Abstract. Verifiable delay functions (VDF) are functions that take a specified number of sequential steps to be evaluated but can be verified efficiently. In this paper, we show that every VDF is provable in PSPACE but every language in PSPACE does not admit to a VDF.

Keywords: Verifiable delay functions · Sequentiality · Turing machine · Space-time hierarchy

1 Introduction

In 1992, Dwork and Naor introduced the very first notion of VDF under a different nomenclature “pricing function” [5]. It is a computationally hard puzzle that needs to be solved to send a mail, whereas the solution of the puzzle can be verified efficiently. Later, the concept of verifiable delay functions was formalized in [4].

Given the security parameter $\lambda$ and delay parameter $T$, the prover needs to evaluate the VDF in time $T$. The verifier verifies the output in $\text{poly}(\lambda, \log T)$-time using some proofs produced by the prover. A crucial property of VDFs, namely sequentiality, ensures that the output can not be computed in time much less than $T$ even in the presence of $\text{poly}(\lambda, T)$-parallelism. VDFs have several applications ranging from non-interactive time-stamping to resource-efficient blockchains, however, are really rare in practice because of the criteria sequentiality. In order to design new VDFs we must find problems that offer sequentiality. To the best of our knowledge so far, all the practical VDFs are based on two inherently sequential algebraic problems – modular exponentiation in groups of unknown order [13,16] (fundamentally known as the time-lock puzzle [14]) and isogenies over super-singular curves [7]. The security proofs of these VDFs are essentially polynomial-time reductions from one of these assumptions to the corresponding VDFs. Thus from the perspective of designers the first hurdle is to find inherently sequential problems.

The main motivation behind this study has been where should we search for such inherently sequential problems in order to design new VDFs? Surprisingly, in this paper, we show that there exist a sequential
algorithm to solve each problem in \textbf{PSPACE}. But we can not derive a VDF out of each of this problem. In fact, quite similarly, breaking a strong belief, Mahmoody et al. shows that the sequence of recursive responses of a random oracle is sequential, but no perfectly sound VDF can be designed using such a sequence only [12].

1.1 Proof Sketch

We show that the class of all VDFs, \textbf{VDF} \subseteq \textbf{PSPACE} in two parts. In order to proof the inclusion \textbf{VDF} \subseteq \textbf{PSPACE} first we model VDFs as a special case of interactive proofs, thus \textbf{VDF} \subseteq \textbf{IP}. Therefore \textbf{VDF} \subseteq \textbf{PSPACE} by the virtue of the seminal result by Shamir \textbf{IP} = \textbf{PSPACE} [15]. We show the opposite exclusion \textbf{PSPACE} \not\subseteq \textbf{VDF} by two incorrect attempts to build VDFs from two different \textbf{PSPACE}-complete languages. The most important finding is that all problems in \textbf{PSPACE} do not turn out to be subexponentially sequential and computationally sound, at the same time.

We consider True-Quantified-Boolean-Formula (TQBF in short) and \textbf{SPACEHALT} as these \textbf{PSPACE}-complete problems. The language TQBF is the set of fully quantified Boolean formula that are true. The sumcheck protocol for TQBF is known to be sound against even computationally unbounded provers. We show that despite its soundness sumcheck protocol fails to achieve subexponential sequentiality which is necessary for any VDF.

The language \textbf{SPACEHALT} is the set of all the tuples \((M, x, 1^S)\) such that the deterministic Turing machine \(M\) halts on input \(x\) in space \(S\). It is a \textbf{PSPACE}-complete language as any language \(L \in \textbf{PSPACE}\) can be reduced to \textbf{SPACEHALT} in polynomial-time. For any \(x \in L\), the reduction \(L \leq_p \text{SPACEHALT}\) is nothing but \(f(x) = (M, x, 1^O(|x|))\). Moreover, \textbf{SPACEHALT} is an inherently sequential language in a sense that \textbf{SPACEHALT} can not be parallelized. If it could be then all the languages in \textbf{PSPACE} could be parallelized by the definition of completeness of a language for a class. But we already know the existence of inherently sequential languages (e.g., time-lock puzzle) that can be recognized in polynomial space.

We prove that VDF derived from \textbf{SPACEHALT} may be subexponentially sequential but not computationally sound.

2 Related Work

In this section, we mention some well-known schemes qualified as VDFs.
The pricing function by Dwork–Naor scheme [5] asks a prover, given a prime \( p \equiv 3 \pmod{4} \) and a quadratic residue \( x \mod p \), to find a \( y \) such that \( y^2 \equiv x \pmod{p} \). The prover has no other choice other than using the identity \( y \equiv x^{(p+1)/4} \pmod{p} \), but the verifier verifies the correctness using \( y^2 \equiv x \pmod{p} \). Evidently, it is difficult to generate difficult instances of this VDF without using larger primes \( p \). Further, the massive parallelism with the prover violates its sequentiality.

In 2018, Dan et al. [4] propose a VDF based on injective rational maps of degree \( T \), where the fastest possible inversion is to compute the polynomial GCD of degree-\( T \) polynomials. They conjecture that it achieves \( (T^2, o(T)) \) sequentiality using permutation polynomials as the candidate map. However, it is a weak form of VDF as the prover needs \( O(T) \)-parallelism in order to evaluate the VDF in time \( T \).

Rivest, Shamir, and Wagner [14] introduced another discipline of VDFs known as time-lock puzzle. These puzzles enable an encryption that can be decrypted only sequentially. Starting with \( N = pq \) such that \( p, q \) are large primes, the key \( y \) is enumerated as \( y \equiv x^{2T} \pmod{N} \). Then the verifier uses the value of \( \phi(N) \) to reduce the exponent to \( e \equiv 2T \pmod{\phi(N)} \) and finds out \( y \equiv x^e \pmod{N} \). On the contrary, without the knowledge of \( \phi(N) \), the only option available to the prover is to raise \( x \) to the power \( 2T \) sequentially. As the verification stands upon a secret, the knowledge of \( \phi(N) \), it is not a VDF as verification should depend only on public parameters.

Wesolowski [16] and Pietrzak [13] circumvent this issue independently. The first one asks the prover to compute an output \( y = x^{2T} \) and a proof \( \pi = x^{2T/l} \), where \( l \) is a \( 2\lambda \)-bit prime chosen at random. The verifier checks if \( y = \pi^l \cdot x^{2T \pmod{l}} \). Hence the verification needs at most \( 2 \log l = 4\lambda \) squaring. Two candidate groups suit well in this scheme—an RSA group \((\mathbb{Z}/N\mathbb{Z})^*\), and the class group of an imaginary quadratic number field. This VDF shines for its short proof which is a single element in underlying group.

Pietrzak’s VDF exploits the identity \( z^r y = (x^r z)^{2T/2} \) where \( y = x^{2T} \), \( z = x^{2T/2} \) and \( r \in \{1, \ldots, 2\lambda\} \) is chosen at random. So the prover is asked to compute the proof \( \pi = \{u_1, u_2, \ldots, u_{\log T}\} \) such that \( u_i = x^{ri+2T/2^i} \). The verifier computes the \( v_i = x^{ri+2T/2^i+2T} \) and checks if \( v_i = u_i^2 \). So the verifier needs \( O(\log T) \) time. Trading-off the size of the proof it optimizes the effort to generate the proof \( \pi \) in \( O(\sqrt{T} \log T) \). As a comparison, Wesolowski’s VDF needs \( O(T/\log T) \) time to do the same. This VDF
uses the RSA group and the class groups of imaginary quadratic number fields.

