$), and verification consists of inverting y and checking if it is x or not. Recently, Abusalah et al. [AKK+19] constructed permutation VDFs in the ROM, but they additionally relied on the assumptions used in the sloth functions of Lenstra and Wesolowski [LW15]. Our work shows that relying on some kind of structured assumption is *necessary* to achieve permutation VDFs.

We next show that in the "tight" regime of sequentiality, as recently studied in the concurrent work of Döttling et al. [DGMV19] (i.e., when the sequentiality parameter σ is quite close to T), even *proofs of sequential work* (PoSW) [DN92, CLSY93, RSW96, MMV13] cannot be based on random oracles. In particular, our result essentially applies to settings where $\sigma \gg T \cdot (1 - 1/t)$ where t is the verification time. A proof of sequential work is a relaxation of a VDF without the uniqueness or the public-verifiability properties. Thus, our lower bound for ruling out tight PoSW also rules out tight VDFs in the ROM. We note, however, that since (even publicly-verifiable) PoSW satisfying weaker notions of sequentiality (e.g., $\sigma = T/2$) are known to exist in the ROM [MMV13], it is not clear whether this lower bound for PoSW can be extended to rule out (non-tight) VDFs, and we leave this as an intriguing open question.

We note that both of our lower bounds are proven in settings that have already been studied in previous (or concurrent) work. Namely, by ruling out permutation VFDs in the ROM, our first main result complements the previous work on (what we refer to as) permutation VDFs [LW15, AKK+19] by showing that the random oracle alone is not sufficient to realize such strong VDFs. Moreover, our second result shows that when it comes to the tight regime of sequentiality (studied in the concurrent work of [DGMV19]), VDFs as well as more relaxed notions like proofs of sequential work cannot be based on symmetric primitives in a black-box way.

1.1.1 Our Techniques

At a technical level, the proofs of our lower bounds start from 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 perfectly-unique VDFs where the VDF is a *permutation* on its domain $\mathcal{X} = \mathcal{Y}$, which we refer to as a "permutation VDF" (cf., [LW15, KJG+16, AKK+19]), we can use the impossibility result of [MMV11] as a black-box by reducing the task of constructing time-lock puzzles in the ROM to constructing permutation VDFs in the ROM.

For the more general case of perfectly-unique VDFs (that are not necessarily permutations) we cannot use the result of [MMV11] as a black-box, but we can still adapt the ideas from their work that are reminiscent of similar techniques also used in [Rud88, BKSY11, MM11]. Namely, our attacker will sample full executions of the evaluation function Eval in its head, while respecting answers to queries that it has already learned from the real oracle. At the end of each simulated execution, it will ask all previously-unasked queries in a *single round* to the oracle (and use those values in subsequent simulated executions). We show that using just $O(t)$ rounds of this form, we can argue that in most of these rounds, the adversary does 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 *perfect uniqueness* property, the answer in those executions should be the correct one. Finally, by taking a majority vote over the executions, we obtain the correct answer with high probability. Observe that this argument critically relies on *perfect uniqueness*. The main open question remaining is whether a similar lower bound for *computational uniqueness* holds for VDF in the ROM or not. In this setting, the security requirement is that no *efficient* adversary can find a different value $y' \neq \mathsf{Eval}(x)$ with a proof π' that passes verification. (See Section 3.1.)

We then adapt this technique to additionally rule out *tight* proofs of sequential work in the ROM. Roughly, a scheme satisfies tight sequentiality if no adversary running in sequential time $\sigma = T - T^{\rho}$ for constant $\rho < 1$ can compute (y, π) such that $\mathsf{Verify}(x, y, \pi) = 1$. More generally, we consider a setting of parameters where $\sigma > T \cdot (1 - 1/2t)$, where t denotes the number of queries made by Verify. In this setting, we can construct an algorithm that simulates the answers to *some* but *not* all of the oracle queries made by the evaluation algorithm. The algorithm answers the remaining queries based on the real oracle evaluations. To simplify the description, in the following, assume that the setup algorithm is essentially nonexistent, and that the public parameter is fixed and publicly known ahead of time. Since the scheme is assumed to be *tightly* sequential, as long as the algorithm simulates sufficiently-many queries (e.g., $T/2t$ queries), it is possible to reduce the number of rounds of queries made to the real oracle (i.e., from T to $T - T/2t$). Moreover, if the number of "simulated responses" is small enough and if the set of queries for which the algorithm simulates is chosen at random, then there is a good chance that none of the simulated queries are asked during verification. If both of these properties hold simultaneously, then the algorithm successfully computes a response that verifies, thus breaking tight sequentiality. We provide the formal analysis in Section 3.2. However, the formal version of this argument needs to also incorporate the query complexity of the setup algorithm as well, leading to a weaker attack that only applies when $\sigma > T \cdot (1 - 1/2(s+t))$. However, in Section 4, we describe how to extend our lower bounds on tight proofs of sequential work (and correspondingly, tight VDFs) to additionally rule out tight proofs of sequential work with an *expensive* setup phase as well as tight proofs of sequential work in the random *permutation* model. In particular, we show how to leverage preprocessing to essentially eliminate the dependency of the attack's online phase on the query complexity of the setup.

1.2 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 (accompanied with a possibly non-unique 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, for every input x, there might be many possible pairs (y, π) that the verifier would accept, and as a result, in this setting there is no longer a need to distinguish between the output y and the proof π. Even more generally, proofs of work need not be publicly-verifiable [DN92], and one could only require sequentiality against adversaries who do not know a secret verification key generated during the setup. 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 [MMV13, CP18, DLM19], while our work rules out a broad class of VDFs in the same model.

