Verifiable Isogeny Walks: Towards an Isogeny-based Postquantum VDF

Jorge Chavez-Saab[1], Francisco Rodríguez-Henríquez[1,2], and Mehdi Tibouchi[3]

[1] Computer Science Department, Cinvestav IPN, Mexico City, Mexico
[2] Cryptography Research Centre, Technology Innovation Institute, Abu Dhabi, United Arab Emirates
[3] NTT Corporation, Tokyo, Japan

Abstract. In this paper, we investigate the problem of constructing postquantum-secure verifiable delay functions (VDFs), particularly based on supersingular isogenies. Isogeny-based VDF constructions have been proposed before, but since verification relies on pairings, they are broken by quantum computers. We propose an entirely different approach using succinct non-interactive arguments (SNARGs), but specifically tailored to the arithmetic structure of the isogeny setting to achieve good asymptotic efficiency. We obtain an isogeny-based VDF construction with postquantum security, quasi-logarithmic verification, and requiring no trusted setup. As a building block, we also construct non-interactive arguments for isogeny walks in the supersingular graph over $\mathbb{F}_{p^2}$, which may be of independent interest.

Keywords: Isogeny-based cryptography · Postquantum cryptography · Verifiable delay functions · Supersingular elliptic curves · SNARGs · Verifiable computation

1 Introduction

A *Verifiable Delay Function* (VDF) is a cryptographic primitive first formalized by Boneh, Bonneau, Bünz and Fisch in 2019 [9], which has since gathered increasing interest due to its various applications such as power-efficient blockchains, benchmarking, and randomness beacons (these and other applications are discussed in [9,15]).

The intuitive idea of a VDF is that it acts as a function whose value is uniquely determined at the moment that we pick an input, but no one is able to compute its output faster than a guaranteed prescribed wall-clock time T. To achieve this, it is crucial that the only known approaches for computing a VDF must be inherently sequential, such that no reasonable amount of parallelism could be effective on speeding up the VDF evaluation. At the same time, we also require the peculiar feature that the VDF's output must be efficiently and publicly verifiable, meaning that any other party can confirm its correctness, without relying on secret parameters nor on repeating the lengthy evaluation work that was required to produce it in the first place.

Constructing an ordinary delay function is a simple task, as it suffices to use T iterations of any function that can be composed with itself and whose output is unpredictable (such as a hash function). However, achieving an efficient verification is usually a much bigger challenge. For this, one may rely on general techniques for verifiable computation, specifically on *Succinct Non-interactive Arguments* (SNARGs) which allow for efficient proofs of any computation, where the time complexity of the proof construction is asymptotically close to that of the original computation, and its verification is polylogarithmic. This paper presents a quantum-resistant VDF, whose evaluation involves isogeny walks over supersingular elliptic curves that can be publicly verified by means of a SNARG-based validation process.

In terms of quantum security, none of the current VDF constructions proposed as of today manage to achieve an exponential time gap between evaluation and verification while still being based on commonly studied postquantum assumptions. Our construction benefits from being derived from isogeny-based cryptography, which provides security guarantees that have been carefully scrutinized in the postquantum setting for over a decade. These studies add confidence in our security assumptions as well as in providing accurate estimates of how fast isogeny evaluations can be performed using optimized software and hardware libraries.

Previous Work: The usage of SNARGs for constructing a VDF was first proposed by Boneh et al. [9], and independently by Döttling et al. [16]. The concept of verifiable computation branched out from probabilistic checkable proofs, as proposed by Babai et al. [3], and its development towards proofs that are short, efficient and non-interactive began with Micali's work [26].

In the context of verifiable delay functions, as of today the only isogeny-based construction was proposed by De Feo et al. in 2019 [15]. The main computational task for the evaluation of this VDF is that of finding images of points under a fixed large degree isogeny of a supersingular elliptic curve, whereas its verification essentially consists of performing a bilinear pairing computation. This verification is much more efficient than a SNARG-based verification, but the trade-off is that a quantum attacker can compute the VDF output by solving an associated discrete logarithm problem rather than going through the intended isogeny evaluation. Moreover, the construction has the added drawbacks of requiring a trusted setup and the setup itself being slow (requiring about as much time as an evaluation).

More recently, Leroux [24] proposed an isogeny-based verifiable random function that makes use of a proof of knowledge of a secret isogeny. While it is pointed out that this proof provides an exponential gap between prover and verifier, it cannot be adapted to a VDF since it only convinces the verifier that the prover knows *some* isogeny without any guarantee that the isogeny was somehow derived from an input. The evaluation of the random function actually maps the input to points, and then evaluates the images of those points, but this is not quantum-resistant unless it is treated as a single-use function.