Feo et al. [7] presents two VDFs based on isogenies of super-singular elliptic curves. They start with five groups \( \langle G_1, G_2, G_3, G_4, G_5 \rangle \) of prime order \( N \) with two non-degenerate bilinear pairing maps \( e_{12} : G_1 \times G_2 \to G_5 \) and \( e_{34} : G_3 \times G_4 \to G_5 \). Also there are two group isomorphisms \( \phi : G_1 \to G_3 \) and \( \phi : G_4 \to G_2 \). Given all the above descriptions as the public parameters along with a generator \( P \in G_1 \), the prover needs to find \( \phi(Q) \), where \( Q \in G_4 \), using \( T \) sequential steps. The verifier checks if \( e_{12}(P, \phi(Q)) = e_{34}(\phi(P), Q) \) in \( \text{poly}(\log T) \) time. It runs on supersingular curves over \( \mathbb{F}_p \) and \( \mathbb{F}_{p^2} \) as two candidate groups. While being inherently non-interactive, a major drawback as mentioned by the authors themselves is that its setup may be as slow as the evaluation.

Mahmoody et al. [12] have recently ruled out the possibility of having perfectly unique VDFs using random oracles only.

Table 1. Comparison among the existing VDFs. \( T \) is the targeted time bound, \( \lambda \) is the security parameter, \( \Delta \) is the number of processors. All the quantities may be subject to \( O \)-notation, if needed.

| VDF (by authors) | Eval Sequential | Eval Parallel | Verify | Setup | Proof size |
|------------------|----------------|--------------|--------|-------|------------|
| Dwork and Naor [5] | \( T \) | \( T^{2/3} \) | \( T^{2/3} \) | \( T \) | – |
| Dan et al. [4] | \( T^2 \) | \( T - o(T) \) | \( \log T \) | \( \log T \) | – |
| Wesolowski [16] | \( (1 + \frac{1}{\log T})T \) | \( (1 + \frac{2}{\Delta \log T})T \) | \( \lambda^4 \) | \( \lambda^3 \) | \( \lambda^3 \) |
| Pietrzak [13] | \( (1 + \frac{1}{\sqrt{T}})T \) | \( (1 + \frac{1}{\sqrt{T}})T \) | \( \log T \) | \( \lambda^4 \) | \( \log T \) |
| Feo et al. [7] | \( T \) | \( T \) | \( \lambda^3 \) | \( T \log \lambda \) | – |

3 Preliminaries

We start with the notations.

3.1 Notations

We denote the security parameter with \( \lambda \in \mathbb{Z}^+ \). The term \( \text{poly}(\lambda) \) refers to some polynomial of \( \lambda \), and \( \text{negl}(\lambda) \) represents some function \( \lambda^{-\omega(1)} \). If any randomized algorithm \( \mathcal{A} \) outputs \( y \) on an input \( x \), we write \( y \leftarrow^R \mathcal{A}(x) \).

By \( x \leftarrow^S \mathcal{X} \), we mean that \( x \) is sampled uniformly at random from \( \mathcal{X} \). For a string \( x \), \( |x| \) denotes the bit-length of \( x \), whereas for any set \( \mathcal{X} \), \( |\mathcal{X}| \)
denotes the cardinality of the set $\mathcal{X}$. If $x$ is a string then $x[i \ldots j]$ denotes the substring starting from the literal $x[i]$ ending at the literal $x[j]$. We consider an algorithm $A$ as efficient if it runs in probabilistic polynomial time (PPT).

### 3.2 Verifiable Delay Function

We borrow this formalization from [4].

**Definition 1. (Verifiable Delay Function).** A VDF $V = (\text{Setup, Eval, Verify})$ that implements a function $\mathcal{X} \rightarrow \mathcal{Y}$ is specified by three algorithms.

- **Setup($1^\lambda, T$) → pp** is a randomized algorithm that takes as input a security parameter $\lambda$ and a targeted time bound $T$, and produces the public parameters $pp$. We require $\text{Setup}$ to run in $\text{poly}(\lambda, \log T)$ time.

- **Eval($pp, x$) → $(y, \pi)$** takes an input $x \in \mathcal{X}$, and produces an output $y \in \mathcal{Y}$ and a (possibly empty) proof $\pi$. $\text{Eval}$ may use random bits to generate the proof $\pi$. For all $pp$ generated by $\text{Setup}(\lambda, T)$ and all $x \in \mathcal{X}$, the algorithm $\text{Eval}(pp, x)$ must run in time $T$.

- **Verify($pp, x, y, \pi$) → $\{0, 1\}$** is a deterministic algorithm that takes an input $x \in \mathcal{X}$, an output $y \in \mathcal{Y}$, and a proof $\pi$ (if any), and either accepts (1) or rejects (0). The algorithm must run in $\text{poly}(\lambda, \log T)$ time.

Before we proceed to the security of VDFs we need the precise model of parallel adversaries [4].

**Definition 2. (Parallel Adversary)** A parallel adversary $A = (A_0, A_1)$ is a pair of non-uniform randomized algorithms $A_0$ with total running time $\text{poly}(\lambda, T)$, and $A_1$ which runs in parallel time $\sigma(T) < T - o(T)$ on at most $\text{poly}(\lambda, T)$ number of processors.

Here, $A_0$ is a preprocessing algorithm that precomputes some state based only on the public parameters, and $A_1$ exploits this additional knowledge to solve in parallel running time $\sigma$ on $\text{poly}(\lambda, T)$ processors.

The three desirable properties of a VDF are now introduced.

**Definition 3. (Correctness)** A VDF is correct with some error probability $\varepsilon$, if for all $\lambda, T$, parameters $pp$, and $x \in \mathcal{X}$, we have

$$\Pr \left[ \text{Verify}(pp, x, y, \pi) = 1 \mid pp \leftarrow \text{Setup}(1^\lambda, T) \atop x \overset{\$}{\leftarrow} \mathcal{X} \atop (y, \pi) \leftarrow \text{Eval}(pp, x) \right] = 1 - \text{negl}(\lambda).$$
**Definition 4. (Soundness)*** A VDF is sound if for all non-uniform algorithms $A$ that run in time $\text{poly}(T, \lambda)$, we have

$$\Pr \left[ y \neq \text{Eval}(pp, x) \right. \left. \text{Verify}(pp, x, y, \pi) = 1 \bigg| pp \leftarrow \text{Setup}(1^\lambda, T) \right. \left. (x, y, \pi) \leftarrow A(1^\lambda, T, pp) \right] \leq \text{negl}(\lambda).$$

We call the VDF *perfectly* sound if this probability is 0.

**Definition 5. (Sequentiality)*** A VDF is $(\Delta, \sigma)$-sequential if there exists no pair of randomized algorithms $A_0$ with total running time $\text{poly}(T, \lambda)$ and $A_1$ which runs in parallel time $\sigma$ on at most $\Delta$ processors, such that

$$\Pr \left[ y = \text{Eval}(pp, x) \right. \left. \text{state} \leftarrow A_0(1^\lambda, T, pp) \right. \left. x \leftarrow \mathcal{X} \right. \left. y \leftarrow A_1(\text{state}, x) \right] \leq \text{negl}(\lambda).$$

We reiterate an important result from [4] but as a lemma.

**Lemma 1.** ($T \in \text{SUBEXP}(\lambda)$). If $T > 2^{o(\lambda)}$ then there exists an adversary that breaks the sequentiality of the VDF with non-negligible advantage.

**Proof.** $A$ observes that the algorithm $\text{Verify}$ is efficient. So given a statement $x \in \mathcal{X}$, $A$ chooses an arbitrary $y \in \mathcal{Y}$ as the output without running $\text{Eval}(x, pp, T)$. Now, $A$ finds the proof $\pi$ by a brute-force search in the entire solution space with its $\text{poly}(T)$ number of processors. In each of its processors, $A$ checks if $\text{Verify}(x, pp, T, y, \pi_i) = 1$ with different $\pi_i$. The advantage of $A$ is $\text{poly}(T)/2^{\Omega(\lambda)} \geq \text{negl}(\lambda)$ as $T > 2^{o(\lambda)}$. □

So we need $T \leq 2^{o(\lambda)}$ to restrict the advantage of $A$ upto $2^{o(\lambda)}/2^{\Omega(\lambda)} = 2^{-\Omega(\lambda)}$.

### 3.3 The Complexity Classes

We start with the definition of Turing machine in order to discuss complexity classes. We consider Turing machines with a read-only input tape and read-write work tape.