Time-lock puzzles. Time-lock puzzle [RSW96] are closely related to VDFs as they are also based on the notion of sequentiality. 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 might 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. In an independent and concurrent work, Döttling et al. [DGMV19] introduce and provide an in-depth study of tight verifiable delay functions. Their work both provides a positive construction of tight VDFs (from algebraic assumptions) as well as a negative result on the existence of *tight* VDFs in the random oracle model. In this work, we show that the lower bound on tight VDFs also extends to (even *privately*-verifiable) proofs of sequential work. At the same time, we note that (even publicly-verifiable) proofs of sequential work do exist in the *non-tight* regime (e.g., $\sigma = T/2$) in the ROM [MMV13]. Thus, whether or not this lower bound in the random oracle model on tight VDFs can be extended to arbitrary (*non-tight*) VDFs or not still remains 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 $\lambda^{-\omega(1)}$. For a distribution D, we write $x \leftarrow D$ to denote that x is drawn from D. For a randomized algorithm Alg, we write $y \leftarrow \mathsf{Alg}(x)$ to denote the process of computing y by running Alg on input x with (implicitly-defined) randomness

r of the appropriate length (based on the length of x). For a finite set $\mathcal{S}$, we write $x \xleftarrow{\$} \mathcal{S}$ to denote that x is sampled uniformly at random from $\mathcal{S}$. 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.

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

- $\mathsf{Setup}(1^\lambda, T) \to \mathsf{pp}$: On input the the security parameter λ and the time bound T, the setup algorithm outputs the public parameters pp. The public parameter determines a (uniformly) *samplable* input space $\mathcal{X}_{\mathsf{pp}}$ and an output space $\mathcal{Y}_{\mathsf{pp}}$. When the context is clear, we simply denote them as $\mathcal{X}$ and $\mathcal{Y}$.
- $\mathsf{Eval}(\mathsf{pp}, x) \to (y, \pi)$: 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. Moreover, in case Eval is randomized, the first output y should be determined by x and pp (and be independent of the randomness used). We will typically refer to y as the "output" of the VDF on x, and since y is unique, when the context is clear, we simply write $y = \mathsf{Eval}(\mathsf{pp}, x)$ 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 (1 means accept and 0 means reject).

Moreover, the algorithms must satisfy the following efficiency requirements:

- The setup algorithm Setup runs in time $\mathrm{poly}(\lambda, \log T)$.
- The evaluation algorithm Eval runs in time T.
- The verification algorithm Verify runs in time $\mathrm{poly}(\lambda, \log T)$.

For simplicity of notation, in the following sections, we sometimes write s (resp., t) to denote the running time of Setup (resp., Verify). This additionally allow us to state our results more generally while also explicitly stating how our results depend on the precise bounds on the running time of Setup and Verify.

Completeness. We now define the completeness requirement on VDFs.

Definition 2.2 (Completeness of VDF)**.** A VDF $\Pi_{\mathsf{VDF}} = (\mathsf{Setup}, \mathsf{Eval}, \mathsf{Verify})$ has completeness error γ if for all $\lambda \in \mathbb{N}$ and $T \in \mathbb{N}$,

$$\Pr\left[\mathsf{Verify}(\mathsf{pp}, x, y, \pi) = 1 \;:\; \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, T), x \xleftarrow{\$} \mathcal{X}_{\mathsf{pp}} \\ (y, \pi) \leftarrow \mathsf{Eval}(\mathsf{pp}, x) \end{array}\right] \geq 1 - \gamma.$$

Unless stated otherwise, we assume $\gamma = 0$. For our first lower bound, we need perfect completeness $\gamma = 0$, but our second lower bound in the tight regime directly extends to more generalized settings where the completeness error γ is $\mathrm{negl}(\lambda)$ (or even a small constant).

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

Definition 2.3 (Uniqueness of VDF)**.** A VDF $\Pi_{\mathsf{VDF}} = (\mathsf{Setup}, \mathsf{Eval}, \mathsf{Verify})$ satisfies *uniqueness* for a class $\mathcal{A}$ of adversaries with error $\varepsilon(\lambda)$, if for all adversaries $\mathsf{Adv} \in \mathcal{A}$, we have that

$$\Pr\left[y \neq \mathsf{Eval}(\mathsf{pp}, x) \wedge \mathsf{Verify}(\mathsf{pp}, x, y, \pi) \ : \ \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, T) \\ (x, y, \pi) \leftarrow \mathsf{Adv}(1^\lambda, 1^T, \mathsf{pp}) \end{array} = 1\right] \leq \varepsilon(\lambda).$$

We say that Π_{VDF} satisfies *statistical uniqueness* if $\mathcal{A}$ is the set of all (computationally unbounded) adversaries and $\varepsilon(\lambda)$ is negligible. We say Π_{VDF} satisfies *perfect uniqueness* if we further require $\varepsilon(\lambda) = 0$. We say that Π_{VDF} is *computationally unique* if $\mathcal{A}$ is the class of $\mathrm{poly}(\lambda, T)$-time adversaries and $\varepsilon(\lambda)$ is negligible.

Definition 2.4 (Sequentiality of VDF)**.** A VDF $\Pi_{\mathsf{VDF}} = (\mathsf{Setup}, \mathsf{Eval}, \mathsf{Verify})$ is σ-sequential (where σ may be a function of λ, T and t) if for all adversaries $\mathsf{Adv} = (\mathsf{Adv}_0, \mathsf{Adv}_1)$, where $\mathsf{Adv}_0, \mathsf{Adv}_1$ both run in total time $\mathrm{poly}(\lambda, T)$ and Adv_1 runs in *parallel* time at most σ, we have that

$$\Pr\left[y = \mathsf{Eval}(\mathsf{pp}, x) \ : \ \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, T) \\ \mathsf{st}_{\mathsf{Adv}} \leftarrow \mathsf{Adv}_0(1^\lambda, 1^T, \mathsf{pp}) \\ x \xleftarrow{\$} \mathcal{X}, y \leftarrow \mathsf{Adv}_1(\mathsf{st}_{\mathsf{Adv}}, x) \end{array}\right] = \mathrm{negl}(\lambda).$$

We can view Adv_0 as a "preprocessing" algorithm that precomputes some initial state $\mathsf{st}_{\mathsf{Adv}}$ based on the public parameters and Adv_1 as the "online" adversarial evaluation algorithm.

