Two Sequential Squaring Verifiable Delay Function

Souvik Sur

Department of Computer Science and Engineering,
Indian Institute of Technology Kharagpur,
Kharagpur, West Bengal, India
souviksur@iitkgp.ac.in

Abstract. A Verifiable Delay Function (VDF) is a function that takes a specified sequential time to be evaluated, but can be efficiently verified. VDFs are useful in several applications ranging from randomness beacons to sustainable blockchains but are really rare in practice. Most of the VDFs are based on algebraic assumptions like time-lock puzzle in unknown group orders [8] and isogenies over pairing groups [4]. The number of modulo squaring required for verification in the time-lock puzzle based VDFs are proportional to their security parameter. This paper proposes a verifiable delay function that requires only 2-modulo squaring for verification. So the sequential effort required for verification is independent of the security parameter.

Keywords: Verifiable Delay Functions · Sequentiality · Soundness · Time-lock Puzzle

1 Introduction

In 1992, Dwork and Naor [3] introduced the very first notion of VDF under a different nomenclature “pricing function”. Basically 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 [1]. A verifiable delay function is a function with the domain $\mathcal{X}$ and the range $\mathcal{Y}$, that takes a specified number of sequential steps $T$ to be evaluated (irrespective of the amount of parallelism) and can be verified efficiently (even without parallelism) and publicly. In order to avoid exponential (processors) adversary $T = 2^{o(\lambda)}$ at most.

Along with the formalization, a candidate VDF using injective rational maps has been proposed in [1]. However, it is a weak VDF, i.e, the prover needs a certain amount of parallelism to evaluate. Wesolowski [8] and Pietrzak [6] come up with two VDFs separately, although based on the same hardness assumption of time-lock puzzle [7]. Feo et al. [4] propose a VDF based on super-singular elliptic curves defined over finite fields.

Each of these schemes has the advantages over the others, however the time-lock puzzle based ones [8,6] shine for their simplicity. The basic idea is to compute $g^{2^T}$ in a multiplicative modulo group $(\mathbb{Z}/N\mathbb{Z})^\times$ of unknown order. Still one
crucial constraint for both of these VDFs is that the effort by the verifier is proportional to the security parameter $\lambda$. In particular Pietrzak’s VDF [6] imposes $O(\sqrt{T}\log T)$ time (as mentioned above $T = 2^{o(\lambda)}$) during verification whereas Wesolowski’s VDF takes $\log_2 \ell$ time such that $\ell$ is at most a $2\lambda$-bits prime. In section 4 we show that this prime should be at least $2\lambda$-bits in order to avoid an attacker $A_W$ breaking the soundness with the success probability $\geq 1/\ell$. This enforces Wesolowski’s VDF to incur at least $2\lambda$-time during verification. We believe that the key reason behind this proportional verification effort is neither of these two VDFs can verify if $g^{2T}$ is really the $2T$-th power of $g$.

Thus the question is that can we have faster verification? In particular, can we design a VDF that takes $\delta$-time such that $\delta$ is reasonably small and independent of the security parameter $\lambda$?

2 Related Work

In this section, we mention some well-known schemes qualified as VDFs, and summarize their features in Table I Sect. 6.1.

The pricing function by Dwork–Naor scheme [3] asks a prover, given a prime $p \equiv 3 \pmod{4}$ and a quadratic residue $x$ modulo $p$, 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 $y^2 \equiv x \pmod{p}$. The drawback of this design is that the delay parameter $T = \log p$. Thus the difference between the evaluation and the verification may be made up by a prover with poly($T$)-processors by parallelizing the field multiplications.

In 2018, Boneh et al. [1] 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 VDF as it needs $O(T)$ processors to evaluate the output in time $T$.