**Definition 6. (Turing machine).** A Turing Machine is a tuple $\text{TM} = \langle Q, \Gamma, q_0, F, \delta \rangle$ with the following meaning,

1. $Q$ is the finite and nonempty set of states.
2. $\Gamma$ is the finite and non-empty set of tape alphabet symbols including the input alphabet $\Sigma$.

3. $q_0 \in Q$ is the initial state.

4. $F \subseteq Q$ is the set of halting states.

5. $\delta: \{Q \setminus F\} \times \Gamma \rightarrow Q \times \Gamma \times D$ is the transition functions where $D = \{-1, 0, +1\}$ is the set of directions along the tape.

Throughout the paper we assume that the initial state $q_0$, one of the final states $q_F$, and the tape alphabet $\Gamma = \{0, 1, \vdash\}$ are implicit to the description of a TM. Here $\Sigma = \{0, 1\}$ and $\vdash$ marks the left-end of the tape. Thus $\langle Q, F, \delta \rangle$ suffices to describe any TM.

**Definition 7. (Configuration).** A configuration of a TM is a triple $(q, z, n)$ where, at present,

1. $q \in Q$ is the state of TM.

2. $z \in \Gamma^*$ is the content of the tape.

3. $n \in \mathbb{Z}$ is the position of the head at the tape.

$(q_0, x, 0)$ denotes the starting configuration for an input string $x$ instead of $(q_0, \vdash x, 0)$ (w.l.o.g.).

**Definition 8. ($\tau$-th Configuration $\xrightarrow{\tau}$).** The relation $\xrightarrow{\tau}$ is defined as,

1. $(q, z, n) \xrightarrow{\tau} (q', z', n + d)$ where $z' = \ldots z[n-1]|b|z[n+1]\ldots$ if $\delta(q, z[n]) = (q', b, d)$.

2. $\alpha \xrightarrow{\tau+1} \beta$ if there exists a $\beta$ such that $\alpha \xrightarrow{\tau} \beta \xrightarrow{1} \gamma$.

We denote $\alpha \xrightarrow{0} \alpha$. and $\alpha \xrightarrow{\tau} \beta$ if $\alpha \xrightarrow{\tau} \beta$ for some $\tau \geq 0$. We say that the TM halts on a string $x$ with the output $y$ if $(q_0, x, 0) \xrightarrow{\ast} (q_F, y, n)$ such that $q_F \in F$.

**Definition 9. (Time and Space Complexity).** We say that a TM computes a function $f: \Sigma^* \rightarrow \Sigma^*$ in time $\tau$ and space $\sigma$ if $\forall x \in \Sigma^*$, $(q_0, x, 0) \xrightarrow{\tau} (q_F, f(x), n)$ using (at most) $\sigma$ different cells on the working tape (excluding the input tape).

We call a language $L$ is decidable by a TM if and only if there exists a TM that accepts all the strings belong to $L$ and rejects all the strings belong to $\overline{L} = \Sigma^* \setminus L$. We say that a language $L$ is reducible to another language $L'$ if and only if there exists a function $f$ such that $f(x) \in L'$ if and only if $x \in L$. If the function $f$ is computable in $\text{poly}(|x|)$-time then we call it as a polynomial time reduction $L \leq_p L'$.

In order to discuss the complexity classes we follow the definitions provided in [1].
Definition 10. (DSPACE). Suppose \( f : \mathbb{N} \to \mathbb{N} \) be some function. A language \( \mathcal{L} \) is in \( \text{DSPACE}[f(n)] \) if and only if there is a TM that decides \( \mathcal{L} \) in space \( O(f(n)) \).

Definition 11. (The Class PSPACE).

\[
\text{PSPACE} = \text{DSPACE}[\text{poly}(n)].
\]

Definition 12. (PSPACE-complete). A language is PSPACE-complete if it is in PSPACE and every language in PSPACE is reducible to it in polynomial time.

3.4 Interactive Proof System

Goldwasser et al. were the first to show that the interactions between the prover and randomized verifier recognizes class of languages larger than \( \text{NP} \) [10]. They named the class as \( \text{IP} \) and the model of interactions as the interactive proof system. Babai and Moran introduced the same notion of interactions in the name of Arthur-Merlin games however with a restriction on the verifiers’ side [2]. Later, Goldwasser and Sipser proved that both the models are equivalent [11]. Two important works in this context that motivate our present study are by the Shamir showing that \( \text{IP} = \text{PSPACE} \) [15] and by the Goldwasser et al. proving that \( \text{PSPACE} = \text{ZK} \), the set of all zero-knowledge protocols. We summarize the interactive proof system from [3].

An interactive proof system \((\mathcal{P} \leftrightarrow \mathcal{V})\) consists of a pair of TMs, \(\mathcal{P}\) and \(\mathcal{V}\), with common alphabet \(\Sigma = \{0, 1\}\). \(\mathcal{P}\) and \(\mathcal{V}\) each have distinguished initial and quiescent states. \(\mathcal{V}\) has distinguished halting states out of which there is no transitions. \(\mathcal{P}\) and \(\mathcal{V}\) have a common communication tape.

i. \(\mathcal{P}\) and \(\mathcal{V}\) have a common read-only input tape.

ii. \(\mathcal{P}\) and \(\mathcal{V}\) each have a private random tape and a private work tape.

iii. \(\mathcal{P}\) and \(\mathcal{V}\) have a common communication tape.

iv. \(\mathcal{V}\) is polynomially time-bounded. This means \(\mathcal{V}\) halts on input \(x\) in time \(\text{poly}(|x|)\). \(\mathcal{V}\) is in quiescent state when \(\mathcal{P}\) is running.

v. \(\mathcal{P}\) is computationally unbounded but runs in finite time. This means \(\mathcal{P}\) may compute any arbitrary function \(\{0, 1\}^* \to \{0, 1\}^*\) on input \(x\) in time \(f(|x|)\). Feldman proved that “the optimum prover lives in \(\text{PSPACE}\)” \(^1\).

\(^1\)We could not find a valid citation.
vi. The length of the messages written by $P$ into the common communication tape is bounded by $\text{poly}(|x|)$. Since $V$ runs in $\text{poly}(|x|)$ time, it can not write messages longer than $\text{poly}(|x|)$.

Execution begins with $P$ in its quiescent state and $V$ in its start state. $V$'s entering its quiescent state arouses $P$, causing it to transition to its start state. Likewise, $P$’s entering its quiescent state causes $V$ to transition to its start state. Execution terminate when $V$ enters in its halting states. Thus $(P \leftrightarrow V)(x) = 1$ denotes $V$ accepts $x$ and $(P \leftrightarrow V)(x) = 0$ denotes $V$ rejects $x$.

**Definition 13. (Interactive Proof System $(P \leftrightarrow V)$ ).** $(P \leftrightarrow V)$ is an interactive proof system for the language $L \subseteq \{0, 1\}^*$ if

\begin{align*}
\text{(Correctness). } (x \in L) &\implies \Pr[(P \leftrightarrow V)(x)] = 1 \geq 1 - \text{negl}(|x|). \\
\text{(Soundness). } (x \notin L) &\implies \forall P', \Pr[(P' \leftrightarrow V)(x)] = 1 < \text{negl}(|x|).
\end{align*}

The class of interactive polynomial-time $\text{IP}$ is defined as the class of the languages that have an interactive proof system. Thus

$$\text{IP} = \{L | L \text{ has an } (P \leftrightarrow V)\}.$$  

Alternatively and more specifically,

**Definition 14. (The Class $\text{IP}$).**

$$\text{IP} = \text{IP}[\text{poly}(n)].$$

For every $k$, $\text{IP}[k]$ is the set of languages $L$ such that there exist a probabilistic polynomial time TM $V$ that can have a $k$-round interaction with a prover $P : \{0, 1\}^* \rightarrow \{0, 1\}^*$ having these two following properties

\begin{align*}
\text{(Correctness). } (x \in L) &\implies \Pr[(P \leftrightarrow V)(x)] = 1 \geq 1 - \text{negl}(|x|). \\
\text{(Soundness). } (x \notin L) &\implies \forall P', \Pr[(P' \leftrightarrow V)(x)] = 1 < \text{negl}(|x|).
\end{align*}

