sensors

MDPI

Article

Verifiable Delay Function and Its Blockchain-Related Application: A Survey

Qiang Wu [1], Liang Xi [2], Shiren Wang [2], Shan Ji [3,*], Shenqing Wang [3] and Yongjun Ren [1]

[1] Engineering Research Center of Digital Forensics, Ministry of Education, School of Computer Science, Nanjing University of Information Science & Technology, Nanjing 210044, China
[2] Beijing Institute of Computer Technology & Application, Beijing 100082, China
[3] College of Computer Science & Technology, Nanjing University of Aeronautics & Astronautics, Nanjing 211106, China
* Correspondence: shanji2022082022@163.com; Tel.: +86-1885-550-8210

Abstract: The concept of verifiable delay functions has received attention from researchers since it was first proposed in 2018. The applications of verifiable delay are also widespread in blockchain research, such as: computational timestamping, public random beacons, resource-efficient blockchains, and proofs of data replication. This paper introduces the concept of verifiable delay functions and systematically summarizes the types of verifiable delay functions. Firstly, the description and characteristics of verifiable delay functions are given, and weak verifiable delay functions, incremental verifiable delay functions, decodable verifiable delay functions, and trapdoor verifiable delay functions are introduced respectively. The construction of verifiable delay functions generally relies on two security assumptions: algebraic assumption or structural assumption. Then, the security assumptions of two different verifiable delay functions are described based on cryptography theory. Secondly, a post-quantum verifiable delay function based on super-singular isogeny is introduced. Finally, the paper summarizes the blockchain-related applications of verifiable delay functions.

Keywords: verifiable delay function; blockchain; algebraic assumption; structural assumption

Citation: Wu, Q.; Xi, L.; Wang, S.; Ji, S.; Wang, S.; Ren, Y. Verifiable Delay Function and Its Blockchain-Related Application: A Survey. *Sensors* **2022**, *22*, 7524. https://doi.org/10.3390/s22197524

Academic Editor: Francesco Longo

Received: 3 September 2022
Accepted: 29 September 2022
Published: 4 October 2022

1. Introduction

The concept of a verifiable delay function was first proposed in 2018 by Boneh et al. [1], who proposed several candidate structures for constructing verifiable delay functions and it is an important tool to add time delay in decentralized applications [2–5]. To be exact, the verifiable delay function is a function $f : \mathcal{X} \to \mathcal{Y}$ that takes a prescribed wall-clock time to compute, even on a parallel processor, ond outputs a unique result that can effectively output the verification. In short, even if it is evaluated on a large number of parallel processors and still requires evaluation of f in a specified number of sequential steps. Most importantly, given an input x and an output y, anyone must quickly verify the output $y = f(x)$. That is to say, for all $x \in \mathcal{X}$ and $y \in \mathcal{Y}$, this function $f : \mathcal{X} \to \mathcal{Y}$ satisfies the following requirements:

$$x \to x^2 \to x^{2^2} \to x^{2^3} \to \ldots \to x^{2^T} \bmod N = y. \tag{1}$$

The verifiable delay function is a cryptographic function that requires to be computed in T sequential steps and produces a unique, efficiently and publicly verified output [6]. Because the verifiable delay function satisfies the characteristic of sequentiality, the iterated value does depend on the order of the iterated elements. Choose the tuple (N, x, T) as the puzzle, and the verifiable delay function is defined as

$$e := 2^T \bmod \varphi(N), \quad y := x^e \bmod N. \tag{2}$$

where $N = p \cdot q$ is an RSA modulus [7], $x \in Z_N^*$ is a random seed, $T \in N$ is time parameter and knows the group order $\varphi(N) = (p-1) \cdot (q-1)$.

Although verifiable delay functions have been roughly described in the review of verifiable delay functions published by Boneh et al. [8], the summary is not comprehensive with the emergence of more candidate structures of verifiable delay functions. In addition, the application of different kinds of verifiable delay functions in the blockchain is not explained in detail. Therefore, this paper makes a more comprehensive and detailed summary.

The verifiable delay function has several important characteristics, such as being T-sequential, uniqueness and effective verifiability, as shown in Table 1.

Table 1. Characteristics of verifiable delay functions.

Characteristics	Description
T-Sequentiality	The function cannot be calculated in a sequential steps less than T to obtain the final result, even given a large amount of parallelism.
Uniqueness	For the input of any verifiable delay functions, only one unique output result shall pass the inspection. Meanwhile, it is necessary to ensure that the probability of the verifier passes the verification because of the proof, but the output result is not the correct result is negligible.
Effective verifiability	The calculation results can be efficiently verified so that the honest party can calculate.

The remainder of this paper is organized as follows. Section 2 introduces the descriptions of verifiable delay functions. Section 3 describes verifiable delay functions based on various algebraic assumptions. In Section 4, verifiable delay functions based on various structural assumptions are introduced in detail. Section 5 the postquantum-secure verifiable delay function. Section 6 describes applications of verifiable delay functions combined with blockchain. Section 7 gives a summary.

2. Descriptions of Verifiable Delay Functions

The concept of the verifiable delay function first proposed by Boneh and Fisch. The verifiable delay function requires a specified number of sequential steps to evaluate and will produce a function with a unique output that can be validated effectively and publicly. Next, a triple of algorithm (*Setup*, *Eval*, *Verify*) of verifiable delay functions are described as follows [9]. The algorithm flow is shown in Figure 1 and the different types of verifiable delay functions are described in Table 2.

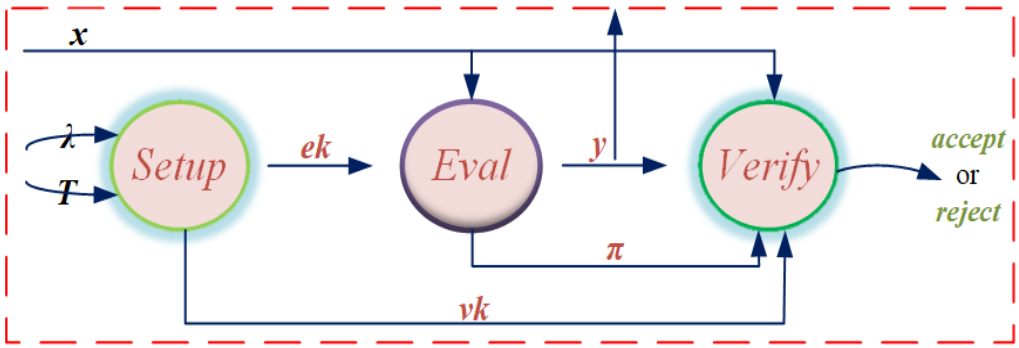

Figure 1. The algorithmic flow of verifiable delay functions.

$Setup(\lambda, T) \rightarrow pp = (ek, vk)$ is a randomized algorithm that takes a delay parameter T and a security parameter λ as input and outputs public parameters pp composed of the evaluation key ek and the verification key vk. Because $Setup$ algorithm is limited by security parameter λ, the running time cannot be too long. In addition, $Setup$ algorithm

usually needs a secret random as a parameter to ensure meaningful security, so it is difficult to avoid that the scheme needs a trusted setup to select the random.

Table 2. Classification of verifiable delay functions.

Classification	Description
Weak verifiable delay functions	The function cannot be calculated in a sequential steps less than T to obtain the final result, even given a large amount of parallelism.
Incremental verifiable delay functions	All verifiable delay functions need to require the *Eval* algorithm to be completed in at least T steps. If the delay parameter T is not uniquely determined in the $Setup$ algorithm, but is allowed to be determined in the Eval algorithm, then the verifiable delay function can be called as incremental verifiable delay function.
Decodable verifiable delay functions	For any verifiable delay function scheme, as long as a random input element x can be obtained from the output value y in reverse, the verifiable delay function can be called a decodable verifiable delay function. Another output value π for the proof is empty.
Trapdoor verifiable delay functions	If there is an algorithm that enables the party who knows a certain secret key value *sk* to calculate the output value of the verifiable delay function through the *Eval* algorithm too quickly, then the function is a trapdoor verifiable delay function.

$Eval(ek, x) \rightarrow (y, \pi)$ is a slow cryptographic algorithm that takes the evaluation key ek and a random seed $x \in \mathcal{X}$ as input and outputs a $y \in \mathcal{Y}$ together with a possibly empty proof π. To ensure sequentiality, $Eval$ must run in time T with no more than a polynomial logarithm of T parallel processors.

$Verify(vk, x, y, \pi) \rightarrow \{accept, reject\}$ is a deterministic cryptographic algorithm, in which the algorithm inputs verification key vk, random seed x, outputs y and proof π. If $f(x) = y$, output $accept$; Otherwise output $reject$. $Verify$ is much faster than $Eval$ and it must run in total time polynomial in $log(T)$ and λ.