Rivest, Shamir, and Wagner [7] introduced another discipline of VDFs known as time-lock puzzle. These puzzles enables 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 $y \equiv x^{2^T} \pmod{N}$. Then the verifier, uses the value of $\phi(N)$ to reduce the exponent to $e \equiv 2^T \pmod{\phi(N)}$ 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 $2^T$ 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 [8] and Pietrzak [6] circumvent this issue independently. The first one asks the prover to compute an output $y = x^{2^T}$ and a proof $\pi = x^{(2^T/\ell)}$, where $\ell$ is a $2\lambda$-bit prime chosen at random. The verifier checks if $y = \pi^t \cdot x^{(2^T \pmod{\ell})}$. Hence the verification needs at most $2\log \ell = 4\lambda$ squaring. However in Sect. 4 we show that it should be at least instead of at most. Two candidate groups suits well in this scheme – an RSA group $(\mathbb{Z}/N\mathbb{Z})^\times$, 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 \( u_i = x^{r_i + 2T/2^i} \). The verifier computes the \( v_i = x^{r_i \cdot 2^{T/2^i} + 2T} \) 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. [4] 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 \( T \) with two non-degenerate bilinear pairing maps \( e_{12} : G_1 \times G_2 \rightarrow G_5 \) and \( e_{34} : G_3 \times G_4 \rightarrow G_5 \). Also there are two group isomorphisms \( \phi : G_1 \rightarrow G_3 \) and \( \phi : G_4 \rightarrow G_2 \).

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 super-singular curves over \( \mathbb{F}_p \) and \( \mathbb{F}_{p^2} \) as two candidate groups. While being inherently non-interactive, there are two drawbacks as mentioned by the authors themselves. First, it requires a trusted setup, and second, the setup phase may turn out to be slower than the evaluation.

Mahmoody et al. recently rule out the possibility of having a VDF out of random oracles only [5].

3 Preliminaries

Now we mention the notations and terminology used in this paper.

3.1 Notation

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 \( A \) outputs \( y \) on an input \( x \), we write \( y \overset{R}{\leftarrow} A(x) \). By \( x \overset{R}{\in} \mathcal{X} \), we mean that \( x \) is sampled uniformly at random from \( \mathcal{X} \). For an element \( x \), \( |x| \) denotes the bit-length of \( x \), whereas for any set \( \mathcal{X} \), \( |\mathcal{X}| \) denotes the cardinality of the set \( \mathcal{X} \).

We consider \( A \) as efficient if it runs in probabilistic polynomial time (PPT). We assume (or believe) a problem to be hard if it is yet to have an efficient algorithm for that problem. We say that an algorithm \( A \) runs in parallel time \( T \) with \( \Gamma \) processors if it can be implemented on a PRAM machine with \( \Gamma \) parallel processors running in time \( T \).

3.2 Verifiable Delay Function

We borrow this formalization from [1].
Definition 1. (Verifiable Delay Function). A VDF is a triple of algorithms \((\text{Setup}, \text{Eval}, \text{Verify})\) that implements a function \(X \rightarrow Y\).

- **Setup**\((1^\lambda, T) \rightarrow \text{pp}\) is a randomized algorithm that takes as input a security parameter \(\lambda\) and a delay parameter \(T\), and produces the public parameters \(\text{pp}\). We require \(\text{Setup}\) to run in \(\text{poly}(\lambda, \log T)\)-time.
- **Eval**\((\text{pp}, x) \rightarrow (y, \pi)\) takes an input \(x \in X\), and produces an output \(y \in Y\) and a (possibly empty) proof \(\pi\). Eval may use random bits to generate the proof \(\pi\). For all \(\text{pp}\) generated by \(\text{Setup}(1^\lambda, T)\) and all \(x \in X\), the algorithm \(\text{Eval}(\text{pp}, x)\) must run in parallel time \(T\) with \(\text{poly}(\lambda, \log T)\) processors.
- **Verify**\((\text{pp}, x, y, \pi) \rightarrow \{0, 1\}\) is a deterministic algorithm that takes an input \(x \in X\), an output \(y \in Y\), and a proof \(\pi\) (if any), and either accepts (1) or rejects (0). The algorithm must run in \(\text{poly}(\lambda, \log T)\) time.

**Subexponentiality of** \(T\)  As Verify is efficient, we need \(T \leq 2^{o(\lambda)}\), otherwise a prover with \(\text{poly}(T)\) processors will always be able to brute-force \(2^\lambda\) possible solutions. \(T \leq 2^{o(\lambda)}\) enforces the complexity of this brute-force approach to be \(2^{\lambda/2^{o(\lambda)}} = 2^{\Omega(\lambda)}\).