### 4 Fiat–Shamir Transformation

Any interactive protocol $L \in \text{IP}$ can be transformed into a non-interactive protocol if the messages from the verifier $V$ are replaced with the response of a random oracle $H$. This is known as Fiat–Shamir transformation (FS) [8]. In particular, the $i$-th message from $V$ is computed as $y_i := H(x, x_1, y_1, \ldots, x_i, y_{i-1})$ where $x_i$ denotes the $i$-th response of $P$. When $H$ is specified in the public parameters of a $k$-round protocol, the
transcript $x, x_1, y_1, \ldots, x_k, y_{k-1}$ can be verified publicly. Thus, relative to a random oracle $H$, a $k$-round interactive proof protocol $(P \leftrightarrow V)$ can be transformed into a two-round non-interactive argument $(P_{FS} \leftrightarrow V_{FS})$ where $P_{FS}$ sends the entire transcript $x, x_1, y_1, \ldots, x_k, y_k$ to $V_{FS}$ in a single round. Under the assumption that $H$ is one-way and collision-resistant, $V_{FS}$ accepts $x \in \mathcal{L}$ in the next round if and only if $V$ accepts. Here we summarize two claims on Fiat–Shamir transformation stated in [6].

**Lemma 2.** If there exists an adversary $A$ who breaks the soundness of the non-interactive protocol $(P_{FS} \leftrightarrow V_{FS})$ with the probability $p$ using $q$ queries to a random oracle then there exists another adversary $A'$ who breaks the soundness of the $k$-round interactive protocol $(P \leftrightarrow V)$ with the probability $p/q^k$.

**Proof.** See [9] for details. □

**Lemma 3.** Against all non-uniform probabilistic polynomial-time adversaries, if a $k$-round interactive protocol $(P \leftrightarrow V)$ achieves $\text{negl}(|x|^k)$-soundness then the non-interactive protocol $(P_{FS} \leftrightarrow V_{FS})$ has $\text{negl}(|x|)$-soundness.

**Proof.** Since, all the adversaries run in probabilistic polynomial time, the number of queries $q$ to the random oracle must be upper-bounded by $\text{poly}(|x|)$. Putting $q = |x|^c$ for any $c \in \mathbb{Z}^+$ in lemma. 2, it follows the claim. □

## 5 VDF Characterization

In this section, we investigate the possibility to model VDFs as a language in order to define its hardness. It seems that there are two hurdles,

**Eliminating Fiat–Shamir** The prover $P$ in Def. 18, generates the proof $\pi := f(x, y, T, H(x, y, T))$ using Fiat–Shamir transformation where $y := \text{Eval}(x, pp, T)$. Unless Fiat–Shamir is eliminated from VDF, its hardness remains relative to the random oracle $H$. Sect. 5.1 resolves this issue.

**Modelling Parallel Adversary** How to model the parallel adversary $A$ (Def. 2) in terms of computational complexity theory? We model $A$ as a special variant of Turing machines described in Def. 16.

We address the first issue now.
5.1 Interactive VDFs

We introduce the interactive VDFs in order to eliminate the Fiat–Shamir. In the interactive version of a VDF, the \( V \) replaces the randomness of Fiat–Shamir heuristic. In particular, a non-interactive VDF with the Fiat–Shamir transcript \( (x, x_1, y_1, \ldots, x_k, y_k) \) can be translated into an equivalent \( k \)-round interactive VDF allowing \( V \) to choose \( y_i \)'s in each round.

**Definition 15. (Interactive Verifiable Delay Function).** An interactive verifiable delay function is a tuple \((\text{Setup}, \text{Eval}, \text{Open}, \text{Verify})\) that implements a function \( \mathcal{X} \rightarrow \mathcal{Y} \) as follows,

- **Setup**\((1^\lambda, T) \rightarrow pp\) is a randomized algorithm that takes as input a security parameter \( \lambda \) and a delay parameter \( T \), and produces the public parameters \( pp \) in \( \text{poly}(\lambda, \log T) \) time.
- **Eval**\((pp, x) \rightarrow y\) takes an input \( x \in \mathcal{X} \), and produces an output \( y \in \mathcal{Y} \). For all \( pp \) generated by \( \text{Setup}(\lambda, T) \) and all \( x \in \mathcal{X} \), the algorithm \( \text{Eval}(pp, x) \) must run in time \( T \).
- **Open**\((x, y, pp, T, t) \rightarrow \pi\) takes the challenge \( t \) chosen by \( V \) and computes a proof \( \pi \) (possibly recursively) in \( \text{poly}(\lambda) \) rounds of interaction with \( V \). In general, for some \( k \in \text{poly}(\lambda) \), \( \pi = \{\pi_1, \ldots, \pi_k\} \) can be computed as \( \pi_{i+1} := \text{Open}(x_i, y_i, pp, T, t_i) \) where \( x_i \) and \( y_i \) depend on \( \pi_i \). Observing \( (x_i, y_i, \pi_i) \) in the \( i \)-th round, \( V \) chooses the challenge \( t_i \) for the \((i + 1)\)-th round. Hence, \( \text{Open} \) runs for \( k \)-rounds.
- **Verify**\((pp, x, y, \pi) \rightarrow \{0, 1\}\) is a deterministic algorithm that takes an input \( x \in \mathcal{X} \), an output \( y \in \mathcal{Y} \), and the proof vector \( \pi \) (if any), and either accepts (1) or rejects (0). The algorithm must run in \( \text{poly}(\lambda, \log T) \) time.

All the three security properties remain same for the interactive VDF. Sequentiality is preserved by the fact that \( \text{Open} \) runs after the computation of \( y := \text{Eval}(x, pp, T) \). For soundness, we rely on lemma. 3. The correctness of interactive VDF's implies the correctness of the non-interactive version as the randomness that determines the proof is not in the control of \( P \). Therefore, an honest prover always convinces \( V \).

Although the interactive VDFs do not make much sense as publicly verifiable proofs in decentralized distributed networks, it allow us to analyze its hardness irrespective of any random oracle.

In order to model parallel adversary, we consider a well-known variant of Turing machine that suits the context of parallelism. We describe the variant namely parallel Turing machine as briefly as possible from (Sect. 2 in cf.[17])

11
5.2 Parallel Turing Machine

Intuitively, a parallel Turing machine has multiple control units (CU) (working collaboratively) with a single head associated with each of them working on a common read-only input tape [17], and a common read-write work tape.

Definition 16. (Parallel Turing Machine). A parallel Turing machine is a tuple $\text{PTM} = (Q, \Gamma, \Sigma, q_0, F, \delta)$ where

1. $Q$ is the finite and nonempty set of states.
2. $\Gamma$ is the finite and non-empty set of tape alphabet symbols including the input alphabet $\Sigma$.
3. $q_0 \in Q$ is the initial state.
4. $F \subseteq Q$ is the set of halting states.
5. $\delta : 2^Q \times \Gamma \rightarrow 2^{Q \times D} \times \Gamma$ where $D = \{-1, 0, +1\}$ is the set of directions along the tape.

A configuration of a PTM is a pair $c = (p, b)$ of mappings $p : \mathbb{Z}^+ \rightarrow 2^Q$ and $b : \mathbb{Z}^+ \rightarrow \Gamma$. The mapping $p(i)$ denotes the set of states of the CUs currently pointing to the $i$-th cell in the input tape and $b(i)$ is the symbol written on it. So it is impossible for two different CUs pointing to the same cell $i$ while staying at the same state simultaneously. During transitions $c' = (M'_i, b'(i)) = \delta(c) = \delta(p(i), b(i))$, the set of CUs may be replaced by a new set of CUs $M'_i \subseteq Q \times D$. The $p'(i)$ in the configuration $c'$ is defined as $p'(i) = \{q \mid (q, +1) \in M'_{i-1} \lor (q, 0) \in M'_i \lor (q, -1) \in M'_{i+1}\}$.

