

Separating One-Way Functions from Verifiable Delay Functions

Hamza Abusalah¹, Nicholas Brandt², Ziyi Guan³, Kristina Hostáková⁴

¹ Aalborg University, Aalborg, Denmark, hamzaabusalah@gmail.com

² `[[alloc] init] crypto@nicholasbrandt.de`

³ MIT, Cambridge, MA, USA, ziyiguan@mit.edu

⁴ ETH Zurich, Zurich, Switzerland, kristina.hostakova@inf.ethz.ch

Timed cryptography captures computations that inherently require a prescribed amount of time, even in the presence of parallelism, and has emerged as a central notion due to its broad applications in both practice and theory. A natural question is how timed cryptographic primitives relate to traditional cryptography, in particular to primitives in Minicrypt.

Guan and Montgomery (CRYPTO 2024) show that sequential functions do not imply one-way functions (OWFs) via black-box reductions. Their result, however, leaves open whether adding public verifiability (a substantially stronger requirement) suffices to imply OWFs. We study this question via verifiable delay functions (VDFs), which require substantial sequential time to evaluate, yet produce a unique output that is efficiently and publicly verifiable.

We show that VDFs do not imply OWFs via fully black-box, relativizing reductions. Combined with prior results showing that VDFs cannot be constructed in the random oracle model, this establishes that there is no black-box implication in either direction between VDFs and OWFs.

Our proof develops a new oracle separation technique that combines the random oracle with a carefully designed complexity-theoretic oracle inspired by Kolmogorov complexity.

Contents

1	Introduction	2
1.1	Related Work	2
1.2	Our Techniques	3
2	Preliminaries	4
3	VDF Definition	6
4	Technical Overview	11
4.1	The Initial Idea	11
4.2	Breaking PRGs (Resolving Circularity)	12
4.3	Proving Sequentiality	14
4.4	PRGs are Still Broken	15
5	The Formal Result	16
5.1	The Oracle Distribution	16
5.2	The Main Result	19
5.3	Part One: Relativized PRGs Don't Exist	19
5.4	Part Two: Relativized VDFs Exist	22
5.5	Part Three: Relativized PRGs from One-Way Functions	30
6	Acknowledgments	32
7	AI Disclosure	32

1 Introduction

Traditional cryptographic primitives are usually built by leveraging computational hardness, measured by the total amount of *work* required to perform a computation. One-way functions (OWFs) capture a minimal form of such hardness and form the foundation of Minicrypt. Timed cryptography, by contrast, is built on *sequential hardness*: the requirement that a computation takes substantial time even when parallel resources are available. Motivated by applications in decentralized computation, this alternative form of hardness has received growing attention. It raises a natural question:

Does sequential hardness imply one-way functions?

Sequential hardness can be instantiated in a variety of models. In the random oracle model (ROM), it can be enforced by repeatedly evaluating the random oracle function [MMV13]. Beyond the ROM, sequential hardness has been instantiated under assumptions including the hardness of repeated squaring in groups of unknown order, worst-case non-parallelizing languages together with indistinguishability obfuscation, succinct arguments for P together with sequential functions, and lattice- and isogeny-based assumptions [RSW96; BN00; GMP⁺06; LW15; Wes19; Pie19; DMP⁺19; MT19; Sha19; DGM⁺20; EFK⁺20b; BDG⁺20; EFK⁺20a; LV20; KLX20; BHR⁺21; FKP⁺21; JMR⁺21; HHK⁺22; BCH⁺22; CSRT22; HHK⁺23; LM23; CLM23; OKC⁺25]. The converse direction—whether sequential hardness implies standard cryptographic hardness—is less understood. Guan and Montgomery [GM24] initiated a systematic study of this question and showed that generic sequential functions, without additional algebraic or structural properties, do not imply OWFs via black-box reductions.

We study sequential hardness in combination with *public verifiability*, a property reminiscent of those found in other cryptographic primitives, such as zero-knowledge proof systems. Specifically, we focus on verifiable delay functions (VDFs) [BBB⁺18], which require substantial sequential time to evaluate, produce a computationally unique output, and allow efficient public verification. Public verifiability is a natural strengthening to consider: for sufficiently strong proof systems, such as zero-knowledge proofs, public verifiability together with simulatability is known to imply OWFs [Ost91; OW93; Ps05; CHK25; HN24; LMP24]. This leads us to ask whether VDFs imply OWFs. We answer this question negatively:

Theorem 1 (informal). There exists an oracle relative to which verifiable delay functions exist, while one-way functions do not.

Our oracle separation rules out fully black-box constructions of OWFs from VDFs. In the other direction, prior works [MSW20; DGM⁺20; GRY25; AAK⁺25] show that VDFs do not exist in the random oracle model, even though OWFs do. Together, these results show that VDFs and OWFs are incomparable under fully black-box, relativizing reductions.

1.1 Related Work

VDFs are motivated by both practical applications and theoretical connections. They have found applications in randomness beacons [LW15; BBB⁺18; EFK⁺20a; Sta20], computational time-stamping [LSS20], resource-efficient blockchains [CP19], proofs of replication [ABB⁺16], and short-lived signatures [ABC22]. On the theoretical side, several works connect VDF constructions to approaches for proving hardness of total search problems [BPR15; HY17; LV20; BCH⁺22].

To place our result in the broader landscape of timed cryptography, we compare VDFs with three related primitives: proofs of sequential work (PoSWs) [MMV13], sequential functions [MMV13], and time-lock puzzles (TLPs) [RSW96; BGJ⁺16]. We organize the comparison around what is known about the relationship between each primitive and OWFs.

Proofs of sequential work. The primitive most closely related to VDFs is that of proofs of sequential work (PoSW), which relax the definition of VDFs by removing the uniqueness requirement on the output.

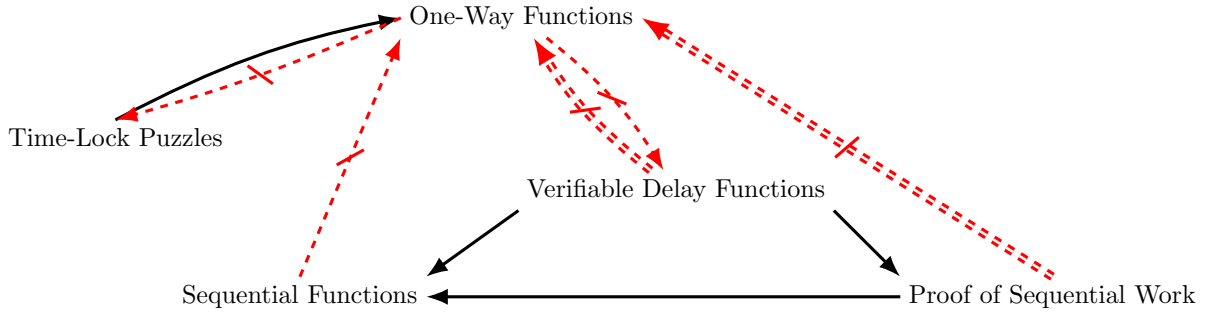


Fig. 1: Known black-box implications and separations involving timed cryptographic primitives and one-way functions. Solid black arrows denote black-box implications; dashed red arrows with a tick denote black-box separations; double dashed red arrows with a tick highlight the new separations established in this work, directly or as consequences of Theorem 1.

Unlike VDFs, PoSWs can be constructed in the random oracle model [MMV13; CP18; AKK⁺19; DLM19; AFG⁺22; AC23; Abu25]. Since every VDF is, in particular, a PoSW, Theorem 1 immediately implies a separation between PoSWs and OWFs in the same oracle model.

Sequential functions. Moving to a more basic notion, sequential functions are arguably the simplest timed cryptographic primitive. As mentioned earlier, it is known that there exists an oracle model in which sequential functions can be constructed, yet one-way functions do not exist [GM24]. This previously known separation also follows from Theorem 1, since every VDF is a sequential function. In contrast, sequential functions can be constructed in the random oracle model [MMV13].

Time-lock puzzles. Time-lock puzzles (TLPs), however, behave differently from generic sequential functions. A TLP is a puzzle that can be quickly generated, but solving it requires a prescribed amount of time, even when parallel adversaries are allowed. Similar to VDFs, TLPs are designed to enforce sequential computation that cannot be substantially sped up via parallelism. A key distinction, however, is that VDFs require efficient public verification of a time-expensive computation, whereas TLPs require efficient generation of a puzzle corresponding to a privately chosen solution under a time-expensive computation. In contrast to sequential functions and PoSWs, TLPs are known not to exist in the random oracle model [MMV11]. Moreover, [BGJ⁺16] shows that TLPs imply one-way functions. Taken together, these results highlight a fundamental distinction between sequential functions, TLPs, and VDFs.

We summarize the current state-of-the-art on known black-box implications and separations involving timed cryptographic primitives and OWFs in Figure 1.

1.2 Our Techniques

To prove Theorem 1, we use pseudorandom generators (PRGs) as an intermediate step. The construction of Haitner, Reingold, and Vadhan [HRV10] transforms any OWF into a PRG in a fully black-box way. Therefore, this transformation continues to hold relative to arbitrary oracle distributions. It is therefore enough to construct an oracle distribution relative to which VDFs exist but PRGs do not.

We use three oracles. A random oracle F , together with a verification oracle V , yields a VDF based on iterated random-oracle evaluation. To rule out PRGs, we begin with a complexity oracle $L^{F,V}$ inspired by Kolmogorov complexity, which identifies strings that admit short descriptions relative to (F, V) . If a candidate PRG accesses only (F, V) , its seed and code give such a short description of every output,

allowing $L^{F,V}$ to distinguish PRG outputs from uniformly random strings. The central challenge is extending this idea to PRGs that themselves query the complexity oracle. We explain how we resolve this circularity in Section 4.2.

After introducing the complexity oracle, proving sequentiality for VDFs requires additional care: we need to make sure that the presence of the complexity oracle does not give speedups to VDF adversaries. In fact, we show that any adversary with access to all three oracles can be simulated by an adversary with access only to F . Thus, any successful attack on the VDF would contradict the inherent sequential hardness of iterated random-oracle evaluation; see Section 4.3. Together, these arguments yield an oracle world relative to which VDFs exist while PRGs, and therefore OWFs, do not.

Remark 1 (Why the two-oracle shortcut does not suffice). At this point, one might try to avoid the circularity from the previous section by using the asymmetric two-oracle technique of [HR04]: restrict the candidate PRG to oracle access to (F, V) , and give only its distinguisher access to the standard Kolmogorov-complexity oracle $L^{F,V}$. This indeed suffices to break the PRG.

It does not, however, resolve the sequentiality problem. A fully black-box security reduction may invoke this distinguisher, so the VDF must remain sequential against adversaries with access to $(F, V, L^{F,V})$. This does not follow from the usual random-function lower bound: a single query to $L^{F,V}$ may depend on unbounded computations with oracle access to (F, V) , and may therefore implicitly inspect arbitrarily many points in the relevant F -slice. In fact, writing $O = (F, V, L^{F,V})$, proving sequentiality in this setting would in particular imply $P^O \neq NP^O$ for this specific structured oracle, a (variant of a) major open question in complexity theory. [ABK06; ABK⁺06]

The paper is structured as follows:

- Section 2 provides the formal notations and definitions that we use.
- Section 3 elaborates on the definition of VDFs.
- Section 4 explains the overall proof idea along with the technical challenges we need to resolve.
- Section 5 states and proves our formal VDF–OWF separation.

2 Preliminaries

Notation. When possible, we try to associate the style of a variable with its type, so that it can be parsed context-independently. We may deviate from this style if semantic conventions clearly suggest a different style. In particular, integers are denoted by lowercase Latin letters, e.g. n, k, i, j (exceptions include λ, T). Strings are usually denoted by lowercase Latin letters towards the end of the alphabet, e.g. x, y, z (exceptions include Π, Φ). Sets and languages are denoted by uppercase Latin letters, e.g. L . Functions are denoted by lowercase Fraktur/Greek letters or sans-serif Latin letters, e.g. f, δ, F . (Fraktur functions are typically functions from/to natural numbers whereas Greek functions map to the reals.) Probability distributions are denoted by uppercase calligraphic letters, e.g. $\mathcal{D}$. For algorithms (e.g. Turing machines or circuits) we use the function symbol, e.g. A , in an overloaded notation to refer both to the function and the code (description) of the algorithm. This allows us to refer to the description length $|A|$ and the execution $A(x)$ of an algorithm A without notational overhead.

We denote the empty string by ε . For any language $L \subseteq \{0, 1\}^*$ and any integer $k \in \mathbb{N}$ we denote by $[L]_{\leq k} := L \cap \{0, 1\}^{\leq k}$ the set of strings in L of length at most k . Similarly, for any function $F : \{0, 1\}^* \times \dots \times \{0, 1\}^* \rightarrow \{0, 1\}^*$ we denote by $[F]_{k_1, k_2, \dots} : \{0, 1\}^{\leq k_1} \times \{0, 1\}^{\leq k_2} \times \dots \rightarrow \{0, 1\}^*$ the restriction of F to inputs of length at most k_i on the i -th coordinate. For consistency, we define $[F]_{k_1, \dots}(x) := \perp$ for any $x \notin \{0, 1\}^{\leq k_1} \times \dots$. For any function $F : \{0, 1\}^* \rightarrow \{0, 1\}^* \cup \{\perp\}$ and any language $L \subseteq \{0, 1\}^*$ we denote by $F(L) := \{F(x) \mid x \in L\}$ the image of L under F . For each function $F : X \rightarrow X$ and each $T \in \mathbb{N}$ let $F^{(T)}(x) := F(F^{(T-1)}(x))$ and let $F^{(0)}(x) := x$. Boolean oracles are typically presented through their characteristic sets.

We use standard asymptotic notation, e.g. $o(1)$ (technically $o(\lambda \mapsto 1)$) to denote the set of vanishing functions, and $o(\lambda)$ (technically $o(\lambda \mapsto \lambda)$) to denote the set of sublinear functions.

Uniform distribution over random functions.

Definition 1. *Let*

$$F := \left\{ F : \{0, 1\}^* \times \mathbb{N} \rightarrow \{0, 1\}^* \mid \forall \lambda, T \in \mathbb{N} : F(\{0, 1\}^\lambda, T) \subseteq \{0, 1\}^\lambda \right\}$$

be the set of (indexed length-preserving) functions over finite strings. We denote by $\mathcal{F}$ the uniform distribution over F where for each $\lambda, T \in \mathbb{N}$ and each $x, y \in \{0, 1\}^\lambda$ it holds that $\Pr[F \leftarrow \mathcal{F} : F(x, T) = y] := 2^{-\lambda}$.

Computational models. The specific details of the computational model do not matter much for the “spirit” of our results, due to various simulation theorems between different computational models. Nonetheless, we pick two concrete models to be able to prove rigorous statements. Concretely, we consider Turing machines (TMs) for our primitives and Boolean circuits for adversaries. We generally assume a TM has a fixed number of tapes, one of which is a read-only input tape, one a write-only output tape, and the rest are read-write work tapes. A non-uniform TM has an additional advice tape that is initialized with an advice string that depends (exclusively but arbitrarily) on the input length (but not on the input itself). This naturally extends to oracle machines with a fixed number of dedicated oracle tapes.

Let M be a deterministic Turing machine. For any $x \in \{0, 1\}^*$ denote by $M(x) \in \{0, 1\}^* \cup \{\perp\}$ the content of the output tape after M has entered a terminal state, or $\perp$ if M does not terminate on input x . We assume that any program $F \in \{0, 1\}^*$ can be padded to any length $\ell \geq |F|$ while preserving its functionality. To denote the runtime of M on input $(\lambda, \dots)$ (resp. $(\lambda, T, \dots)$) we write $t_M(\lambda)$ (resp. $t_M(\lambda, T)$).

Remark 2 (Resource bounds). For both computational models we specify the resource bounds as functions of the first input, typically λ , or the first two inputs, typically (λ, T) —not in the length of the overall input. This notation makes it easier to express the concrete resource bounds for specific invocations of the involved algorithms/circuits.

For example, let $t : \mathbb{N} \rightarrow \mathbb{N}$ be a function and A a t -time-bounded TM; then for any $x \in \{0, 1\}^*$, on input (λ, x) , written as $A(\lambda, x)$, we require that A halts after at most $t(\lambda)$ steps, regardless of the length of x .

Circuits. On the other hand, we consider (uniform and non-uniform) bounded-fan-in Boolean circuits. In this work, the most important properties of a Boolean circuit are its size s and depth d . By $C_{s,d}$ we denote the class of all Boolean circuits with size at most s and depth at most d . For any functions $\mathfrak{s}, \mathfrak{d} : \mathbb{N}^2 \rightarrow \mathbb{N}$ we denote the class of all *uniform* families of Boolean circuits with size at most $\mathfrak{s}$ and depth at most $\mathfrak{d}$ by

$$C_{\mathfrak{s}, \mathfrak{d}}^U := \left\{ (C_{\lambda, T})_{\lambda, T \in \mathbb{N}} \mid \exists \text{TM } M \forall \lambda, T \in \mathbb{N} : M(\lambda, T) = C_{\lambda, T} \wedge C_{\lambda, T} \in C_{\mathfrak{s}(\lambda, T), \mathfrak{d}(\lambda, T)} \right\}. \quad (1)$$

In other words, a uniform circuit family has a (deterministic) circuit generator TM that on input (λ, T) outputs a circuit $C_{\lambda, T}$ of size at most $\mathfrak{s}(\lambda, T)$ and depth at most $\mathfrak{d}(\lambda, T)$.

The class of *non-uniform* families of Boolean circuits with size at most $\mathfrak{s}$ and depth at most $\mathfrak{d}$ is defined as

$$C_{\mathfrak{s}, \mathfrak{d}}^{NU} := \left\{ (C_{\lambda, T})_{\lambda, T \in \mathbb{N}} \mid \forall \lambda, T \in \mathbb{N} : C_{\lambda, T} \in C_{\mathfrak{s}(\lambda, T), \mathfrak{d}(\lambda, T)} \right\}. \quad (2)$$

If $\mathfrak{d}$ is not specified we set $\mathfrak{d}(\lambda, T) := \infty$.

Similarly to TMs, circuits can be equipped with oracle gates. In this case, the size and depth of an oracle circuit only account for the number of oracle gates in the circuit.

Let C be an oracle circuit of depth d , let x be an input for C , and let $O_1, \dots, O_m$ be fixed oracles. We write $C^{O_1, \dots, O_m}(x) \stackrel{i}{\hookrightarrow} Q_i$ to denote that the circuit C on input x makes queries $Q_i = \{(j_1^i, q_1^i), \dots, (j_k^i, q_k^i)\}$ to the oracles $O_1, \dots, O_m$ in its i -th round of oracle gates, for each round $i \in [d]$. Here, j_ℓ^i denotes the index of the oracle, i.e., (j_k^i, q_k^i) means that q_k^i is asked to oracle $O_{j_\ell^i}$.

Complexity measures. There are several notions of complexity, each describing the smallest compression of a given string in some computational model. The most basic notion of Solomonoff–Kolmogorov–Chaitin complexity [Sol60; Kol63; Cha69] is the length of the smallest program (witness program) that produces a given string. An obvious question is in which programming language the witness programs are interpreted. To solve this issue one typically specifies a universal Turing machine (UTM) relative to which the complexity notion is defined. Because there exist strong invariance theorems,⁵ the particular choice of UTM is mostly irrelevant. For this reason, we omit the explicit notion of a UTM and simply use the notation $\Pi^O(z)$ to denote the output of program Π on input z given access to oracle O .

For a given string x we say that the program that produces x is the *witness program* for x and denote it by Π_x . (If not further specified, we assume that Π_x is the lexicographically first program of minimal length that produces x .)

Pseudorandom Generators.