**Interactive VDFs**  Def. 1 uses Fiat–Shamir heuristic to generate the proof \(\pi\). It is the non-interactive version of VDFs where the verifier does not interact with the prover. Fiat–Shamir heuristic replaces this interaction in any public coin interactive proof using a random oracle. The interactive versions of VDFs asks the prover to compute the proof \(\pi\) on challenges chosen by the verifier.

The three desirable properties of a VDF are now described.

**Definition 2.** (Correctness) A VDF is correct, if for all \(\lambda, T\), parameters \(\text{pp}\), and \(x \in X\), we have

\[
\Pr \left[ \text{Verify}(\text{pp}, x, y, \pi) = 1 \mid \begin{align*}
\text{pp} &\leftarrow \text{Setup}(1^\lambda, T) \\
(x, y, \pi) &\leftarrow \text{Eval}(\text{pp}, x)
\end{align*} \right] = 1.
\]

**Definition 3.** (Soundness) A VDF is sound if for all non-uniform algorithms \(A\) that run in time \(\text{poly}(T, \lambda)\), we have

\[
\Pr \left[ y \neq \text{Eval}(\text{pp}, x) \mid \begin{align*}
\text{Verify}(\text{pp}, x, y, \pi) &\leftarrow 1 \\
(x, y, \pi) &\leftarrow A(1^\lambda, T, \text{pp})
\end{align*} \right] \leq \text{negl}(\lambda).
\]

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

**Definition 4.** (Sequentiality) A VDF is \((\Gamma, \sigma)\)-sequential if for all pair of randomized algorithms \(A_0\) with total running time \(\text{poly}(T, \lambda)\) and \(A_1\) which runs in parallel time \(\sigma(T)\) on at most \(\Gamma\) processors, we have,

\[
\Pr \left[ y = \text{Eval}(\text{pp}, x) \mid \begin{align*}
\text{pp} &\leftarrow \text{Setup}(1^\lambda, T) \\
\text{state} &\leftarrow A_0(1^\lambda, T, \text{pp}) \\
x &\leftarrow X \\
y &\leftarrow A_1(\text{state}, x)
\end{align*} \right] \leq \text{negl}(\lambda).
\]
Here, $A_0$ is a preprocessing algorithm that precomputes some state based only on the public parameters, and $A_1$ exploits this additional knowledge in $\text{Eval}(x, pp)$ in parallel running time $\sigma$ on $\Gamma$ processors.

4 Lower Bound for Verification of Wesolowski’s VDF

First we give an attacker $A_W$ that breaks the soundness of non-interactive version of Wesolowski’s VDF with probability $1/\ell$. Two important observations relevant to the verification of this VDF are,

1. The sampled prime $\ell \leftarrow H_{\text{prime}}(\text{bin}(g)||\text{bin}(y))$ is independent of the delay parameter $T$. So the prover may claim that $y = g^{2^T}$ for any arbitrary values of $T$.
2. The equivalence $y = \pi^\ell \cdot g^{r}$ implies $y = g^{2^T}$ if and only if $\pi = g^{(2^T/\ell)}$ which is never verified by the verifier.

4.1 Strategy of $A_W$

$A_W$ chooses a small $(2\lambda + 1$-bit suffices in this context) number $\tau$ and computes $y = g^\tau$. Then (s)he samples the prime $\ell$ in order to compute $\pi = g^{(\tau/\ell)}$. Now (s)he finds a number $T > \log_2 \tau$ such that $\tau \mod \ell = 2^T \mod \ell$. In particular $A_w$ computes,

1. $g \leftarrow H_{G}(x)$.
2. $y = g^\tau$ such that $\tau \leftarrow \mathbb{Z}_{(2\lambda + 1)} \setminus \mathbb{Z}_{2\lambda}$.
3. $\ell \leftarrow H_{\text{prime}}(\text{bin}(g)||\text{bin}(y))$.
4. $\pi = g^{(\tau/\ell)}$.
5. $T \leftarrow \lceil \log_2 \tau \rceil$.
6. While $(\tau \mod \ell \neq 2^T \mod \ell)$ do $T \leftarrow T + 1$.