Definition 2.5 (Decodable VDF [BBBF18])**.** Let t be a function of λ and T. A VDF $\Pi_{\mathsf{VDF}} = (\mathsf{Setup}, \mathsf{Eval}, \mathsf{Verify})$ 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 $\mathsf{Dec}(\mathsf{pp}, y) = x$.

We say that Π_{VDF} is *strongly* decodable, if for all $y' \neq y = \mathsf{Eval}(\mathsf{pp}, x)$, it holds that $\mathsf{Dec}(\mathsf{pp}, y) \neq x$. Finally, we say a VDF is *efficiently* (strongly) decodable if it is (strongly) t-decodable for $t = \mathrm{poly}(\lambda, \log T)$.

Remark 2.6 (Strongly Decodable VDFs and Perfect Uniqueness)**.** Strong decodability (Definition 2.5) implies *perfect* uniqueness (Definition 2.3). However, the reverse is not true in general.

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" to simulate the behavior of a random oracle as follows:

- If the oracle has not been queried on $x \in \{0,1\}^*$, sample $y \xleftarrow{\$} \mathcal{R}$. The oracle returns y and remembers the mapping (x, y).

[2]There are multiple possible ways to model a random oracle in the literature. We describe three possibilities here. Sometimes, the range $\mathcal{R}$ is $\{0,1\}^\lambda$ where λ is a security parameter; other times, it is simply $\{0,1\}$, and sometimes it is a "length preserving" mapping that maps each input x to a string of the same length.

- If the oracle was previously queried on $x \in \{0,1\}^*$, return the previously-chosen value of $y \in \mathcal{R}$ associated with x.

Remark 2.8 (Hardness in the Random Oracle Model)**.** Note that constructions with unconditional security in the ROM use the oracle as the *only* source of hardness. In particular, as stated above, the number of rounds of queries to the oracle model the cost of the parallel computation. If one allows other sources of computational hardness, existing constructions of VDFs trivially also exist relative to the ROM (by ignoring the random oracle).

Definition 2.9 (VDFs in the ROM)**.** We define uniqueness and sequentiality of a VDF in the ROM by allowing the components $\mathsf{Setup}, \mathsf{Eval}, \mathsf{Verify}$ of a VDF to be oracle-aided algorithms in the ROM and adjusting their notion of time and parallel time (in Definitions 2.3 and 2.4 according to Definition 2.7 and Remark 2.8). In particular, 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). Furthermore, for the uniqueness property, we require 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.

3 Lower Bounds for VDFs in the Random Oracle Model

In this section, we first show that perfectly unique VDFs (Definition 2.3) are impossible in the random oracle model. Then, as a corollary, we obtain barriers for strongly decodable VDFs as well. In particular, if a VDF in the ROM is perfectly unique, it means that for *every* sampled random oracle $O \leftarrow \mathcal{O}$, perfect uniqueness holds. Moreover, our result shows that a recent construction of [AKK+19] for reversible VDFs that is of the form of a permutation function and uses the sloth functions from [LW15] cannot be modified to only rely on a random oracle.

We then turn our attention to the case of tight VDFs (and more generally, tight proofs of sequential work) and show that they cannot be constructed in the ROM as well.

3.1 Case of Perfectly Unique VDFs

In this section, we give a lower bound for perfectly unique VDFs in the random oracle model. Specifically, we prove the following theorem:

Theorem 3.1 (Lower Bounds for Perfectly Unique VDFs in the ROM)**.** *Suppose* $\Pi_{\mathsf{VDF}} = (\mathsf{Setup}, \mathsf{Eval}, \mathsf{Verify})$ *is a VDF in the ROM with perfect uniqueness and perfect completeness 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* Adv *that breaks sequentiality (Definition 2.4) by asking a total of* $2T \cdot (s+t)$ *queries in* $2(s+t)$ *rounds of queries.*

Before proving Theorem 3.1, we observe that this result already rules out the possibility of constructing strongly decodable VDFs (which are forced to be perfectly unique; see Remark 2.6) in the ROM. In fact, a special case of this theorem for the class of "permutation VDFs" can be derived from the impossibility result of [MMV11] for time-lock puzzles [RSW96].[3] We first define this class of special VDFs below and prove their impossibility in the ROM as a warm up.

[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.

3.1.1 Warm Up: The Special Case of Permutation VDFs

Permutation VDFs. As a special case of strongly decodable VDFs, one can further restrict the mapping from $\mathcal{X}$ to $\mathcal{Y}$ to be a *permutation* (instead of just being an injective function). Indeed, the recent construction of [AKK+19] has this exact property as they construct decodable/reversible VDFs where the evaluation function is a permutation on its domain. Indeed, [AKK+19] constructs proofs of sequential work in the ROM, which when combined with with the sloth function from [LW15] leads to permutation VDFs with weakly efficient verification (similarly to the sloth function itself [LW15]). Thus, it was left open whether random oracle would suffice for permutation VDFs or not. Our result rules this possibility out, even if the verification is slightly more efficient than the evaluation.

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* $O(T \cdot (s+t))$ *queries.*

Proof. To prove Proposition 3.2, we show how to construct a time-lock puzzle from a permutation VDF. The result then follows from the lower bound of Mahmoody et al. [MMV11] who showed an impossibility result for time-lock puzzles in the ROM. The construction is as follows. The puzzle generator would first run the setup algorithm Setup of the VDF to get the public parameter pp. Then, it samples $y \xleftarrow{\$} \mathcal{X} = \mathcal{Y}$ (here, we use the fact that $\mathcal{X}$ is efficiently-samplable, and that $\mathcal{X} = \mathcal{Y}$) and sets $x \leftarrow \mathsf{Dec}(\mathsf{pp}, y)$. It outputs x as the puzzle (and keeps y as the solution). Since Setup and Dec for a VDF are both efficient (i.e., run in time $\mathrm{poly}(\lambda, t)$), the puzzle generator is also efficient.