Without loss of generality, the cell 1 is observed in order to find the halting condition of PTM. We say that a PTM halts on a string if and only if $p(1) \subseteq F$ after some finite time. The notion of decidability by a PTM is exactly same as in TM. We denote $\text{PTM}(s, t, h)$ as the family of all languages for which there is a PTM recognizing them using space $s$, time $t$ and $h$ processors. Thus languages decidable by a TM is basically decidable by a PTM$(s, t, 1)$. Assuming $\text{TM}(s, t)$ is the set of languages recognized by a TM in space $s$ and time $t$, we mention Theorem 15 from (cf. [17]) without the proof.

We observe that the parallel adversary $\mathcal{A}$ defined in Def. 2 is essentially a PTM having $\text{poly}(\lambda, T)$ processors running on $\text{poly}(\lambda, T)$ space in time $\sigma(T)$. We will refer such a PTM with $\text{poly}(\lambda, T)$-PTM (w.l.o.g.) in our subsequent discussions.

5.3 VDF As A Language

Now we characterize VDFs in terms of computational complexity theory. We observe that, much like $(\mathcal{P} \leftrightarrow \mathcal{V})$, VDFs are also proof system for the
languages,
\[ L = \left\{ (x, y, T) \mid \begin{array}{l}
\text{pp} \leftarrow \text{Setup}(1^\lambda, T) \\
x \in \{0, 1\}^\lambda \\
y \leftarrow \text{Eval}(\text{pp}, x)
\end{array} \right\}. \]

\( P \) tries to convince \( V \) that the tuple \((x, y, T) \in L\) in polynomially many rounds of interactions. In fact, Pietrzak represents his VDF using such a language (Sect. 4.2 in cf. [13]) where it needs \( \log T \) (i.e., \( \text{poly}(\lambda) \)) rounds of interaction. However, by design, the VDF is non-interactive. It uses Fiat–Shamir transformation.

Thus, a VDF closely resembles an \((P \leftrightarrow V)\) except on the fact that it stands sequential (see Def. 5) even against an adversary (including \( P \)) possessing subexponential parallelism. Notice that a \( \text{poly}(\lambda, T) \)-PTM (see Def. 16) precisely models the parallel adversary described in Def. 2. In case of interactive proof systems, we never talk about the running time of \( P \) except its finiteness. On the contrary, \( P \) of a VDF must run for at least \( T \) time in order to satisfy its sequentiality. Hence, we define VDF as follows,

**Definition 17. (Verifiable Delay Function \((P \leftrightarrow V)\)).** For every \( \lambda \in \mathbb{Z}^+ \), \( T \in 2^{o(\lambda)} \) and for all \( s = (x, y, T) \in \{0, 1\}^{2\lambda+\lceil\log T\rceil} \), \( \langle P \leftrightarrow V \rangle \) is a verifiable delay function for a language \( L \subseteq \{0, 1\}^* \) if

\[
\begin{align*}
\text{(Correctness).} & \quad (s \in L) \implies \Pr[\langle P \leftrightarrow V \rangle(s) = 1] \geq 1 - \text{negl}(\lambda). \\
\text{(Soundness).} & \quad (s \notin L) \implies \forall A, \Pr[\langle A \leftrightarrow V \rangle(s) = 1] \leq \text{negl}(\lambda). \\
\text{(Sequentiality).} & \quad (s \in L) \implies \forall B, \Pr[\langle B \leftrightarrow V \rangle(s) = 1] \leq \text{negl}(\lambda).
\end{align*}
\]

where,

i. \( P : \{0, 1\}^* \rightarrow \{0, 1\}^* \) is a TM that runs in time \( \geq T \),
ii. \( A : \{0, 1\}^* \rightarrow \{0, 1\}^* \) is a TM that runs in time \( \text{poly}(\lambda, T) \),
iii. \( B \) is a \( \text{poly}(\lambda, T) \)-PTM (see Def. 16) that runs in time < \( T \).

Further we define the class of all verifiable delay functions as,

**Definition 18. (The Class VDF).**

\[ \text{VDF} = \text{VDF}[\text{poly}(\lambda)]. \]

For every \( k \in \mathbb{Z}^+ \), \( \text{VDF}[k] \) is the set of languages \( L \) such that there exists a probabilistic polynomial-time TM \( V \) that can have a \( k \)-round interaction with

i. \( P : \{0, 1\}^* \rightarrow \{0, 1\}^* \) is a TM that runs in time \( \geq T \),
ii. $A : \{0,1\}^* \rightarrow \{0,1\}^*$ is a TM that runs in time $\text{poly}(\lambda, T)$,

iii. $B$ is a $\text{poly}(\lambda, T)$-PTM (see Def. 16) that runs in time $< T$.

satisfying these three following properties,

(Correctness). $(s \in L) \implies \Pr[\langle P \leftrightarrow V \rangle(s) = 1] \geq 1 - \text{negl}(\lambda)$.

(Soundness). $(s \notin L) \implies \forall A, \Pr[\langle A \leftrightarrow V \rangle(s) = 1] \leq \text{negl}(\lambda)$.

(Sequentiality). $(s \in L) \implies \forall B, \Pr[\langle B \leftrightarrow V \rangle(s) = 1] \leq \text{negl}(\lambda)$.

6 PSPACE-hardness

Although the definitions of IP and VDF appear quite similar, the key differences are,

1. IP does not demand for sequentiality. In fact, one of the most elegant IP, the sumcheck protocol for UNSAT is known to be parallelizable in nature. The sumcheck protocol asks $P$ to compute the sum $\sum_{z \in \{0,1\}^n} f(z) = y$ of polynomial $f$ of small degree. Therefore, $y$ can be computed in $2^{O(n)}$-time sequentially but in $O(2^n / \Gamma)$-time parallelly when $P'$ has $\Gamma$ number of parallel processors. As IP allows a malicious prover $P'$ to have unbounded computational power (so processors), sequentiality can not be achieved against $P'$.

2. IP demands for statistical soundness i.e., no prover has non-negligible advantage to convince $V$ with a false proof. On the contrary, VDF asks for computational soundness only i.e., no prover running in $\text{poly}(\lambda, T)$-time has non-negligible advantage. As we see, statistical soundness implies computational soundness.

These two observations together imply that VDFs are special kind of interactive proofs that are sequential but with a bit relaxed notion of soundness.

Therefore, the proof for VDF $\subseteq$ IP = PSPACE is straightforward.

Theorem 1. VDF $\subseteq$ PSPACE.

Proof. An honest prover $P$ needs to run for time $T$ to decide if a tuple $(x, y, T) \in L$ for all $L \in \text{VDF}$. By lemma. 1, $T$ can be at most $2^{o(\lambda)}$. By lemma. 7, a TM with $S$-space may run for $|Q|S^2$-time. Therefore, a TM with even $o(\lambda)$-space suffices to decide $L$. Hence, VDF $\subseteq$ PSPACE. $\Box$

The standard way to prove PSPACE $\subseteq$ VDF is to derive a VDF from a PSPACE-complete language [15]. Existence of such a VDF would
imply that there is an inherently sequential PSPACE-complete problem whose solution is sound also. We claim that such a PSPACE-complete problem that commits sequentiality and soundness together, hardly exists. We present two flawed VDFs in order to show this.

6.1 A Sound But Non-sequential Approach

Our goal is to check if a statistically sound interactive proof evokes a VDF. Thus, following [15], we attempt to derive a VDF from TQBF.

**Definition 19.** (True-Quantified-Boolean-Formula TQBF). Let, 
\[ \Psi = Q_1x_1, \ldots, Q_nx_n \phi(x_1, \ldots, x_n) \] 
be a quantified Boolean formula of \( n \) variables and \( m \) clauses such that all \( Q_i \in \{\exists, \forall\} \) and \( \phi \) is in 3-CNF (w.l.o.g.). The language \( \text{TQBF} \) is defined as the set of all quantified Boolean formula that are true. Formally,

\[ \text{TQBF} = \{ \Psi(x_1, \ldots, x_n) = 1 \}. \]

**Sumcheck Protocol for TQBF** We summarize the sumcheck protocol for TQBF from [15].

Given a quantified Boolean formula (QBF) \( \Psi \), first we arithmetize \( \Psi \) to obtain a polynomial \( f \) as follows,

1. \( \forall x_n \phi(x_1, \ldots, x_n) \) evaluate \( \prod_{x_n \in \{0, 1\}} f(x_1, \ldots, x_n) \).
2. \( \exists x_n \phi(x_1, \ldots, x_n) \) evaluate \( \sum_{x_n \in \{0, 1\}} f(x_1, \ldots, x_n) \).