2.1. Weak Verifiable Delay Functions

Definition 1. *(Weak verifiable delay functions.) The system $\mathcal{V}$ = (Setup, Eval, Verify) is a weak verifiable delay function if the verifiable delay function allows Eval to achieve $O(T)$ parallelism. $(T^2, o(T))$-sequentiality can only be meaningful for a weak verifiable delay function if Eval is allowed strictly less that $T - o(T)$ on fewer than T^2 parallel processors, otherwise the honest computation of Eval would require more parallelism than even the adversary is allowed.*

The weak verifiable delay function can be constructed based on the existence of degree T injective rational maps [10] that cannot be inverted faster than computing polynomial greatest common denominators of degree T polynomials.

Injective rational maps. Define the reverse problem of an injective rational map $F = (f_1, \ldots, f_m)$ on algebraic sets $\mathcal{Y} \subseteq \mathbb{F}_q^n$ to $\mathcal{X} \subseteq \mathbb{F}_q^m$, where each $f_i : \mathbb{F}_q^n \rightarrow \mathbb{F}_q$ is a rational function in $\mathbb{F}_q(\mathcal{X}_1, \ldots, \mathcal{X}_n)$, for $i = \{1, \ldots, m\}$. An algebraic set $\mathcal{Y}$ is the set of vanish points of some set of polynomial S.

Boneh et. al. abstract weak verifiable delay functions from an injective rational map. First, let F: $\mathbb{F}_q^n \rightarrow \mathbb{F}_q$ be a rational function that is an injective map from $\mathcal{Y}$ to $\mathcal{X} := F(\mathcal{Y})$. At the same time, $\mathcal{X}$ is required to be efficiently sampleable and F can be evaluated efficiently for all $y' \in \mathcal{Y}$. If you need to use the injective rational map function F in the verifiable delay function, you must guarantee $|\mathcal{X}| > \lambda T^3$ to prevent brute force attacks, where a delay parameter T and a security parameter λ as input to the $Setup$ algorithm.

Verifiable delay functions construct a weak verifiable delay function by function family $\mathcal{F} := (q, F, \mathcal{X}, \mathcal{Y})_{\lambda,T}$ with a security parameter λ and a delay parameter T as input parameters.

$Setup(\lambda, T) \rightarrow pp = ((q, F), (q, F))$ is a randomized algorithm that takes a delay parameter T and a security parameter λ as input and choose a $(q, F, \mathcal{X}, \mathcal{Y}) \in \mathcal{F}$, then outputs public parameters pp composed of the (q, F).

$Eval((q, F), x') \rightarrow (y', \pi)$ is a slow cryptographic algorithm that takes the (q, F) and a random seed $x' \in \mathcal{X} \subseteq \mathbb{F}_q^m$ as input and compute a $y' \in \mathcal{Y}$ together with a possibly empty proof π.

$Verify((q, F), x', y', \pi) \rightarrow \{accept, reject\}$ is a deterministic cryptographic algorithm, in which the algorithm inputs (q, F), random seed x', outputs y' and proof π. If $F(x') = y'$, output *accept*; Otherwise output *reject*.

In order to ensure that the solution y' is unique, F is required to be injective on Y.

2.2. Incremental Verifiable Delay Functions

Definition 2. *(Incremental verifiable delay functions.) The system $\mathcal{V}$ = (Setup, Eval, Verify) is a incremental verifiable delay function if the time parameter T of the verifiable delay function is not uniquely determined and is allowed to be determined in the output π of Eval, which does not generate additional proofs.*

Since the verifiable delay function is a sequential function [11], Boneh et. al. propose the use of tight incremental verifiable computation to construct an incremental verifiable delay function construction. Next, here's how to build an incremental verifiable delay function with a tight incremental verifiable computation.

Incremental verifiable computation was first proposed by Valiant [12]. After that, Bitansky et al. [13] applied it to distributed computions and to other proof systems. The incremental verifiable computation is to guarantee that the prover can generate a proof that a certain state is indeed the current state of the computation at every incremental step of the computation. The proof is updated after every step of the computation to produce a new proof. Iterative sequence functions can be implemented via tight incremental verifiable computation, which captures the primitives required by verifiable delay functions.

Let $f_\lambda : \mathbb{N} \times \mathcal{X} \rightarrow \mathcal{X}$ be an interated sequential function with round function g_λ having (T, ϵ)-sequentiality. An incremental verifiable computation system for an interated sequential function f_λ is polynomial time algorithm ($IVCGen$, $IVCProve$, $IVCVerify$) that satisfy completeness, succinctness and soundness.

Completeness.

$$\forall x \in \mathcal{X}, \Pr\left[\begin{array}{l|r} IVCVerify(vk, x, y, k, \pi) = \text{Yes} & (vk, ek) \xleftarrow{R} IVCGen(\lambda, f) \\ & (y, \pi) \xleftarrow{R} IVCProve(ek, k, x) \end{array}\right] = 1 \quad (3)$$

Succinctness. The length of a proof is bounded by $ploy(\lambda, log(kT))$.

Soundness. The soundness satisfied by the incremental verifiable computation is sub-exponential soundness. For all algorithm $\mathcal{A}$ running in time $2^{o(\lambda)}$.

$$\Pr\left[\begin{array}{l|r} IVCVerify\, y(vk, x, y, k, \pi) = \text{Yes} & (vk, ek) \xleftarrow{R} IVCGen(\lambda, f) \\ f(k, x) \neq y & (x, y, k, \pi) \xleftarrow{R} \mathcal{A}(\lambda, vk, ek) \end{array}\right] < negl(\lambda) \quad (4)$$

Next, we introduce the verifiable delay function construction based on tight incremental verifiable computation. Let a family f_λ, where each f_λ, $\mathbb{N} \times \mathcal{X}_\lambda \rightarrow \mathcal{X}_\lambda$ is defined by $f_\lambda(k, x) = g_\lambda^k(x)$. Here g_λ is a (T, ϵ)-sequential function on an efficiently sampleable domain of size $O(2^\lambda)$.

$Setup(\lambda, T) \rightarrow pp = ((ek, k), vk)$ is a randomized algorithm that takes a delay parameter T and a security parameter λ as input and outputs public parameters pp composed of the evaluation key ek, a largest integer k and the verification key vk. Generate (ek, vk) by running $IVCGen(\lambda, f_\lambda)$.

$Eval((ek,k),x) \rightarrow (y,\pi)$ is a slow cryptographic algorithm that takes the evaluation key ek, a largest integer k and a random seed x as input and runs $IVCProve(ek, k, x)$, and outputs a y together with a possibly empty proof π.

$Verify(vk,x,y,\pi) \rightarrow \{accept, reject\}$ is a deterministic cryptographic algorithm, in which the algorithm runs and outputs $IVCVerify(vk, x, y, k, \pi)$. If $f_\lambda(x,k) = y$, output *accept*; Otherwise output *reject*.

Since T is fixed in the public parameters pp. However, it is also possible to directly assign the T to *Eval* algorithm. Therefore, a tight incremental verifiable computation based the verifiable delay function is an incremental verifiable delay function.

2.3. Decodable Verifiable Delay Functions

Definition 3. *(Decodable verifiable delay functions.) The system $\mathcal{V}$ = (Setup, Eval, Verify) is a decodable verifiable delay function if there is an algorithm in the verifiable delay function that can decode input x backwards from output y. If the decoding is efficient then no additional proof π is required [14].*

Using a slow and easy to verify function with exponentiation [15] based calculations in a finite group can be constructed a decodable verifiable delay function. Boneh et al. propose a simple exponentiation-based decodable verifiable delay functions with bounded pre-computation. However, the adversary cannot run a long pre-computation between the time the public parameter pp is exposed and the time that the verifiable delay function is computed.

Next, we introduce a decodable verifiable delay function based on an exponentiation in a finite group. Let $L = \{\iota_1, \iota_2, \ldots, \iota_T\}$ be the first T odd promes, namely $\iota_1 = 3, \iota_2 = 5$, etc. Let P be the product of the primes in L, namely $P := \iota_1\iota_2 \ldots \iota_T$.

$Setup(\lambda, T, b) \rightarrow pp = (ek, vk)$ is a randomized algorithm that takes a delay parameter T, a security parameter λ and a preprocessing security parameter b as input and outputs public parameters pp composed of the evaluation key ek and the verification key vk.

In algorithm *Setup*, let a integer module N multiplicative group $\mathbb{G} := (\mathbb{Z}/N\,\mathbb{N})^*$ and a random hash function $H : \mathbb{Z} \rightarrow \mathbb{G}$. The algorithm needs to compute $h_i \leftarrow H(i) \in \mathbb{G}$, for $i \in \{1, 2, \ldots, b = 2^{30}\}$ then compute $g_i := h_i^{1/P}$. It outputs the evaluation key $ek := (\mathbb{G}, H, g_1, g_2, \ldots, g_b)$ and the verification key $vk := (\mathbb{G}, H)$.

$Eval(ek, x) \rightarrow y$ is a slow cryptographic algorithm that takes the evaluation key ek and a random seed x as input and outputs a y.

In algorithm *Eval*, using random hash function to map a random seed x to a size of λ random subset $L_x \subseteq L$ and random subset S_x of λ values in $\{1, 2 \ldots, b = 2^{30}\}$. At the same time, let P_x be the product of all prime numbers in L_x. Let g be $g := \Pi_{i \in S_x} g_i \in \mathbb{G}$ and the seed solution y is simply $y \leftarrow g^{P/P_x} \in \mathbb{G}$.