We can now use the result of [MMV11] which shows that any time-lock puzzle in the ROM where the puzzle-generation algorithm makes k queries and the puzzle-solving algorithm makes T queries can be broken by an adversary making $O(k)$ rounds of queries and a total of $O(k \cdot T)$ queries. For the time-lock puzzle based on the permutation VDF, $k = s + t$, where s is the number of queries made by the Setup algorithm and t is the number of queries made by the Dec algorithm. □

3.1.2 Proof of Theorem 3.1 in the General (Perfectly Unique) Case

We now give the proof of Theorem 3.1. It 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. We begin with an informal overview of the proof before providing the formal analysis.

Proof overview. We begin with an informal description of the main ideas behind the lower bound. Our goal is to construct an adversary that can efficiently find an output that passes verification, while asking fewer than T *rounds* of queries to the random oracle. To do so, we consider an algorithm that implements the honest evaluation algorithm, but instead of issuing queries to the actual random oracle, the algorithm will instead *sometimes* simulate the outputs from those queries itself (i.e., by sampling from the output distribution of the random oracle). Of course, if one of these "faked" or "simulated" queries was asked to the oracle by another algorithm in the system (e.g., by Setup or Verify), then our algorithm will almost certainly fail. On the other hand, if none of the simulated queries were asked to the oracle, then our attacker wins (because the set of "faked" queries and real queries together form an oracle that is consistent). This means that as long as the number of queries Setup and Verify make are much smaller than T and our algorithm is able to "identify" or

"learn" those queries with fewer than T rounds of queries to the random oracle, then the algorithm succeeds in breaking sequentiality of the VDF.

Proof of Theorem 3.1. Without loss of generality, assume that Eval asks no repeated queries in a single execution. We construct an attacker Adv that breaks sequentiality of the VDF as follows. Our adversary is entirely online (i.e., there is no separate preprocessing step).

1. At the beginning of the game, the adversary Adv receives the public parameters pp and a challenge $x \in \mathcal{X}$ from the sequentiality challenger.
2. Initialize a query set $Q_{\mathsf{Adv}} \leftarrow \varnothing$ and a set of query-answer pairs $P_{\mathsf{Adv}} \leftarrow \varnothing$.
3. Let $d = 2(s+t)+1$.
4. For $i \in [d]$, do the following:
   (a) Initialize $P^{(i)}_{\mathsf{Adv}} \leftarrow \varnothing$ and $Q^{(i)}_{\mathsf{Adv}} \leftarrow \varnothing$.
   (b) Execute $(y_i, \pi_i) \leftarrow \mathsf{Eval}(\mathsf{pp}, x)$ where the random oracle queries (made by Eval) are answered using the following procedure. On every oracle query q:
      - If $q \in Q_{\mathsf{Adv}}$, then reply with the value r where $(q, r) \in P_{\mathsf{Adv}}$.
      - Otherwise, choose a uniformly random value $r \xleftarrow{\$} \mathcal{R}$ (where $\mathcal{R}$ is the range of the random oracle $\mathcal{O}$) and add (q, r) to $P^{(i)}_{\mathsf{Adv}}$ and add q to $Q^{(i)}_{\mathsf{Adv}}$.
   (c) If $i < d$, then in *one round*, for all $(q, \star) \in P^{(i)}_{\mathsf{Adv}}$, query the real oracle $\mathcal{O}$ to get $r \leftarrow \mathcal{O}(q)$ as the answer; and then add (q, r) to P_{Adv} and add q to Q_{Adv}.
5. Output $\mathrm{Maj}(y_1, ..., y_d)$ where Maj denotes the majority operation (which outputs $\perp$ if no majority exists).

Algorithm 1: The adversary Adv that breaks sequentiality of the VDF

We now show that Adv in Algorithm 1 satisfies the properties needed in Theorem 3.1. Let Q_S be the queries made by the setup algorithm $\mathsf{Setup}(1^\lambda, T)$ to sample pp and Q_V be the queries made by $\mathsf{Verify}(\mathsf{pp}, x, y)$ where $y = \mathsf{Eval}(\mathsf{pp}, x)$ is the true solution.

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

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

Claim 3.3. *If $\mathcal{I} = \{i : H_i \ \ holds\}$, then $|\mathcal{I}| \leq s + t$.*

Proof. If event H_i occurs as a result of some query q, then at the end of round i, Adv queries the oracle $\mathcal{O}$ on the input q. By construction, since $q \in (Q^{(i)}_{\mathsf{Adv}} \cap (Q_S \cup Q_V)) \setminus Q_{\mathsf{Adv}}$, it must be the case that Adv makes a *new* query on an input that was previously queried by either the Setup or Verify algorithms (but *not* queried in any of the previous rounds). However, since Setup and Verify together ask a combined total of (at most) $s + t$ queries, this event cannot happen more than $s + t$ times. □

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 Adv 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 perfect 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 $\mathsf{Verify}^{O'}(\mathsf{pp}, x, y, \pi) = 1$. □

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 Adv. □

3.2 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 (see [DGMV19] for more discussion on this tightness notion) 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 $\sigma = T/2$). The following definition derives publicly-verifiable proofs of sequential work [DN92, CLSY93, RSW96, MMV13] as a relaxation of VDFs.[4]

Definition 3.5 (Publicly-Verifiable Proofs of Sequential Work)**.** A *publicly-verifiable* proof of sequential work is a relaxation of a VDF where the uniqueness property is not needed. Hence, there is no need to distinguish between an output y and a proof π. In particular, $y = \pi$ can be the only (not-necessarily-unique) output of Eval that is still sequentially hard to compute.