Definition 2 (Relativized Pseudorandom Generator). Let $\mathcal{O}$ be an oracle distribution, and let $\text{PRG}^{\mathcal{O}}: \{0,1\}^* \rightarrow \{0,1\}^*$ be a deterministic polynomial-time oracle algorithm. Suppose there is a polynomial-time computable output-length function $\ell: \mathbb{N} \rightarrow \mathbb{N}$ such that $\ell(\lambda) > \lambda$ and $\text{PRG}^{\mathcal{O}}(\{0,1\}^\lambda) \subseteq \{0,1\}^{\ell(\lambda)}$. For a non-uniform probabilistic polynomial-time oracle distinguisher D , define

$$\text{Adv}_{\text{PRG},D}^{\mathcal{O}}(\lambda) := \left| \Pr_{s \leftarrow U_\lambda} [D^{\mathcal{O}}(1^\lambda, \text{PRG}^{\mathcal{O}}(s)) = 1] - \Pr_{z \leftarrow U_{\ell(\lambda)}} [D^{\mathcal{O}}(1^\lambda, z) = 1] \right|.$$

Both probabilities include the internal randomness of D . We call $\text{PRG}^{\mathcal{O}}$ a pseudorandom generator relative to $\mathcal{O}$ if this advantage is negligible for every such D . We call $\text{PRG}^{\mathcal{O}}$ an infinitely often secure pseudorandom generator relative to $\mathcal{O}$ if, for every such D and every $c \in \mathbb{N}$, $\text{Adv}_{\text{PRG},D}^{\mathcal{O}}(\lambda) \leq \lambda^{-c}$ for infinitely many λ .

One-Way Functions.

Definition 3 (One-Way Function). A deterministic polynomial-time algorithm $f: \{0,1\}^* \rightarrow \{0,1\}^*$ is a one-way function if, for every non-uniform probabilistic polynomial-time adversary A , the function

$$\epsilon_A(\lambda) := \Pr_{x \leftarrow U_\lambda} [f(A(\lambda, f(x))) = f(x)]$$

is negligible in λ , where the probability also includes the internal randomness of A . We say that one-way functions exist if such an algorithm exists.

Definition 4 (Relativized One-Way Function). Let $\mathcal{O}$ be an oracle distribution. A deterministic polynomial-time oracle algorithm $f^{\mathcal{O}}: \{0,1\}^* \rightarrow \{0,1\}^*$ is a one-way function relative to $\mathcal{O}$ if for every non-uniform probabilistic polynomial-time oracle adversary A , the function

$$\epsilon_A(\lambda) := \Pr_{O \leftarrow \mathcal{O}, x \leftarrow U_\lambda} [f^{\mathcal{O}}(A^{\mathcal{O}}(\lambda, f^{\mathcal{O}}(x))) = f^{\mathcal{O}}(x)]$$

is negligible in λ . We say that one-way functions exist relative to $\mathcal{O}$ if such an oracle algorithm exists.

3 VDF Definition

In this section, we give a VDF definition that is formal enough to allow for proving a rigorous impossibility result. Along the way, we also formalize relevant aspects of the VDF definition that pertain to adversarial speedups and the set of permissible time parameters that allow for VDF instantiations by avoiding trivial attacks.

⁵ See for example Section 2.1 resp. Section 3.1 in [LV08] for plain resp. prefix-free complexity.

Syntax. On a high level, verifiable delay functions (VDFs) are functions that are “slow to compute but efficient to verify.” A VDF is a tuple (**Setup**, **Eval**, **Verify**) of randomized algorithms such that **Setup**, on input a security parameter λ and time parameter T , outputs public parameters $\mathbf{pp}$; **Eval**, on input $\lambda, T, \mathbf{pp}$, and a bit-string x , returns an output y and a short proof π ; and **Verify**($\mathbf{pp}, x, y, \pi$) accepts or rejects a proof π that vouches for the correctness of y .

Semantic properties. A VDF must satisfy completeness, efficiency, soundness, and sequentiality. Informally still, completeness requires that the honest evaluation makes the verifier accept; efficiency requires **Eval** to be computable in “sequential time” $t_{\text{Eval}}(\lambda, T)$ and **Verify** to be computable in “total time” $t_{\text{Verify}}(\lambda, T)$ that is “substantially” smaller than $t_{\text{Eval}}(\lambda, T)$; soundness means that no “efficient” adversary can find a proof for an input x that is deemed valid for an output y' different from the correct output y produced by **Eval**; sequentiality ensures that no “massively parallel” algorithm can compute the function in time “significantly less” than $t_{\text{Eval}}(\lambda, T)$. The precise meaning of these properties depends on the computational model.

Computational models. Timed cryptographic primitives, such as PoSWs [MMV13], TLPs [RSW96; BGJ⁺16], and VDFs [BBB⁺18] are often predicated on the existence of an “atomic” computational unit, often in some ideal oracle model, e.g. a hash function in the random oracle model (ROM) or a group operation in the generic group model (GGM).⁶ This conceptual view is appealing because it allows a clean and simplified (atomic) definition of sequentiality to—informally—require T executions of the atomic computation step.

However, not all computational models exhibit this atomicity property; in particular circuits and (multi-tape) TMs in the standard model enjoy linear speedups via alphabet compression. For example, any function that can be computed in time $T(\lambda) \geq 2\lambda$ can also be computed in time $T(\lambda)/2$ by increasing the alphabet size (for TMs) or allowing larger gates (in circuits). Other models are not atomic and do not allow for speedups. An example of such a model is the (unit-cost) random access machine (RAM) because RAMs have time hierarchy theorems [SZ76; BG97] that preclude universal linear speedups. More precisely, they allow for a linear speedup, but the speedup factor depends on the function itself (whereas linear speedup models allow one to speed up any (at least linear) function by any constant factor).

Definitional fragmentation. Note that the simplified (atomic) definition of sequentiality cannot be fulfilled in a speedup model,⁷ while a weaker (speedup) notion of sequentiality (e.g. $o(T)$) may not be optimal in an atomic model. For example, in the ROM, optimally secure sequential functions are possible [MMV13; CP18]: while the honest function evaluation takes T oracle rounds, every adversary making up to $T - 1$ rounds fails to compute the function’s output with overwhelming probability. This seems to suggest that the notion of sequentiality must be tailored to the given computational model. We believe this state is unsatisfactory and that a VDF definition should be agnostic to the computational model—as long as it has notions of depth/size, or equivalently, sequential/total time.

Our approach to defining VDFs. To be flexible enough to capture different types of models, our definition, which we formally give in Definitions 5 and 6, is relativized over an oracle distribution. To capture standard model VDFs, one simply instantiates our definition with an empty oracle.

For concreteness, we model VDF algorithms as TMs and adversaries as (non-uniform) circuit families. We stress that these concrete models can be replaced by any model that admit a notion of sequential/total time or depth/size. Here, we equate sequential time with depth $\mathfrak{d}$, and total time with size $\mathfrak{s}$.

The VDF syntax as well as completeness, efficiency and soundness are straightforward and essentially follow existing definitions in the literature. However, sequentiality is more subtle. To provide the necessary

⁶ VDFs are impossible in both the ROM [MSW20; GRY25; AAK⁺25] and the GGM [RSS20].

⁷ Because an adversary with a larger alphabet (i.e., better hardware) can evaluate the VDF in time $T/2 \preceq T$.

flexibility to capture various models, our sequentiality “proto-definition” is parameterized over the size (total time) and depth (sequential time) of the adversary, the success probability of the adversary, and two additional parameters: a speedup factor $\mathfrak{o}_\mathfrak{d}$ and a size-bound factor $\mathfrak{o}_\mathfrak{s}$. The purpose of these parameters is to exclude trivial attacks that (depending on the model) would make the definition uninstantiable. Finally, to obtain a non-parameterized definition for a specific computational model, we quantify over suitable parameters of the proto-definition that are consistent with the model and exclude trivial attacks.⁸

Excluding trivial attacks from the definition. Sequentiality should prevent any adversary from computing the VDF output faster than the (honest) evaluation algorithm. As VDFs allow for public parameters $\mathbf{pp}$, such an adversary is modeled in two stages: the first “offline” stage (of size $\mathfrak{s}_0$) only has access to the (honestly) generated $\mathbf{pp}$ alongside the security and time parameters λ, T ; the second “online” stage (of size $\mathfrak{s}_1$ and depth $\mathfrak{d}_1$) has access to the first stage’s state as well as the VDF input x for which it must compute the VDF output. The definition of sequentiality must exclude two unavoidable trivial attacks.

- The offline brute-force verification attack: the offline adversary must have total time strictly less than $2^{\mathfrak{t}_{\text{Verify}}(\lambda, T)} \mathfrak{t}_{\text{Verify}}(\lambda, T)$, as otherwise the (adversarial) offline stage could simply check each preimage–image–proof triple and store the results in a lookup table.
- The honest online evaluation attack: the (adversarial) online stage must use sequential time strictly less than $\mathfrak{t}_{\text{Eval}}(\lambda, T)$, as otherwise the adversary could simply run Eval itself, thus making the definition unsatisfiable.

To exclude these attacks, we lower and upper bound the set of permissible time parameters $P_{\mathfrak{d}_1, \mathfrak{o}_\mathfrak{d}, \mathfrak{o}_\mathfrak{s}}(\lambda) \subseteq \mathbb{N}$ for each given security parameter in Equation (6) in our Definition 5.

Upper bounding T : brute-force attacks. Boneh, Boneau, Bünz, and Fisch [BBB⁺18] require T to be at most a subexponential, i.e., $2^{o(\lambda)}$, in λ . Unfortunately, they do not specify how exactly this upper bound is quantified, i.e., whether it is part of the VDF scheme, or it is somehow quantified in the security definitions (sequentiality and soundness). Their justification for this restriction is that if $T \in 2^{\text{poly}(\lambda)}$ were allowed, then the offline adversary would have time $2^{\lambda^{c'}}$ for any constant c' , and thus it would be able to break any potential cryptographic assumption underlying the VDF construction.

While this restriction is certainly necessary on an intuitive level, its necessity does not formally follow from the trivial attacks mentioned above. As such, this subexponential upper bound on T is not inherent to the definition of VDFs: it could be the case that a VDF exists that supports sequentiality for exponential T simply because an offline adversary that runs in time exponential in λ might be able to break any underlying cryptographic assumption, but not the VDF itself. Naturally, a VDF satisfying sequentiality for exponential T is a stronger primitive than one satisfying sequentiality only for subexponential T , which makes the former hard to construct.

Nevertheless, we follow the intuition of [BBB⁺18] and adopt a subexponential upper bound on T in our VDF definition as $2^{\mathfrak{o}_\mathfrak{s}(\lambda)\lambda}$ where $\mathfrak{o}_\mathfrak{s}$ is a parameter in our definition. (Quantified existentially over vanishing functions $\mathfrak{o}_\mathfrak{s} \in o(1)$.) Along with the instantiability of the definition we find this choice useful because VDFs whose preimage length, image length and proof size are independent of T (and polynomial in λ), i.e., $|x|, |y|, |\pi| \leq \lambda^c$ can only be sequential for subexponential T , for otherwise for $T \geq 2^{\lambda^c}$, an offline adversary could test all preimage–image–proof triples in time $2^{3\lambda^c} \mathfrak{t}_{\text{Verify}}(\lambda, T) \leq T^{c'} \lambda^{c'}$ for some constants $c, c' \geq 0$ and sufficiently large λ .

Lower bounding T : adversarial speedups. While the (subexponential) upper bound on T is required by the offline brute-force verification attack, a lower bound on T is required to exclude the trivial honest online evaluation attack. Intuitively, we need to require $\mathfrak{d}_1(\lambda, T) \leq \mathfrak{t}_{\text{Eval}}(\lambda, T)$, so that the (online) adversary

⁸ This is the same approach as in traditional cryptographic definitions: e.g. an OWF is secure if and only if it is $(t, \epsilon := 1/q)$ -secure for all polynomials t, q , i.e., a t -time adversary breaks security with probability ϵ . Here, OWF “security” is the non-parameterized property, and (t, ϵ) -security is the parameterized “proto-definition”.

cannot simply evaluate the VDF honestly. While for atomic models a natural bound is $\mathfrak{d}_1(\lambda, T) \leq \mathfrak{t}_{\text{Eval}}(\lambda, T) := T$, this bound results in an uninstantiable definition in a speedup model, because an adversary with better hardware could evaluate the VDF in time $T/2 \leq T$. So, our parameterized definition includes a speedup parameter $\mathfrak{o}_{\mathfrak{d}}$ that controls the allowed speedup. That is, our definition requires $\mathfrak{d}_1(\lambda, T) \leq \mathfrak{o}_{\mathfrak{d}}(\lambda) \mathfrak{t}_{\text{Eval}}(\lambda, T)$.

Instantiations. If we now want to reconstruct the atomic definition we simply instantiate our parameterized definition with $\mathfrak{o}_{\mathfrak{d}} := 1$. Together with $\mathfrak{t}_{\text{Eval}}(\lambda, T) := T$, this collapses to the canonical lower bound $\mathfrak{d}_1(\lambda, T) \leq \mathfrak{o}_{\mathfrak{d}}(\lambda) \mathfrak{t}_{\text{Eval}}(\lambda, T) = T$ that we discussed above.

For linear speedup models, we (existentially) quantify our parameterized definition over some vanishing function $\mathfrak{o}_{\mathfrak{d}} \in o(1)$ so that $\mathfrak{o}_{\mathfrak{d}}(\lambda) \mathfrak{t}_{\text{Eval}}(\lambda, T)$ scales sublinearly with respect to $\mathfrak{t}_{\text{Eval}}(\lambda, T)$. This rules out adversaries that honestly evaluate the VDF in time $\mathfrak{t}_{\text{Eval}}(\lambda, T)/a$ for *any* constant $a \in \mathbb{N}$.

For other models, e.g. RAMs we can (universally) quantify over all constant speedup factors $\mathfrak{o}_{\mathfrak{d}}(\lambda) := 1/a$ for each $a \in \mathbb{N}$. This rules out adversaries that honestly evaluate the VDF in time $\mathfrak{t}_{\text{Eval}}(\lambda, T)/a$ for *some* constant $a \in \mathbb{N}$ (depending on $\mathfrak{t}_{\text{Eval}}$).

While these parameter choices might seem very specific to the actual computational model, a conservative choice is to simply use our default instantiation for linear speedup models.

Definition 5 (Relativized Verifiable Delay Function). *A verifiable delay function (VDF) is a tuple of randomized uniform oracle algorithms $\text{VDF} := (\text{Setup}, \text{Eval}, \text{Verify})$ where*

- $\text{Setup}(\lambda, T) \rightarrow \text{pp}$: *On input a security parameter $\lambda \in \mathbb{N}$ and a time parameter $T \in \mathbb{N}$, it outputs public parameters $\text{pp} := (1^\lambda, T, \text{pp}')$ which define an efficiently samplable and recognizable domain X and a range Y .*
- $\text{Eval}(\text{pp}, x) \rightarrow (y, \pi)$: *On input pp and a value x , it outputs y and a (possibly empty) proof π . The output y is independent of the randomness of Eval and is determined only by pp and x .*
- $\text{Verify}(\text{pp}, x, y, \pi) \in \{0, 1\}$: *On input pp , values x, y , and a proof π , it outputs deterministically a bit indicating acceptance (1) or rejection (0).*

We require VDF to satisfy completeness, soundness, sequentiality, and efficiency as defined below.

Completeness. *We say that VDF is $(\mathfrak{s}, \delta)$ -complete relative to an oracle distribution $\mathcal{O}$ iff each non-uniform $\mathfrak{s}(\lambda, T)$ -size-bounded adversary A succeeds with probability at most $\delta(\lambda)$. Formally,*

$$\forall A \in C_{\mathfrak{s}(\lambda, T)}^{\text{NU}} \exists \lambda_0 \forall \lambda \geq \lambda_0 \forall T \in [2^\lambda] : \Pr \left[\begin{array}{l} \mathcal{O} \leftarrow \mathcal{O} \\ \text{pp} \leftarrow \text{Setup}^{\mathcal{O}}(\lambda, T) \\ x := A^{\mathcal{O}}(\text{pp}) \\ (y, \pi) \leftarrow \text{Eval}^{\mathcal{O}}(\text{pp}, x) \end{array} : \text{Verify}^{\mathcal{O}}(\text{pp}, x, y, \pi) = 0 \right] \leq \delta(\lambda) . \quad (3)$$

We say that VDF is complete relative to $\mathcal{O}$ if for each $c, d \in \mathbb{N}$ it is $(T^c \lambda^c, 1/\lambda^d)$ -complete. We say a VDF is perfectly complete relative to $\mathcal{O}$ if it is $(\infty, 0)$ -complete.

Efficiency. *We say that VDF is efficient iff there exists some $c \in \mathbb{N}$ such that for each λ, T, pp'*

- $\text{Setup}(\lambda, T)$ runs in time at most $\log(T)^c \lambda^c$,
- $\text{Eval}(\text{pp}, x)$ runs in time at most $\mathfrak{t}_{\text{Eval}}(\lambda, T) \leq T \lambda^c$,
- $\text{Verify}(\text{pp}, x, y, \pi)$ runs in time at most $\log(T)^c \lambda^c$

where $\text{pp} := (\lambda, T, \text{pp}')$.

Soundness. We say that VDF is $(\mathfrak{s}, \delta)$ -sound relative to an oracle distribution $\mathcal{O}$ iff each non-uniform $\mathfrak{s}(\lambda, T)$ -size-bounded adversary A succeeds with probability at most $\delta(\lambda)$. Formally,

$$\forall A \in C_{\mathfrak{s}(\lambda, T)}^{\text{NU}} \exists \lambda_0 \forall \lambda \geq \lambda_0 \forall T \in [2^\lambda] : \Pr \left[\begin{array}{l} O \leftarrow \mathcal{O} \\ \text{pp} \leftarrow \text{Setup}^O(\lambda, T) \\ (x, y', \pi') := A^O(\text{pp}) \\ (y, \pi) \leftarrow \text{Eval}^O(\text{pp}, x) \end{array} : \begin{array}{l} y' \neq y \\ \text{Verify}^O(\text{pp}, x, y', \pi') = 1 \end{array} \right] \leq \delta(\lambda). \quad (4)$$

We say that VDF is sound relative to $\mathcal{O}$ if for each $c, d \in \mathbb{N}$ it is $(T^c \lambda^c, 1/\lambda^d)$ -sound. We say a VDF is perfectly sound relative to $\mathcal{O}$ if it is $(\infty, 0)$ -sound.

Sequentiality. We say that VDF is $(\mathfrak{s}_0, \mathfrak{s}_1, \mathfrak{d}_1, \delta, \mathfrak{o}_\mathfrak{d}, \mathfrak{o}_\mathfrak{s})$ -sequential relative to an oracle distribution $\mathcal{O}$ iff for each non-uniform $\mathfrak{s}_0(\lambda, T)$ -size-bounded adversary A_0 , each non-uniform $\mathfrak{d}_1(\lambda, T)$ -depth-bounded $\mathfrak{s}_1(\lambda, T)$ -size-bounded adversary A_1 , each permissible T , (A_0, A_1) succeeds with probability at most $\delta(\lambda)$. Formally,

$$\forall A_0 \in C_{\mathfrak{s}_0}^{\text{NU}} \forall A_1 \in C_{\mathfrak{s}_1, \mathfrak{d}_1}^{\text{NU}} : \Pr \left[\begin{array}{l} O \leftarrow \mathcal{O} \\ \text{pp} \leftarrow \text{Setup}^O(\lambda, T) \\ \text{state} := A_0^O(\text{pp}) \\ x \leftarrow X \\ y' := A_1^O(\text{pp}, x, \text{state}) \\ (y, \pi) \leftarrow \text{Eval}^O(\text{pp}, x) \end{array} : y' = y \right] \leq \delta(\lambda) \quad (5)$$

where

$$P_{\mathfrak{d}_1, \mathfrak{o}_\mathfrak{d}, \mathfrak{o}_\mathfrak{s}}(\lambda) := \{ T \in \mathbb{N} \mid \mathfrak{d}_1(\lambda, T) \leq \mathfrak{o}_\mathfrak{d}(\lambda) t_{\text{Eval}}(\lambda, T) \wedge \mathfrak{s}_0(\lambda, T), \mathfrak{s}_1(\lambda, T) \leq 2^{\mathfrak{o}_\mathfrak{s}(\lambda)\lambda} \} \quad (6)$$

is the set of permissible time parameters T that exclude inherent attacks. Uniform sequentiality is defined analogously where $A_0 \in C_{\mathfrak{s}_0}^{\text{U}}$ and $A_1 \in C_{\mathfrak{s}_1, \mathfrak{d}_1}^{\text{U}}$.

Non-parameterized sequentiality. We say that VDF is sequential relative to $\mathcal{O}$ if there exists some $\epsilon \in \mathbb{R}$ such that for each $c, d \in \mathbb{N}$ there exist $\mathfrak{o}_\mathfrak{d}, \mathfrak{o}_\mathfrak{s} \in o(1)$ s.t. it is $(T^c \lambda^c, T^\epsilon \lambda^c, 1/\lambda^d, \mathfrak{o}_\mathfrak{d}, \mathfrak{o}_\mathfrak{s})$ -sequential.