$Verify(vk,x,y,\pi) \rightarrow \{accept, reject\}$ is a deterministic cryptographic algorithm, in which the algorithm inputs verification key vk, random seed x, outputs y and proof π. If $f(x) = y$, output *accept*; Otherwise output *reject*.

In algorithm *Verify*, let h be $h := \Pi_{i \in S_x} H(i) \in \mathbb{G}$ and if and only if $y^{P_x} = h \in \mathbb{G}$, where P_x and S_x are calculated by the algorithm $Eval(ek, x)$.

The preprocessing parameter b in an exponentiation-based the decodable verifiable delay function ensures the security of the construction. The construction requires a trusted setup [16,17], but can be eliminated by choosing a random number large enough.

2.4. Trapdoor Verifiable Delay Functions

Definition 4. *(Trapdoor verifiable delay functions.) The system $\mathcal{V}$ = (Keygen, Trapdoor, Eval, Verify) is a trapdoor verifiable delay function if there is a secret key sk that can quickly get the output of Eval through the input of Eval. In other words, the trapdoor verifiable delay function can bypass the delay parameter to quickly calculate the result through the trapdoor [18].*

Given a pair of Alice's public-secret keys (pk, sk), where pk is Alice's public key and sk is the secret key. Alice is able to quickly evaluate trap [19,20] functions $Trapdoor_{sk}$ on x with a secret key sk. Let T be an implicit time function about the security parameter λ and x be a piece of data. Except for Alice, everyone else can only compute the public evaluation function $Eval_{pk}$ with the public key pk in T-sequential steps and the calculation is slow, but the result between $Eval_{pk}$ and $Trapdoor_{sk}$ is equal. A trapdoor verifiable delay function comsists of four algorihtm $(Keygen, Trapdoor, Eval, Verify)$.

$Genkey(\lambda) \rightarrow (pk, sk)$ is a key generation algorithm that takes a security parameter λ as input and outputs Alice's public key pk and the secret key sk. Meanwhile, Alice's public key is publicly valid, and the secret key is known only to Alice herself.

$Trapdoor_{sk}(x, T) \rightarrow (y, \pi)$ is a slow cryptographic algorithm that takes an implicit time function T about the security parameter λ and a piece of data x as input, and uses the secret key sk to output y together with a possibly empty proof π. The function T is a sequence of sequential steps required to compute the same output y without knowledge of the secret key sk.

$Eval_{pk}(x, T) \rightarrow (y', \pi')$ is a slow cryptographic algorithm to evaluate the function on x using only the public key pk. It produces an output y associated with y' and a possibly empty proof π'. This procedure is meant to be infeasible in time less than T (this will be expressed precisely in the security requirements).

$Verify(x, T, y, \pi) \rightarrow \{accept, reject\}$ is a deterministic cryptographic algorithm to verify if y is indeed the correct output for x, associated with the public key pk and the evaluation time T, possibly with the help of the proof π.

The time delay T is a function of the security parameter λ and T is an input to each algorithm, so the security parameter λ is implicitly an input to each of these procedures. Generate a public-secret key pair (pk, sk) through the key generation algorithm $Genkey$. Given a piece of data x and time delay parameter T, let $Trapdoor_{sk}(x, T) \rightarrow (y, \pi)$ and $Eval_{pk}(x, T) \rightarrow (y', \pi')$. If $y = y'$ and $Verify(x, T, y, \pi) = Verify(x, T, y', \pi')$ output *accept*; Otherwise output *reject*.

3. Verifiable Delay Functions Based on Algebraic Assumptions

3.1. Construction Based on Finite Abelian Groups of Unknown Order

Verifiable delay functions can be constructed by showing Rivests-Shamir-Wagner (RSW) when the time-lock puzzle [21,22] is publicly verifiable. To be precise, giving a statistically sound public-coin protocol [23] to prove that a tuple (T, N, x, y) satisfies $y = x^t (mod N)$ verifiers do not know the decomposition of N and its running time is mainly to solve the puzzle, where the time $t = 2^T$ is a power of two. This construction solves an instance of the time-lock puzzle, and computes a proof of correctness, which allows anyone to efficiently verify the result.

Pietrzak [24] proposed a verifiable delay function based on finite abelian groups [25,26] of unknown order consisting of four algorithm $(Setup, Genkey, Sloth, Verify)$.

$Setup(1^\lambda) \rightarrow N$ inputs the statistical security parameter 1^λ output N, where the λ defines another security parameter λ_{RSA} specifying the bitlength of an λ_{RSA} modulus and N is the single λ_{RSA} bit RSA modulus of public parameters. The $N := p \cdot q$ is composed of two $\lambda_{RSA}/2$-bit secure prime numbers p and q randomly selected by the $Setup$ algorithm.

$Genkey(N, T) \rightarrow (x, T)$ samples a random number $x \in QR^+$ and outputs (x, T).

Define $QR_N \stackrel{\text{def}}{=} \{z^2 mod N : z \in Z_N^*\}$ as quadratic residues and the signed quadratic residues [27] are the group $QR^+ \stackrel{\text{def}}{=} \{|x| : x \in QR_N\}$. In a verifiable delay function, calculating x^{2^T} is difficult in $(Z_N^*, \cdot)$. Pietrzak uses $(QR_N^+, \circ)$ instead of $(Z_N^*, \cdot)$. Because $|QR_N| = |Z_N^*|/4$, the probability that a random number in ϵ is also in QR_N is $1/4$. Therefore, if one can break the assumption with probability ϵ over QR_N, the assumption can also be broken over Z_N^* with probability $\epsilon/4$. Then, they uses $(QR_N, \cdot)$ instead of $(QR_N^+, \circ)$ in the proof. This approach can make the proof more efficient because the multiplication mod N in QR_N is more convenient and simpler than the $\circ$ operation in QR_N^+.

Since $(QR_N, \cdot)$ and $(QR^+, \circ)$ are isomorphic, it is proved $(QR_N, \cdot)$ means $(QR^+, \circ)$ has the same security.

Let random number $x \in QR_N$ and $y = x^{2^T} mod N$ in $(QR_N, \cdot)$, and $x' = |x|$ and $y' = (x' = |y|$ in $(QR_N^+, \circ)$, where $y' = |y|$ and $y = |y'|^{-1}$, and $y \in y', N - y'$. Although it is not certain whether $y = y'$ or $y = N - y'$, y has a $1/2$ probability of getting the correct value. This shows that given an algorithm that calculates x^{2^T} in QR_N^+ with probability ϵ in time T, it is possible to obtain an algorithm that calculates x^{2^T} in QR_N^+ in time when time T and probability $\epsilon/2$ are essentially the same.

$Sloth(N, (x, T)) \rightarrow (y, \pi)$ is a slow algorithm that takes the N and a random seed x and time delay parameter T as input and outputs a y together with π, where $y = x^{2^T}$ is the solution of the RSW time-lock puzzle in QR_N^+ and $\pi = \{u_i\}_i \in [T]$ is a possibly empty proof that y has been correctly evaluated. It is derived by applying the Fiat-Shamir heuristic [28,29] to the protocol.

$Verify(N, (x, T), (y, \pi)) \rightarrow \{accept, reject\}$ is a deterministic cryptographic algorithm to verify if x, y and all u_i are in QR_N^+. If these are not the case output *reject*. Otherwise, all x_i and y_i should be calculated, and $y_{T+1} \stackrel{?}{=} x_{T+1}^2$ should be judged. If all the above are satisfied, output *accept*.

3.2. Construction Based on Elliptic Curve Cryptography

De Feo et al. [30] designed a new verifiable delay function using isogenic and bilinear pairs [31,32] of super-singular elliptic curves [33], and this framework is non-interactive in nature, the output can be effectively verified without additional proofs. Before describing this structure, let's introduce some basic factors of super-singular curves, pairings and isogenies.

Elliptic curves on finite fields are described in detail in [34–36] and their use in cryptography is described in detail in [37–39]. In addition, the ideal class group of quadratic imaginary fields are explained in [40] and the maximal orders of quaternion algebras are introduced in [41,42].

Let C be an elliptic curve defined over a finite field $\mathbb{F}_q$ characterized by p and the order of $C(\mathbb{F}_q)$ is $\#C(\mathbb{F}_q) = q + 1 - L$, where L is the trace of the Frobenius endomorphism π. If and only when p divided L, the curve can be called a super-singular elliptic curve. Each super-singular curve is isomorphic to the curve defined on $\mathbb{F}_{p^2}$, and for the fixed prime number p, there are only a finite number of super-singular curves until isomorphism.

Weil pairing [43] $e_N : C[N] \times C[N] \rightarrow \mu_N$ with bilinear pairs is defined on super-singular curves are used to describe verifiable delay functions. That pairing needs to be satisfied the compatibility condition $e_N(\varphi(P), Q) = e_N(P, \tilde{\varphi}(Q))$ for any isogeny $\varphi : C \rightarrow C'$ and points $P \in C[N], Q \in C'[N]$.