$A_W$ announces the tuple $(x, G, y, \pi, T)$ to get verified. The verifier will find that $\pi^\ell \cdot g^{2^T \mod \ell} = \pi^\ell \cdot g^r \mod \ell = (g^{\tau/\ell})^\ell g^r \mod \ell = y$.

For any arbitrary $T$, $\Pr[A_W \text{ wins}] = \Pr[\tau \mod \ell = 2^T \mod \ell] = 1/\ell$. Inclusion of $T$ in $\ell \leftarrow H_{\text{prime}}(\text{bin}(g)||\text{bin}(y)||\text{bin}(T))$ requires $A_W$ to include only this sampling of $\ell$ within the while loop incurring the same success probability $1/\ell$. In order to have $\Pr[A_W \text{ wins}] \leq \text{negl}(\lambda)$ we must have $\ell \geq 2^{2\lambda}$. As verification needs $O(\log_2 \ell)$ modulo squaring, against $A_W$ it needs $\geq 2\lambda$ modulo squaring.

5 $\delta$-Squaring Verifiable Delay Function

As before, $\lambda \in \mathbb{Z}^+$ denotes the security parameter, $T \in 2^{o(\lambda)}$ denotes the delay parameter. The three algorithms that specify our VDF are,
5.1 The Setup(1\(\lambda\), \(T\)) Algorithm

This algorithm outputs the public parameters \(\text{pp} = \langle H, N \rangle\) having the following meanings.

1. \(N = pq\) where
   (a) \((p - 1)/2\) and \((q - 1)/2\) are primes.
   (b) \(\log_2 p \approx \log_2 q\) and are small multiples of \(\lambda\).
2. \(\delta \in \mathbb{Z}^+\) is a parameter that tunes the sequential effort during verification.
   We show that \(\delta = 2\) is necessary and sufficient. Thus \(\delta\) is fixed even before we run the \text{Setup}. Still we mention \(\delta\) in \(\text{pp}\), otherwise we had to introduce \(\delta\) as another implicit parameter like \(\lambda\) or \(T\).
3. Theorem A in [2] states that for a finite group \(G\) having \(k\) elements of maximal order \(m\) holds
   \(|G| \leq \frac{mk^2}{\phi(m)}\) where \(\phi()\) Euler’s totient function. When \(m\) is a power of 2 the ratio \(\frac{\phi(m)}{m} = \frac{1}{2}\) so \(k \geq \sqrt{|G|}/2\). Hence, there are at least \(2^{(T+\delta)/2}\) elements in the group \((\mathbb{Z}/2^{T+\delta+2} \mathbb{Z})^\times\) that have maximal order. So the cardinality of \(|S| = 2^{(T+\delta)/2} - p(q - 1) - q(p - 1) = 2^{(T+\delta)/2} \approx 2^{2^{o(\lambda)}}\).
   We take \(H : \mathcal{X} \rightarrow S\) maps \(x \in \mathcal{X}\) to the set \(S\).

None of the public parameters needs to be computed. The factorization of \(N\) opens up a trapdoor to this VDF and is unknown to the prover, the verifier and the third-party who runs the \text{Setup}.

5.2 The Eval(\(\text{pp}, x, T\)) Algorithm

The prover,
1. maps \(g \leftarrow H(x)\).
2. computes \(y = g^{(2^T+1)} \mod N\).

Announce \((x, T, y)\).

5.3 The Verify(\(\text{pp}, x, T, y, \bot\)) Algorithm

The verifier,
1. maps \(g \leftarrow H(x)\).
2. accepts if \((y \cdot g^{-1})^{2^\delta} \mod N = 1 \mod (2^{T+\delta+2} \mod N)\), rejects otherwise.

Verification needs no proof, so \(\pi = \bot\).

6 Efficiency Analysis

Here we discuss the efficiencies of both the prover \(P\) and the verifier \(V\) in terms of number of modulo squaring and the memory requirement.
Proof Size The proof size is essentially zero as the proof $\pi = \bot$ is empty. The output $y$ is an element in the group $(\mathbb{Z}/N\mathbb{Z})^\times$. So the size of the output is $\log_2 N$-bits.

Prover’s Efficiency The prover $P$ needs at least $T$ sequential time in order to compute $g^{2^T}$ such that $g = H(x)$. Additionally, one multiplication is needed to compute $g^{2^T+1}$.