Remark 3. Note that the oracle distribution in Definition 5 is a single distribution and the sampled oracle O does *not* depend on the security parameter.

Remark 4. For the non-parameterized sequentiality notion, and depending on the computational model, there are a few upper bounds on the depth of the online adversary with respect to which sequentiality is instantiable. The bound $T - 1$ is instantiable in atomic models (e.g., [Pie19; Wes19]), and in non-atomic models (e.g. circuits), and excluding the security parameter for clarity, the following bounds are instantiable: $(1 - o(1))T$ (e.g., [DGM⁺20; LV20]), $(1 - \epsilon)T$ (e.g., [BBB⁺18; EFK⁺20b; EFK⁺20a]), and T^ϵ (e.g., [AAF⁺25]) for constant $\epsilon \geq 0$. Note that $(1 - \epsilon)T$ is only achievable in models without linear speedup. Thus, we consider the conservative bound T^ϵ that has been extensively used in the time-lock puzzles and timed commitments literature (e.g., [BGJ⁺16; MT19; TCL⁺21; CJS⁺21; CJ23; AA24]) in addition to the recent work on VDFs [AAF⁺25]. More precisely, we allow for adversaries of depth $T^\epsilon \lambda^c$ for any $c \in \mathbb{N}$. However, stronger quantifications of up to $T/\log(T)^{\omega(1)} \cdot \lambda^c$ also result in instantiable definitions. As our techniques have no depth overhead (see the proof of Lemma 3), our oracle yields a VDF (Lemma 3) for any instantiable definition.

Definition 6 (Verifiable Delay Function). A standard model VDF is a VDF that is complete, sound and sequential relative to the empty set $\emptyset$, i.e., relative to the singleton oracle distribution $\mathcal{O} = \delta_\emptyset$. Importantly, in this case the depth and size bounds of the adversaries apply to their overall depth and size, not just to their oracle gates.

4 Technical Overview

Our task is to construct an oracle distribution relative to which a VDF exists but no OWF does. To this end, we define three oracles: a random length-preserving function F , a Boolean verification oracle V , and a Boolean complexity oracle $L^{F,V}$ that depends on the first two. At a high level, the first two oracles give us a VDF, while the complexity oracle rules out PRGs and thereby also OWFs.

We start off with a natural and simple definition of the three oracles and provide an intuition why such oracles are promising candidates for our oracle separation. Thereafter, we point out subtle, but crucial technical problems with this simple approach. We overcome them by iteratively modifying our oracle definitions. By the end of this section, we derive the final version of our oracles, which we then use in our separation results formally presented in Section 5.

4.1 The Initial Idea

Let us start by ensuring that a VDF exists. To this end, let λ be the VDF security parameter and T the VDF time parameter. Consider two oracles: a random function

$$F : \{0, 1\}^* \times \mathbb{N} \rightarrow \{0, 1\}^* : (x, T) \mapsto y \quad (7)$$

which is length-preserving for each T , (i.e., $\forall \lambda, T \in \mathbb{N} : F(\{0, 1\}^\lambda, T) \subseteq \{0, 1\}^\lambda$), and a (Boolean) verification oracle

$$V := \{(x, y, T) \mid y = F^{(T)}(x, T)\}. \quad (8)$$

Equipped with the two oracles F, V , our VDF construction (**Setup**, **Eval**, **Verify**) is straightforward. On input (λ, T) , **Setup** outputs $\text{pp} := (\lambda, T)$. On input (pp, x) , **Eval** iterates the function $F(\cdot, T)$ T -times to produce an image y (the proof π is empty). On input (pp, x, y, π) , **Verify** outputs whatever V outputs on (x, y, T) . The constructed VDF is efficient, and completeness and soundness hold perfectly. It is also easy to see that sequentiality holds against any adversary that makes at most $T - 1$ sequential queries.

Let us turn our attention to OWFs now, and analyze what we know about OWFs relative to (F, V) . A random oracle F is a strong OWF, and adding a verification oracle V does not break the OWF. This is because the verification oracle V is sparse and without computing T sequential queries to $F(\cdot, T)$ any V query can be answered with 0. (This simulation is also used by Wee [Wee06] to replace the verification oracle for a sparse NP-relation with the zero function.) Since it is not possible to rule out OWFs relative to only (F, V) because they exist unconditionally, we need to add a third oracle whose primary goal is to break any OWF construction, yet it does not help in breaking the VDF.

Since the random oracle F is an OWF, a natural idea is to additionally provide the inverse function F^{-1} as an oracle. Then F is obviously not an OWF anymore. However, Winternitz [Win84] proves that the Davies–Meyer construction $F(x, T) \oplus x$ is an OWF even relative to F and F^{-1} . Hence, F^{-1} does not help us to break *every* OWF and we need to find a different strategy.

Switching to PRGs. Our first important observation is that we can prove the VDF-OWF separation through PRGs. The fully black-box construction of Haitner, Reingold, and Vadhan [HRV10] transforms any OWF f into a PRG PRG^f . Both the construction and its security reduction relativize, so the same implication holds relative to an arbitrary oracle distribution $\mathcal{O}$: if $f^{\mathcal{O}}$ is one-way relative to $\mathcal{O}$, then $\text{PRG}^{f,\mathcal{O}}$ is pseudorandom relative to $\mathcal{O}$. More precisely, an oracle distinguisher $D^{\mathcal{O}}$ that breaks $\text{PRG}^{f,\mathcal{O}}$ can be turned into an oracle algorithm $I^{\mathcal{O}}$ that inverts $f^{\mathcal{O}}$. Thus, an oracle distribution relative to which no PRG exists cannot support an OWF. We formalize this implication in Lemma 4.

Hence, our goal from now on is to find a third oracle that rules out any PRG construction, yet does not break the VDF construction defined above.

Kolmogorov Complexity Oracle. This is where we employ ideas from Kolmogorov complexity theory. Intuitively, the Kolmogorov complexity of a bit string x is the length of the shortest program producing x (the lexicographically first, hence unique, such program is called the *witness program* for x). More precisely, the (traditional) Kolmogorov complexity of a bit string relative to oracles F, V is defined as:

$$K^{F,V}(x) := \min\{|II| \mid II \in \{0,1\}^* \wedge II^{F,V}(\varepsilon) = x\}. \quad (9)$$

Using this notion of complexity, we define the following complexity oracle consisting of all *low-complexity* bit strings (also called the Minimum Kolmogorov Problem (MKP) in the literature):

$$L^{F,V} := \{x \mid K^{F,V}(x) \leq |x| - 2\}. \quad (10)$$

The constant 2 in the definition is arbitrary; our proofs work analogously for any positive integer.

Remark 5. Note that the set $L^{F,V}$ is undecidable. This is however not an issue for our separation result. Recall that we need to prove that there *exists* a tuple of oracles relative to which there are VDFs but no PRGs. We do not need an efficient way to construct them.

Now, why should such a complexity oracle help to break any PRG? Without loss of generality, let us consider a length-doubling PRG. The high-level idea is that any PRG image (output of the PRG) is of low complexity. Namely, the witness program of a PRG image could be described by a short seed s (much shorter than the PRG output length) and the constant-size description of the PRG code PRG. Let us denote such a program by $\Pi_{s,\text{PRG}}$. On the other hand, one can prove that a random bit string of some length ℓ has complexity $\approx \ell$ with high probability (see Lemma 5). Therefore, a PRG distinguisher can simply query the complexity oracle $L^{F,V}$ on its own input z (a PRG image or a random bit string). If z has high complexity, it must be a random bit string; otherwise it is likely a PRG image.

4.2 Breaking PRGs (Resolving Circularity)

We now explain why this rather simple idea does not work and how we can fix it. Consider an arbitrary PRG construction that makes calls to the complexity oracle $L^{F,V}$. Let $\Pi_{s,\text{PRG}}$ be the program that stores the seed s and the PRG code PRG, and outputs $z := \text{PRG}^{F,V,L^{F,V}}(s)$. One might hope that $\Pi_{s,\text{PRG}}$ is a short witness program for a PRG output z ; thus z has low complexity. However, for $\Pi_{s,\text{PRG}}$ to actually be a valid witness program, the complexity notion $K^{F,V}$ needs to provide oracle access to *all oracles* needed to correctly evaluate the PRG code. This is where the above intuition falls short. Recall that we want to rule out any PRG construction that exists relative to F, V and $L^{F,V}$, i.e., we have $\Pi_{s,\text{PRG}}^{F,V,L^{F,V}}$. But the witness programs inside the definition of the Kolmogorov complexity are only relativized with respect to the oracles F and V . Hence, a PRG construction that calls the complexity oracle $L^{F,V}$ does not constitute a valid witness program for z . We are facing a circularity issue!

For the same PRG breaking strategy, we need to redefine the complexity notion (and hence the complexity oracle) so that $\Pi_{s,\text{PRG}}$ with access to all three oracles constitutes a valid witness program. This in particular means we need to find a way to answer calls of $\Pi_{s,\text{PRG}}$ to the complexity oracle.

The first (naive) attempt. We start off by observing that one can decide whether a bit string x has low or high Kolmogorov complexity (i.e., $x \in L^{F,V}$) through the following two-step process: (1) *Pre-processing:* Consider all candidate witness programs II for the bit string x , i.e., all programs with $|II| \leq |x| - 2$. Run all such (exponentially many) programs relative to F, V (on the empty input ε) in parallel in order to obtain an output bit string y_{II} for each candidate witness program II ; (2) *Membership testing:* Check and output 1 if and only if x equals one of the output bit strings y_{II} .

To build on this observation, let $\Phi_\ell^{F,V}$ denote a *pre-processing program* that takes candidate witness programs of length at most ℓ and maps them to their outputs relative to F, V , or outputs $\perp$ if they do

not halt, i.e.,

$$\Phi_\ell^{F,V} : \{0,1\}^{\leq \ell} \rightarrow \{0,1\}^* \cup \{\perp\} : \Pi \mapsto \begin{cases} y_\Pi := \Pi^{F,V}(\varepsilon), & \text{if } \Pi^{F,V}(\varepsilon) \text{ halts;} \\ \perp & \text{otherwise.} \end{cases}$$

Let us denote the image of $\Phi_\ell^{F,V}$ (restricted to bit strings) by

$$\text{Im}^*(\Phi_\ell^{F,V}) := \Phi_\ell^{F,V}(\{0,1\}^{\leq \ell}) \setminus \{\perp\}.$$

The observation presented above suggests that one could answer queries to the complexity oracle that are of length ℓ , by testing membership in $\text{Im}^*(\Phi_\ell^{F,V})$.

Extending the witness. The central problem in the naive attempt is that it considers a *single* pre-processing program. To gain more flexibility, our idea is to make the pre-processing program Φ part of the witness program, and hence allow the preprocessing program Φ to depend on x arbitrarily. Concretely, a witness program (for a given bit string x) is now a *pair* (Π, Φ) consisting of a *base program* Π and a *pre-processing program* Φ , and the complexity notion is the following:

$$\tilde{K}_{\text{pre}}^{F,V}(x) := \min \left\{ |\Pi| + |\Phi| \mid \Pi, \Phi \in \{0,1\}^* \wedge \Pi^{F,V,\text{Im}^*(\Phi^{F,V})}(\varepsilon) = x \right\},$$

where $\text{Im}^*(\Phi^{F,V}) := \Phi^{F,V}(\{0,1\}^*) \setminus \{\perp\}$. Note that this notion of complexity is well-defined (i.e., not circular). Let $\tilde{L}_{\text{pre}}^{F,V}$ denote the respective complexity oracle.⁹ Now, to prove that a PRG output z has low complexity, we need to find a short base program Π that produces z (that will be the PRG description and a short hardcoded seed), and a suitable pre-processing program Φ .

Universal pre-processing. The key insight is that the $\tilde{L}_{\text{pre}}^{F,V}$ oracle can now itself be computed by the two-step process of pre-processing and membership testing outlined at the beginning of this section.

Let $\hat{\Phi}$ be the *universal* pre-processing program for $\tilde{L}_{\text{pre}}^{F,V}$ that takes as input a triple (Π, Φ, x) consisting of a *base program* Π , a *pre-processing program* Φ and a bit string x . The function $\hat{\Phi}$ either outputs the bit string x , or $\perp$. Intuitively, $\hat{\Phi}$ outputs x if the base program and the pre-processing program are sufficiently short (concretely $|\Pi| + |\Phi| \leq |x| - 2$) and x is the output of $\Pi(\varepsilon)$ with oracle access to F, V and $\text{Im}^*(\Phi^{F,V})$, formally:

$$\hat{\Phi}^{F,V}(\Pi, \Phi, x) := \begin{cases} x & \text{if } x = \Pi^{F,V,\text{Im}^*(\Phi^{F,V})}(\varepsilon) \wedge |\Pi| + |\Phi| \leq |x| - 2 \\ \perp & \text{otherwise} \end{cases}. \quad (11)$$

Let $\text{Im}^*(\hat{\Phi}^{F,V}) := \hat{\Phi}^{F,V}(\{0,1\}^*) \setminus \perp$. To see that $\text{Im}^*(\hat{\Phi}^{F,V}) = \tilde{L}_{\text{pre}}^{F,V}$ consider the following: Indeed, if x is a bit string in the image of $\hat{\Phi}^{F,V}$, then there must exist a short witness pair (Π, Φ) such that $x = \Pi^{F,V,\text{Im}^*(\Phi^{F,V})}(\varepsilon)$ which is exactly what we needed in order to show that $x \in \tilde{L}_{\text{pre}}^{F,V}$. In the other direction, if $x \in \tilde{L}_{\text{pre}}^{F,V}$, there exists a witness pair (Π, Φ) such that their length is small and $x = \Pi^{F,V,\text{Im}^*(\Phi^{F,V})}(\varepsilon)$. Hence, it holds that $\hat{\Phi}^{F,V}(\Pi, \Phi, x) = x$ which implies $x \in \text{Im}^*(\hat{\Phi}^{F,V})$.

No PRG exists. Through our iterative process, we found a third oracle $\tilde{L}_{\text{pre}}^{F,V}$ which breaks any PRG. Let us summarize the argumentation here. For any PRG output $z := \text{PRG}^{F,V,\tilde{L}_{\text{pre}}^{F,V}}(s)$ we can now find a short witness pair $(\Pi_{\text{PRG},s}, \hat{\Phi})$, where $\Pi_{\text{PRG},s}$ has the PRG code PRG and the PRG seed s hardcoded and $\hat{\Phi}$ is the universal preprocessing function. Because $\tilde{L}_{\text{pre}}^{F,V} = \text{Im}^*(\hat{\Phi}^{F,V})$, the witness pair is a valid witness according to the complexity notion $\tilde{K}_{\text{pre}}^{F,V}$ and hence z has low complexity. A PRG distinguisher can now

⁹ The tilde above K emphasizes that this notion of complexity is neither the original definition, nor our final modification of the complexity notion (the same holds for the complexity oracle).

simply query the complexity oracle $\tilde{L}_{\text{pre}}^{F,V}$ on its own input z ; if it is a PRG image, it must have low complexity; if it is a random string it has high complexity with high probability.

4.3 Proving Sequentiality

After finding a third oracle, the complexity oracle $\tilde{L}_{\text{pre}}^{F,V}$, that allows us to break any PRG, we need to reconsider our VDF construction and check that all VDF properties still hold. Let us focus on sequentiality of our construction relative to $\mathcal{O} = (F, V, \tilde{L}_{\text{pre}}^{F,V})$ as this is the only tricky property to prove.

At a high level, the proof strategy is the following: assume there exists a VDF sequentiality adversary A which has access to all three oracles. Transform this adversary A into an adversary B which has almost the same output distribution as A while making only few queries to F (and only on inputs in $\{0, 1\}^\lambda \times \{T\}$). Such an adversary B then successfully breaks the sequentiality of a random function which, as we prove in Lemma 10, is a contradiction. The adversary B simulates the VDF sequentiality game for A , which in particular means that B needs to answer all oracle queries made by A .

Let us first discuss the F oracle queries made by A . Note that the winning condition of the VDF sequentiality game only depends on the $\{0, 1\}^\lambda \times \{T\}$ “slice” of the oracle F . Therefore, B can resample all values of F outside of the $\{0, 1\}^\lambda \times \{T\}$ slice without changing the output distribution of A . To this end, we define a new oracle F' which on inputs in $\{0, 1\}^\lambda \times \{T\}$ actually queries F , but lazily samples all other inputs. We can also define a new verification oracle V' which instead of F calls F' . The oracles F' and V' are then used by B to simulate F and V oracle queries of A in the sequentiality game.¹⁰

On any query x made by A to the complexity oracle, the adversary B needs to decide whether the bit string x has low or high complexity. The canonical way to do this would be for B to exhaustively run all possible base programs and then check if one of them outputs x . However, according to our complexity notion $\tilde{K}_{\text{pre}}^{F',V'}$, base programs are run relative to the random oracle F' . In particular, this means that B would have to answer all F' oracle queries made by all possible base programs, which would result in B having to make prohibitively many queries to F in the $\{0, 1\}^\lambda \times \{T\}$ slice. Hence, we would not yield the desired contradiction.

Simulating short queries. Since we do not know how to simulate A ’s queries to $\tilde{L}_{\text{pre}}^{F',V'}$ with only few sequential queries to F , we weaken the complexity notion by limiting the base programs’ access to F' and V' . More precisely, a base program for a bit string x is allowed to make queries to F' only on *shorter* bit strings than x (say on bit strings of length up to $|x|/6$):

$$\tilde{K}_{\text{restrict}}^{F',V'}(x) := \min \left\{ |\Pi| + |\Phi| \mid \Pi, \Phi \in \{0, 1\}^* \wedge \Pi^{[F']_k, [V']_k}, \text{Im}^*(\Phi^{[F']_k, [V']_k})(\varepsilon) = x \right\},$$

where $k := |x|/6$ and $[\cdot]_k$ denotes the restriction to bit strings of length $\leq k$. If a restricted oracle is queried on a longer bit string, it outputs $\perp$. The complexity oracle $\tilde{L}_{\text{restrict}}^{F,V}$ is then defined with respect to this restricted complexity notion.

The weakening of the complexity notion allows B to simulate “short” queries of A to the complexity oracle. Namely, if A makes a query of length up to 5λ to the complexity oracle, candidate base programs can only make queries to F' of length up to $5\lambda/6$. In other words, the queries that base programs can make to F' stay outside the $\{0, 1\}^\lambda \times \{T\}$ slice and hence, B can use the lazy-sampling strategy without making *any* query to F because queries outside of the relevant slice are answered through lazy-sampling.

Simulating long queries. We cannot use this argument for long queries of A to the complexity oracle. If A makes a query x which is of length greater than 5λ , say 7λ , a base program for x can make queries to F' up to length $7\lambda/6$. In particular, this means that the queries might be in the relevant slice $\{0, 1\}^\lambda \times \{T\}$ resulting in B having to make undesirable F oracle calls.

¹⁰ This is not entirely true for V' as B answers those queries differently in the online phase. We refer the reader to Figure 4 in Section 5.4 for an accurate description.

The idea to overcome this issue is to make all long queries a sequentiality adversary A could possibly make of *low complexity by definition*. That is, artificially add the set of all possible long queries, let us call it J (see Definition 11), to the complexity oracle. With such a modification, B can answer all long queries of A by always outputting 1 without any query to F .

In order to maintain the invariant that the image of the universal pre-processing program is equal to the complexity oracle, we also need to adjust the definition of the complexity notion itself (by giving the preprocessing program access to J) and accordingly also the universal preprocessing program. We refer to Section 5 for the formal definitions of the complexity notion, complexity oracle and the universal preprocessing program. For the purpose of this technical overview, let us denote the final version of the complexity oracle $A^{F,V,J}$.

Completing the reduction. Let us summarize the simulation strategy of the adversary B . Queries of A to the oracle F (resp. V) are answered according to F' (resp. V'). Upon receiving a long query to the complexity oracle $A^{F,V,J}$, B simply answers 1. Recall that long queries are now of low complexity by definition. Since the definition of the complexity oracle changed, we need to reconsider the simulation strategy for short queries.

As discussed above, the idea is that B can use the canonical simulation strategy (i.e., execute all candidate base programs) without making F queries. However, according to the new complexity notion, base programs have access to an oracle which depends on the set of long queries J . Hence, for this simulation strategy to work, B needs a way to compute the set J without making too many queries to F . We present such an algorithm denoted as Gen^F in Section 5.1. At a high level, this algorithm diagonalizes against all long queries all sequentiality adversaries could make to the third oracle and outputs a set of those long queries. Equipped with such an algorithm, the adversary B can execute Gen^F in its offline phase, and compute the complexity oracle restricted to short queries. Upon receiving a short query to the complexity oracle (either in the off-line or on-line phase), B simply checks whether the bit string is in the precomputed set or not.