Verifiable delay functions from super-singular curves. Let $X_1, X_2, Y_1, Y_2, \mathbb{G}$ be groups of prime order N. Let $e_X : X_1 \times X_2 \rightarrow \mathbb{G}$ and $e_Y : Y_1 \times Y_2 \rightarrow \mathbb{G}$ be non degenerate bilinear pairings, where a pair of bijections $\varphi : X_1 \rightarrow Y_1$ and $\tilde{\varphi} : Y_2 \rightarrow X_2$ and φ and $\tilde{\varphi}$ are group isomorphisms. Let $\mathfrak{g}$ be any generator of X_1 and $(N, X_1, X_2, Y_1, Y_2, \mathbb{G}, e_X, e_Y, \mathfrak{g}, \varphi(\mathfrak{g}))$ be the public parameters. The verifiable delay function is the map $\tilde{\varphi}$ and the maps $\varphi, \tilde{\varphi}$ are also part of the public parameters. To verify the output, one checks that $e_X(\mathfrak{g}, \tilde{\varphi}(Q)) = e_Y(\varphi(\mathfrak{g}), Q)$, where $Q \in Y_1$ is the point at which *Eval* calculates the value.

De Feo et al. propose verifiable delay functions for super-singular curves over prime field $\mathbb{F}_p$ and $\mathbb{F}_{p^2}$, using super-singular elliptic curves for the pairing groups, and isogenies of prime power degree for the maps φ, $\tilde{\varphi}$. Next, we mainly introduce verifiable delay functions with super-singular curves over a prime field $\mathbb{F}_p$.

Let p be prime so that $p + 1$ contains the larger prime factor N. Let degree $l = 2$, $p = 7 mod 8$ or a small prime such that $(\frac{-p}{l}) = 1$. Take the super-singular elliptic curve $C/\mathbb{F}_p$, and denote by $e_N(\cdot, \cdot)$ the Weil pairing on $C[N]$.

$Setup(\lambda, T) \rightarrow (ek, vk) = (\tilde{\varphi}, (C, C', \mathfrak{g}, \varphi(\mathfrak{g})))$ is a randomized algorithm that takes a delay parameter T and a security parameter λ as input. First, a super-singular curve $C/\mathbb{F}_p$

needs to be chosen and a direction needs to be chosen on the horizontal l-isogeny graph to compute a cyclic isogeny $\varphi : C \to C'$ of degree l^T and its dual $\tilde{\varphi}$ in this algorithm. Next, the algorithm chooses a generator $\mathfrak{g}$ of $X_1 = v^{-1}(\tilde{C}[N] \cap \tilde{C}(\mathbb{F}_p)))$ and compute $\varphi(\mathfrak{g})$, where $\mu \in \mathbb{F}_p$ is a non quadratic residue and $\tilde{C}$ is a quadratic twist of C. Finally, the algorithm outputs public parameters $(ek, vk) = (\tilde{\varphi}, (C, C', \mathfrak{g}, \varphi(\mathfrak{g})))$, where (ek, vk) composed of the evaluation key ek and the verification key vk, and $(\tilde{\varphi}, (C, C', \mathfrak{g}, \varphi(\mathfrak{g})))$ composed of the map $\tilde{\varphi}$, the cyclic isogeny $C \to C'$, generator of X_1 and the map $\varphi(\mathfrak{g})$ of generator $\mathfrak{g}$.

$Eval(\tilde{\varphi}, Q \in Y_1) \to \tilde{\varphi}(Q)$ is a slow cryptographic algorithm that takes the map $\tilde{\varphi}$ and a point $Q \in Y_1$ as input and outputs $\tilde{\varphi}(Q)$.

$Verify(C, C', \mathfrak{g}, Q, \varphi(\mathfrak{g}), \tilde{\varphi}(Q)) \to \{accept, reject\}$ is a deterministic cryptographic algorithm. The algorithm needs to verify that $\tilde{\varphi}(Q) \in X_2 = C[N] \cap C(\mathbb{F}_p)$ and $e_N(\mathfrak{g}, \tilde{\varphi}(Q)) = e_N(\varphi(\mathfrak{g}), Q)$. If all the above are satisfied, output *accept*; Otherwise, output *reject*.

4. Verifiable Delay Functions Based on Structural Assumptions

Ephraim et al. design continuous verifiable delay functions based on a high arity tree [44], where each intermediate state of the computation can be verified and proofs of the computation can be efficiently merged. It is a verifiable delay function based on the assumption of tree structure constructed on the basis of the repeated square [45] assumption. The continuous verifiable delay function only depends on the Fiat-Shamir heuristic for a constant round proof system. Next, we introduce continuous verifiable delay functions based on high arity trees.

First, the computational steps correspond to a specific traversal of a $(k+1)$-ary tree of height $h = log_k^B$. Each node in the $(k+1)$-ary tree is related to a statement (x, y, T, π) of the underlying verifiable delay functions, where the output value $y = x^{2^T}$ and the possible empty proof π are the corresponding proofs of correctness. If x is the node's input, the difficulty $T = k^{h-l}$ is determined by its height in $(k+1)$-ary tree and l is root node.

Next, they define a tree structure. Starting from the root node with input x_0 and difficulty $T = k^h$, divide the tree structure into k segments $x_1, x_2, \ldots, x_k$ similar to the verifiable delay functions structure. In a tree-based verifiable delay functions structure, only calculating the input x of a leaf node from the previous state can guarantee that each step of calculation corresponds to the calculation of a single leaf's statement.

Before introducing continuous verifiable delay functions, let's review unique verifiable delay functions. Next, the interactive proof that language $\mathcal{L}_{N,B}$ corresponding to repeated squares is transformed into unique verifiable delay functions by using the Fiat-Shamir heuristic, where

$$\mathcal{L}_{N,B} = \left\{ (x_0, y, T) : \begin{array}{c} y^2 = (x_0)^{2^{T+1}} \bmod N, x_0 \text{ is valid and } T \leq B_3 \\ y = \perp, \text{othersize} \end{array} \right\} \tag{5}$$

A unique verifiable delay function is composed of the following four algorithm $(uVDF.Gen, uVDF.Sample, uVDF.Eval, uVDF.Verify)$.

$uVDF.Gen(1^\lambda) \to pp = (N, B, k, d, hash)$ is a randomized algorithm that takes a statistical security parameter 1^λ as input and outputs public parameters pp composed of the RSA modulus N, the upper bound B, the arity k, a constant d and a hash function $hash$, where $hash \leftarrow \mathcal{H}$, $k = \lambda$ and $B = B(\lambda)$.

$uVDF.Sample(1^\lambda, pp) \to x_0$ takes a statistical security parameter 1^λ and the public parameters pp and sample and output a random element $x_0 \leftarrow Z_N^*$ such that $gcd(x_0 \pm 1, N) = 1$ and $x_0 = |x_0|$.

$uVDF.Eval(1^\lambda, pp, (x_0, T)) \to (y, \pi)$ is a slow cryptographic algorithm that takes a statistical security parameter 1^λ, the public parameters pp, a random element x_0 and the time delay parameter T as input and outputs a y together with a possibly empty proof π.

$uVDF.Verify(1^\lambda, pp, (x_0, T), (y, \pi)) \to \{0, FS - Verify(pp, (x_0, T), (y, \pi))\}$ is a deterministic cryptographic algorithm. If all the above are satisfied, output *accept*. Otherwise, output $FS - Verify(pp, (x_0, T), (y, \pi))$. The $FS - Verify$ is a verification algorithm for

Fiat-Shamir transformations defined on the protocol for language $\mathcal{L}_{N,B}$ relative to some hash family $\mathcal{H}$. For details of the algorithm, see [46]. Then, we review the definition of a continuous verifiable delay function and describe it in detail.

Definition 5. *(Continuous verifiable delay functions.) Let* $B, l : \mathbb{N} \to \mathbb{N}$ *and* $\epsilon \in (0,1)$. *A* (B, l, ϵ)*-continuous verifiable delay function is a tuple* $(cVDF.Gen, cVDF.Sample, cVDF.Eval, cVDF.verify)$ *such that* $(cVDF.Gen, cVDF.Sample, cVDF.Eval)$ *is a* (a, B, l, ϵ)*-iteratively sequential function,* $(cVDF.Eval, cVDF.verify)$ *is a B-sound function.*

At a high level, the continuous verifiable delay function will iteratively compute each leaf node in a (pp_{uVDF}, d', g)-puzzle tree, where $pp_{uVDF} = (N, B, k, d, hash)$ are the public parameters of the underlying unique verifiable delay function and g is the starting point of the tree given by $uVDF.Sample$.

Next, we define a frontier. At a high level, for a leaf s, the frontier of s will correspond to the state of the continuous verifiable delay function upon reaching s. Specifically, it will contain all nodes whose values have been computed at that point, but whose parents' values have not yet been computed.