Thus, a QBF \( \Psi = \forall x_1 \exists x_2 \ldots \forall x_n \phi(x_1, \ldots, x_n) \) \( \in \text{TQBF} \) if and only if \( h(x_1, \ldots, x_n) = \prod_{x_1 \in \{0, 1\}} \sum_{x_2 \in \{0, 1\}} \ldots \prod_{x_n \in \{0, 1\}} f(x_1, \ldots, x_n) \neq 0 \). As \( \phi \) is in 3-CNF, degree of \( f \) is \( O(n^3) \). But, due to the presence of \( \prod \) operator in \( h \), its degree and number of coefficients can be \( O(2^n) \) in the worst-case.

It is resolved with the observation that \( x^k = x \) for all \( k \geq 1 \) as \( x \in \{0, 1\} \). It allows to define a linearization operator \( L_n \) as follows,

\[ L_n f(x_1, \ldots, x_n) = x_n \cdot f(x_1, \ldots, x_{n-1}, 1) + (1 - x_n) \cdot f(x_1, \ldots, x_{n-1}, 0). \]

Thus, in order to keep the degree and size of \( h \) in \( \text{poly}(n) \), we sprinkle the linearization operators in between \( h \) as,

\[ h' = \prod_{x_1 \in \{0, 1\}} L_1 \sum_{x_2 \in \{0, 1\}} L_1 L_2 \prod_{x_3 \in \{0, 1\}} \ldots \prod_{x_n \in \{0, 1\}} L_1 L_2 \ldots L_n f(x_1, \ldots, x_n). \]

The size of \( h' \) is \( O(n^2) \) as there are exactly \( n(n + 3)/2 \) operators.
The sumcheck protocol for TQBF asks the prover $P$ to prove that $h'(x_1, \ldots, x_n) = y \neq 0 \mod p$ for a prime $p \geq 2^n 3^m$ in at most $n(n+3)/2$-rounds. In each round, $P$ strips one operator. Let us denote the operator before $x_i$ in $h'$ with $\otimes_i \in \{\prod_i, \sum_i, L_i\}$. The protocol is defined recursively as follows,

Suppose the partial sum of $h'(r_1, r_2, \ldots, r_{i-1}, x_i, \ldots, x_n) = y'$ where $r_j$ is chosen from the finite field $F_p$ uniformly at random. For all $i$,

**Case 1:** If $\otimes_i = \sum_i$ then $P$ sends a univariate polynomial $s(x_i) = h'(r_1, r_2, \ldots, r_{i-1}, x_i, \ldots, x_n)$. The verifier $V$ rejects if $s(0) + s(1) \neq y'$, otherwise asks $P$ to prove in the next round that $s(r_i) = h'(r_1, r_2, \ldots, r_{i-1}, r_i, x_{i+1}, \ldots, x_n)$ for some $r_i \in R F_p$.

**Case 2:** If $\otimes_i = \prod_i$ Exactly same as case (1) except that $V$ rejects if $s(0) \cdot s(1) \neq y'$ instead of $s(0) + s(1) \neq y'$.

**Case 3:** If $\otimes_i = L_i$ then $P$ sends a univariate polynomial $s(x_i) = h'(r_1, r_2, \ldots, r_{i-1}, x_i, \ldots, x_n)$. The verifier $V$ rejects if $r_i \cdot s(0) + (1 - r_i) \cdot s(1) \neq y'$, otherwise asks $P$ to prove in the next round that $s(r_i) = h'(r_1, r_2, \ldots, r_{i-1}, r_i, x_{i+1}, \ldots, x_n)$ for some $r_i \in R F_p$.

**Lemma 4.** For all adversaries $A$ even with computationally unbounded power the probability the $V$ accepts a false $y \neq h'(x_1, \ldots, x_n)$ is at most $\frac{3mn + n^2}{p}$.

**Proof.** For case (1) and (2), the degree of the polynomial $s(x_i)$ is 1 as $L_x$ linearize $s$. For case (3), the degree of $s(x_i)$ can be at most 2.

By Schwartz-Zippel lemma, every two distinct univariate polynomials of degree $\leq d$ over a field $F$ agree in at most $d$ points. So, the probability that $V$ accepts a wrong $y$ has this two components,

i. For the inner $L_x$, this probability is $\leq \frac{2}{p}$.

ii. For the final $L_x$, this probability is $\leq \frac{3m}{p}$.

Therefore, by the union bound, the total probability that $V$ accepts a wrong $y$ is at most,

$$\frac{n}{p} + \frac{3mn}{p} + \frac{2}{p} \sum_{i=1}^{n-1} i = \frac{3mn + n^2}{p}.$$ 

\[\square\]

**Argument Against Sequentiality** Although TQBF raises an interactive proof with statistical soundness, we are not sure if it admits a
subexpontentially long sequential computation too. The reason is that the maximum number of sequential steps required to evaluate \( h' \) in the worst-case is the number of operators in it i.e., \( n(n+3)/2 \). So, in order to setup a VDF for sequential time \( T \), it needs to sample a QBF of length \( \Omega(\sqrt{T}) \) as the public parameter. To support efficient execution in practice, cryptographic protocols should allow polynomially long public parameters only. Therefore, such a VDF may work for \( T \in \text{poly}(\lambda) \) but not for \( T \in 2^{o(\lambda)} \).

6.2 A Sequential But Unsound Approach

In this section, we present another wrong attempt to derive a VDF which turns out to be sequential but not computationally sound.

**Definition 20. (SPACEHALT).** Suppose \( M \) is a Turing machine, \( x \in \Sigma^* \) is an input string and \( S \in \mathbb{N} \). The language \( \text{SPACEHALT}^2 \) is the set of all the tuples \( (\langle M \rangle, x, 1^S) \) such that the \( \text{TM} M \) halts on input \( x \) in space \( S \). Formally,

\[
\text{SPACEHALT} = \{(\langle Q, F, \delta \rangle, x, 1^S) | (q_0, x, 0) \xrightarrow{\cdot} (q_F, y, n) \text{ in space } S\}.
\]

**Lemma 5.** \( \text{SPACEHALT} \) is \( \text{PSPACE-complete} \).

*Proof.* We show that any language \( \mathcal{L} \in \text{PSPACE} \) is reducible to \( \text{SPACEHALT} \) in polynomial-time. Suppose \( \mathcal{L} \in \text{PSPACE} = \text{DSPACE}(\text{poly}(n)) \) is decided by a \( \text{TM} M \). Then the function \( f(x) = (\langle M \rangle, x, 1^{\text{poly}(|x|)}) \) is a polynomial-time reduction from \( \mathcal{L} \) to \( \text{SPACEHALT} \). \( \square \)

**Lemma 6.** If time-lock puzzle is inherently sequential then \( \text{SPACEHALT} \) is inherently sequential.

*Proof.* We prove this by contradiction. Suppose \( (\langle M \rangle, x, T) \in \text{SPACEHALT} \) and we parallelize the simulation of \( M \) with another \( \text{TM} \hat{M} \). Then any \( \text{TM} \) that decides a language \( \mathcal{L} \in \text{PSPACE} \) must be parallelizable using the \( \text{TM} \hat{M} \). Then it means that there exists no inherently sequential language in \( \text{PSPACE} \). But we already know the existence of languages (e.g., time-lock puzzle [14]) which is sequential but can be evaluated in polynomial space. \( \square \)

**Lemma 7.** \( \text{SPACEHALT} \) is decidable in at most \( |Q|S2^S \) time.

\(^2\) See \( \text{SPACE TMSAT} \) (cf. Def. 4.9) in [1].
Proof. There are $2^S$ different strings that could appear in the work tape of $M$. The head could be in any of $S$ different places and the $M$ could be in one of $|Q|$ different states. So the total number of configurations is $|Q|S2^S$.