4.4 PRGs are Still Broken

We need to argue that no PRG can exist relative to our final oracles $(F, V, A^{F,V,J})$, where J is the set of long queries.

As discussed at the beginning of the technical overview, a naive approach to distinguish PRG images from random bit strings would be to simply query the complexity oracle $A^{F,V,J}$ on the input bit string $z \in \{0,1\}^{2\lambda}$ (a PRG image or a random bit string). That is, a PRG distinguisher $\mathcal{D}$ would simply be defined as $\mathcal{D}(z) := A^{F,V,J}(z)$.

This naive distinguisher might not work relative to our weakened complexity oracle due to the oracle restrictions in our complexity notion. To see why, let us consider a PRG construction $\text{PRG}^{F,V,A^{F,V,J}}$ that on input a seed $s \in \{0,1\}^\lambda$ produces an output $z \in \{0,1\}^{2\lambda}$ in time $t_{\text{PRG}}(\lambda) \leq \lambda^2$. During the computation, the algorithm can make queries to its oracles F , V and $A^{F,V,J}$ of length up to $t_{\text{PRG}}(\lambda) \leq \lambda^2$. However, according to our restricted complexity notion, base programs for the bit string $z \in \{0,1\}^{2\lambda}$ can make oracle queries only up to length $|z|/6$. Hence, we have

$$|z|/6 = 2\lambda/6 = \lambda/3 < \lambda^2.$$

In other words, we cannot argue that the code of the PRG together with the seed constitutes a valid base program for z .

The remedy for this problem is to consider a *randomized* distinguisher that appends a sufficiently long random bit string r to its input z and queries the result $y := z||r$ to the complexity oracle $A^{F,V,J}$. Now, base programs for y can make much longer queries to their oracles (concretely $|y|/6 = (|z| + |r|)/6$), but the queries made during the PRG execution are still the same.

Finally, we need to argue that adding the set J to the complexity oracle does not result in too many “false positives” for the PRG distinguisher (i.e., not too many random strings end up having low complexity). In Lemma 7, we prove that the set J is sparse; hence, there are not too many bit strings added to the complexity oracle and hence the distinguishing strategy presented above still works.

5 The Formal Result

In this section, we formalize the VDF–OWF separation outlined in Section 4. Our main theorem, Theorem 2, constructs an oracle distribution $\mathcal{O}_{\text{sep}}$ relative to which there exists a relativized, perfectly complete, perfectly sound, and sequential VDF, while no relativized OWF exists.

We first define $\mathcal{O}_{\text{sep}}$ in Section 5.1 and then state the main theorem in Section 5.2. Its proof uses three lemmas. The first shows that no infinitely often secure PRG exists relative to $\mathcal{O}_{\text{sep}}$, the second constructs a VDF relative to $\mathcal{O}_{\text{sep}}$, and the third shows that the fully black-box construction of PRGs from OWFs continues to hold relative to arbitrary oracle distributions. Together, these lemmas prove the claimed VDF–OWF separation.

5.1 The Oracle Distribution

In this section, we formally define our oracle distribution $\mathcal{O}_{\text{sep}}$. The oracles $\mathbf{F}$ and $\mathbf{V}$ have already been defined earlier in this paper. We restate them here for completeness and focus on defining the complexity oracle.

Definition 7 (Oracle $\mathbf{F}$). *Oracle $\mathbf{F}: \{0,1\}^* \times \mathbb{N} \rightarrow \{0,1\}^*$ is a function sampled according to the distribution $\mathcal{F}$ (the uniform distribution over a set of indexed length-preserving functions over finite strings, cf. Definition 1).*

Definition 8 (Verification Oracle $\mathbf{V}$). *The verification oracle $\mathbf{V}$ defined with respect to $\mathbf{F}$ is a Boolean oracle defined by the set*

$$\mathbf{V} := \{(x, y, T) \mid y = \mathbf{F}^{(T)}(x, T)\}.$$

Definition 9 (Complexity Notion). *For any oracle $\mathbf{F}$, corresponding oracle $\mathbf{V}$, and any set $J \subseteq \{0,1\}^*$, we define the complexity notion*

$$\mathbf{K}_{\text{long}}^{\mathbf{F}, \mathbf{V}, J}(x) := \min \left\{ |\Pi| + |\Phi| \mid \begin{array}{l} \Pi, \Phi \in \{0,1\}^* \\ \Pi^{[\mathbf{F}]_k, [\mathbf{V}]_k, \mathbf{M}}(\varepsilon) = x \\ \mathbf{M} := \Phi^{[\mathbf{F}]_k, [\mathbf{V}]_k, [J]_k}((\{0,1\}^*)^3) \setminus \{\perp\} \end{array} \right\}$$

where $k := \lfloor |x|/6 \rfloor$ and $[\cdot]_k$ denotes the restriction to bit strings of length $\leq k$.

Definition 10 (Complexity Oracle). *We define the set $L_{\text{long}}^{\mathbf{F}, \mathbf{V}, J}$ of all bit strings with low $\mathbf{K}_{\text{long}}^{\mathbf{F}, \mathbf{V}, J}$ -complexity as follows*

$$L_{\text{long}}^{\mathbf{F}, \mathbf{V}, J} := \{x \mid \mathbf{K}_{\text{long}}^{\mathbf{F}, \mathbf{V}, J}(x) \leq |x| - 2\}.$$

The Boolean complexity oracle, which we denote $A^{\mathbf{F}, \mathbf{V}, J}$, is $L_{\text{long}}^{\mathbf{F}, \mathbf{V}, J}$ augmented by J , i.e.,

$$A^{\mathbf{F}, \mathbf{V}, J} := L_{\text{long}}^{\mathbf{F}, \mathbf{V}, J} \cup J.$$

In order to fix an oracle distribution for our separation result, we still need to specify a set J . As discussed in the technical overview (cf. Section 4), we need J to be the set of “long canonical” queries a sequentiality adversary could make to the complexity oracle.

Collecting long queries. Figure 2 depicts the procedure $\text{Gen}^F(\lambda)$ that collects all long queries of all possible sequentiality adversaries up to security parameter λ . The basic idea is borrowed from sophisticated diagonalization techniques on resource hierarchies in complexity theory; see [FS07] for a survey.

The algorithm Gen^F is designed to facilitate the sequentiality proof of our construction. So, let us first describe its role in proving sequentiality to understand the intuition behind the design of Gen^F . Suppose we knew a particular (constant size) adversary that breaks the sequentiality of our VDF construction. Then we could emulate this adversary on a given security parameter λ (and time parameter T and preimage x) and collect all of its long queries in a set J . Then, we add those queries artificially to our complexity oracle (by the term $\cup J$ in Definition 10). This makes the complexity oracle essentially useless to this adversary, because each (long) query response would be 1. Consequently, we could easily transform such an adversary into an adversary on our construction that only makes few queries to the random function F —which is a contradiction to Lemma 10.

The crucial point is that we need not know the description of the adversary because we can simulate all possible adversaries. For input $\lambda \in \mathbb{N}$ the algorithm $\text{Gen}^F(\lambda)$ simulates all possible adversaries up to description length $\log_2(\lambda)$ (on all inputs $\lambda, T \in [2^\lambda], x \in \{0, 1\}^\lambda$) and collects their long queries. A minor subtlety is that we assume the adversary has a constant-size description; this is true for uniform adversaries but not necessarily non-uniform adversaries. Thus, in Lemma 14 we show that a non-uniform adversary implies a uniform adversary, and hence it suffices to simulate uniform adversaries in Gen^F .

For our formal proof we require that the algorithm Gen^F on certain inputs $\lambda \in \mathbb{N}$ can be computed by a uniform algorithm with only few queries to the random function F .

Lemma 1 (Query-Complexity of Gen^F). There exists a uniform (computationally unbounded) oracle algorithm E that for each oracle F , on input any $\lambda \in \{8^\kappa \mid \kappa \in \mathbb{N}\}$ outputs $G_{\kappa-1} := \text{Gen}^F(\lambda/8)$ making at most $2^{5/8 \cdot \lambda}$ F -queries.

Proof. We count the calls to F during the computation of $G_\kappa := \text{Gen}^F(\lambda)$:

- There are at most $\lfloor \log_2 \lambda \rfloor$ iterations of the outer loop (Line 2).
- In Line 2 of Sim_0 each of the at most λ (valid) programs Π on each of the at most 2^λ time parameters T makes at most $\sum_\ell |Q_\ell| \leq \hat{s}_0(\lambda) := 2^\lambda$ queries to all oracles.
- In Line 2 of Sim_1 each of the at most λ (valid) programs Π on each of the at most 2^λ time parameters T and each of the at most 2^λ preimages x makes at most $\sum_\ell |Q_\ell| \leq \hat{s}_1(\lambda) := 2^\lambda$ queries to all oracles.
- For each query $(2, q = (x, y, T))$ of any candidate program Π to the verification oracle V , the algorithm E queries F on (x^i, T) for each $i \in [T]$ where $x^i := (F(\cdot, T))^{(i)}(x)$. The query response is 1 iff $x^T = y$.

Overall, for any $\lambda \in \mathbb{N}$ at most

$$\underbrace{\log_8 \lambda}_{\kappa} \cdot \left(\underbrace{\lambda}_{\Pi} \cdot \underbrace{2^\lambda}_{T} \cdot \underbrace{2^\lambda}_{\# \text{ of queries}} \cdot \underbrace{2^\lambda}_{V \text{ query}} + \underbrace{\lambda}_{\Pi} \cdot \underbrace{2^\lambda}_{T} \cdot \underbrace{2^\lambda}_x \cdot \underbrace{2^\lambda}_{\# \text{ of queries}} \cdot \underbrace{2^\lambda}_{V \text{ query}} \right) \leq 2^{5\lambda} = 2^{5 \cdot 8^\kappa} \quad (12)$$

queries are sufficient to compute G_κ . Let E emulate the computation of $\text{Gen}^F(\lambda/8)$ as described in Figure 2. Consequently, to compute $G_{\kappa-1}$ at most $2^{5 \cdot 8^{\kappa-1}} = 2^{5/8 \cdot \lambda}$ queries are sufficient. $\square$

The oracle distribution. We are now ready to fix our oracle distribution used for the separation result.

Definition 11 (Separation Oracle). We fix an oracle distribution

$$\mathcal{O}_{\text{sep}} := \left\{ \left(F, V, A^{F, V, G^F} \right) \mid F \leftarrow \mathcal{F} \right\} \quad (13)$$

where F , V and $A^{F, V, G^F} := L_{\text{long}}^{F, V, G^F} \cup G^F$ are defined in Definitions 7, 8 and 10 respectively. Here, $G^F := \bigcup_{\lambda \in \mathbb{N}} \text{Gen}^F(\lambda)$ (cf. Figure 2) is the set of all “long canonical” queries (previously called J) to the complexity oracle A^{F, V, G^F} .

$\text{Gen}^F(\lambda_{\max})$	
<pre> 1 : $G_0 := \emptyset; \kappa_{\max} := \lfloor \log_8 \lambda_{\max} \rfloor$ 2 : L1 : for $\kappa \in [\kappa_{\max}]$ 3 : // simulate all first-stage adversaries 4 : $G_{\text{first}} := G_{\kappa-1} \cup \text{Sim}_0^F(\kappa, G_{\kappa-1})$ 5 : // simulate all second-stage adversaries 6 : $G_{\kappa} := G_{\text{first}} \cup \text{Sim}_1^F(\kappa, G_{\text{first}})$ 7 : // G_{κ} now contains all long queries of first-stage and second-stage adversaries up to λ 8 : return $G_{\kappa_{\max}}$ </pre>	
$\text{Sim}_0^F(\kappa, G_{\text{prev}})$	$\text{Sim}_1^F(\kappa, G_{\text{prev}})$
<pre> 1 : $G_{\text{new}} := \emptyset; \lambda := 8^{\kappa}$ 2 : L0.1 : for $\Pi \in \{0, 1\}^{\lfloor \log_2 \lambda \rfloor}, T \in [2^{\lambda}]$ 3 : $\tilde{Q} := G_{\text{prev}}$ 4 : $\tilde{A} := \emptyset$ // current third oracle state 5 : L0.2 : for $i \in [\hat{\mathbf{d}}_0(\lambda, T)]$ 6 : // Q_i are the queries of the i-th round 7 : $\Pi^{F, V, \tilde{A}}(\lambda, T) \xrightarrow{i} Q_i$ 8 : // if $\Pi(\lambda, T)$ makes too many queries, ignore it 9 : if $\sum_{\iota=1}^i Q_{\iota} \geq \hat{\mathbf{s}}_0(\lambda, T)$ then continue L0.1 10 : for $(j, q) \in Q_i$ 11 : if $j = 3 \wedge q \geq 5\lambda$ then $\tilde{Q} := \tilde{Q} \cup \{q\}$ 12 : if $j = 3 \wedge q \in \tilde{Q} \cup L_{\text{long}}^{F, V, G_{\text{prev}}}$ then 13 : $\tilde{A} := \tilde{A} \cup \{q\}$ 14 : endfor 15 : L0.2 : endfor 16 : $G_{\text{new}} := G_{\text{new}} \cup \tilde{Q}$ 17 : L0.1 : endfor 18 : return G_{new} </pre>	<pre> 1 : $G_{\text{new}} := \emptyset; \lambda := 8^{\kappa}$ 2 : L1.1 : for $\Pi \in \{0, 1\}^{\lfloor \log_2 \lambda \rfloor}, T \in [2^{\lambda}], x \in \{0, 1\}^{\lambda}$ 3 : $\tilde{Q} := G_{\text{prev}}$ 4 : $\tilde{A} := \emptyset$ // current third oracle state 5 : L1.2 : for $i \in [\hat{\mathbf{d}}_1(\lambda, T)]$ 6 : // Q_i are the queries of the i-th round 7 : $\Pi^{F, V, \tilde{A}}(\lambda, T, x) \xrightarrow{i} Q_i$ 8 : // if $\Pi(\lambda, T, x)$ makes too many queries, ignore it 9 : if $\sum_{\iota=1}^i Q_{\iota} \geq \hat{\mathbf{s}}_1(\lambda, T)$ then continue L1.1 10 : for $(j, q) \in Q_i$ 11 : if $j = 3 \wedge q \geq 5\lambda$ then $\tilde{Q} := \tilde{Q} \cup \{q\}$ 12 : if $j = 3 \wedge q \in \tilde{Q} \cup L_{\text{long}}^{F, V, G_{\text{prev}}}$ then 13 : $\tilde{A} := \tilde{A} \cup \{q\}$ 14 : endfor 15 : L1.2 : endfor 16 : $G_{\text{new}} := G_{\text{new}} \cup \tilde{Q}$ 17 : L1.1 : endfor 18 : return G_{new} </pre>

Fig. 2: Algorithms to enumerate all long queries made by all potential sequentiality adversaries. Here, $\hat{\mathbf{d}}_0(\lambda, T) := 2^{\lambda}$ and $\hat{\mathbf{s}}_0(\lambda, T) := 2^{\lambda}$ resp. $\hat{\mathbf{d}}_1(\lambda, T) := 2^{\lambda}$ and $\hat{\mathbf{s}}_1(\lambda, T) := 2^{\lambda}$ are universal depth and size upper bounds for the first resp. second stage of permissible sequentiality adversaries from Definition 5. We use the notation (j, q) , where $j \in \{1, 2, 3\}$ corresponds to the oracle index for F, V, $\tilde{A}$ respectively.

5.2 The Main Result

Theorem 2 (VDF–OWF Separation). Relative to $\mathcal{O}_{\text{sep}}$ from Definition 11 there exists a relativized, perfectly complete, perfectly sound, and sequential VDF (according to Definition 5), while no relativized one-way function exists.

We split the proof of Theorem 2 into the following three lemmas.

Lemma 2 (PRGs Do Not Exist). Relative to $\mathcal{O}_{\text{sep}}$ from Definition 11, no relativized infinitely often secure PRG exists according to Definition 2.

Lemma 3 (VDFs Exist). Relative to $\mathcal{O}_{\text{sep}}$ from Definition 11 there exists a relativized, perfectly complete, perfectly sound, and sequential VDF (according to Definition 5).

Lemma 4 (Relativized PRGs from OWFs). Let $\mathcal{O}$ be any oracle distribution and let $f^{\mathcal{O}}$ be a one-way function relative to $\mathcal{O}$ according to Definition 4. Then there exists a deterministic polynomial-time oracle algorithm

$$\text{PRG}^{f, \mathcal{O}} : \{0, 1\}^* \longrightarrow \{0, 1\}^*$$

that is length doubling, runs in time $O(n^2)$ on seeds of length n , and is infinitely often secure relative to $\mathcal{O}$ according to Definition 2.

Moreover, the construction and its security reduction are fully black-box. In particular, if a non-uniform probabilistic polynomial-time oracle distinguisher violates the infinitely-often security condition, the reduction yields a non-uniform probabilistic polynomial-time oracle inverter I for which

$$\Pr [O \leftarrow \mathcal{O}, x \leftarrow U_\lambda : f^{\mathcal{O}}(I^{\mathcal{O}}(\lambda, f^{\mathcal{O}}(x))) = f^{\mathcal{O}}(x)]$$

is non-negligible in λ (for all sufficiently large λ).

We prove the three lemmas in Sections 5.3 to 5.5, respectively.

Proof of Theorem 2. By Lemma 3, a VDF exists relative to $\mathcal{O}_{\text{sep}}$. If a one-way function existed relative to $\mathcal{O}_{\text{sep}}$, Lemma 4 would yield an infinitely often secure PRG relative to the same oracle distribution, contradicting Lemma 2. $\square$

Corollary 1 (Black-Box Separation). There is no fully black-box construction of an OWF from a VDF that relativizes to arbitrary oracle distributions, in the sense of Definition 12.

Proof. Such a construction, applied to the VDF that exists relative to $\mathcal{O}_{\text{sep}}$, would yield an OWF relative to $\mathcal{O}_{\text{sep}}$, contradicting Theorem 2. $\square$

5.3 Part One: Relativized PRGs Don't Exist

We first state and prove a few technical lemmas, and then give the proof of Lemma 2.

Lemma 5 (Most strings have large complexity). Fix any oracle $\mathcal{O}$. Then $\Pr[x \leftarrow \{0, 1\}^\lambda : K_{\text{long}}^{\mathcal{O}}(x) \leq \lambda - \Delta] \leq 2^{1-\Delta}$.

Proof. Denote by $L_{\text{long}}^{\mathcal{O}}[\lambda - \Delta] := \{x \in \{0, 1\}^\lambda \mid K_{\text{long}}^{\mathcal{O}}(x) \leq \lambda - \Delta\}$ the set of strings with complexity deficiency at least Δ . Suppose

$$\Pr [x \leftarrow \{0, 1\}^\lambda : K_{\text{long}}^{\mathcal{O}}(x) \leq \lambda - \Delta] \geq 2^{1-\Delta} \tag{14}$$

$$\iff |L_{\text{long}}^{\mathcal{O}}[\lambda - \Delta]| \geq 2^{\lambda+1-\Delta} . \tag{15}$$

Let Π_x denote the lexicographically first (hence unique) witness program for string x . (The mapping $x \mapsto \Pi_x$ is bijective.) Let

$$W^O[\lambda - \Delta] := \{ \Pi_x \mid x \in \{0, 1\}^\lambda \wedge \mathsf{K}_{\text{long}}^O(x) \leq \lambda - \Delta \} \subseteq \{0, 1\}^{\leq \lambda - \Delta} \quad (16)$$

denote the set of all witness programs for strings in $L_{\text{long}}^O[\lambda - \Delta]$. Then

$$2^{\lambda+1-\Delta} \geq |W^O[\lambda - \Delta]| = |L_{\text{long}}^O[\lambda - \Delta]| \geq 2^{\lambda+1-\Delta}. \quad (17)$$

□

Claim 1. For each F and each $\lambda \in \mathbb{N}$ it holds that $\text{Gen}^{\mathsf{F}}(\lambda) \subseteq \text{Gen}^{\mathsf{F}}(\lambda + 1)$ by definition of Gen .