Proofs of sequential work in the ROM. While Definition 2.9 defines the notion of a VDF in the ROM, the same definition extends to the setting of (publicly-verifiable) proofs of sequential work in the ROM as well. Namely, we measure the running time in terms of the total number of oracle queries and the parallel time by the the number of rounds of oracle queries. We now show how to adapt our techniques for ruling our perfectly-unique VDFs in the ROM to also rule out tight proofs of sequential work.

Theorem 3.6 (Attacking Proofs of Sequential Work in the ROM)**.** *Suppose* $\Pi_{\mathsf{PSW}} = (\mathsf{Setup}, \mathsf{Eval}, \mathsf{Verify})$ *is a publicly-verifiable proof of sequential 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* $1 < G < T$ *there is an adversary* Adv *that asks a total of at most* $T - G$ *queries and breaks sequentiality (Definition 2.4) with probability at least* $1 - (s + t) \cdot G/T$*.*

Corollary 3.7 (Ruling Out Tight Proofs of Sequential Work in the ROM)**.** *Let* λ *be a security parameter and* T *be the time bound parameter. For any choice of* $s, t = \mathrm{poly}(\lambda, \log T)$*, there does not exist a proof of sequential work in the ROM with sequentiality* $\sigma = T \cdot (1 - 1/2(s+t))$*. More generally, for any constant* $0 < \rho < 1$*, there does not exist a proof of sequential work in the ROM with sequentiality* $\sigma = T - T^{\rho}$*.*

[4] Definition 3.5 is even more general as it allows a setup phase.

Proof. The first statement follows by instantiating Theorem 3.6 with $G = T/2(s+t)$. For this setting of parameters, Theorem 3.6 implies an adversary that breaks sequentiality with probability at least $1/2$ and making only $T - G = T \cdot (1 - 1/2(s+t))$ queries. For the more general statement, we take T to be a sufficiently large polynomial in the security parameter λ such that $s, t < T^{1-\rho}/4$. Note that this is always possible since $s, t = \mathrm{poly}(\lambda, \log T)$; that is, both s, t are poly-logarithmic in T and $0 < \rho < 1$ is constant. Then, we can again set $G = T - \sigma = T^{\rho}$ and appeal to Theorem 3.6 to obtain an adversary that makes $T - G = \sigma$ queries and breaks sequentiality with probability at least $1/2$. □

Remark 3.8 (Secretly-Verifiable Tight Proofs of Sequential Work)**.** We note that Theorem 3.6 also holds for secretly-verifiable proofs of sequential work as well. The proof for the secretly-verifiable setting is identical to that for the publicly-verifiable setting. For simplicity of notation we write the proof for the publicly-verifiable version (Definition 2.4) which uses pp as both the evaluation and the verification key.

Remark 3.9 (Other Extensions)**.** The extension from Remark 3.8 on ruling out secretly-verifiable tight proofs of sequential work relies on the same proof as that of Theorem 3.6. In Section 4, we show two extensions of this result by adapting the proof of Theorem 3.6. These include extending the lower bound to: (1) the setting where the Setup algorithm is "slow" (i.e., runs in time proportional to T); and (2) the random permutation model (rather than the random oracle model). We refer to Section 4 for the details on those extensions.

Proof of Theorem 3.6. Again, without loss of generality, we assume that Eval asks no repeated queries in a single execution. The attacker's algorithm Adv is defined as follows:

1. At the beginning of the game, the adversary Adv receives the public parameters pp and a challenge $x \in \mathcal{X}$ from the sequentiality challenger.
2. Pick a random set $\mathcal{S} \subseteq [T]$ of size $T - G$.
3. Execute $(y, \pi) \leftarrow \mathsf{Eval}(\mathsf{pp}, x)$ where the i^{th} oracle query q_i is answered as follows:
   - If $i \in \mathcal{S}$, compute the response $r_i \leftarrow \mathcal{O}(q_i)$ from the true oracle $\mathcal{O}$.
   - Otherwise sample a uniformly random $r_i \leftarrow \mathcal{R}$ as the response for the query q_i.
4. Output (y, π).

Algorithm 2: The adversary Adv that breaks sequentiality for the proof of sequential work

To analyze Algorithm 2, we compare the output of the attacker's experiment (Real) with the output in an "ideal" experiment (Ideal) where all of the oracle queries are answered using the real oracle. We define these two experiments below:

1. Sample $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, T)$.
2. Sample $x \xleftarrow{\$} \mathcal{X}$.
3. In Experiment Real, run Algorithm 2 to obtain a pair (y, π). In Experiment Ideal, run Algorithm 2, except use the real oracle $\mathcal{O}$ to answer *all* of the oracle queries made by Eval (in Step 3 of Algorithm 2).
4. The output of the experiment is 1 if $\mathsf{Verify}(\mathsf{pp}, x, y, \pi) = 1$ and 0 otherwise.

Figure 1: The Real and Ideal experiments

In the following, we write $\Pr_{\mathsf{real}}[\cdot]$ (resp., $\Pr_{\mathsf{ideal}}[\cdot]$) to denote the probability of an event E in the Real (resp., Ideal) experiment.

Events. Let Q_V be the set of oracle queries made by Verify and Q_S be the set of oracle queries made by Setup. Let Q_{Adv} be the set of oracle queries q_i that appears in Step 3 of Algorithm 2 where $i \notin \mathcal{S}$. Namely, these are the set of oracle queries q_i that Adv answers with uniformly random values in Real. We now define the following two 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).
- Let B be the "bad" event where $(Q_V \cup Q_S) \cap Q_{\mathsf{Adv}} \neq \varnothing$. Namely, this is 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 (by perfect completeness of the underlying proof of sequential work[5]), as all of the oracle queries q_i are computed using the real oracle $\mathcal{O}(q_i)$.

Claim 3.10. $\Pr_{\mathsf{ideal}}[W] = 1$.