By the pigeonhole principle, if $M$ is run for further steps, it must visit a configuration again resulting into looping. We design a VDF from the language SPACEHALT.

**VDF From SPACEHALT** The design of this VDF is based on two fundamental observations on Turing machines.

- the running time $T$ of $M$ having $S$ space is bounded by $|Q|S2^S$ (by lemma. 7).
- $M$ continues to stay within the set of halting states (either accepting or rejecting) once it reaches at one of them.

We specify the algorithms for VDFs as follows,

**Setup** $(1^\lambda, T) \rightarrow pp$ It samples a Turing machine, $M = (Q, F, \delta)$ such that $2^{O(\lambda)} \leq |Q| \leq 2^{\text{poly}(\lambda)}$ and $|F| \leq \text{poly}(\lambda)$. Although the description of $M$ is exponentially large, it suffices to provide $\delta$ as a $\text{poly}(\lambda)$-size circuit that outputs the next state $q_{i+1}$ on an input $q_i$. Without loss of generality, we assume an encoding $Q = \{0, 1\}^{\lceil \log |Q| \rceil}$ with the implicit initial state $q_0 = 0^{\lceil \log |Q| \rceil}$.

**Eval** $(x, pp) \rightarrow (y, \pi)$ The prover $P$ computes the $T$-th state $q_T$ of $M$ on the input $x$ starting from the initial state $q_0$ and sends it to the verifier $V$. Formally, $q_T := \delta^T(q_0, x)$. Now, $V$ asks $P$ to provide another state $q_t$ from this sequence of states $q_0, \delta(q_0), \ldots, \delta^T(q_0)$ such that $(T - t) \leq \lambda$.

With $q_t$, $V$ also asks for the tape content $z$ of $M$ at time $t$.

**Verify** $(x, pp, \phi, \pi) \rightarrow \{0, 1\}$ Using Verify, $V$ checks if $y = q_T$ in $(T - t) \leq \lambda$ steps, as follows.

### 6.3 Efficiency

Here we discuss the time and the memory required by the prover and the verifier.

**Proof Size** The output $\phi = q_T$ needs $\log |Q|$-bits where $\lambda \leq \log |Q| \leq \text{poly}(\lambda)$. The proof $\pi = (q_{T-t}, z)$ requires $\log |Q| + t$-bits as $|z| = t$.

**P’s Effort** $P$ needs $T$ time to find $q_T$, then $T - t$ time to find $q_{T-t}$ and finally $t$ time to find $z$. By the Theorem 3 it requires $2T$ time in total.

**V’s Effort** $V$ needs only $t$ time to find $q_T$ from $q_{T-t}$ using $z$. Deciding $q \in F$ is already shown to be efficient.
Algorithm 1 Eval from SPACEHALT

1: \((q_0, x, 0) \xrightarrow{T} (q_T, y, n)\).
2: Obtains \(t \geq T - \lambda\) from \(V\).
3: \((q_0, x, 0) \xrightarrow{t} (q_t, y_t, n_t)\).
4: Initialize a string \(z := y_t[n_t]\).
5: for \(t \leq i \leq T\) do
6: \((q_i, y_i, n_i) \xrightarrow{1} (q_{i+1}, y_{i+1}, n_{i+1})\).
7: \(z := z \| y_{i+1}[n_{i+1}]\).
8: end for
9: \(y := q_T\).
10: \(\pi : (q, z)\)
11: return \((x, y, T, \pi)\)

Algorithm 2 Verify for SPACEHALT

1: \((q_t, z, 0) \xrightarrow{T-t} (q_T, y, n)\).
2: if \(y = q_T\) then
3: return 1;
4: else
5: return 0;
6: end if

7 Security

We claim that the derived VDF is correct and sequential but not sequential.

Theorem 2. The constructed VDF is correct.

Proof. If \(P\) has run Eval honestly \((q_0, x, 0) \xrightarrow{T} (q_T, y, n)\). The integer \(t\) is solely determined by the input statement \(x\) and the output \(q_T\). Thus \(V\) will always find \(y = q_T\) by computing \((q_t, z, 0) \xrightarrow{T-t} (q_T, y, n)\).

Theorem 3. If there is an adversary \(A\) breaking the sequentiality of this VDF with the probability \(p\) then there is a Turing machine \(A'\) breaking the sequentiality of the language SPACEHALT with the same probability.

Proof. Suppose a TM halts on input \(x\) in space \(S\) and in time \(T \leq |Q|S2^{S}\). Thus to decide if the string \((M, x, 1^S) \in\) SPACEHALT in time \(< T\), \(A'\) passes it to \(A\) that runs in time \(< T\). When \(A\) outputs \((q_T, \pi)\), \(A'\) checks if \(q_T \in F\) or not and decides the string \((M, x, 1^S)\). Observe that deciding \(q_T \in F\) is efficient as \(|F| \in \text{poly}(\lambda)\).

As \(A\) breaks the sequentiality of this VDF with the probability \(p\) so \(\text{Pr}[q_T \in F] = p\). Hence \(A'\) decides SPACEHALT in time \(< T\) with the probability \(p\) violating the sequentiality of SPACEHALT.
While everything seems perfect in the above construction of the VDF, there exists an adversary that breaks the soundness of this VDF certainly.

**Theorem 4.** There exists an adversary $A$ who breaks the soundness of the VDF derived from SPACEHALT.

**Proof.** The key observation made by $A$ is that the verifier $V$ never checks if the state $q_t$ in proof $\pi$ has correctly been computed starting from the state $q_0$ in $t$-steps. However, $A$ knows that the maximum distance from the state $q_t$ and $q_T$ is upper-bounded by $\lambda$.

Therefore, $A$ chooses an arbitrary configuration $\alpha$ of $M$. (S)he never runs Eval. Rather, by simulating $M$, (s)he computes another configuration $\alpha \xrightarrow{\lambda} \beta$ remembering the sequence of all the states and the scanned input symbols in these $\lambda$ steps. This can be done efficiently as $Q$ can be encoded using only $\text{poly}(\lambda)$-bits. Suppose, the sequence of states and the scanned symbols are denoted as $Q$ and $Z$. Observe that $|Q| = |Z| = \lambda$. Finally, $A$ announces the state in the configuration $\beta$ as the output $q_T$.

When $A$ obtains the offset $t$ from $V$, $A$ fixes the $q_t := Q[t - \lambda - 1]$ and $z := Z[t - \lambda - 1, \ldots, \lambda]$. Essentially, $A$ sets the $(t - \lambda)$-th state in $Q$ as the state $q_t$ in the proof $\pi$. Similarly, the tape content $z$ in $\pi$ is the sequence of last $(T - t)$ symbols in $Z$.

Clearly, $V$ will be convinced as $(q_t, z, 0) \xrightarrow{T-t} (q_T, z', n)$. 

\[ \square \]

**Argument Against Soundness** Theorem 4 suggests that the VDF is not computationally sound, however, is sequential for any $T \in 2^{\text{poly}(\lambda)}$. This argument can be supported with an informal claim that if such a sequence of states can be verified efficiently with soundness then a chain of hashes (i.e., $H^T(x)$) may also raise VDFs. Unfortunately, we do not know no efficient verification algorithm for hash-chains.

## 8 Frequently Asked Questions

In this section, we would like to clear some obvious confusions in our results.

Q.1) Does Lemma 6 suggest that every problem in $\text{PSPACE}$ is inherently sequential?

A.1) Yes and no both. Because our interpretation of this theorem is that we call a problem inherently sequential as long as we do not know
an efficiently parallelizable for the same. For example, the time-lock puzzle or the isogenies over the super-singular curves. Similarly, having a parallel algorithm for a problem does not really deny the chance of having another sequential algorithm for the same. Because a parallelizable language in \textit{PSPACE} also reduces to \textit{SPACEHALT}. So, we believe that every language in \textit{PSPACE} has sequential algorithm and parallelizable algorithm together.