Definition 6. *(Frontier.) For a node* s *in a* (pp_{uVDF}, d', g)*-puzzle tree, the frontier of* s*, denoted* $frontier(s)$*, is the set of pairs* $(s', val(s'))$ *for nodes* s' *that are left siblings of any of the ancestors of* s*. We note that* s *is included as one of its ancestors.*

Next, we review the formal details of continuous verifiable delay functions, which is a tuple $(cVDF.Gen, cVDF.Sample, cVDF.Eval, cVDF.Verify)$.

$cVDF.Gen(1^\lambda) \to pp = (pp_{uVDF}, d', hgt)$ is a randomized algorithm that takes a statistical security parameter 1^λ as input and outputs public parameters pp composed of the $pp_{uVDF} = (N, B, k, d, hash)$, a constant d' and a tree height $hgt = \lceil log_k(B) \rceil - d'$.

$cVDF.Sample(1^\lambda, pp) \to v = (g, 0^h, \phi)$ takes a statistical security parameter 1^λ and public parameters pp as input and outputs a random element v, where $g \leftarrow uVDF.Sample(1^\lambda, pp_{uVDF})$ is sampled by the $Sample$ algorithm of unique verifiable delay functions.

$cVDF.Eval(1^\lambda, pp, v) \to v'$ takes a state v corresponding to a leaf s in the tree and computes the next state v' corresponding to the next leaf. Each state v will be of the form (g, s, F), where s is the current leaf in the tree, F is the frontier of s and g is the starting point of the tree given by $uVDF.Sample$.

$cVDF.Verify(1^\lambda, pp, (v, T), v') \to \{accept, reject\}$ verifies the state v by recursively running this verification algorithm and whether v' is correct. Output *accept* if all the check conditions of the continuous verifiable delay function are satisfied; Otherwise output *reject*.

The heart of our construction is the $cVDF.Eval$ functionality which takes a state v corresponding to a leaf s in the tree and computes the next state v' corresponding to the next leaf. Each state v will be of the form (g, s, F), where s is the current leaf in the tree and F is the frontier of s. Then, $cVDF.Eval(1^\lambda, pp, (g, s, frontier(s))$ will output $(g, s+1, frontier(s+1))$. There are three phases of the algorithm $cVDF.Eval$. First, it checks that its input is well-formed. It then computes $val(s)$ using $frontier(s)$, and then computes $frontier(s+1)$ using both $frontier(s)$ and $val(s)$.

5. Post-Quantum Verifiable Delay Functions

In 2021, Jorge Chavez-Saab et al. [47] researched the problem of constructing post-quantum secure verifiable delay functions, especially verifiable delay functions based on super-singular isogeny. They propose an arithmetic structure specifically for homologous settings using succinct non-interacting arguments (SNARGs) [48] to achieve good asymptotic efficiency. This isogeny-based verifiable delay functions has the advantages of post-quantum security [49], quasi-logarithmic verification, and does not require a trusted setup. Since the *Eval* algorithm for postquantum verifiable delay functions involves isogeny walks on super-singular elliptic curves that can be publicly verified through a SNARG-

based verification process. Formally, a verifiable delay function is composed of three main algorithm:

$Setup(\lambda, T) \rightarrow pp = (ek, vk)$ takes a delay parameter T and a security parameter λ as input and outputs public parameters pp composed of the evaluation key ek and the verification key vk.

$Eval(ek, x) \rightarrow (y, \pi)$ takes the evaluation key ek and a certain input x as input and calculates an output y and a possibly empty proof π.

The function involves computing walks of length T in the 2-isogeny graph of super-singular curves on $\mathbb{F}_{p^2}$, where $p^2 \equiv 9 mod 16$ (which is required to apply Kong's square root algorithm [50]) and $p = poly(T)$. Given a time delay parameter T, and the evaluator needs to compute a walk of length T on the 2-isogeny graph, where the exact path is determined by a string s and the path is not backtracking. Given the two v-invariant curves v_i and v_{i+1}, they are 2-isogenous over $\mathbb{F}_{p^2}$ if and only if the modular polynomial $\Phi(v_i, v_{i+1})$ vanishes. For a fixed v_i the next curve in the path can be calculated by finding the root of $\Phi(v_i, A)$. If $A = v_{i-1}$ is a known root of $\Phi(A) = A^3 + aA^2 + bA + c$ then $\Phi(A)$ can rewrite $\Phi(A) = (A - v_{i-1})(A^2 + (a + v_{i-1})A + b + av_{i-1} + v_{i-1}^2)$ and focus on finding the roots of the quadratic factor. After the square root is calculated, the evaluator selects the symbol using the input string, resulting in a definite traversal process.

$Verify(vk, x, y, \pi) \rightarrow \{accept, reject\}$ inputs verification key vk, a certain input x, a output y and a proof π. If $f(x) = y$, output *accept*; Otherwise output *reject*.

Since the postquantum verifiable delay functions is constructed over SNARG, a deterministic process and a fixed symbol are required for the SNARG verification process. For the validation process, the evaluator keeps track the results of the evaluation and construct an SNARG, and the values resulting from the evaluation process must be satisfied.

De-Feo et al. proposed a new verifiable delay function framework based on the assumption of elliptic curve cryptography, and instantiated this framework using super-singular elliptic curves and bilinear pairs. The structure of this verifiable delay function is non-interactive in nature, and the output can be effectively verified without additional proofs. However, the only secure way to instantiate a verifiable delay function requires a trusted setup to perform a random isogeny traversal. In fact, this setup needs to start with super-singular elliptic curves with unknown autohomomorphic rings. In order to explain how to implement the proposed verifiable delay function on elliptic curves with commutative self-homomorphic rings, Shani later used the idea of verifiable delay functions based on isogeny and pairing proposed by De-Feo et al. to construct developed a verifiable delay function based on isogeny without pairing. The scheme is a combination of a time-lock puzzle and a trapdoor verifiable delay function.

However, neither scheme is quantum secure. Thus, Chavez-Saab et al. studied the problem of constructing post-quantum secure verifiable delay functions, especially verifiable delay functions based on super-singular isogeny. They propose an arithmetic structure specifically for homologous settings using SNARGs to achieve good asymptotic efficiency. This isogeny-based verifiable delay function has the advantages of post-quantum security, quasi-logarithmic verification, and does not require a trusted setup.

This verifiable delay function construction finds codomain curves, which are computed from any three-point evaluation, so the problem in the verifiable delay function setting could be regarded as a general problem. The precomputation time allowed in the setting is given before learning the isogeny to be evaluated, suggesting that this verifiable delay function construction uses a different isogeny for each input. This verifiable delay function relies on a very weak assumption, so it is more secure. Starting from a public curve means we do not need a trusted setup. Next, we analyze two security properties of post-quantum verifiable delay functions based on super-singular elliptic curves: sequentiality and soundness.

Sequentiality. Any protocol that does not prescribe isogeny walk in some way is insecure in terms of sequentiality. The evaluator can be asked to provide SNARG proof of any large degree of isogeny and consider this to be a good proof of sequentiality even if

the output is not unique. However, if the evaluator is free to choose the path, this does not constitute a proof of sequential computation. The sequentiality of post-quantum verifiable delay functions relies on isogeny shortcut problem of De Feo et al.. If no pair of random algorithms $\mathcal{A}_0$ (running in total time $poly(T, \lambda)$) and $\mathcal{A}_1$ (running in parallel time less than T) can win the following sequential game with non-negligible probability, then the post-quantum verifiable delay functions based on super-singular elliptic curves is sequential. The $Setup$ must use secret randomness to choose the starting curve, and the $Setup$ is left with choosing isogeny and generators, both of which can use public randomness. Furthermore, $\mathcal{A}_0$ allows for $poly(T)$ computations, so it can compute isogeny on generators. Therefore, setting aside the choice of starting curve, the $Setup$ can be absorbed into $\mathcal{A}_0$, which proves that the verifiable delay function of the post-quantum is sequential

Soundness. The soundness of post-quantum verifiable delay functions based on super-singular elliptic curves completely depends on the SNARG proofs. Succinctness and non-interactiveness are achieved by generating SNARG through a transform that acts on any probabilistic checkable proofs. As long as the hash function is collision-resistant, it is straightforward to prove that the soundness of the structure reduces to the soundness of the original probabilistic checkable proofs. Therefore, the post-quantum verifiable delay functions construction based on super-singular elliptic curves is sufficiently soundness.

6. Verifiable Delay Functions Applications

With the proposal of more and more verifiable delay function schemes, the application of verifiable delay function in distributed systems is also well known. Next, as shown in Figure 2, this paper introduces several particularly important applications. In Table 3, the blockchain-related applications of verifiable delay functions are described.