Lemma 6. For each F and each $\lambda \in \{8^\kappa \mid \kappa \in \mathbb{N}\}$ it holds that $[G^{\mathsf{F}}]_{5\lambda} = [\text{Gen}^{\mathsf{F}}(\lambda/8)]_{5\lambda}$.

Proof. Recall that $G^{\mathsf{F}} := \bigcup_{\lambda \in \mathbb{N}} \text{Gen}^{\mathsf{F}}(\lambda)$. By definition $\text{Gen}^{\mathsf{F}}(8^{\kappa-1}) = \text{Gen}^{\mathsf{F}}(\lambda/8) \subseteq G^{\mathsf{F}}$.

Let $q \in G^{\mathsf{F}} \cap \{0, 1\}^{\leq 5\lambda}$. This means that there exists some $\lambda' = 8^{\kappa'} \in \mathbb{N}$ and $\Pi \in \{0, 1\}^{\lfloor \log_2 \lambda' \rfloor}$, $T \in [2^{\lambda'}]$, $x \in \{0, 1\}^{\lambda'}$ such that q is added to $\tilde{Q}$ in Line 11 of Sim_0 by Π, T , or in Line 11 of Sim_1 by Π, T, x . Because these two lines are only invoked if $|q| \geq 5\lambda'$, it must hold that $|q| \geq 5\lambda'$. Hence,

$$5\lambda \geq |q| \geq 5\lambda' \implies \lambda' \leq \lambda \iff \kappa' \leq \kappa \iff \lambda' \leq \lambda/8. \quad (18)$$

Then q must have been added already to $\text{Gen}^{\mathsf{F}}(\lambda') \subseteq \text{Gen}^{\mathsf{F}}(\lambda/8)$ by Claim 1. □

Lemma 7 (Sparsity of G^{F}). For each F and each $\lambda \in \{8^\kappa \mid \kappa \in \mathbb{N}\}$ it holds that $|[G^{\mathsf{F}}]_{5\lambda}| \leq 2^\lambda$.

Proof. Observe that $G^{\mathsf{F}} \cap \{0, 1\}^{5\lambda} \subseteq [G^{\mathsf{F}}]_{5\lambda} \subseteq [\text{Gen}^{\mathsf{F}}(\lambda/8)]_{5\lambda}$ by Lemma 6. In $\text{Gen}^{\mathsf{F}}(\lambda/8)$ there are

- at most $\lfloor \log_8(\lambda/8) \rfloor$ iterations of the outer loop (Line 2 of Gen in Figure 2),
- at most $2^{\lfloor \log_2(\lambda/8) \rfloor}$ choices for Π in Sim_0 ,
- at most $2^{\lambda/8}$ choices for T in Sim_0 ,
- at most $2^{\lambda/8}$ queries $\bigcup_i Q_i$ in Sim_0 ,
- at most $2^{\lfloor \log_2(\lambda/8) \rfloor}$ choices for Π in Sim_1 ,
- at most $2^{\lambda/8}$ choices for T in Sim_1 ,
- at most $2^{\lambda/8}$ choices for x in Sim_1 ,
- at most $2^{\lambda/8}$ queries $\bigcup_i Q_i$ in Sim_1 .

Overall, $\text{Gen}^{\mathsf{F}}(\lambda/8)$ has size at most

$$\log_8(\lambda/8) \cdot \lambda/8 \cdot 2^{\lambda/8} \cdot 2^{\lambda/8} \cdot (1 + 2^{\lambda/8}) \leq 2^\lambda \quad (19)$$

for all $\lambda \in \mathbb{N}$. □

Lemma 8. For each $J \subseteq \{0, 1\}^*$ it holds that $\widehat{\Phi}^{\mathsf{F}, \mathsf{V}, J}((\{0, 1\}^*)^3 \setminus \{\perp\}) = A^{\mathsf{F}, \mathsf{V}, J}$.

Proof. Recall that $A^{\mathsf{F}, \mathsf{V}, J} := L_{\text{long}}^{\mathsf{F}, \mathsf{V}, J} \cup J = \{x \in \{0, 1\}^* \mid \mathsf{K}_{\text{long}}^{\mathsf{F}, \mathsf{V}, J}(x) \leq |x| - 2\} \cup J$. To simplify the notation, let $k := \mathfrak{k}(|x|)$, $\mathsf{F}_k := [\mathsf{F}]_k$, $\mathsf{V}_k := [\mathsf{V}]_k$, $J_k := [J]_k$. We first show that $\widehat{\Phi}^{\mathsf{F}, \mathsf{V}, J}((\{0, 1\}^*)^3 \setminus \{\perp\}) \subseteq A^{\mathsf{F}, \mathsf{V}, J}$. Let $x \in \widehat{\Phi}^{\mathsf{F}, \mathsf{V}, J}((\{0, 1\}^*)^3 \setminus \{\perp\})$. By definition of $\widehat{\Phi}^{\mathsf{F}, \mathsf{V}, J}((\{0, 1\}^*)^3)$ it holds that $x \in J \subseteq A^{\mathsf{F}, \mathsf{V}, J}$, or there exists a pair (Π, Φ) such that $x = \widehat{\Phi}^{\mathsf{F}, \mathsf{V}, J}(\Pi, \Phi, x)$. By definition of $\widehat{\Phi}^{\mathsf{F}, \mathsf{V}, J}$ it holds that $\Pi^{\mathsf{F}_k, \mathsf{V}_k, \mathsf{M}}(\varepsilon) = x$ and $|\Pi| + |\Phi| \leq |x| - 2$. This means that (Π, Φ) is a short witness program for x and therefore it holds that $x \in L_{\text{long}}^{\mathsf{F}, \mathsf{V}, J} \subseteq A^{\mathsf{F}, \mathsf{V}, J}$.

We conclude by showing that $A^{\mathsf{F}, \mathsf{V}, J} \subseteq \widehat{\Phi}^{\mathsf{F}, \mathsf{V}, J}((\{0, 1\}^*)^3 \setminus \{\perp\})$. Let $x \in A^{\mathsf{F}, \mathsf{V}, J}$. By definition of $A^{\mathsf{F}, \mathsf{V}, J}$ it holds that $x \in J \subseteq \widehat{\Phi}^{\mathsf{F}, \mathsf{V}, J}((\{0, 1\}^*)^3 \setminus \{\perp\})$, or $x \in L_{\text{long}}^{\mathsf{F}, \mathsf{V}, J}$. By definition of $L_{\text{long}}^{\mathsf{F}, \mathsf{V}, J}$ there exists a pair (Π, Φ) such that $\Pi^{\mathsf{F}_k, \mathsf{V}_k, \mathsf{M}}(\varepsilon) = x$ and $|\Pi| + |\Phi| \leq |x| - 2$. By definition of $\widehat{\Phi}^{\mathsf{F}, \mathsf{V}, J}$ it holds that $x = \widehat{\Phi}^{\mathsf{F}, \mathsf{V}, J}(\Pi, \Phi, x)$. By definition of $\widehat{\Phi}^{\mathsf{F}, \mathsf{V}, J}((\{0, 1\}^*)^3)$ it follows that $x \in \widehat{\Phi}^{\mathsf{F}, \mathsf{V}, J}((\{0, 1\}^*)^3 \setminus \{\perp\})$. □

We are now ready to prove Lemma 2.

Proof of Lemma 2. Let us fix a PRG construction PRG relative to the oracles $\mathcal{O}_{\text{sep}}$. Without loss of generality, assume the PRG is length-doubling and runs in time $t_{\text{PRG}}(\lambda) \leq \lambda^2$. We construct a distinguisher D as follows. On input $z \in \{0, 1\}^{2\lambda}$ the distinguisher

- samples randomness $r \leftarrow \{0, 1\}^{\lambda^3}$,
- queries $y := z \parallel r$ to the oracle A^{F, V, G^F} , and
- returns the result, 1 meaning pseudorandom, and 0 meaning random.

First, from Lemma 5, we know that random strings $y \leftarrow \{0, 1\}^{2\lambda + \lambda^3}$ have high complexity with high probability, i.e., most strings are not in $L_{\text{long}}^{F, V, G^F}$. Moreover, from Lemma 7, we know that the set of long queries G^F is sparse and hence a random string is not in $A^{F, V, G^F} := L_{\text{long}}^{F, V, G^F} \cup G^F$ with high probability. Hence, if $z \in \{0, 1\}^{2\lambda}$ is a random string, D outputs 0 with high probability.

Second, we prove that for every PRG image $z \in \text{PRG}^{F, V, A^{F, V, G^F}}(\{0, 1\}^\lambda)$, the distinguisher D always outputs 1. Let $z := \text{PRG}^{F, V, A^{F, V, G^F}}(s)$ for some seed $s \in \{0, 1\}^\lambda$ and let $r \in \{0, 1\}^{\lambda^3}$ be arbitrary. Then there exists a program $\Pi_{s, r, \text{PRG}}$ that has the bit strings s and r , and the code of PRG hardcoded, and produces $y = z \parallel r$ given access to the oracles F , V , and A^{F, V, G^F} . Our goal is to show that y has low $K_{\text{long}}^{F, V, J}$ complexity (cf. Definition 9) and hence D indeed outputs 1.

We set the base program for y to be $\Pi_y := \Pi_{s, r, \text{PRG}}$ and choose the preprocessing program for y to be the universal preprocessing program

$$\widehat{\Phi}^{F, V, J}(\Pi, \Phi, x) := \begin{cases} x & \text{if } (x = \Pi^{[F]_k, [V]_k, M}(\varepsilon) \wedge |\Pi| + |\Phi| \leq |x| - 2) \vee x \in J, \\ \perp & \text{otherwise,} \end{cases} \quad (20)$$

where $k := |x|/6$ and $M := \text{Im}^*(\Phi^{[F]_k, [V]_k, [J]_k})$.

To see why $(\Pi_y, \widehat{\Phi})$ is a valid witness program for y (and hence y has low complexity), we need to show that the witness program is sufficiently small, and that it outputs y correctly. The size of the witness program is

$$|\Pi_y| + |\widehat{\Phi}| = |s| + |r| + |\text{PRG}| + \Theta(1) = \lambda + \lambda^3 + \Theta(1) \leq 2\lambda + \lambda^3 - 2 = |y| - 2$$

for sufficiently large λ . Hence, the witness program is small enough.

It remains to show that $(\Pi_y, \widehat{\Phi})$ is a valid witness program for y . The base program Π_y runs in time $\mathcal{O}(\lambda^2 + \lambda^3) = \mathcal{O}(\lambda^3)$ but makes queries to its oracles on strings of length at most $t_{\text{PRG}}(\lambda) \leq \lambda^2$. Hence, for sufficiently large λ , the base program Π_y 's maximal query length λ^2 is smaller than the length restriction inside the complexity notion which is $k := \frac{|y|}{6} = \frac{2\lambda + \lambda^3}{6}$. This implies

$$y = \Pi_y^{F, V, A^{F, V, G^F}}(\varepsilon) = \Pi_y^{[F]_k, [V]_k, [A^{F, V, G^F}]_k}(\varepsilon) \stackrel{\text{Lemma 8}}{=} \Pi_y^{[F]_k, [V]_k, [\text{Im}^*(\widehat{\Phi}^{F, V, G^F})]_k}(\varepsilon).$$

The last equality follows from Lemma 8, which shows that $A^{F, V, G^F} = \text{Im}^*(\widehat{\Phi}^{F, V, G^F})$. It remains to show that $[\text{Im}^*(\widehat{\Phi}^{F, V, G^F})]_k \subseteq \text{Im}^*(\widehat{\Phi}^{[F]_k, [V]_k, [G^F]_k})$.

Let $x \in [\text{Im}^*(\widehat{\Phi}^{F, V, G^F})]_k$. Then we know that $|x| \leq k$ and by the definition of the universal preprocessing function, we have that either (1) $x \in G^F$ or (2) there exist (Π, Φ) such that

$$\Pi^{[F]_{k'}, [V]_{k'}, M}(\varepsilon) = x,$$

where $k' = \frac{|x|}{6} \leq \frac{k}{6}$ and $M = \text{Im}^*(\Phi^{[F]_{k'}, [V]_{k'}, [G^F]_{k'}})$.

In case (1), we know that $x \in G^F \Rightarrow x \in [G^F]_k$ as $x \leq k$. In case (2), we observe that $[[F]_{k'}]_{k'} = [F]_{k'}$ as $k' \leq k$. Analogously for other oracles. Hence $x \in \text{Im}^*(\widehat{\Phi}^{[F]_k, [V]_k, [G^F]_k})$, i.e., $(\Pi_y, \widehat{\Phi})$ is a valid witness program for y . $\square$

5.4 Part Two: Relativized VDFs Exist

We argue that relative to $\mathcal{O}_{\text{sep}}$ from Definition 11 there exists a relativized, perfectly complete, perfectly sound, and sequential VDF (according to Definition 5). Our proof consists of three steps. First, we construct a *weak* VDF, i.e., a VDF that is only sequential on infinitely many security parameters and only against uniform adversaries. Second, we show how to turn this weak VDF into a fully sequential VDF. Third, we show how to lift uniform sequentiality to non-uniform sequentiality. Finally, we combine the three steps to prove Lemma 3.

Weak VDF. To start, we construct a weak VDF relative to $\mathcal{O}_{\text{sep}}$ as follows:

- **Setup** ^{$\mathcal{O}$} (λ, T) outputs $\text{pp} := (\lambda, T)$,
- **Eval** ^{$\mathcal{O}$} (pp, x) outputs $(y, \pi) := ((F(\cdot, T))^{(T)}(x), \varepsilon)$,
- **Verify** ^{$\mathcal{O}$} (pp, x, y, π) outputs 1 if and only if $V(x, y, T) = 1$

with an oracle $\mathcal{O} = (F, V, A^{F, V, G^F})$ sampled from $\mathcal{O}_{\text{sep}}$. This VDF construction is perfectly complete and perfectly sound.

Lemma 9 (Infinitely-Often Sequentiality). The VDF $\text{VDF}' = (\text{Setup}, \text{Eval}, \text{Verify})$ is uniform infinitely often sequential on any security parameter $\lambda \in \{8^\kappa \mid \kappa \in \mathbb{N}\}$ relative to $\mathcal{O}_{\text{sep}}$.

At a high level, the proof of Lemma 9 is by contradiction. Assume there exists a uniform adversary A that breaks the sequentiality of VDF' on infinitely many security parameters $\lambda \in \{8^\kappa \mid \kappa \in \mathbb{N}\}$. Then, we turn that adversary into an adversary B that infinitely often breaks the sequentiality of a random iterated function $F^{(T)}(\cdot, T)$ with few (sequential) queries to F on $\{0, 1\}^\lambda \times \{T\}$ only—which is impossible as formalized by Lemma 10. The main challenge is that the adversary A can make many queries to the oracles F , V , and A^{F, V, G^F} and we need to simulate the answers to these queries using few calls to F on $\{0, 1\}^\lambda \times \{T\}$. The calls that A makes to F can be simulated directly because A makes only few sequential queries by assumption. The calls that A makes to V can be simulated by 0; this induces a negligible simulation error. This is because a sequentiality adversary A cannot make sufficiently many sequential queries to F to find the valid image of any preimage. The calls that A makes to A^{F, V, G^F} are a bit more tricky. Notice that, as mentioned in Section 4.3, any “long” query of the uniform A has answer 1 *by definition* of the oracle A^{F, V, G^F} . This is because the uniform adversary A was simulated during the construction of the oracle A^{F, V, G^F} and hence its long queries are contained in the set G^F of long queries and thus in $A^{F, V, G^F} \supseteq G^F$. Moreover, any short query can be simulated without any calls to F on $\{0, 1\}^\lambda \times \{T\}$. This is because testing membership of a short query $x \in \{0, 1\}^{\leq 5\lambda}$ in A^{F, V, G^F} can be done using oracle access to $F(\cdot, T)$ only on inputs of length at most $5\lambda/6$, i.e., using only calls in $\{0, 1\}^{\leq 5\lambda/6} \times \{T\}$.

Before we prove Lemma 9 formally, we state and prove two technical lemmas which we will need in our proof.

Lemma 10 (Random Sequential Function). Let $s_0, s_1, d_1 \in \mathbb{N}$. For each s_0 -size-bounded oracle circuit B_0 , for each d_1 -depth-bounded s_1 -size-bounded oracle circuit B_1 , and each $\lambda \in \mathbb{N}$ and $T \geq d_1$ it holds that

$$\Pr \left[\begin{array}{l} f \leftarrow [\{0, 1\}^\lambda \rightarrow \{0, 1\}^\lambda] \\ x \leftarrow \{0, 1\}^\lambda \\ \text{state} := B_0^f(\lambda, T) \\ y := B_1^f(\lambda, T, x, \text{state}) \end{array} : y = f^{(T)}(x) \right] \leq \frac{d_1 \cdot s_1 + 1}{2^\lambda - s_0} . \quad (21)$$

Proof. Let B_1 be a depth d_1 adversary. Then there is a depth T adversary B_1' that behaves like B_1 in the first d_1 rounds and additionally queries y in the T -th round.

Let $F := \{\{0,1\}^\lambda \rightarrow \{0,1\}^\lambda\}$ be the set of all functions from $\{0,1\}^\lambda$ to $\{0,1\}^\lambda$. Let B_0, B'_1 be a sequentiality adversary. Because the adversary is not computationally bounded, an optimal preprocessing adversary B_0 simply stores all preimage–image pairs that it queries. Thus, w.l.o.g. we can assume that the state contains exactly the preimage–image pairs that B_0 queries, i.e.,

$$\text{state} := \left\{ (x_j^i, f(x_j^i)) \mid i \in [d_0], (1, x_j^i) \in Q_i \xrightarrow{i} B_0^f(\lambda, T) \right\}. \quad (22)$$

Let

$$p_i := \Pr \left[\begin{array}{c} x \leftarrow \{0,1\}^\lambda \\ f \leftarrow F \\ y_i := (1, f^{(i)}(x)) : (1, y_i) \in Q_i^* \\ B_1^f(\lambda, T, x, \text{state}) \xrightarrow{i} Q_i \end{array} \right] \quad (23)$$

be the probability that the i -th intermediate image is queried in the i -th round by the adversary. Let $Q_i^* := \bigcup_{\ell=1}^i Q_\ell$ be the set of all queries that the adversary has made *up to* round i . The proof proceeds by induction: We prove the statement

$$p_i \leq \frac{\sum_{\ell=1}^i v_\ell}{2^\lambda - s_0} \quad (24)$$

where $v_\ell := \begin{cases} s_1 & \text{if } 1 \leq \ell \leq d_1 \\ 1 & \text{if } \ell = T \\ 0 & \text{otherwise} \end{cases}$. We observe that the probability of querying the i -th image is small

conditioned on the fact that the $(i-1)$ -th image was not queried before. That is,

$$\Pr \left[\begin{array}{c} x \leftarrow \{0,1\}^\lambda \\ f \leftarrow F \\ y_i := f^{(i)}(x) : (1, y_i) \in Q_i \\ B^f(\lambda, T, x, \text{state}) \xrightarrow{i} Q_i \end{array} \middle| (1, y_{i-1}) \notin Q_{i-1}^* \right] \leq \frac{v_i}{2^\lambda - s_0}. \quad (25)$$

In the first round of queries the adversary knows at most s_0 many preimage–image pairs. It follows immediately that the initial probability is bounded by

$$p_1 \leq \frac{s_1}{2^\lambda - s_0} = \frac{v_1}{2^\lambda - s_0}. \quad (26)$$

For the induction step we bound

$$\Pr[(1, y_i) \in Q_i] \leq \Pr[(1, y_i) \in Q_i \mid (1, y_{i-1}) \notin Q_{i-1}^*] + \Pr[(1, y_{i-1}) \in Q_{i-1}^*] \quad (27)$$

$$\stackrel{(25)}{\leq} \frac{v_i}{2^\lambda - s_0} + \Pr[(1, y_{i-1}) \in Q_{i-1}^*] \quad (28)$$

$$\stackrel{(24)}{\leq} \frac{\sum_{\ell=1}^i v_\ell}{2^\lambda - s_0}. \quad (29)$$

Thus, overall we see that

$$\Pr[(1, y_T) \in Q_{d_1}^*] \leq \frac{\sum_{\ell=1}^T v_\ell}{2^\lambda - s_0} \leq \frac{d_1 \cdot s_1 + 1}{2^\lambda - s_0}. \quad (30)$$

□

Lemma 11 (Complexity Oracle Restriction). For each function F , each oracle $J \subseteq \{0,1\}^*$ and each length $\lambda \in \mathbb{N}$ it holds that

$$\left[L_{\text{long}}^{F, V, J} \right]_{5\lambda} = \left[L_{\text{long}}^{[F]_k, [V]_k, [J]_k} \right]_{5\lambda} \quad (31)$$

where $k := \mathfrak{k}(5\lambda)$ and $L_{\text{long}}^{\mathbf{F}, \mathbf{V}, J}$ is defined in Definition 9.