The following lemma states that until event B happens, the two experiments are identical.

Lemma 3.11. $\Pr_{\mathsf{real}}[B] = \Pr_{\mathsf{ideal}}[B]$. *Moreover, conditioned on the event B not happening, the two experiments are identically distributed. 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 the 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 evaluation 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 perfect sense) and *consistently* according to their own distribution, it means that until event B happens, the two games have the same exact distributions. Therefore, by the "fundamental lemma of game

[5]Note that this argument extends also to the setting where we have completeness error γ (i.e., completeness holds with probability $1 - \gamma$ over the choice of the public parameters). This modification introduces a γ loss in the adversary's advantage in the real game.

playing" [Mau02, BR04] it holds that $\Pr_{\mathsf{real}}[B] = \Pr_{\mathsf{ideal}}[B]$ and that conditioned on the event B not happening, the two experiments are identically distributed. □

We now observe that the probability of the event B is small in the ideal game, and conclude that it is indeed small in both games.

Claim 3.12. $\Pr_{\mathsf{ideal}}[B] \leq (s+t) \cdot \frac{G}{T}$.

Proof. In the ideal experiment, the set $\mathcal{S}$ is independent of all other components in the experiment, so we can choose $\mathcal{S}$ *at the end* of the experiment (*after* Q_S and Q_V have been determined). By definition of Q_{Adv}, we have that $|Q_{\mathsf{Adv}}| \leq G$. Therefore, for any query $q \in Q_S \cup Q_V$ that is also queried by $\mathsf{Eval}(\mathsf{pp}, x)$, the probability that $q \in Q_{\mathsf{Adv}}$ is at most G/T. The claim now follows by a union bound. □

The above claims complete the proof of Theorem 3.6, 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 $\Pr_{\mathsf{ideal}}[W] = 1$, therefore, we conclude that

$$\Pr_{\mathsf{real}}[W] \geq \Pr_{\mathsf{ideal}}[W] - \Pr_{\mathsf{ideal}}[B] \geq 1 - (s+t) \cdot \frac{G}{T}.$$

□

4 Extensions to the Lower Bounds

In this section, we briefly discuss several extensions of our lower bounds on perfectly unique VDFs and tight proofs of sequential work by extending the proofs of Theorems 3.1 and 3.6.

4.1 Handling Expensive Setup Phase

Our lower bounds in Theorems 3.1 and 3.6 construct an adversary whose running time depends on both t (the verification time) as well as s (the setup time). When these bounds t, s are only $\mathrm{poly}(\lambda, \log T)$, the running time of our attacker is much smaller compared to T, and thus, breaks the sequentiality of the scheme. However, one might argue that since the setup is executed only once, it is reasonable to consider a scenario where s is potentially as large as T. In this case, *both* of our lower bounds in Theorems 3.1 and 3.6 become meaningless.

Case of expensive *public* setup. If the setup algorithm is public coin (i.e, the randomness to Setup is publicly known), then both Theorems 3.1 and 3.6 directly extend to the setting where $s = \mathrm{poly}(\lambda, T)$. Specifically, in this setting, the setup queries can be discovered publicly (i.e., by emulating an execution of Setup). Thus, the adversary can learn the oracle's values on the set of queries Q_S made by Setup. In the online phase, the adversary carries out its attack by *incorporating* the knowledge of Q_S when answering oracle queries. In other words, if during the emulation of the Eval algorithm, the adversary encounters a query $q \in Q_S$, it will use the known answer (saved as part of the state $\mathsf{st}_{\mathsf{Adv}}$) instead of asking it from the oracle or guessing its answer.

Case of expensive *private* setup for Theorem 3.6. When the setup algorithm uses *private* randomness, the above argument for extending Theorems 3.1 and 3.6 no longer applies. Nonetheless, we can still show how the proof of Theorem 3.6 (for ruling our tight proofs of sequential work) can be extended to this setting as well.

Theorem 4.1 (Attacking Tight Proofs of Sequential Work with Expensive Setup in the ROM)**.** *Suppose* $\Pi_{\mathsf{PSW}} = (\mathsf{Setup}, \mathsf{Eval}, \mathsf{Verify})$ *is a publicly-verifiable proof of sequential work in the ROM in which (for a concrete choice of* λ, T*),* Setup *runs in time* s*,* Eval *runs in time* T*,* Verify *runs in time* t*, and completeness error is at most* γ *(see Definition 2.2). Then, for any* $\varepsilon < 1$ *and* $1 < G < T$*, there is an adversary* $\mathsf{Adv} = (\mathsf{Adv}_0, \mathsf{Adv}_1)$ *where* Adv_0 *runs in time* $\mathrm{poly}(s, T, t, 1/\varepsilon)$ *and* Adv_1 *makes at most* $T - G$ *queries (and run in total time* $\mathrm{poly}(T)$*) and breaks sequentiality (Definition 2.4) with probability at least* $1 - \varepsilon - t \cdot G/T - \gamma$*.*

Before proving Theorem 4.1, we first derive a corollary, formally stating the range of tight sequentiality for which we rule out PoSWs in the ROM.

Corollary 4.2 (Ruling Out Tight Proofs of Sequential Work with Expensive Setup in the ROM)**.** *Let* λ *be a security parameter and* T *be the time bound parameter. For any choice of* $s = \mathrm{poly}(\lambda, T), t = \mathrm{poly}(\lambda, \log T)$*, and completeness error* $\gamma = \mathrm{negl}(\lambda)$*, there does not exist a proof of sequential work in the ROM with sequentiality* $\sigma = T \cdot (1 - 1/2t)$ *and completeness error* γ*. In particular, for any constant* $0 < \rho < 1$*, there does not exist a proof of sequential work in the ROM with sequentiality* $\sigma = T - T^{\rho}$ *and completeness error* γ*.*

Proof. The proof is identical to that of Corollary 3.7, except here, we use the attack algorithm from Theorem 4.1 that relies on a preprocessing step with time complexity $\mathrm{poly}(\lambda, T)$. The only difference is that we now also need to make sure $\varepsilon + \gamma < 1/2$, which is easily satisfied whenever $\gamma = \mathrm{negl}(\lambda)$. For example, take $\varepsilon = 1/3$. □

We now give the proof of Theorem 4.1.