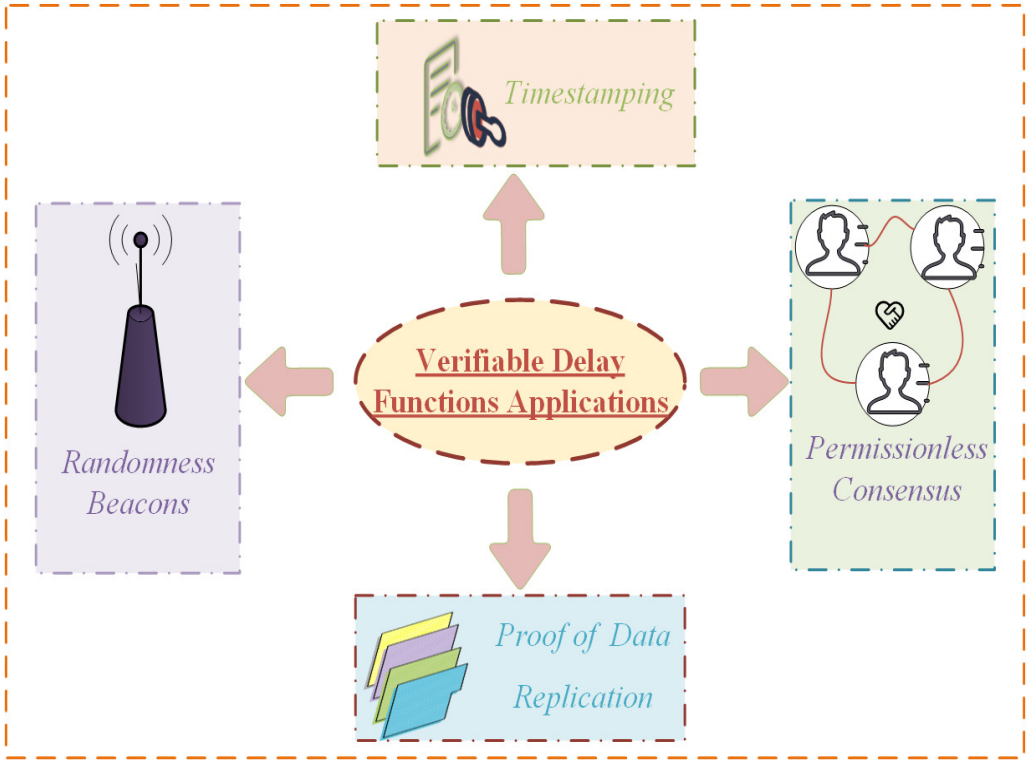

Figure 2. Verifiable delay functions applications.

Computational Timestamping. Almost all know proof-of-stake [51] systems face the problem of long-range attack [52]. In a proof-of-stake protocol, a group of stakeholders has voting rights proportional to their stake at any time. Assume that the majority of stakeholders have no reason to break the system, since the stakeholders themselves are incentivized. However, when too many stakeholders sell their stake, they can collude to

create a new longer system to replace history. An external timestamp mechanism [53] can prove to the user that the true history of the system is much older.

Incremental verifiable delay functions can provide computational evidence that a given version of the state system is older by proving that long-running verifiable delay function computations has been performed on the true history after divergence from the fraud history. This has the potential to detect long-range attack without relying on external timestamping mechanisms. In 2019, Landerreche et al. [54] presented the first treatment of non-interactive publicly verifiable timestamping schemes and a secure timestamping scheme based on a verifiable delay function is proved. The timestamping scheme [55] consists of sequence of verifiable delay function proofs linked to each other by a cryptographic hash function, modeled as a sequence of random oracles. Add verifiable delay functions to the sequence to increase the structure, thus preserving the safety guarantees of the structure.

Table 3. Blockchain-related applications of verifiable delay functions.

Applications	Solution	Purpose
Timestamping	A verifiable delay function is equivalent to a proof of the passage of time, with the input and output of a verifiable delay function on-chain to prove the history of a given block.	Mitigating long-range attacks.
Randomness Beacon	The time delay parameter T of the verifiable delay function is set long enough, and the latest block header is used as part of the input in the verifiable delay function, and the final output is the random beacon result.	Enhancing the security of public verifiable random numbers.
Permissionless Consensus	Combine proofs-of-resource with incremental verifiable delay functions and use the product of resource proved and delay induced as a measure of blockchain quality.	Solving nothing-at-stake attacks.
Proof of Replication	Decodable verifiable delay functions generate multiple puzzles, and then the solutions of these puzzles are combined with all replicas to generate new replicas.	Preventing dynamic generation of replicas.

Public Randomness Beacons. Verifiable delay functions are useful for methods of obtaining random numbers from public sources. For example, constructing public randomness beacons [56] from stock prices, election audit or proof-of-work blockchains [57]. In the stock market, a strong enough stock trader can change the random output of stock prices by influencing the market trend, which greatly affects the fairness of the stock market [58]. The verifiable delay function can increase the security of public verifiable nonces by adding a long enough delay to calculate the beacon, which helps ensure that malicious traders do not have enough time to adjust the market. In a proof-of-work blockchain's computational puzzle solving, miners continuously mine and publish puzzles for monetary rewards. However, similarly powerful enough stock traders may manipulate stock prices, sufficiently powerful miners may manipulate beacon results by refusing to publish blocks [59] that produce unfavorable beacon outputs.

In addition, the verifiable delay function can also add some random beacon schemes involving multiple parties. For example, in "commit-and-reveal" [60], the attacker can wait until the end of the reveal phase to decide whether to reveal his or her commitment. If the following three conditions are met: discard the commit phase; integrate everyone's input at the end of the protocol and put it into the verifiable delay function instead of directly as the result of a random number; set the time parameter T long enough and later than the deadline for the last submission; then even the last-minute submitter has no way of knowing the result of the random number.

Resource Efficient Blockchains. When the blockchain [61] is forked, the consensus participants will choose to mortgage assets on different forked chains to participate in the block generation for their own interests, so that the forked chain may always exist and there will be more and more forks. Seriously endanger the consistency of the system and the consumption of resources. However, resource-efficient mining suffers from nothing-at-stake attacks. Intuitively, since mining is not computationally expensive, miners can easily try to produce many individual forks.

To prevent nothing-at-stake attacks [62], random beacons are used to select a new leader at intervals. At the same time, a verifiable delay function can also be used to increase the security of random beacons in the consensus protocol that uses random beacons to select a new leader. Most of the random number schemes used by these protocols remain secure only when there is a majority of honest participants. Utilizing a verifiable delay function can improve the participation of any honest party.

In addition to the electoral scheme, Cohen proposes to combine proof of resources [63] with an incremental verifiable delay function and use the product of proven resources and induced delay as a measure of blockchain [64,65] quality. This scheme prevents nothing-at-stake attacks by simulating the proof-of-work mining process. The timing of mining blocks is unpredictable, and each miner competes with each other to be the first to mine a block. The difference from the proof-of-work [66] is that this scheme does not actually need to consume too much time resources for parallel computing, and only has certain space resources when entering mining.

Proof of Data Replication. Proof of data replication [67–69] is a special type of proof of storage of data that allows a client to verify that it has a unique replica of some data stored on an untrusted server, even if the data is readily available from a public storage source. Proof of data replication is meant to prove that the server has a replica of the data, not that it owns the data. Boneh et al. proposed to provide a publicly verifiable solution using a decodable verifiable delay function that is asymmetric in time. The decodable verifiable delay function prevents the server from dynamically computing the client's replica when challenged to prove that it correctly stored a replica of the data.

Gritti proposes a publicly verifiable proof of data replication scheme using verifiable delayed functions, and explains the scheme in detail along with a security proof. Given a unique replicator identifier id. Then the server divides the file into b-bit sized file blocks Bi and calculate $B_{id} = B_i(i||id)$ where H is a collision-resistant hash function $H : \{0,1\} \rightarrow \{0,1\}^b$ and $i \in \{1, \ldots, n\}$. The output value y_i can be obtained by taking the calculated B_{id} as part of the $Eval$ algorithm input of the decodable verifiable delay function. The server stores all y_i and the client continuously randomly selects i to challenge the server to return y_i. If the server can respond to the corresponding result to the client within the specified time and the client can obtain B_i by decoding y_i in a very short time to complete the verification. If the server does not respond to the client correctly, y_i must be calculated in T steps, but this calculation cannot be completed within the specified time.

Verifiable delay functions are widely used in blockchains. However, verifiable delay functions based on finite groups of unknown order are insecure against an adversary with access to a quantum computer. Quantum computers can easily compute the order of a group using Shor's algorithm, making it easy to break the application of verifiable delay functions in blockchains. In addition, the post-quantum secure verifiable delay function based on supersingular elliptic curves needs to be verified by SNARG, and the structure

is in its infancy and can only be limited to the field of elliptic curves. Therefore, it is an open problem to develop a verifiable delay function that has a simple structure in quantum computers and can be safely applied to blockchain.

7. Conclusions and Perspectives

Verifiable delay function is a basic and important tool in the field of cryptography, and it has been widely used in distributed systems. This article firstly introduces the types of verifiable delay functions, and describes the construction schemes of different kinds of verifiable delay functions in detail. Secondly, in the algebraic assumption scheme, verifiable delay function schemes based on finite abelian groups of unknown order and super-singular elliptic curve cryptography scheme are introduced respectively, both of which are not quantum secure. Thirdly, the unique verifiable delay functions and the continuous verifiable delay functions based on tree-structure assumptions are introduced. The continuous verifiable delay function achieves effective verification of the output of each intermediate iteration through continuous iteration, and can efficiently incorporate proofs of final computation. Fourthly, this paper presents a postquantum secure verifiable delay function scheme based on super-singular isogeny, And the security analysis of the verifiable delay function is carried out. Finally, the application of verifiable delay function in different aspects of the blockchain is summarized.