Proof. Let $\lambda \in \mathbb{N}$. Then

$$\left[L_{\text{long}}^{\mathbf{F}, \mathbf{V}, J} \right]_{5\lambda} := \left\{ x \in \{0, 1\}^{\leq 5\lambda} \mid \mathbf{K}_{\text{long}}^{\mathbf{F}, \mathbf{V}, J}(x) \leq |x| - 2 \right\} \quad (32)$$

$$= \left\{ x \in \{0, 1\}^{\leq 5\lambda} \mid \begin{array}{l} \Pi, \Phi \in \{0, 1\}^* \\ \Pi^{[\mathbf{F}]_{k_x}, [\mathbf{V}]_{k_x}, \mathbf{M}}(\varepsilon) = x \\ \mathbf{M} := \Phi^{[\mathbf{F}]_{k_x}, [\mathbf{V}]_{k_x}, [\mathbf{J}]_{k_x}} \\ k_x := \mathfrak{k}(|x|) \leq k \end{array} \right\} \quad (33)$$

$$= \left\{ x \in \{0, 1\}^{\leq 5\lambda} \mid \begin{array}{l} \Pi, \Phi \in \{0, 1\}^* \\ \Pi^{[\mathbf{F}]_{k_x}, [\mathbf{V}]_{k_x}, \mathbf{M}}(\varepsilon) = x \\ \mathbf{M} := \Phi^{[\mathbf{F}]_{k_x}, [\mathbf{V}]_{k_x}, [\mathbf{J}]_{k_x}} \\ k_x := \mathfrak{k}(|x|) \leq k \end{array} \right\} \quad (34)$$

$$= \left[L_{\text{long}}^{[\mathbf{F}]_k, [\mathbf{V}]_k, [\mathbf{J}]_k} \right]_{5\lambda} \quad (35)$$

□

Lemma 12 (Pigeonhole Principle). Let $\mathcal{X}$ and $\mathcal{Y}$ be two distributions on the spaces X and Y , respectively. Let $\mathfrak{f} : X \times Y \rightarrow \{0, 1\}$ be a predicate. Let $\alpha, \beta \in \mathbb{R}_{\geq 0}$. It holds that

$$\Pr[x \leftarrow \mathcal{X}, y \leftarrow \mathcal{Y} : \mathfrak{f}(x, y) = 1] \leq \alpha \quad (36)$$

$$\implies \Pr[x \leftarrow \mathcal{X} : \Pr[y \leftarrow \mathcal{Y} : \mathfrak{f}(x, y) = 1] \geq \beta] \leq \alpha/\beta. \quad (37)$$

Proof. Let $X_{\geq \beta} := \{x \in X : \Pr[y \leftarrow \mathcal{Y} : \mathfrak{f}(x, y) = 1] \geq \beta\}$. By the implication's conditional it holds that

$$\alpha \geq \Pr[x \leftarrow \mathcal{X}, y \leftarrow \mathcal{Y} : \mathfrak{f}(x, y) = 1] \quad (38)$$

$$= \sum_{x' \in X} \Pr[x \leftarrow \mathcal{X} : x = x'] \cdot \Pr[y \leftarrow \mathcal{Y} : \mathfrak{f}(x', y) = 1] \quad (39)$$

$$\geq \sum_{x' \in X_{\geq \beta}} \Pr[x \leftarrow \mathcal{X} : x = x'] \cdot \Pr[y \leftarrow \mathcal{Y} : \mathfrak{f}(x', y) = 1] \quad (40)$$

$$\geq \sum_{x' \in X_{\geq \beta}} \Pr[x \leftarrow \mathcal{X} : x = x'] \cdot \beta \quad (41)$$

$$= \beta \cdot \Pr[x \leftarrow \mathcal{X} : x \in X_{\geq \beta}]. \quad (42)$$

That is,

$$\Pr[x \leftarrow \mathcal{X} : x \in X_{\geq \beta}] \leq \alpha/\beta. \quad (43)$$

□

Lemma 13 (Borel–Cantelli [Bor09; Can17]). Let (Ω, E, μ) be a probability space. For each countable sequence of events $(A_\lambda)_{\lambda=0}^\infty \in E^\omega$ it holds that

$$\sum_{\lambda=0}^\infty \mu(A_\lambda) < \infty \iff \mu\left(\bigcap_{\lambda_0=0}^\infty \bigcup_{\lambda=\lambda_0}^\infty A_\lambda\right) = 0. \quad (44)$$

Now we can prove the main claim that our construction $\mathbf{VDF}'$ is a weakly sequential VDF relative to $\mathcal{O}_{\text{sep}}$.

Oracle $F'(x', T')$	Oracle $V'(x', y', T')$
1 : if $T' = T \wedge x' = \lambda$	1 : $x'^{T'} := (F'(\cdot, T'))^{(T')}(x')$
2 : return $F(x', T')$	2 : return 1 iff $y' = x'^{T'}$
3 : if $F'[x', T'] = \perp$	
4 : $F'[x', T'] \leftarrow \{0, 1\}^\lambda$	
5 : return $F'[x', T']$	

Fig. 3: Oracles F' and V' .

Proof of Lemma 9. Uniform infinitely often sequentiality formally means

$$\exists \mathcal{O}, \epsilon \forall c, d \in \mathbb{N} \exists \mathfrak{o}_d, \mathfrak{o}_s \in o(1) \forall A_0 \in C_{T^c \lambda^c}^U \forall A_1 \in C_{T^c \lambda^c, T^\epsilon \lambda^c}^U \forall \lambda_0 \exists \lambda \geq \lambda_0 \forall T \in P_{\mathfrak{d}_1, \mathfrak{o}_d, \mathfrak{o}_s}(\lambda) \neq \emptyset :$$

$$p_{\lambda, T, A} := \Pr \left[\begin{array}{l} \mathcal{O} \leftarrow \mathcal{O} \\ \text{pp} \leftarrow \text{Setup}^{\mathcal{O}}(\lambda, T) \\ \text{state} := A_0^{\mathcal{O}}(\text{pp}) \\ x \leftarrow \{0, 1\}^\lambda \\ y' := A_1^{\mathcal{O}}(x, \text{state}) \\ (y, \pi) \leftarrow \text{Eval}^{\mathcal{O}}(\text{pp}, x) \end{array} : y' = y \right] \leq 1/\lambda^d .$$

Suppose towards a contradiction that

$$\forall \mathcal{O}, \epsilon \exists c, d \in \mathbb{N} \forall \mathfrak{o}_d, \mathfrak{o}_s \in o(1) \exists A_0 \in C_{T^c \lambda^c} \exists A_1 \in C_{T^c \lambda^c, T^\epsilon \lambda^c} \exists \lambda_0 \forall \lambda \geq \lambda_0 \exists T \in P_{\mathfrak{d}_1, \mathfrak{o}_d, \mathfrak{o}_s}(\lambda) \neq \emptyset : p_{\lambda, T, A} \geq 1/\lambda^d . \quad (45)$$

This holds in particular for $\mathcal{O} := \mathcal{O}_{\text{sep}}$ and $\epsilon := 1/2$. Fix such $c, d \in \mathbb{N}$, let $\mathfrak{o}_d(\lambda) := 1/\lambda$ and $\mathfrak{o}_s(\lambda) := 1/\log_2 \lambda$, and fix the corresponding $A_0 \in C_{T^c \lambda^c}$ and $A_1 \in C_{T^c \lambda^c, T^\epsilon \lambda^c}$ and $\lambda_0 \in \mathbb{N}$ from Equation (45). Then let

$$\lambda := \min \left\{ \lambda \in \mathbb{N} \left| \frac{1}{\lambda^d} \geq \frac{2^{\mathfrak{o}_s(\lambda)\lambda(\epsilon+c)} \lambda^{2c+1}}{2^{\lambda-2^{5/8} \cdot \lambda - 2^{\mathfrak{o}_s(\lambda)\lambda(c+1)}} \lambda^c} + \frac{2^{\mathfrak{o}_s(\lambda)\lambda(\epsilon+c)} \lambda^{2c}}{2^{\lambda-2^{\mathfrak{o}_s(\lambda)\lambda(c+1)}} \lambda^c - 2^{\mathfrak{o}_s(\lambda)\lambda(\epsilon+c)} \lambda^{2c}} \right. \right\} + \lambda_0 \geq \lambda_0 \quad (46)$$

and fix any $T \in P_{\mathfrak{d}_1, \mathfrak{o}_d, \mathfrak{o}_s}(\lambda) \neq \emptyset$ s.t. $T \geq \lambda^{2c}$. Such a T exists because the permissible lower bound is $\mathfrak{d}_1(\lambda, T) = T^\epsilon \lambda^c \leq T \lambda^{\Theta(1)} \iff T \geq \lambda^{\Theta(1)}$ and the permissible upper bound is $T \leq 2^{\lambda/\log_2 \lambda}$.

We first observe that because the winning condition of the sequentiality game only depends on the $\{0, 1\}^\lambda \times \{T\}$ slice of the oracle F , we can sample all values of F outside of $\{0, 1\}^\lambda \times \{T\}$ without changing the output distribution of A , i.e., when running A with oracles $F', V', A^{F', V', G^{F'}}$ where F' and V' are as defined in Figure 3, we get that

$$\begin{aligned} p'_{\lambda, T, A} &:= \Pr_{F, F', x} \left[A^{F', V', A^{F', V', G^{F'}}}(\lambda, T, x) = (F(\cdot, T))^{(T)}(x) \right] \\ &= \Pr_{F, F', x} \left[A^{F, V, A^{F, V, G^F}}(\lambda, T, x) = (F(\cdot, T))^{(T)}(x) \right] . \end{aligned} \quad (47)$$

Now we use the VDF sequentiality adversary A to construct an adversary B against the sequentiality of random functions whose success probability is then bounded by Lemma 10.

Concretely, for any VDF uniform sequentiality adversary $A = (A_0, A_1)$ with oracle access to $F', V', A^{F', V', G^{F'}}$ we construct a uniform adversary $B := (B_0, B_1)$ such that B has oracle access to F only on inputs in

$\{0, 1\}^\lambda \times \{T\}$, and, on input a random $x \in \{0, 1\}^\lambda$, it outputs $y = (F(\cdot, T))^{(T)}(x)$ with probability

$$p'_{\lambda, T, B} := \Pr \left[\begin{array}{l} \mathbf{f} \leftarrow \left\{ \mathbf{f} : \{0, 1\}^\lambda \rightarrow \{0, 1\}^\lambda \right\} \\ \mathbf{F} : (x, T) \mapsto \mathbf{f}(x) \\ \text{state} := \mathbf{B}_0^F(\lambda, T) \\ x \leftarrow \{0, 1\}^\lambda \\ y := \mathbf{f}^{(T)}(x) \\ y' := \mathbf{B}_1^F(\lambda, T, x, \text{state}) \end{array} : y' = y \right] \geq p'_{\lambda, T, A} - \xi(\lambda) \quad (48)$$

where

$$\xi(\lambda) := \frac{2^{\mathfrak{o}_s(\lambda)\lambda(\epsilon+c)} \lambda^{2c}}{2^\lambda - 2^{\mathfrak{o}_s(\lambda)\lambda(c+1)} \lambda^c - 2^{\mathfrak{o}_s(\lambda)\lambda(\epsilon+c)} \lambda^{2c}} \quad (49)$$

is the simulation error. Furthermore, by construction of B , we will show that

$$\mathbf{A}_0 \in C_{T^c \lambda^c} \wedge \mathbf{A}_1 \in C_{T^c \lambda^c, T^\epsilon \lambda^c} \Rightarrow \mathbf{B}_0 \in C_{T^{c+1} \lambda^c + 2^{5/8 \cdot \lambda}} \wedge \mathbf{B}_1 \in C_{T^c \lambda^c, T^\epsilon \lambda^c} . \quad (50)$$

That is, B 's size and depth are $\mathfrak{s}_0(\lambda, T) := T^{c+1} \lambda^c + 2^{5/8 \cdot \lambda}$, $\mathfrak{s}_1(\lambda, T) := T^c \lambda^c$ and $\mathfrak{d}_1(\lambda, T) := T^\epsilon \lambda^c \lesssim T$.

To upper bound $p'_{\lambda, T, B}$, we use Lemma 10 which states that a random function cannot be sped up (even with preprocessing). This lemma implies that

$$p'_{\lambda, T, B} \leq \frac{T^\epsilon \lambda^c \cdot T^c \lambda^c + 1}{2^\lambda - (T^{c+1} \lambda^c + 2^{5/8 \cdot \lambda})} \leq \frac{2^{\mathfrak{o}_s(\lambda)\lambda(\epsilon+c)} \lambda^{2c} + 1}{2^\lambda - 2^{5/8 \cdot \lambda} - 2^{\mathfrak{o}_s(\lambda)\lambda(c+1)} \lambda^c} . \quad (51)$$

Combining Equations (45), (48) and (51) yields the contradiction

$$\frac{1}{\lambda^d} \lesssim \frac{2^{\mathfrak{o}_s(\lambda)\lambda(\epsilon+c)} \lambda^{2c} + 1}{2^\lambda - 2^{5/8 \cdot \lambda} - 2^{\mathfrak{o}_s(\lambda)\lambda(c+1)} \lambda^c} + \frac{2^{\mathfrak{o}_s(\lambda)\lambda(\epsilon+c)} \lambda^{2c}}{2^\lambda - 2^{\mathfrak{o}_s(\lambda)\lambda(c+1)} \lambda^c - 2^{\mathfrak{o}_s(\lambda)\lambda(\epsilon+c)} \lambda^{2c}} \quad (52)$$

to Equation (46).

It remains to construct B and prove its success probability, i.e., Equation (48), and its size and depth bounds, i.e., Equation (50).

Description of B . Formally, B is depicted in Figure 4. Here we give a high-level overview. The first stage B_0 , on input a security parameter λ and time parameter T and with oracle access to F , first diagonalizes over all possible uniform sequentiality adversaries and collects their long queries in the set J . This is executed by invoking the uniform algorithm E guaranteed by Lemma 1. Then, B_0 computes the set A' which is defined to be the union of J and the complexity oracle $L_{\text{long}}^{[F]_{\mathfrak{t}(5\lambda)}, [V]_{\mathfrak{t}(5\lambda)}, J}}$, viewed as a set. Then B_0 runs the preprocessing phase A_0 of the original sequentiality adversary on input $\text{pp} := (\lambda, T)$ with oracle access to $F', V', A^{F', V', G^{F'}}$ and answers its queries q as follows.

- If q is a query to $A^{F', V', G^{F'}}$, then if q is long, i.e., $|q| \geq 5\lambda$, answer with 1, and otherwise answer with 1 if and only if $q \in A'$.
- If q is a query to F' , i.e., q is of the form $q = (x', T')$, answer with $F(q)$ if and only if $q \in \{0, 1\}^\lambda \times \{T\}$, and otherwise answer with a lazily sampled random value from $\{0, 1\}^\lambda$.
- If q is a query to V' , i.e., q is of the form (x', y', T') , query the T' successors of x' according to F' and answer with 1 if and only if y' is the T' -th successor, i.e., for each $i \in [T']$ compute $x'^i := (F'(\cdot, T'))^{(i)}(x')$ and answer with 1 if and only if $y' = x'^{T'}$.

Finally B_0 outputs its state $\text{state}_B := (A', \text{state}_A)$, which contains the set A' and state_A , the state of A_0 .

The second stage B_1 , on input security parameter λ , time parameter T , preimage $x \in \{0, 1\}^\lambda$ and state state_B , and with oracle access to F , runs the online sequentiality adversary A_1 with oracle access to $F', V', A^{F', V', G^{F'}}$ and inputs pp , x , and state state_A . Now B_1 answers all queries A_1 makes exactly the same way B_0 answers them, except for V' queries $q = (x', y', T')$ where $T' = T$. In this case, B_1 checks,

$B_0^F(\lambda, T)$	$B_1^F(\lambda, T, x, \text{state}_B)$
1 : $F' := []$ // a global variable storing F' 's function table 2 : $\text{pp} := (\lambda, T)$ 3 : $J := E^{F'}(\lambda, T) = \text{Gen}^{F'}(\lambda/8)$ 4 : $A' := L_{\text{long}}^{[F']_{\mathfrak{k}(5\lambda)}, [V']_{\mathfrak{k}(5\lambda)}, J} \cup J$ 5 : where $\mathfrak{k}(5\lambda) = \lfloor 5\lambda/6 \rfloor$ 6 : $\text{state}_A := A_0^{F', V', A^{F', V', G^{F'}}}(\text{pp})$ 7 : return $\text{state}_B := (A', \text{state}_A)$	1 : $(A', \text{state}_A) := \text{state}_B$ 2 : $y' := A_1^{F', V', A^{F', V', G^{F'}}}(\text{pp}, x, \text{state}_A)$ 3 : for every V' query (x', y', T') that A_1 makes 4 : if $(x', T') \in \{0, 1\}^\lambda \times \{T\}$ then 5 : Answer 1 iff $\exists (x'_0 := x', x'_1, \dots, x'_T := y')$ 6 : s.t. $\forall i \in [T'] : x'_i = F'[x'_{i-1}, T'] \neq \perp$ 7 : else 8 : Answer with $V'(x', y', T')$ 9 : return y'
Oracle $A^{F', V', G^{F'}}(q)$	
1 : if $q \in \{0, 1\}^{\geq 5\lambda}$ then return 1, else return 1 iff $q \in A'$	

Fig. 4: Algorithm $B := (B_0, B_1)$. All oracle queries during the execution of B are answered by oracles F', V' and $A^{F', V', G^{F'}}$ unless otherwise answered explicitly. Oracles F', V' are defined in Figure 3.

without querying F' , that $y' = (F'(\cdot, T'))^{(T)}(x')$ and answers 1 if and only if the check passes, i.e., B_1 answers 1 if and only if there exists $(x'_0 := x', x'_1, \dots, x'_T := y')$ such that for every $i \in [T']$, it holds that $x'_i = F'[x'_{i-1}, T'] \neq \perp$ where F' stores the lazily sampled function table of F' . Finally B_1 outputs what A_1 outputs.

Size and depth of B . We prove the claims made in Equation (50) about the size and depth parameters of B_0 and B_1 . By definition of B_0 , it makes F queries only when it runs algorithms E in Line 3 and A_0 in Line 6. The former is guaranteed by Lemma 1 to cost at most $2^{5/8 \cdot \lambda}$ queries and the latter costs at most T queries per query A_0 makes. A_0 can make at most $T^c \lambda^c$ queries. Therefore, the total number of F queries B_0 makes is at most $2^{5/8 \cdot \lambda} + T \cdot T^c \lambda^c$, which is an upper bound on its size as claimed in Equation (50). It is crucial to note that B_0 makes no F queries when it computes the set A' in Line 4: L_{long} makes F' queries on input length $\mathfrak{k}(5\lambda) = \lfloor 5\lambda/6 \rfloor \neq \lambda$, and hence such queries are lazily sampled by design (see oracle F' in Figure 4).

The second stage B_1 , by design, makes F queries only when A_1 makes an F' query, and furthermore, B_1 makes at most one F query for every such F' query. Crucially, when A_1 makes a query to V' the adversary B_1 makes no queries to F' . Hence, given that A_1 is $T^\epsilon \lambda^c$ -depth-bounded and $T^c \lambda^c$ -size-bounded, B_1 is also $T^\epsilon \lambda^c$ -depth-bounded and $T^c \lambda^c$ -size-bounded as claimed in Equation (50).

Correctness of B . Towards showing Equation (48), we show that B correctly simulates the oracles for A with probability at least $1 - \xi(\lambda)$.

We first examine B_0 's simulation of A_0 's oracle queries. For A_0 's queries to F' and V' , by inspection, B_0 answers correctly. The less obvious case is when q is a query A_0 makes to $A^{F', V', G^{F'}}$. We analyze this in two cases:

- Long query: $q \in \{0, 1\}^{\geq 5\lambda}$. In this case, B_0 answers 1. This is exactly how $A^{F', V', G^{F'}}$ answers q . Note that as A is a valid sequentiality adversary, it is simulated in the computation of $G^{F'}$, and hence, whenever A_0 makes a long query q , such a q is added to $\tilde{Q}$ in Line 11 of Sim_0^F in Figure 2, and finally to $G^{F'}$. By definition of $A^{F', V', G^{F'}}$ it holds that $q \in A^{F', V', G^{F'}}$, i.e., 1 is the correct answer.