Proof of Theorem 4.1. Here, Adv_0 will perform a precomputation and compute the so-called "ε-heavy" queries [BM17, IR89] of the setup algorithm. More precisely, we aim to find queries that are asked by the setup and have noticeable chance of being queried again during the evaluation. In particular, Algorithm Adv_0 outputs a set Q consisting of input-output pairs of the oracle $\mathcal{O}$. The online algorithm Adv_1 then follows Algorithm 2 from the proof of Theorem 3.6, except whenever Eval makes a query q where $q \in Q$, Adv_1 replies with the precomputed value of $\mathcal{O}(q)$ in Q. We now describe Adv_0:

1. Initialize a set $Q \leftarrow \varnothing$.
2. Repeat the following procedure $k = s/\varepsilon = \mathrm{poly}(\lambda, T)$ times:
   (a) Sample $x \xleftarrow{\$} \mathcal{X}$.
   (b) Compute $(y, \pi) \leftarrow \mathsf{Eval}(\mathsf{pp}, x)$. Whenever Eval makes a query q to the oracle $\mathcal{O}$, add the pair $(q, \mathcal{O}(q))$ to Q.
3. Output $\mathsf{st}_{\mathsf{Adv}} = Q$.

Algorithm 3: The adversary Adv_0 that precomputes the ε-heavy queries of the Setup algorithm.

We define Adv_1 as in Algorithm 2, except in Step 3, if the query q_i is contained in $\mathsf{st}_{\mathsf{Adv}} = Q$, then the adversary always replies with the precomputed value of $\mathcal{O}(q_i)$ in Q. Otherwise, it uses the same procedure as in Algorithm 3. The rest of the analysis proceeds similar to that in the proof of Theorem 3.6. Namely, let Q_S denote the set of oracle queries made by Setup and Q_V be the set of queries made by Verify in the real/ideal experiments. We define the events W and B exactly as in the proof of Theorem 3.6. We now show an analog of Claim 3.12:

Claim 4.3. $\Pr_{\mathsf{ideal}}[B] \leq \varepsilon + t \cdot \frac{G}{T}$.

Proof. Recall that the event B occurs if $(Q_V \cup Q_S) \cap Q_{\mathsf{Adv}} \neq \varnothing$. We consider the following two setting. (All probabilities are stated in the ideal experiment.)

- Consider the event that $Q_V \cap Q_{\mathsf{Adv}} \neq \varnothing$. By the same argument as in the proof of Claim 3.12, this event occurs with probability at most $|Q_V| \cdot G/T \leq t \cdot G/T$ over the randomness of $\mathcal{S}$.
- Consider the probability that $Q_S \cap Q_{\mathsf{Adv}} \neq \varnothing$. Take any query $q \in Q_S$ and consider the event H_q that $q \in Q_{\mathsf{Adv}}$ and $q \notin \mathsf{st}_{\mathsf{Adv}}$. By construction of Algorithm 3, for H_q to happen, the first k iterations of Adv_0 should *not* query q during Eval, and yet q *is* queried in the next (actual) execution of Eval. However, it is easy to show that for any Bernoulli variable (of arbitrary probability α) the probability of missing it k times and hitting it on the $(k+1)^{\text{st}}$ iteration is at most $1/k$. Since Setup makes at most s queries, $|Q_S| \leq s$, so by a union bound,

$$\Pr[Q_S \cap Q_{\mathsf{Adv}} \neq \varnothing] \leq \frac{s}{k} = \varepsilon.$$

The claim now follows by a union bound. □

The rest of the proof of Theorem 4.1 proceeds identically to the proof of Theorem 3.6 and noting that the probability of W in the ideal game is $1 - \gamma$. □

4.2 Extending to the Random Permutation Oracle Model

Random oracles can be used to instantiate all symmetric primitives (including the ideal cipher model [CPS08, HKT11]) with one exception: the random permutation oracle R that implements a random permutation on $\{0,1\}^n$ for all $n \in \mathbb{N}$. Indeed there are impossibility results showing that such a construction does not exist [Rud88, MM11]. We can extend the proof of Theorem 3.6 to rule out constructions of tight proofs of sequential work in the random permutation oracle as well by developing a preprocessing attack and then using standard techniques based on the fact that a random permutation oracle and a random oracle over a large domain is statistically indistinguishable to any query-bounded algorithm [IR89].

Case of large input domains. More formally, let n be such that $(T+s+t)^2/2^n \leq \varepsilon$. Then, first suppose the only domain used by the three algorithms $(\mathsf{Setup}, \mathsf{Eval}, \mathsf{Verify})$ is $\{0,1\}^n$. In this case, the probability that a random oracle used by these three algorithms $\mathsf{Setup}, \mathsf{Eval}, \mathsf{Verify}$ has *any* collision during the course of their execution is at most $(T+s+t)^2/2^n \leq \varepsilon$. However, whenever there are no collisions, there is no way to distinguish between random permutations or random oracles. Therefore, our attack in Theorem 3.6 would automatically work (as is) up to a loss of ε in the success probability. This argument also works if the three algorithms $(\mathsf{Setup}, \mathsf{Eval}, \mathsf{Verify})$ ask their queries from input domains with *different* size as long as for any domain $\{0,1\}^n$ that they query, n satisfies $(T+s+t)^2/2^n \leq \varepsilon$.