Verifiable delay functions have been extensively studied, and the research results obtained cover various branches of verifiable delay function research. However, the application of the verifiable delay function in the existing research results has not been fully integrated into the blockchain, and the application in the blockchain still needs further research. We hope that this work will stimulate new practical applications of verifiable delay functions in blockchains and continue to investigate theoretically optimal verifiable delay function constructions in the future.

Author Contributions: Q.W., Y.R. and S.J. conceived the mechanism design and wrote the paper, S.W. (Shenqing Wang) built the models. L.X. and S.W. (Shiren Wang) developed the mechanism, Y.R. and S.J. revised the manuscript. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Key R&D Program of China (Grant No. 2021YFB2700500), and it was also supported by the National Natural Science Foundation of China (Grant No. 62072249). This work was also supported by the National Key R&D Program of Guangdong Province (Grant No. 2020B0101090002), and the Natural Science Foundation of Jiangsu Province (Grant No. BK20200418, BE2020106). Shan Ji is the corresponding authors.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Boneh, D.; Bonneau, J.; Bünz, B.; Fisch, B. Verifiable delay functions. In Proceedings of the Annual International Cryptology Conference, Santa Barbara, CA, USA, 18–22 August 2018; pp. 757–788.
2. Rotem, L. Simple and efficient batch verification techniques for verifiable delay functions. In Proceedings of the Theory of Cryptography Conference, Raleigh, NC, USA, 8–11 November 2021.
3. Zhou, M.; Lin, X.; Liu, A.; Che, Y. An improved blockchain consensus protocol with distributed verifiable delay function. In Proceedings of the 2021 IEEE International Conference on Electronic Technology, Communication & Information, Changchun, China, 27–29 August 2021; pp. 330–337.
4. Öztürk, E. Design and implementation of a low-latency modular multiplication algorithm. *IEEE Trans. Circuits Syst.* **2020**, *67*, 1902–1911. [CrossRef]
5. Lombardi, A.; Vaikuntanathan, V. Fiat-Shamir for repeated squaring with applications to PPAD-hardness and VDFs. In Proceedings of the 40th Annual International Cryptology Conference, Santa Barbara, CA, USA, 17–21 August 2020; pp. 632–651.

6. Döttling, N.; Garg, S.; Malavolta, G.; Vasudevan, P.N. Tight verifiable delay functions. In Proceedings of the 12th International Conference on Security and Cryptography for Networks, Amalfi, Italy, 14–16 September 2020; pp. 65–84.
7. Raghunandan, K.R.; Aithal, G.; Shetty, S. Comparative analysis of encryption and decryption techniques using mersenne prime numbers and phony modulus to avoid factorization attack of RSA. In Proceedings of the 2019 International Conference on Advanced Mechatronic Systems, Kusatsu, Japan, 26–28 August 2019; pp. 152–157.
8. Boneh, D.; Benedikt, B.; Ben, F. A survey of two verifiable delay functions. In Proceedings of the International Association for Cryptologic Research, Brisbane, QLD, Australia, 2–6 December 2018; pp. 712–725.
9. Medley, L.; Quaglia, E.A. Collaborative verifiable delay functions. In Proceedings of the 17th International Conference on Information Security and Cryptology, Virtual Event, 12–14 August 2021; pp. 507–530.
10. Blanc, J.; Canci, J.K.; Elkies, N.D. Moduli spaces of quadratic rational maps with a marked periodic point of small order. *Int. Math. Res. Not.* **2015**, *2015*, 12459–12489. [CrossRef]
11. Moradi, M. On sequential decoding metric function of polarization-adjusted convolutional (PAC) codes. *IEEE Trans. Commun.* **2021**, *69*, 7913–7922. [CrossRef]
12. Valiant, P. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In *Theory of Cryptography, Proceedings of the Fifth Theory of Cryptography Conference, New York, NY, USA, 19–21 March 2008*; Springer: Berlin/Heidelberg, Germany, 2008; pp. 1–18.
13. Bitansky, N.; Canetti, R.; Chiesa, A.; Tromer, E. Recursive composition and bootstrapping for SNARKs and proof-carrying data. In Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing, Palo Alto, CA, USA, 2–4 June 2013; pp. 111–120.
14. Gritti, C. Publicly verifiable proofs of data replication and retrievability for cloud storage. In Proceedings of the 2020 International Computer Symposium Conference, Tainan, Taiwan, 17–19 December 2020; pp. 431–436.
15. Abadi, A.; Kiayias, A. Multi-instance publicly verifiable time-lock puzzle and its applications. In Proceedings of the 25th International Conference on Financial Cryptography and Data Security, Virtual Event, 1–5 March 2021; pp. 541–559.
16. Burdges, J.; Feo, L.D. Delay encryption. In Proceedings of the 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, 17–21 October 2021; pp. 302–326.
17. Ren, Y.J.; Zhu, F.J.; Kumar, S.P.; Wang, T.; Wang, J. Data query mechanism based on hash computing power of blockchain in Internet of Things. *Sensors* **2020** *20*, 207. [CrossRef]
18. Wesolowski, B. Efficient verifiable delay functions. *J. Cryptol.* **2020**, *33*, 2113–2147. [CrossRef]
19. Santos, R.G.; Machovsky-Capuska, G.E.; Andrades, R. Plastic ingestion as an evolutionary trap: Toward a holistic understanding. *Science* **2021**, *373*, 56–60. [CrossRef]
20. Isfandbod, M.; Martínez-Pañeda, E. A mechanism-based multi-trap phase field model for hydrogen assisted fracture. *Int. J. Plast.* **2021**, *144*, 103044. [CrossRef]
21. Raikwar, M.; Gligoroski, D. R3V: Robust round robin VDF-based consensus. In Proceedings of the 2021 3rd Conference on Blockchain Research & Applications for Innovative Networks and Services, Paris, France, 27–30 September 2021; pp. 81–88.
22. Jiang, P.; Qiu, B.; Zhu, L. Toward reliable and confidential release for smart contract via ID-based TRE. *IEEE Internet Things J.* **2022**, *9*, 11422–11433. [CrossRef]
23. Acharya, J.; Canonne, C.L.; Tyagi, H. Inference under information constraints II: Communication constraints and shared randomness. *IEEE Trans. Inf. Theory* **2020**, *66*, 7856–7877. [CrossRef]
24. Pietrzak, K. Simple verifiable delay functions. In Proceedings of the 10th Innovations in Theoretical Computer Science Conference, San Diego, CA, USA, 10–12 January 2019; pp. 1–15.
25. Pan, R.; Abel, R.J.R.; Bunjamin, Y.A. Difference matrices with five rows over finite abelian groups. *Des. Codes Cryptogr.* **2022**, *90*, 367–386. [CrossRef]
26. Li, F.; Yue, Q.; Wu, Y. LCD and self-Orthogonal group codes in a finite abelian p -group algebra. *IEEE Trans Inf. Theory* **2020**, *66*, 2717–2728. [CrossRef]
27. Hong, S.; Park, H.; No, J.S.; Helleseth, T.; Kim, Y.S. Near-optimal partial hadamard codebook construction using binary sequences obtained from quadratic residue mapping. *IEEE Trans Inf. Theory* **2014**, *60*, 3698–3705. [CrossRef]
28. Bettaieb, S.; Bidoux, L.; Blazy, O.; Gaborit, P. Zero-knowledge reparation of the véron and AGS code-based identification schemes. In Proceedings of the 2021 IEEE International Symposium on Information Theory, Melbourne, VIC, Australia, 12–20 July 2021; pp. 55–60.
29. Liu, Y.; Zhou, Y.; Sun, S.; Wang, T.; Zhang, R.; Ming, J. On the security of lattice-based Fiat-Shamir signatures in the presence of randomness leakage. *IEEE Trans. Inf. Forensics Secur.* **2021**, *16*, 1868–1879. [CrossRef]
30. De-Feo, L.; Masson, S.; Petit, C.; Sanso, A. Verifiable delay functions from super-singular isogenies and pairings. In Proceedings of the International Conference on the Theory & Application of Cryptology & Information Security, Kobe, Japan, 8–12 December 2019; pp. 248–277.
31. Ali, I.; Chen, Y.; Ullah, N.; Afzal, M.; Wen, H.E. Bilinear pairing-based hybrid signcryption for secure heterogeneous vehicular communications. *IEEE Trans. Veh. Technol.* **2021**, *70*, 5974–5989. [CrossRef]
32. Ren, Y.J.; Leng, Y.; Cheng, Y.P.; Wang, J. Secure data storage based on blockchain and coding in edge computing. *Math. Biosci. Eng.* **2019**, *16*, 1874–1892. [CrossRef]
33. Onuki, H. On oriented super-singular elliptic curves. *Finite Fields Their Appl.* **2021**, *69*, 101777. [CrossRef]