- Short query: $q \in \{0, 1\}^{\leq 5\lambda}$. In this case, B_0 answers 1 if and only if $q \in A'$. This answer is consistent with $q \in A^{F', V', G^{F'}}$ because of Claim 2.

Claim 2. $[A^{F', V', G^{F'}}]_{5\lambda} = [A']_{5\lambda}$.

Proof. Observe

$$[A^{F', V', G^{F'}}]_{5\lambda} = [L_{\text{long}}^{F', V', G^{F'}} \cup G^{F'}]_{5\lambda} \quad (53)$$

$$= [L_{\text{long}}^{F', V', G^{F'}}]_{5\lambda} \cup [G^{F'}]_{5\lambda} \quad (54)$$

$$= [L_{\text{long}}^{[F']_{\mathfrak{t}(5\lambda)}, [V']_{\mathfrak{t}(5\lambda)}, [G^{F'}]_{\mathfrak{t}(5\lambda)}}]_{5\lambda} \cup [G^{F'}]_{5\lambda} \quad (55)$$

$$= [L_{\text{long}}^{[F']_{\mathfrak{t}(5\lambda)}, [V']_{\mathfrak{t}(5\lambda)}, [[G^{F'}]_{5\lambda}]_{\mathfrak{t}(5\lambda)}}]_{5\lambda} \cup [G^{F'}]_{5\lambda} \quad (56)$$

$$= [L_{\text{long}}^{[F']_{\mathfrak{t}(5\lambda)}, [V']_{\mathfrak{t}(5\lambda)}, [J]_{5\lambda}]_{\mathfrak{t}(5\lambda)}}]_{5\lambda} \cup [J]_{5\lambda} \quad (57)$$

$$= [L_{\text{long}}^{[F']_{\mathfrak{t}(5\lambda)}, [V']_{\mathfrak{t}(5\lambda)}, J}]_{5\lambda} \cup [J]_{5\lambda} \quad (58)$$

$$= [L_{\text{long}}^{[F']_{\mathfrak{t}(5\lambda)}, [V']_{\mathfrak{t}(5\lambda)}, J \cup J}]_{5\lambda} \quad (59)$$

$$= [A']_{5\lambda} \quad (60)$$

where Equations (55) and (58) follow from Lemma 11, and Equation (57) follows from Lemma 6. $\blacksquare$

Next, we examine B_1 's simulation of A_1 's oracle queries. The consistency of B_1 's answers to A_1 's F' queries follows trivially and the consistency of its answers to $A^{F', V', G^{F'}}$ queries follows by an analogous analysis to that of B_0 's response to such oracles. The interesting case is A_1 's queries to V' . Let $q = (x', y', T')$ be a V' query. We distinguish two cases. In case $(x', T') \notin \{0, 1\}^\lambda \times \{T\}$, B_1 answers 1 if and only if y' is the T' -th successor of x' according to F' . This is consistent with V' by definition. In case $(x', T') \in \{0, 1\}^\lambda \times \{T\}$, we distinguish two cases:

- y' is not the T -th successor of x' according to F' , i.e., $y' \neq (F'(\cdot, T))^{(T)}(x')$. In this case, B_1 correctly answers 0 as in this case, by definition, no $x'_0 := x', x'_1, \dots, x'_T := y'$ exist such that for every $i \in [T]$ it holds that $x'_i = F'[x'_{i-1}, T] \neq \perp$.
- y' is the T -th successor of x' according to F' , i.e., $y' = (F'(\cdot, T))^{(T)}(x')$. In this case, if A_1 makes all intermediate F' queries $(x'_0, T), \dots, (x'_{T-1}, T)$ where $x'_0 := x', x'_i$ is the response to query (x'_{i-1}, T) , and y' is the response to (x'_{T-1}, T) , then B_1 would correctly answer with 1. However, there is a possibility that A_1 queries (x', y', T) such that $y' = (F'(\cdot, T))^{(T)}(x')$ *without* making all intermediate queries. Towards computing the probability of this bad event, note that by construction, the variable F' (in B 's execution) contains (a) all F' queries (x', T') made by A_0 in addition to their T' -successors (according to F') and (b) all F' queries made by A_1 up to the current query round. As such, the (partial) truth table of F' captures A_1 's knowledge about F' . Let F'_T be the restriction of F' to entries of the form $(x, T) \in \{0, 1\}^\lambda \times \{T\}$. Thus, the probability that in some round $i \in [\mathfrak{d}_1(\lambda, T)]$, A_1 queries (x', y', T) such that $y' = (F'(\cdot, T))^{(T)}(x')$ without making all intermediate queries is at most

$$\frac{\mathfrak{w}_{1,i}(\lambda)}{2^\lambda - q_i} \quad (61)$$

where $\mathbf{w}_{1,i}(\lambda)$ is the width of the i -th layer of $\mathbf{A}_1$ and q_i is the number of entries in F'_T up to the i -th query round of $\mathbf{A}_1$. Note that $q_i \leq q_1 + \sum_{\ell=1}^{i-1} \mathbf{w}_{1,\ell}(\lambda)$. With a union bound we upper bound the probability of bad by

$$\sum_{i=1}^{\mathfrak{d}_1(\lambda, T)} \frac{\mathbf{w}_{1,i}(\lambda)}{2^\lambda - q_i} \leq \sum_{i=1}^{\mathfrak{d}_1(\lambda, T)} \frac{\mathbf{w}_{1,i}(\lambda, T)}{2^\lambda - q_1 - \sum_{\ell=1}^{i-1} \mathbf{w}_{1,\ell}(\lambda, T)} \quad (62)$$

$$\leq \sum_{i=1}^{\mathfrak{d}_1(\lambda, T)} \frac{\mathfrak{s}_1(\lambda, T)}{2^\lambda - q_1 - (i-1) \cdot \mathfrak{s}_1(\lambda, T)} \quad (63)$$

$$\leq \frac{\mathfrak{d}_1(\lambda, T) \cdot \mathfrak{s}_1(\lambda, T)}{2^\lambda - \mathfrak{s}_0(\lambda, T) \cdot T - \mathfrak{d}_1(\lambda, T) \cdot \mathfrak{s}_1(\lambda, T)} \quad (64)$$

$$\leq \frac{T^\epsilon \lambda^c \cdot T^c \lambda^c}{2^\lambda - T^c \lambda^c T - T^\epsilon \lambda^c \cdot T^c \lambda^c} \quad (65)$$

$$\leq \frac{2^{\mathfrak{o}_s(\lambda) \lambda (\epsilon+c)} \lambda^{2c}}{2^\lambda - 2^{\mathfrak{o}_s(\lambda) \lambda (c+1)} \lambda^c - 2^{\mathfrak{o}_s(\lambda) \lambda (\epsilon+c)} \lambda^{2c}} \quad (66)$$

$$=: \xi(\lambda) \quad (67)$$

for sufficiently large λ .

□

Infinitely often sequentiality to full sequentiality. Such an infinitely often sequential VDF $\mathbf{VDF}'$ can be turned into a fully sequential VDF $\mathbf{VDF}$ by truncating the input, with linear security loss. More concretely, let $\mathbf{VDF}'$ be a VDF that is secure on all powers-of-eight security parameters. We construct a VDF that is secure on all security parameters. Here, $\mathbf{VDF}.\text{Eval}(\lambda, T, x) := \mathbf{VDF}'.\text{Eval}(\lambda', T, x')$ where $\lambda' := 8^{\lceil \log_8 \lambda \rceil} \geq \lambda/8$ and $x' := x \upharpoonright \lambda'$ is the prefix of x of length λ' . $\mathbf{VDF}.\text{Verify}$ is defined analogously. The loss in security is at most linear in λ . Moreover, notice that if T is permissible for $\mathbf{VDF}$ then it is also permissible for $\mathbf{VDF}'$. Concretely, let $c \in \mathbb{N}$ be a speedup exponent and $\mathfrak{o}_s \in o(1)$ be the size bound exponent from the sequentiality of $\mathbf{VDF}$, then $T^c \lambda^c \leq 2^{\mathfrak{o}_s(\lambda) \lambda}$ for any permissible T . It follows that $T^c \lambda'^c \leq T^c \lambda^c \leq 2^{\mathfrak{o}_s(\lambda) \lambda} \leq 2^{\mathfrak{o}'_s(\lambda) \lambda'}$ for $\mathfrak{o}'_s(\lambda') := 8\mathfrak{o}_s(\lambda') \in o(1)$, i.e., $\mathbf{VDF}$ is sequential for $\mathfrak{o}'_s$ as well.

Uniform to non-uniform sequentiality. Finally, by the following lemma we can lift uniform sequentiality to non-uniform sequentiality.

Lemma 14 (Non-Uniform Sequentiality Lifting). Let $\mathbf{VDF}$ be a relativized VDF. If $\mathbf{VDF}$ is uniformly sequential relative to some samplable oracle distribution $\mathcal{O}$, it is also sequential against non-uniform adversaries relative to the same distribution.

The main idea behind this lemma is that a uniform (computationally unbounded) adversary can find the best advice string for a non-uniform adversary. Because the sequentiality game is falsifiable, the uniform adversary can compute the non-uniform adversary's success probability (over $\mathcal{O}$) for each advice string. Thus, the uniform adversary can find the best advice string, and thus has at least the same success probability as the non-uniform adversary.

Proof. We prove that a (non-uniform) adversary against the sequentiality of $\mathbf{VDF}$ can be turned into a uniform adversary against the sequentiality of $\mathbf{VDF}$ with the same success probability.

Counter assumption. Suppose for each $\epsilon \in \mathbb{R}$ there exist $c, d \in \mathbb{N}$ s.t. for each $\mathfrak{o}_d, \mathfrak{o}_s$ there exists a (non-uniform) sequentiality adversary $\mathbf{A} \in C_s^{\text{NU}} \times C_{s, \mathfrak{d}_1}^{\text{NU}}$ that has high success probability for all sufficiently large λ and all permissible T .

Let $\mathfrak{s}(\lambda, T) := T^c \lambda^c$ and $\mathfrak{d}_1(\lambda, T) := T^e \lambda^c$. Let A^* be the following uniform, computationally unbounded circuit family: The first stage A_0^* on input pp computes the circuit that wins the sequentiality game with maximal probability:

$$A' = (A'_0, A'_1) := \underset{\substack{A'_0 \in C_{\mathfrak{s}(\lambda, T)} \\ A'_1 \in C_{\mathfrak{s}(\lambda, T), \mathfrak{d}_1(\lambda, T)}}}{\text{argmax}} p_{\lambda, T, A'} \quad (68)$$

where

$$p_{\lambda, T, A'} := \Pr \left[\begin{array}{l} O \leftarrow \mathcal{O} \\ \text{pp} \leftarrow \text{Setup}^O(\lambda, T) \\ \text{state} := A_0'^O(\text{pp}) \\ x \leftarrow X \\ y' := A_1'^O(\text{pp}, x, \text{state}) \\ (y, \pi) \leftarrow \text{Eval}^O(\text{pp}, x) \end{array} : y' = y \right]. \quad (69)$$

Then, $A_0^*(\text{pp})$ returns $\text{state}^* := (A_0'^O(\text{pp}), A')$.

The second stage A_1^* on input $(\text{pp}, x, \text{state}^*)$ parses $(\text{state}', A') := \text{state}^*$ and returns $A_1'^O(\text{pp}, x, \text{state}')$.

Properties of A^* . Notice A^* does not need to make any queries to its oracle O to compute A' . Thus, A^* has the same query complexity as A' , i.e., $A^* \in C_{\mathfrak{s}}^U \times C_{\mathfrak{s}, \mathfrak{d}_1}^U$. Moreover, A^* 's success probability is at least the success probability of A' which is at least as large as A 's success probability.

This means that VDF is not uniformly sequential. $\square$

We are now ready to prove Lemma 3.

Proof of Lemma 3. Consider the VDF VDF' constructed above, which is perfectly complete and perfectly sound relative to $\mathcal{O}_{\text{sep}}$. By Lemma 9, VDF' is uniform infinitely often sequential on any security parameter $\lambda \in \{8^\kappa \mid \kappa \in \mathbb{N}\}$ relative to $\mathcal{O}_{\text{sep}}$. Truncating the input as described above yields a VDF VDF that is uniformly sequential on all security parameters relative to $\mathcal{O}_{\text{sep}}$ and that remains perfectly complete and perfectly sound. Finally, by Lemma 14, VDF is also sequential against non-uniform adversaries relative to $\mathcal{O}_{\text{sep}}$. $\square$

5.5 Part Three: Relativized PRGs from One-Way Functions

It remains to prove Lemma 4, i.e., a OWF implies a PRG relative to any oracle distribution. The result follows from well-known results in the literature. We use the fully black-box construction of a PRG from any OWF due to Haitner, Reingold, and Vadhan [HRV10], together with the black-box reduction framework of Baecher, Brzuska, and Fischlin [BBF13], to show that the construction remains secure relative to an arbitrary oracle distribution. We first recall the relevant notions and results, from which we derive Corollary 2, and then prove Lemma 4.

Definition 12 (Fully Black-Box Reduction). *Following [BBF13, Definition 2.1], a primitive $A = (\mathcal{F}_A, \mathcal{R}_A)$ consists of a set $\mathcal{F}_A$ of implementations and a relation $\mathcal{R}_A$, where $(g, D) \in \mathcal{R}_A$ means that D breaks the implementation g . Define a primitive B analogously. A fully black-box reduction constructing A from B consists of probabilistic polynomial-time oracle algorithms G and S satisfying:*

- *Correctness. For every $f \in \mathcal{F}_B$, $G^f \in \mathcal{F}_A$.*
- *Security. For every $f \in \mathcal{F}_B$ and every machine D ,*

$$(G^f, D^f) \in \mathcal{R}_A \implies (f, S^{D, f}) \in \mathcal{R}_B.$$

Lemma 15 (Composition of Fully Black-Box Reductions). Let A , B , and C be primitives. If there is a fully black-box reduction constructing A from B , and a fully black-box reduction constructing B from C , then there is a fully black-box reduction constructing A from C .

Proof of Lemma 15. Let (G_{AB}, S_{AB}) and (G_{BC}, S_{BC}) be the two fully black-box reductions. Define their composition by

$$G_{AC}^f := G_{AB}^{G_{BC}^f}.$$

Correctness follows immediately: for every $f \in \mathcal{F}_C$, first $G_{BC}^f \in \mathcal{F}_B$, and hence $G_{AC}^f \in \mathcal{F}_A$. Now suppose that D^f breaks G_{AC}^f . The first security reduction yields an adversary

$$D_B^f := S_{AB}^{D^f, G_{BC}^f}$$

that breaks G_{BC}^f . Applying the second security reduction yields

$$S_{AC}^{D, f} := S_{BC}^{D_B, f},$$

which breaks f . The composed construction and reduction are probabilistic polynomial-time oracle algorithms and use f and D only as black boxes. Thus they satisfy Definition 12. $\square$

Theorem 3 (Haitner–Reingold–Vadhan [HRV10]). For every polynomial-time one-way function f , there is a fully black-box construction of a polynomial-time generator family $G_\lambda^f: \{0, 1\}^{\mathfrak{d}(\lambda)} \rightarrow \{0, 1\}^{\mathfrak{d}(\lambda)+\mathfrak{t}(\lambda)}$, where $\mathfrak{d}$ is polynomial-time computable and polynomially bounded, $\lambda \leq \mathfrak{d}(\lambda)$, and $\mathfrak{t}(\lambda) \geq 1$. The distributions $G_\lambda^f(U_{\mathfrak{d}(\lambda)})$ and $U_{\mathfrak{d}(\lambda)+\mathfrak{t}(\lambda)}$ are computationally indistinguishable for non-uniform probabilistic polynomial-time distinguishers. The construction and its security reduction are fully black-box according to Definition 12.

The lower bound $\mathfrak{d}(\lambda) \geq \lambda$ is without loss of generality. If the original construction uses fewer seed bits, we may append unused uniform bits to both the seed and the output. This preserves the stretch, pseudorandomness, and fully black-box security reduction.

Lemma 16. Let $c', c'' \in \mathbb{N}$ be constants with $c', c'' \geq 1$. If $G: \{0, 1\}^n \rightarrow \{0, 1\}^{n+c''}$ is a PRG, or an infinitely often secure PRG, running in time at most $n^{c'}$, then there is a length-doubling PRG $\widehat{G}: \{0, 1\}^\ell \rightarrow \{0, 1\}^{2\ell}$ running in time $O(\ell^2)$ with the same respective security guarantee. Moreover, the construction of $\widehat{G}$ from G and its security reduction are fully black-box.

Proof. Let $m := \lfloor \ell^{1/c'} \rfloor$ and $t := \lceil \ell/c'' \rceil$. Parse a seed in $\{0, 1\}^\ell$ as $x\|r$, where $x \in \{0, 1\}^m$ and $r \in \{0, 1\}^{\ell-m}$. Starting with $G_0(x) := x$, recursively parse $G_{i-1}(x) = x_i\|z_i$, where $x_i \in \{0, 1\}^m$, and define

$$G_i(x) := G(x_i)\|z_i \quad \text{for } i \in [t].$$

Thus $G_t(x)\|r$ has length $\ell + tc'' \geq 2\ell$. Define $\widehat{G}(x\|r)$ to be its first 2ℓ bits.

A standard hybrid argument over the t invocations of G shows that $\widehat{G}(U_\ell)$ is pseudorandom, with distinguishing loss at most the polynomial factor t . Since $m^{c'} \leq \ell$ and $t = O(\ell)$, the t invocations of G , including the overhead for copying their outputs, take time $O(\ell^2)$. The same argument preserves infinitely-often security. For every seed length n on which G satisfies the required pseudorandomness bound, take $\ell = n^{c'}$; then $\lfloor \ell^{1/c'} \rfloor = n$. Thus, a distinguisher that violates infinitely-often security of $\widehat{G}$ yields, through the same hybrid argument, a non-uniform distinguisher that violates infinitely-often security of G . The construction accesses G only as an oracle. The hybrid argument also gives a probabilistic polynomial-time oracle reduction that converts any distinguisher for $\widehat{G}$ into a distinguisher for G . Hence, the construction and reduction are fully black-box according to Definition 12. $\square$

Corollary 2 (Length-Doubling PRG from OWFs). Every infinitely often secure OWF admits a fully black-box construction of a length-doubling, infinitely often secure PRG that runs in quadratic time in its seed length.

Proof of Corollary 2. Let $\mathfrak{d}$ and G_λ^f be given by Theorem 3. Truncate the output to obtain one bit of stretch. Define a generator G^f on every seed length n as follows. If $n = \mathfrak{d}(\lambda)$ for some $\lambda \leq n$, choose the largest such λ and run the truncated generator G_λ^f ; otherwise, output 0^{n+1} . Because $\mathfrak{d}$ is polynomial-time computable and $\lambda \leq \mathfrak{d}(\lambda)$, this search and the resulting generator run in time polynomial in n . The image of $\mathfrak{d}$ is infinite, and G^f is pseudorandom on all sufficiently large seed lengths in that image. Since $\mathfrak{d}(\lambda)$ is polynomially bounded in λ , these seed lengths are polynomially related to the original security parameter, so negligible advantage in λ remains negligible in n . Hence, G^f is an infinitely often secure one-bit-stretch PRG. Let $n^{c'}$, for some constant $c' \geq 1$, bound its runtime. Applying Lemma 16 with $c'' = 1$ gives an infinitely often secure length-doubling PRG running in time $O(\ell^2)$. Finally, Lemma 15 shows that seed padding, truncation, reindexing, and length doubling preserve the fully black-box reduction. $\square$

Using Corollary 2, it is straightforward to see that a OWF implies a PRG relative to any oracle distribution.

Proof of Lemma 4. By Corollary 2, there is a fully black-box construction PRG^f and a fully black-box security reduction S . Give the construction and the reduction shared access to an additional oracle O . This yields $\text{PRG}^{f,O}$ and an inverter $I^O := S^{D,f,O}$. The hierarchy of black-box reductions in [BBF13, Theorem 2.5] and the relativization result in [BBF13, Theorem 2.9] justify this step for every fixed auxiliary oracle O .