General case. The above argument fails when any of the algorithms ($\mathsf{Setup}, \mathsf{Eval}, \mathsf{Verify}$) ask their queries from any domain $\{0,1\}^n$ where n is small (and as such, $(T+s+t)^2/2^n > \varepsilon$). However, the *total* number of queries over *all* such domains is at most

$$\tau = |\{0,1\}^1| + |\{0,1\}^2| + \cdots + |\{0,1\}^n|,$$

where $(T+s+t)^2/2^n = \varepsilon$, which means

$$\tau = 2 + 4 + \cdots + \frac{(T+s+t)^2}{\varepsilon} < \frac{2(T+s+t)^2}{\varepsilon} = \mathrm{poly}(\lambda, T).$$

Therefore, a preprocessing adversary Adv_0 can ask all of these $\tau = \mathrm{poly}(\lambda, T)$ queries from the real oracle $\mathcal{O}$ and send them together with their answers to the online adversary Adv_1 who will then use the answers to these queries whenever needed without asking them from $\mathcal{O}$ or guessing their answers. In this case, the analysis of our attack is identical to the aforementioned case with inputs drawn from a large domain.

5 Conclusion and Open Questions

In this work, we initiated a formal study of the assumptions behind VDFs and provided new lower bounds on basing VDFs with perfect uniqueness in the random oracle model as well as stronger lower bounds in the tight security regime in which the sequentiality guarantees a very close running time to the honest execution. The second lower bound applies not only to VDFs but also to relaxations of it such as sequential proofs of work. While our first lower bound captures existing notions [LW15, AKK+19], they do not extend to the full range of VDF notions.

The main open question remaining is whether we can extend our first lower bound to rule out VDFs satisfying *computational uniqueness* in the ROM, and ideally do so allowing negligible completeness error as well. Alternatively, the fact that our current lower bound critically relies on the perfect uniqueness property may suggest new approaches to basing VDFs on weaker assumptions. In other words, any approach for constructing VDFs in the ROM must either rely on non-black-box techniques or leverage imperfect soundness in a critical manner. Both of these possibilities represent intriguing avenues for further research.

References

[AKK+19] Hamza Abusalah, Chethan Kamath, Karen Klein, Krzysztof Pietrzak, and Michael Walter. Reversible proofs of sequential work. In *EUROCRYPT*, pages 277–291, 2019.

[AS15] Joël Alwen and Vladimir Serbinenko. High parallel complexity graphs and memory-hard functions. In *STOC*, pages 595–603, 2015.

[BBBF18] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In *CRYPTO*, pages 757–788, 2018.

[BBHM02] Ingrid Biehl, Johannes A. Buchmann, Safuat Hamdy, and Andreas Meyer. A signature scheme based on the intractability of computing roots. *Des. Codes Cryptogr.*, 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 *TCC*, pages 559–578, 2011.

[BM17] Boaz Barak and Mohammad Mahmoody. Merkle's key agreement protocol is optimal: An $o(n^2)$ attack on any key agreement from random oracles. *J. Cryptology*, 30(3):699–734, 2017.

[BR04] Mihir Bellare and Phillip Rogaway. Code-based game-playing proofs and the security of triple encryption. Cryptology ePrint Archive, Report 2004/331, 2004. https://eprint.iacr.org/2004/331.

[CLSY93] Jin-yi Cai, Richard J. Lipton, Robert Sedgewick, and Andrew Chi-Chih Yao. Towards uncheatable benchmarks. In *Structure in Complexity Theory Conference*, pages 2–11, 1993.

[CP18] Bram Cohen and Krzysztof Pietrzak. Simple proofs of sequential work. In *EUROCRYPT*, pages 451–467, 2018.

[CPS08] Jean-Sébastien Coron, Jacques Patarin, and Yannick Seurin. The random oracle model and the ideal cipher model are equivalent. In *CRYPTO*, pages 1–20, 2008.

[DGMV19] Nico Döttling, Sanjam Garg, Giulio Malavolta, and Prashant Nalini Vasudevan. Tight verifiable delay functions. *IACR Cryptology ePrint Archive*, 2019:659, 2019.

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

[DN92] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In *CRYPTO*, pages 139–147, 1992.

[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. In *ASIACRYPT*, pages 248–277, 2019.

[HC19] Matt Howard and Bram Cohen. Chia network announces 2nd VDF competition with $100,000 in total prize money, 2019.

[HKT11] Thomas Holenstein, Robin Künzler, and Stefano Tessaro. The equivalence of the random oracle model and the ideal cipher model, revisited. In *STOC*, pages 89–98, 2011.

[IR89] Russell Impagliazzo and Steven Rudich. Limits on the provable consequences of one-way permutations. In *STOC*, pages 44–61, 1989.

[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 *USENIX Security Symposium*, pages 279–296, 2016.

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

[LW15] Arjen K. Lenstra and Benjamin Wesolowski. A random zoo: sloth, unicorn, and trx. *IACR Cryptology ePrint Archive*, 2015:366, 2015.

[Mau02] Ueli Maurer. Indistinguishability of random systems. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 110–132. Springer, 2002.

[MM11] Takahiro Matsuda and Kanta Matsuura. On black-box separations among injective one-way functions. In *TCC*, pages 597–614, 2011.

[MMV11] Mohammad Mahmoody, Tal Moran, and Salil P. Vadhan. Time-lock puzzles in the random oracle model. In *CRYPTO*, pages 39–50, 2011.

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

[MSW19] Mohammad Mahmoody, Caleb Smith, and David J. Wu. A note on the (im)possibility of verifiable delay functions in the random oracle model. Cryptology ePrint Archive, Report 2019/663, 2019. https://eprint.iacr.org/2019/663/20190606:045724.

[MSW20] Mohammad Mahmoody, Caleb Smith, and David J. Wu. Can verifiable delay functions be based on random oracles? In *ICALP*, 2020.

[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, 1996.

[Rud88] Steven Rudich. *Limits on the Provable Consequences of One-way Functions*. PhD thesis, EECS Department, University of California, Berkeley, 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 *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 379–407. Springer, 2019.