34. Saouter, Y. Constructions of LDPCs from Elliptic Curves over finite fields. *IEEE Commun. Lett.* **2017**, *21*, 2558–2561. [CrossRef]
35. Sutter, G.D.; Deschamps, J.; Imana, J.L. Efficient Elliptic Curve point multiplication using digit-serial binary field operations. *IEEE Trans. Ind. Electron.* **2013**, *60*, 217–225. [CrossRef]
36. Merad Boudia, O.R.; Senouci, S.M.; Feham, M. Elliptic Curve-Based Secure Multidimensional Aggregation for Smart Grid Communications. *IEEE Sens. J.* **2017**, *17*, 7750–7757. [CrossRef]
37. Wang, J.; Li, J.; Wang, H.; Zhang, L.Y.; Cheng, L.M.; Lin, Q. Dynamic scalable Elliptic Curve cryptographic scheme and its application to in-vehicle security. *IEEE Internet Things J.* **2019**, *6*, 5892–5901. [CrossRef]
38. Azarderakhsh, R.; Reyhani-Masoleh, A. Parallel and high-speed computations of Elliptic Curve cryptography using hybrid-double multipliers. *IEEE Trans. Parallel Distrib. Syst.* **2015**, *26*, 1668–1677. [CrossRef]
39. Mehrabi, M.A.; Doche, C.; Jolfaei, A. Elliptic Curve cryptography point multiplication core for hardware security module. *IEEE Trans Comput.* **2020**, *69*, 1707–1718. [CrossRef]
40. Lyu, S.; Porter, C.; Ling, C. Lattice reduction over imaginary quadratic fields. *IEEE Trans. Signal Process.* **2020**, *68*, 6380–6393. [CrossRef]
41. Mushtaq, E.; Ali, S.; Hassan, S.A. On decoupled decoding of quasi orthogonal STBCs using quaternion algebra. *IEEE Syst. J.* **2019**, *13*, 1580–1586. [CrossRef]
42. Thomas, F. Approaching dual quaternions from matrix algebra. *IEEE Trans. Robot.* **2014**, *30*, 1037–1048. [CrossRef]
43. Laurian, A.G.; Emmanuel, F.; Nadia, E.M.; Aminatou, P.N. Faster beta Weil pairing on BLS pairing friendly curves with odd embedding degree. *Math. Comput. Sci.* **2022**, *16*, 1–23. [CrossRef]
44. Mann, Z.Á. A comment on "Process placement in multicore clusters: Algorithmic issues and practical techniques". *IEEE Trans. Parallel Distrib. Syst.* **2016**, *27*, 2475–2476. [CrossRef]
45. Nguyen, H.; Nguyen, T.M.N.C.; Nguyen, L.; Custovic, E. An FPGA-based implementation for repeated square-and-multiply polynomials. In Proceedings of the 7th International Conference on Broadband Communications and Biomedical Applications, Melbourne, VIC, Australia, 21–24 November 2011; pp. 173–178.
46. Ephraim, N.; Freitag, C.; Komargodski, I.; Pass, R. Continuous verifiable delay functions. In Proceedings of the 39th Annual International Conference on the Theory & Applications of Cryptographic Techniques, Zagreb, Croatia, 10–14 May 2020; pp. 125–154.
47. Chávez-Saab, J.; Rodríguez-Henríquez, F.; Tibouchi, M. Verifiable isogeny walks: Towards an isogeny-based postquantum VDF. In Proceedings of the International Conference on Selected Areas in Cryptography, Virtual Event, 1–5 March 2021; pp. 441–460.
48. Parno, B.; Howell, J.; Gentry, C.; Raykova, M. Pinocchio: Nearly practical verifiable computation. *Commun. ACM* **2016**, *59*, 103–112. [CrossRef]
49. Akleylek, S.; Soysaldı, M.; Lee, W.K.S.; Hwang, O.; Wong, D.C.K. Novel Postquantum MQ-based signature scheme for Internet of things with parallel implementation. *IEEE Internet Things J.* **2021** *8*, 6983–6994. [CrossRef]
50. Kong, F.; Cai, Z.; Jia, Y.; Li, D. Improved generalized Atkin algorithm for computing square roots in finite fields. *Inform. Process. Lett.* **2006**, *98*, 1–5. [CrossRef]
51. Ren,Y. J.; Zhu, F.J; Wang, J.; Sharma, P.; Ghosh, U. Novel vote scheme for decision-making feedback based on blockchain in internet of vehicles. *IEEE Trans. Intell. Transp. Syst.* **2022**, *23*, 1639–1648. [CrossRef]
52. Feng, X.; Ma, J.; Miao, Y.; Liu, X.; Choo, K.K.R. Regulatable and hardware-based proof of stake to approach nothing at stake and long range attacks. *IEEE Trans. Serv. Comput.* **2022**, *1*, 1–12. [CrossRef]
53. Clarke, M.; Schluter, P.; Reinhold, B.; Reinhold, B. Designing robust and reliable timestamps for remote patient monitoring. *IEEE J. Biomed. Health Inform.* **2015**, *19*, 1718–1723. [CrossRef]
54. Landerreche, E.; Stevens, M.; Schaffner, C. Non-interactive cryptographic timestamping based on verifiable delay functions. In Proceedings of the 24th International Conference on Financial Cryptography & Data Security, Kota Kinabalu, Malaysia, 10–14 February 2020; pp. 541–558.
55. Wu, Q.; Han, Z.; Mohiuddin, G.; Ren, Y.J. Distributed timestamp mechanism based on verifiable delay functions. *Comput. Syst. Sci. Eng.* **2023**, *44*, 1633–1646. [CrossRef]
56. Schindler, P.; Judmayer, A.; Hittmeir, M.; Stifter, N.; Weippl, E. RandRunner: Distributed randomness from trapdoor VDFs with strong uniqueness. In Proceedings of the 2020 Network & Distributed System Security Symposium Conference, San Diego, CA, USA, 23–26 February 2020; pp. 21–25.
57. Ren, Y.Y.; Leng, Y.; Qi, J.; Pradip, K.S.; Wang, J. Multiple cloud storage mechanism based on blockchain in smart homes. *Future Gener. Comput. Syst.* **2021**, *115*, 304–313. [CrossRef]
58. Pierrot, C.; Wesolowski, B. Malleability of the blockchain's entropy. *Cryptogr. Commun.* **2018**, *10*, 211–233. [CrossRef]
59. Gueron, S.; Persichetti, E.; Santini, P. Designing a practical code-based signature scheme from zero-knowledge proofs with trusted setup. *Cryptography* **2022**, *6*, 5–22. [CrossRef]
60. Zhang, H.; Tong, L.; Yu, J.; Lin, J. Blockchain-aided privacy-preserving outsourcing algorithm of bilinear pairings for Internet of things devices. *IEEE Internet Things J.* **2021**, *8*, 15596–15607. [CrossRef]
61. Toyoda, K.; Zhang, A.N. Mechanism design for an incentive-aware blockchain-enabled federated learning platform. In Proceedings of the 2019 IEEE International Conference on Big Data (Big Data), Los Angeles, CA, USA, 9–12 December 2019; pp. 395–403.

62. Li, A.; Wei, X.H.; He, Z. Robust proof of stake: A new consensus protocol for sustainable blockchain systems. *Sustainability* **2020**, *12*, 2824–2839. [CrossRef]
63. Ko, C.H.; Chou, C.C.; Meng, H.Y.; Wei, H.Y. Strategy-proof resource allocation mechanism for multi-flow wireless multicast. *IEEE Trans. Wirel. Commun.* **2015**, *14*, 3143–3156. [CrossRef]
64. Ren, Y.J.; Zhu, K.; Gao, Y.Q.; Xia, J.Y.; Zhou, S. Long-term preservation of electronic record based on digital continuity in smart cities. *Comput. Mater. Contin.* **2021**, *66*, 3271–3287. [CrossRef]
65. Sasikumar, A.; Karthikeyan, B.; Arunkumar, S.; Saravanan, P.; Subramaniyaswamy, V.; Ravi, L. Blockchain-based decentralized user authentication scheme for letter of guarantee in financial contract management. *Malays. J. Comput. Sci.* **2022**, *1*, 62–73.
66. Kumar, G.; Saha, R.; Rai, M.K.; Thomas, R.; Kim, T.H. Proof-of-Work consensus approach in blockchain technology for cloud and fog computing using maximization-factorization statistics. *IEEE Internet Things J.* **2019**, *6*, 6835–6842. [CrossRef]
67. Chen, D.; Yuan, H.; Hu, S.; Wang, Q.; Wang, C. BOSSA: A decentralized system for proofs of data retrievability and replication. *IEEE Trans. Parallel Distrib. Syst.* **2021**, *32*, 786–798. [CrossRef]
68. Schäfer, D.R.; Rothermel, K.; Tariq, M.A. Replication schemes for highly available workflow engines. *IEEE Trans. Serv. Comput.* **2021**, *14*, 559–573. [CrossRef]
69. Nelson, A.; Toth, G.; Linders, D.; Nguyen, C.; Rhee, S. Replication of smart-city Internet of Things assets in a municipal deployment. *IEEE Internet Things J.* **2019**, *6*, 6715–6724. [CrossRef]