Here O is sampled from $\mathcal{O}$, so we also need to average over the oracle. The construction and reduction have oracle-independent polynomial runtime and security loss. Consequently, each step of the fully black-box security proof may first be conditioned on a fixed O and then averaged over $O \leftarrow \mathcal{O}$. The seed lengths introduced by padding, reindexing, and length doubling are polynomially related, so inverse-polynomial advantage remains inverse polynomial. Thus, if a non-uniform probabilistic polynomial-time oracle distinguisher D^O violated infinitely-often security of $\text{PRG}^{f,O}$ relative to $\mathcal{O}$, the same reduction would satisfy

$$\Pr[O \leftarrow \mathcal{O}, x \leftarrow U_\lambda : f^O(I^O(\lambda, f^O(x))) = f^O(x)] \geq \frac{1}{\lambda^c}$$

for some fixed $c \in \mathbb{N}$ and for infinitely many λ . Since D is non-uniform, the resulting inverter I^O is also non-uniform. This contradicts Definition 4. Therefore $\text{PRG}^{f,O}$ is infinitely often secure relative to $\mathcal{O}$. $\square$

6 Acknowledgments

We thank Karen Azari and Chethan Kamath for helpful discussions in the early stages of this work.

7 AI Disclosure

This paper was written entirely by the authors, and all ideas and technical contributions are theirs. The large language models ChatGPT and Claude were used only at the final stage of preparing the manuscript to polish the writing. ChatGPT was additionally used to create Figure 1.

References

- [AA24] H. Abusalah and G. Avitabile. Black-Box Timed Commitments from Time-Lock Puzzles. In E. Boyle and M. Mahmoody, editors, *TCC 2024, Part III*, volume 15366 of *LNCS*, pages 460–493. Springer, Cham, December 2024.
- [AAF⁺25] H. Abusalah, K. Azari, D. Fiore, C. Kamath, and E. Tairi. On Verifiable Delay Functions from Time-Lock Puzzles. *IACR Cryptol. ePrint Arch.*:1782, 2025.

- [AAK⁺25] H. Abusalah, K. Azari, C. Kamath, E. Tairi, and M. von Consbruch. Impossibility of VDFs in the ROM: The Complete Picture. Cryptology ePrint Archive, Paper 2025/1773, 2025.
- [ABB⁺16] F. Armknecht, L. Barman, J.-M. Bohli, and G. O. Karame. Mirror: Enabling Proofs of Data Replication and Retrieval in the Cloud. In T. Holz and S. Savage, editors, *USENIX Security 2016*, pages 1051–1068. USENIX Association, August 2016.
- [ABC22] A. Arun, J. Bonneau, and J. Clark. Short-lived Zero-Knowledge Proofs and Signatures. In S. Agrawal and D. Lin, editors, *ASIACRYPT 2022, Part III*, volume 13793 of *LNCS*, pages 487–516. Springer, Cham, December 2022.
- [ABK06] E. Allender, H. Buhrman, and M. Koucký. What can be efficiently reduced to the Kolmogorov-random strings? *Annals of Pure and Applied Logic*, 138(1):2–19, 2006.
- [ABK⁺06] E. Allender, H. Buhrman, M. Koucký, D. van Melkebeek, and D. Ronneburger. Power from Random Strings. *SIAM Journal on Computing*, 35(6):1467–1493, 2006. eprint: <https://doi.org/10.1137/050628994>.
- [Abu25] H. Abusalah. On Graphs of Incremental Proofs of Sequential Work. In T. Jager and J. Pan, editors, *PKC 2025, Part V*, volume 15678 of *LNCS*, pages 203–233. Springer, Cham, May 2025.
- [AC23] H. Abusalah and V. Cini. An Incremental PoSW for General Weight Distributions. In C. Hazay and M. Stam, editors, *EUROCRYPT 2023, Part II*, volume 14005 of *LNCS*, pages 282–311. Springer, Cham, April 2023.
- [AFG⁺22] H. Abusalah, G. Fuchsbauer, P. Gazi, and K. Klein. SNACKs: Leveraging Proofs of Sequential Work for Blockchain Light Clients. In S. Agrawal and D. Lin, editors, *ASIACRYPT 2022, Part I*, volume 13791 of *LNCS*, pages 806–836. Springer, Cham, December 2022.
- [AKK⁺19] H. Abusalah, C. Kamath, K. Klein, K. Pietrzak, and M. Walter. Reversible Proofs of Sequential Work. In Y. Ishai and V. Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 277–291. Springer, Cham, May 2019.
- [BBB⁺18] D. Boneh, J. Bonneau, B. Bünz, and B. Fisch. Verifiable Delay Functions. In H. Shacham and A. Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 757–788. Springer, Cham, August 2018.
- [BBF13] P. Baecher, C. Brzuska, and M. Fischlin. Notions of Black-Box Reductions, Revisited. In K. Sako and P. Sarkar, editors, *ASIACRYPT 2013, Part I*, volume 8269 of *LNCS*, pages 296–315. Springer, Berlin, Heidelberg, December 2013.
- [BCH⁺22] N. Bitansky, A. R. Choudhuri, J. Holmgren, C. Kamath, A. Lombardi, O. Paneth, and R. D. Rothblum. PPAD is as Hard as LWE and Iterated Squaring. In E. Kiltz and V. Vaikuntanathan, editors, *TCC 2022, Part II*, volume 13748 of *LNCS*, pages 593–622. Springer, Cham, November 2022.
- [BDG⁺20] Z. Brakerski, N. Döttling, S. Garg, and G. Malavolta. Candidate iO from Homomorphic Encryption Schemes. In A. Canteaut and Y. Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 79–109. Springer, Cham, May 2020.
- [BG97] A. Blass and Y. Gurevich. The Linear Time Hierarchy Theorems for Abstract State Machines and RAMs. *JUCS - Journal of Universal Computer Science*, 3(4):247–278, 1997. eprint: <https://doi.org/10.3217/jucs-003-04-0247>.
- [BGJ⁺16] N. Bitansky, S. Goldwasser, A. Jain, O. Paneth, V. Vaikuntanathan, and B. Waters. Time-Lock Puzzles from Randomized Encodings. In M. Sudan, editor, *ITCS 2016*, pages 345–356. ACM, January 2016.
- [BHR⁺21] A. R. Block, J. Holmgren, A. Rosen, R. D. Rothblum, and P. Soni. Time- and Space-Efficient Arguments from Groups of Unknown Order. In T. Malkin and C. Peikert, editors, *CRYPTO 2021, Part IV*, volume 12828 of *LNCS*, pages 123–152, Virtual Event. Springer, Cham, August 2021.
- [BN00] D. Boneh and M. Naor. Timed Commitments. In M. Bellare, editor, *CRYPTO 2000*, volume 1880 of *LNCS*, pages 236–254. Springer, Berlin, Heidelberg, August 2000.
- [Bor09] É. M. Borel. Les probabilités dénombrables et leurs applications arithmétiques. *Rendiconti del Circolo Matematico di Palermo (1884-1940)*, 27(1):247–271, December 1909. Published: 1909/12/01.
- [BPR15] N. Bitansky, O. Paneth, and A. Rosen. On the Cryptographic Hardness of Finding a Nash Equilibrium. In V. Guruswami, editor, *56th FOCS*, pages 1480–1498. IEEE Computer Society Press, October 2015.

- [Can17] F. P. Cantelli. Sulla probabilità come limite della frequenza. *Rendiconti della Reale Accademia dei Lincei. Classe di scienze fisiche, matematiche e naturali*, 26(1):39–45, 1917.
- [Cha69] G. J. Chaitin. On the Simplicity and Speed of Programs for Computing Infinite Sets of Natural Numbers. *J. ACM*, 16(3):407–422, July 1969.
- [CHK25] S. Chakraborty, J. Hulett, and D. Khurana. On Weak NIZKs, One-Way Functions and Amplification. In Y. T. Kalai and S. F. Kamara, editors, *CRYPTO 2025, Part VII*, volume 16006 of *LNCS*, pages 580–610. Springer, Cham, August 2025.
- [CJ23] P. Chvojka and T. Jager. Simple, Fast, Efficient, and Tightly-Secure Non-malleable Non-interactive Timed Commitments. In A. Boldyreva and V. Kolesnikov, editors, *PKC 2023, Part I*, volume 13940 of *LNCS*, pages 500–529. Springer, Cham, May 2023.
- [CJS⁺21] P. Chvojka, T. Jager, D. Slamanig, and C. Striecks. Versatile and Sustainable Timed-Release Encryption and Sequential Time-Lock Puzzles (Extended Abstract). In E. Bertino, H. Shulman, and M. Waidner, editors, *ESORICS 2021, Part II*, volume 12973 of *LNCS*, pages 64–85. Springer, Cham, October 2021.
- [CLM23] V. Cini, R. W. F. Lai, and G. Malavolta. Lattice-Based Succinct Arguments from Vanishing Polynomials - (Extended Abstract). In H. Handschuh and A. Lysyanskaya, editors, *CRYPTO 2023, Part II*, volume 14082 of *LNCS*, pages 72–105. Springer, Cham, August 2023.
- [CP18] B. Cohen and K. Pietrzak. Simple Proofs of Sequential Work. In J. B. Nielsen and V. Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 451–467. Springer, Cham, 2018.
- [CP19] B. Cohen and K. Pietrzak. The Chia Network Blockchain. In 2019.
- [CSRT22] J. Chávez-Saab, F. Rodríguez-Henríquez, and M. Tibouchi. Verifiable Isogeny Walks: Towards an Isogeny-Based Postquantum VDF. In R. AlTawy and A. Hülsing, editors, *SAC 2021*, volume 13203 of *LNCS*, pages 441–460. Springer, Cham, 2022.
- [DGM⁺20] N. Döttling, S. Garg, G. Malavolta, and P. N. Vasudevan. Tight Verifiable Delay Functions. In C. Galdi and V. Kolesnikov, editors, *SCN 20*, volume 12238 of *LNCS*, pages 65–84. Springer, Cham, September 2020.
- [DLM19] N. Döttling, R. W. F. Lai, and G. Malavolta. Incremental Proofs of Sequential Work. In Y. Ishai and V. Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 292–323. Springer, Cham, May 2019.
- [DMP⁺19] L. De Feo, S. Masson, C. Petit, and A. Sanso. Verifiable Delay Functions from Supersingular Isogenies and Pairings. In S. D. Galbraith and S. Moriai, editors, *ASIACRYPT 2019, Part I*, volume 11921 of *LNCS*, pages 248–277. Springer, Cham, December 2019.
- [EFK⁺20a] N. Ephraim, C. Freitag, I. Komargodski, and R. Pass. Continuous Verifiable Delay Functions. In A. Canteaut and Y. Ishai, editors, *EUROCRYPT 2020, Part III*, volume 12107 of *LNCS*, pages 125–154. Springer, Cham, May 2020.
- [EFK⁺20b] N. Ephraim, C. Freitag, I. Komargodski, and R. Pass. SPARKs: Succinct Parallelizable Arguments of Knowledge. In A. Canteaut and Y. Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 707–737. Springer, Cham, May 2020.
- [FKP⁺21] C. Freitag, I. Komargodski, R. Pass, and N. Sirkin. Non-malleable Time-Lock Puzzles and Applications. In K. Nissim and B. Waters, editors, *TCC 2021, Part III*, volume 13044 of *LNCS*, pages 447–479. Springer, Cham, November 2021.
- [FS07] L. Fortnow and R. Santhanam. Time Hierarchies: A Survey. *Electronic Colloquium on Computational Complexity (ECCC)*, 14, January 2007.
- [GM24] J. Guan and H. Montgomery. On Sequential Functions and Fine-Grained Cryptography. In L. Reyzin and D. Stebila, editors, *CRYPTO 2024, Part V*, volume 14924 of *LNCS*, pages 393–428. Springer, Cham, August 2024.
- [GMP⁺06] J. A. Garay, P. D. MacKenzie, M. Prabhakaran, and K. Yang. Resource Fairness and Composability of Cryptographic Protocols. In S. Halevi and T. Rabin, editors, *TCC 2006*, volume 3876 of *LNCS*, pages 404–428. Springer, Berlin, Heidelberg, March 2006.
- [GRY25] Z. Guan, A. Riazanov, and W. Yuan. Breaking Verifiable Delay Functions in the Random Oracle Model. In Y. T. Kalai and S. F. Kamara, editors, *CRYPTO 2025, Part VII*, volume 16006 of *LNCS*, pages 161–191. Springer, Cham, August 2025.

- [HHK⁺22] C. Hoffmann, P. Hubáček, C. Kamath, K. Klein, and K. Pietrzak. Practical Statistically-Sound Proofs of Exponentiation in Any Group. In Y. Dodis and T. Shrimpton, editors, *CRYPTO 2022, Part II*, volume 13508 of *LNCS*, pages 370–399. Springer, Cham, August 2022.
- [HHK⁺23] C. Hoffmann, P. Hubáček, C. Kamath, and T. Krnák. (Verifiable) Delay Functions from Lucas Sequences. In G. N. Rothblum and H. Wee, editors, *TCC 2023, Part IV*, volume 14372 of *LNCS*, pages 336–362. Springer, Cham, 2023.
- [HN24] S. Hirahara and M. Nanashima. One-Way Functions and Zero Knowledge. In B. Mohar, I. Shinkar, and R. O’Donnell, editors, *56th ACM STOC*, pages 1731–1738. ACM Press, June 2024.
- [HR04] C.-Y. Hsiao and L. Reyzin. Finding Collisions on a Public Road, or Do Secure Hash Functions Need Secret Coins? In M. Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 92–105. Springer, Berlin, Heidelberg, August 2004.
- [HRV10] I. Haitner, O. Reingold, and S. Vadhan. Efficiency improvements in constructing pseudorandom generators from one-way functions. In *Proceedings of the Forty-Second ACM Symposium on Theory of Computing, STOC ’10*, 437–446, Cambridge, Massachusetts, USA. Association for Computing Machinery, 2010.
- [HY17] P. Hubáček and E. Yogev. Hardness of Continuous Local Search: Query Complexity and Cryptographic Lower Bounds. In P. N. Klein, editor, *28th SODA*, pages 1352–1371. ACM-SIAM, January 2017.
- [JMR⁺21] S. Jaques, H. Montgomery, R. Rosie, and A. Roy. Time-Release Cryptography from Minimal Circuit Assumptions. In A. Adhikari, R. Küsters, and B. Preneel, editors, *INDOCRYPT 2021*, volume 13143 of *LNCS*, pages 584–606. Springer, Cham, December 2021.
- [KLX20] J. Katz, J. Loss, and J. Xu. On the Security of Time-Lock Puzzles and Timed Commitments. In R. Pass and K. Pietrzak, editors, *TCC 2020, Part III*, volume 12552 of *LNCS*, pages 390–413. Springer, Cham, November 2020.
- [Kol63] A. N. Kolmogorov. On Tables of Random Numbers. *Sankhyā: The Indian Journal of Statistics, Series A (1961-2002)*, 25(4):369–376, 1963.
- [LM23] R. W. F. Lai and G. Malavolta. Lattice-Based Timed Cryptography. In H. Handschuh and A. Lysyanskaya, editors, *CRYPTO 2023, Part V*, volume 14085 of *LNCS*, pages 782–804. Springer, Cham, August 2023.
- [LMP24] Y. Liu, N. Mazon, and R. Pass. A Note on Zero-Knowledge for NP and One-Way Functions. Cryptology ePrint Archive, Report 2024/800, 2024.
- [LSS20] E. Landerreche, M. Stevens, and C. Schaffner. Non-interactive Cryptographic Timestamping Based on Verifiable Delay Functions. In J. Bonneau and N. Heninger, editors, *FC 2020*, volume 12059 of *LNCS*, pages 541–558. Springer, Cham, February 2020.
- [LV08] M. Li and P. Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer New York, New York, NY, 2008.
- [LV20] A. Lombardi and V. Vaikuntanathan. Fiat-Shamir for Repeated Squaring with Applications to PPAD-Hardness and VDFs. In D. Micciancio and T. Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 632–651. Springer, Cham, August 2020.
- [LW15] A. K. Lenstra and B. Wesolowski. A random zoo: sloth, unicorn, and trx. Cryptology ePrint Archive, Report 2015/366, 2015.
- [MMV11] M. Mahmoody, T. Moran, and S. P. Vadhan. Time-Lock Puzzles in the Random Oracle Model. In P. Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 39–50. Springer, Berlin, Heidelberg, August 2011.
- [MMV13] M. Mahmoody, T. Moran, and S. P. Vadhan. Publicly verifiable proofs of sequential work. In R. D. Kleinberg, editor, *ITCS 2013*, pages 373–388. ACM, January 2013.
- [MSW20] M. Mahmoody, C. Smith, and D. J. Wu. Can Verifiable Delay Functions Be Based on Random Oracles? In A. Czumaj, A. Dawar, and E. Merelli, editors, *ICALP 2020*, volume 168 of *LIPICs*, 83:1–83:17. Schloss Dagstuhl, July 2020.
- [MT19] G. Malavolta and S. A. K. Thyagarajan. Homomorphic Time-Lock Puzzles and Applications. In A. Boldyreva and D. Micciancio, editors, *CRYPTO 2019, Part I*, volume 11692 of *LNCS*, pages 620–649. Springer, Cham, August 2019.

- [OKC⁺25] M. Osadnik, D. Kaviani, V. Cini, R. W. F. Lai, and G. Malavolta. Papercraft: Lattice-Based Verifiable Delay Function Implemented. In M. Blanton, W. Enck, and C. Nita-Rotaru, editors, *2025 IEEE Symposium on Security and Privacy*, pages 1603–1621. IEEE Computer Society Press, May 2025.
- [Ost91] R. Ostrovsky. One-way functions, hard on average problems, and statistical zero-knowledge proofs. In *[1991] Proceedings of the Sixth Annual Structure in Complexity Theory Conference*, pages 133–138, 1991.
- [OW93] R. Ostrovsky and A. Wigderson. One-way functions are essential for non-trivial zero-knowledge. In *[1993] The 2nd Israel Symposium on Theory and Computing Systems*, pages 3–17, 1993.
- [Pie19] K. Pietrzak. Simple Verifiable Delay Functions. In A. Blum, editor, *ITCS 2019*, volume 124, 60:1–60:15. LIPIcs, January 2019.
- [Ps05] R. Pass and a. shelat. Unconditional Characterizations of Non-interactive Zero-Knowledge. In V. Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 118–134. Springer, Berlin, Heidelberg, August 2005.
- [RSS20] L. Rotem, G. Segev, and I. Shahaf. Generic-Group Delay Functions Require Hidden-Order Groups. In A. Canteaut and Y. Ishai, editors, *EUROCRYPT 2020, Part III*, volume 12107 of *LNCS*, pages 155–180. Springer, Cham, May 2020.
- [RSW96] R. L. Rivest, A. Shamir, and D. A. Wagner. Time-lock Puzzles and Timed-release Crypto. Technical report, USA, 1996.
- [Sha19] B. Shani. A note on isogeny-based hybrid verifiable delay functions. Cryptology ePrint Archive, Report 2019/205, 2019.
- [Sol60] R. Solomonoff. *A Preliminary Report on a General Theory of Inductive Inference*. AFOSR TN-60-1459. United States Air Force, Office of Scientific Research, 1960.
- [Sta20] StarkWare Industries. Presenting: VeeDo. <https://medium.com/starkware/presenting-veedo-e4bbff77c7ae>, 2020.
- [SZ76] I. H. Sudborough and A. Zalcberg. On Families of Languages Defined by Time-Bounded Random Access Machines. *SIAM Journal on Computing*, 5(2):217–230, 1976. eprint: <https://doi.org/10.1137/0205018>.
- [TCL⁺21] S. A. K. Thyagarajan, G. Castagnos, F. Laguillaumie, and G. Malavolta. Efficient CCA Timed Commitments in Class Groups. In G. Vigna and E. Shi, editors, *ACM CCS 2021*, pages 2663–2684. ACM Press, November 2021.
- [Wee06] H. Wee. Finding Pessiland. In S. Halevi and T. Rabin, editors, *TCC 2006*, volume 3876 of *LNCS*, pages 429–442. Springer, Berlin, Heidelberg, March 2006.
- [Wes19] B. Wesolowski. Efficient Verifiable Delay Functions. In Y. Ishai and V. Rijmen, editors, *EUROCRYPT 2019, Part III*, volume 11478 of *LNCS*, pages 379–407. Springer, Cham, May 2019.
- [Win84] R. S. Winternitz. A Secure One-Way Hash Function Built from DES. In *1984 IEEE Symposium on Security and Privacy*, pages 88–88, 1984.