Verifier’s Efficiency During verification only $\delta$ number of modulo squaring are needed. We have shown that $\delta = 2$ is necessary and sufficient to achieve soundness.

6.1 Performance Comparison

Each of the existing VDFs suffers from at least one limitation. We have already discussed them in Sect. 2, however, reiterate here to show that the proposed VDF overcomes all of them.

Sequentiality Adversary with $\text{poly}(\lambda, T)$-parallelism breaks the sequentiality of the scheme [3]. It shows that our VDF stands sequential against any $\text{poly}(\lambda, T)$-adversary.

Weak VDF We call a VDF is a weak VDF if it demands $\text{poly}(\lambda, T)$-parallelism to compute the Eval in time $T$. The VDF based on injective rational maps is a weak VDF [1]. In our case, no parallelism is required to compute the VDF in time $T$.

Proof The proofs in the Wesolowski’s and Pietrzak’s VDFs consume one and $\log T$ group elements [8,6]. Moreover in Sect. 4 we show that, the proof itself in Wesolowski’s VDF gave us an edge to attack it. Our VDF requires no proof.

Furthermore, for the interactive versions they need 1 and $\log T$ extra round(s) of interaction to communicate the proof [3]. As the verification runs without a proof only 2 rounds suffices to execute our VDF. The prover sends the output $y$ and the verifier replies with its decision.

Slow Setup The Setup in the isogeny-based VDF may turn out to be as slow as the Eval itself [4] questioning the practicality of the scheme. On the other hand, the Setup in our VDF is exactly same as the ones in all other time-lock based VDFs [6].

$\Omega(\lambda)$-Verifiability All the existing VDFs need at least $\lambda$ sequential effort for verification. The attack proposed in Sect. 4 mandates at least $2\lambda$-sequential effort for verification in Wesolowski’s VDF. On the contrary, the most important advantage of our VDF is that the verification requires only 2 modulo squaring. Thus the sequential effort in verification is independent of the parameters $\lambda$ and $T$.

In Table 1 we summarize the above comparison.
Table 1. Comparison among the existing VDFs. $T$ is the delay parameter, $\lambda$ is the security parameter and $\Gamma$ is the number of processors. All the quantities may be subjected to $O$-notation, if needed.

| VDFs (by authors) | Eval Sequential | Eval Parallel | Verify | Setup | Proof size |
|-------------------|----------------|--------------|--------|-------|------------|
| Dwork and Naor \[3\] | $T$ | $T^{2/3}$ | $T^{2/3}$ | $T$ | – |
| Boneh et al. \[1\] | $T^2$ | $> T - o(T)$ | $\log T$ | $\log T$ | – |
| Wesolowski \[8\] | $(1 + \frac{1}{\log T})T$ | $(1 + \frac{2}{\log T})T$ | $\lambda^3$ | $\lambda^3$ | $\lambda^3$ |
| Pietrzak \[4\] | $(1 + \frac{2}{\sqrt{T}})T$ | $(1 + \frac{2}{\sqrt{T}})T$ | $\sqrt{T} \log T$ | $\lambda^3$ | $\log T$ |
| Feo et al. \[4\] | $T^2$ | $T$ | $\lambda^3$ | $T \log \lambda$ | – |
| **Our work** | $(1 + \frac{2}{\log T})T$ | $(1 + \frac{2}{\log T})T$ | $2$ | $\lambda^3$ | – |

7 An Open Problem

Beyond this the best that one can think of is a VDF that verifies spending only one sequential computation without any proof, if possible at all. In case of time-lock puzzle based VDFs the sequential computation is modulo squaring.

References

1. 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)

2. Cocke, W., Venkataraman, G.: On the number of elements of maximal order in a group. The American Mathematical Monthly 126(1), 66–69 (2019). [https://doi.org/10.1080/00029890.2019.1528826](https://doi.org/10.1080/00029890.2019.1528826)

3. 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)

4. 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)

5. 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](https://doi.org/10.4230/LIPIcs.ICALP.2020.83)
6. 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
7. Rivest, R.L., Shamir, A., Wagner, D.A.: Time-lock puzzles and timed-release crypto. Tech. rep., USA (1996)
8. 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