Q.2) Does the answer A.1 suggest that problems in \textit{P} can also be used to derive VDF?

A.2) Yes. Because, the key observation is that the definition of VDF imposes an upper bound on \( T \leq 2^{o(\lambda)} \) but not the lower bound. For arbitrarily small \( T \), the computation can not be arbitrarily hard. For all \( T \leq \text{poly}(\lambda) \), the hardness can not be beyond \textit{P}. In principle, we need sequentially hard problems rather than the computationally hard problems to derive VDFs.

For example, circuit value problem (CVP) is known to be a \textit{P}-complete problem. The algorithm that solves CVP is efficient but is sequential also as we do not know faster parallel algorithm. So, VDFs derived from \textit{P}-complete problems should work for all \( T \leq \text{poly}(\lambda) \).

Q.3) The VDF derived in Sect. 6.2 uses a random oracle \( H \) in \textit{Eval} and \textit{Verify}. Does it mean that all the subsequent claims are relative to a random oracle?

A.3) The definitions of the class VDF [Def. 19] and the interactive VDFs [Def. 18] are independent of any random oracle. As the class VDF is defined to be the set of interactive VDFs, this result is not relative to any random oracle. It could have been only if the class VDF was defined to be the set of non-interactive VDFs.

The random oracle \( H \) is required only in the non-interactive version of this VDF as per the Fiat–Shamir heuristic, but not in the interactive VDF. We have explicitly mentioned how the verifier replaces the random oracle \( H \) in the interactive version in the descriptions of each algorithm \textbf{Setup}, \textbf{Eval} and \textbf{Verify}. Still the interactive version satisfies all the security proofs in Sect. 5.1 which are also independent of any random oracle.

Q.5) Do the flawed attempts to derive VDFs suggest that no problems in \textit{PSPACE} can be used to derive VDFs?

A.5) No. They suggest that all problems in \textit{PSPACE} do not have subexponentially long sequentiality and computational soundness together. Time-lock puzzle probably one among such rare problems that has both the properties. In particular, Pietrzak’s VDF [13] is known to be statistically sound and, of course, sequential. It is derived from
time-lock puzzle. Hence, these flawed attempts actually introduce the notion a subclass \textbf{VDF} full of such special problems within \textbf{PSPACE}.

References

1. Arora, S., Barak, B.: Computational Complexity - A Modern Approach. Cambridge University Press (2009), [http://www.cambridge.org/catalogue/catalogue.asp?isbn=9780521424264](http://www.cambridge.org/catalogue/catalogue.asp?isbn=9780521424264)

2. Babai, L., Moran, S.: Arthur-merlin games: A randomized proof system, and a hierarchy of complexity classes. J. Comput. Syst. Sci. 36(2), 254–276 (1988). [https://doi.org/10.1016/0022-0000(88)90028-1](https://doi.org/10.1016/0022-0000(88)90028-1)

3. Ben-Or, M., Goldreich, O., Goldwasser, S., Hästad, J., Kilian, J., Micali, S., Rogaway, P.: Everything provable is provable in zero-knowledge. In: Goldwasser, S. (ed.) Advances in Cryptology - CRYPTO ’88. 8th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1988, Proceedings. Lecture Notes in Computer Science, vol. 403, pp. 37–56. Springer (1988). [https://doi.org/10.1007/0-387-34799-2_4](https://doi.org/10.1007/0-387-34799-2_4)

4. Boneh, D., Bonneau, J., Bünz, B., Fisch, B.: Verifiable delay functions. In: Shacham, H., Boldyreva, A. (eds.) Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10991, pp. 757–788. Springer (2018). [https://doi.org/10.1007/978-3-319-96884-1_25](https://doi.org/10.1007/978-3-319-96884-1_25)

5. Dwork, C., Naor, M.: Pricing via processing or combatting junk mail. In: Brickell, E.F. (ed.) Advances in Cryptology - CRYPTO ’92. 12th Annual International Cryptology Conference, Santa Barbara, California, USA, August 16-20, 1992, Proceedings. Lecture Notes in Computer Science, vol. 740, pp. 139–147. Springer (1992). [https://doi.org/10.1007/3-540-48071-4_10](https://doi.org/10.1007/3-540-48071-4_10)

6. Ephraim, N., Freitag, C., Komargodski, I., Pass, R.: Continuous verifiable delay functions. In: Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020. Proceedings, Part III. pp. 125–154 (2020). [https://doi.org/10.1007/978-3-030-45727-3_5](https://doi.org/10.1007/978-3-030-45727-3_5)

7. Feo, L.D., Masson, S., Petit, C., Sanso, A.: Verifiable delay functions from supersingular isogenies and pairings. In: Galbraith, S.D., Moriai, S. (eds.) Advances in Cryptology - ASIACRYPT 2019 - 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-12, 2019. Proceedings, Part I. Lecture Notes in Computer Science, vol. 11921, pp. 248–277. Springer (2019). [https://doi.org/10.1007/978-3-030-34578-5_10](https://doi.org/10.1007/978-3-030-34578-5_10)

8. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Advances in Cryptology - CRYPTO ’86. Santa Barbara, California, USA, 1986. Proceedings. pp. 186–194 (1986). [https://doi.org/10.1007/3-540-47721-7_12](https://doi.org/10.1007/3-540-47721-7_12)

9. Goldreich, O., Krawczyk, H.: On the composition of zero-knowledge proof systems. SIAM J. Comput. 25(1), 169–192 (1996). [https://doi.org/10.1137/S0097539791220688](https://doi.org/10.1137/S0097539791220688)
10. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof-systems (extended abstract). In: Sedgewick, R. (ed.) Proceedings of the 17th Annual ACM Symposium on Theory of Computing, May 6-8, 1985, Providence, Rhode Island, USA. pp. 291–304. ACM (1985). https://doi.org/10.1145/22145.22178

11. Goldwasser, S., Sipser, M.: Private coins versus public coins in interactive proof systems. In: Hartmanis, J. (ed.) Proceedings of the 18th Annual ACM Symposium on Theory of Computing, May 28-30, 1986, Berkeley, California, USA. pp. 59–68. ACM (1986). https://doi.org/10.1145/12130.12137

12. Mahmoody, M., Smith, C., Wu, D.J.: Can verifiable delay functions be based on random oracles? In: Czumaj, A., Dawar, A., Merelli, E. (eds.) 47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference). LIPIcs, vol. 168, pp. 83:1–83:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2020). https://doi.org/10.4230/LIPIcs.ICALP.2020.83

13. Pietrzak, K.: Simple verifiable delay functions. In: Blum, A. (ed.) 10th Innovations in Theoretical Computer Science Conference, ITCS 2019, January 10-12, 2019, San Diego, California, USA. LIPIcs, vol. 124, pp. 60:1–60:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2019). https://doi.org/10.4230/LIPIcs.ITCS.2019.60

14. Rivest, R.L., Shamir, A., Wagner, D.A.: Time-lock puzzles and timed-release crypto. Tech. rep., USA (1996)

15. Shamir, A.: IP=PSpace. In: 31st Annual Symposium on Foundations of Computer Science, St. Louis, Missouri, USA, October 22-24, 1990. Volume I. pp. 11–15. IEEE Computer Society (1990). https://doi.org/10.1109/FSCS.1990.89519

16. Wesolowski, B.: Efficient verifiable delay functions. In: Ishai, Y., Rijmen, V. (eds.) Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part III. Lecture Notes in Computer Science, vol. 11478, pp. 379–407. Springer (2019). https://doi.org/10.1007/978-3-030-17659-4_13

17. Worsch, T.: Parallel turing machines with one-head control units and cellular automata. In: Theoretical Computer Science. Lecture Notes in Computer Science, vol. 217, pp. 3–30. Springer (1999). https://doi.org/https://doi.org/10.1016/S0304-3975(98)00148-0