Our Contribution: We present an isogeny-based VDF that is free of the three main drawbacks suffered by the construction of De Feo et al.: the setup is fast, is not required to be trusted, and the construction is postquantum. By using a SNARG verification, we are able to obtain an evaluator with $\tilde{\mathcal{O}}(T)$ time complexity using $\mathcal{O}((\log T)^4)$ parallelism, and a quasi-logarithmic verifier with $\tilde{\mathcal{O}}((\log T)^{4+\log\log T})$ time complexity using no parallelism. This result is of interest not only due to its asymptotic complexity, but also because of the fact that it could directly benefit from future advances in SNARG constructions.

Since our VDF construction performs a random walk in the supersingular isogeny graph over $\mathbb{F}_{p^2}$, it can be seen as an instance of the Charles-Lauter-Goren hash function [12] augmented with a SNARG verification of this random walk. While there exist general-purpose SNARGs that can be applied for any computation in a straightforward fashion (see for example [6,34]), we save as much as possible on overhead by specializing to the isogeny setting and constructing a SNARG over the field $\mathbb{F}_{p^2}$, which verifies the computation at the field-arithmetic level as opposed to the ALU-operation level. This implies that we have to generalize various SNARG results to work efficiently over a prescribed field, which leads us to present a framework that connects the SNARG with the isogeny walk setting in a natural way. Moreover, we describe the process for "ordering" the possible isogenies at each vertex of the isogeny graph so that the walk can be derived from an input string. This was never presented explicitly in [12], and in order to be compatible with our SNARG, our method selects an isogeny using only $\mathbb{F}_{p^2}$ arithmetic (i.e. we refrain from using arbitrary rules that look at the bit-representation of the field elements).

Organization: The remainder of the manuscript is organized as follows. Section 2 contains an overview and background of both isogeny-based cryptography and time-sensitive cryptography. In Section 3 we present the evaluation method of our isogeny-based delay function, and in Section 4 we present its SNARG-based verification. We then provide our security analysis in Section 5, and concluding remarks in Section 6.

2 Background

In this section we present some basic definitions and background material used throughout this paper.

2.1 Elliptic Curves

Basic definitions of elliptic curves over finite fields. Let $\mathbb{F}_p$ be a finite field with p a large odd prime, and let $\mathbb{F}_q = \mathbb{F}_{p^2}$ be its quadratic extension. We denote the algebraic closure of $\mathbb{F}_q$ as $\overline{\mathbb{F}}_q$.

Let E be a Montgomery elliptic curve over $\mathbb{F}_q$, expressed as,

$$E(\mathbb{F}_q): \quad By^2 = x^3 + Ax^2 + x\,, \tag{1}$$

such that $A, B \in \mathbb{F}_q$, $A^2 \neq 4$ and $B \neq 0$. The set of solutions of (1) plus the neutral element $\mathcal{O}$, known as the point at infinity, form an abelian additive group, with the inverse of a point (x, y) being $(x, -y)$. For $N \in \mathbb{N}$, the $N-$torsion subgroup $E[N]$ is defined as the kernel of the multiplication-by-N map.

The cardinality of $E(\mathbb{F}_q)$ is given by $\#E(\mathbb{F}_q) = q + 1 - t$, where t is the trace of E over $\mathbb{F}_q$ and satisfies $|t| \leq 2\sqrt{q}$. An elliptic curve E is said to be supersingular if $p \mid t$, and ordinary otherwise. When $q = p^2$, it follows that $t \in \{0, \pm p, \pm 2p\}$ for any supersingular curve. In the rest of this paper we only consider supersingular curves with $t = -2p$, so that the group order is $(p+1)^2$.

An elliptic curve E is uniquely defined up to isomorphism by its j-invariant, given by

$$j(E(\mathbb{F}_q)) = 256\frac{(A^2-3)^3}{A^2-4} \quad . \tag{2}$$

Since the parameter B does not appear at all in (2), it follows that Montgomery curves are completely determined by the parameter A up to isomorphism. Let $E'/\mathbb{F}_q$ be another elliptic curve with parameter $A' \in \mathbb{F}_q$. Then, E is isomorphic to E' over $\mathbb{F}_q$, denoted by $E \cong E'$, if and only if $j(E) = j(E')$. Appropriate values for A always result in supersingular curves. In particular, $A = 0$ and $A = 6$ correspond to supersingular elliptic curves whenever $p \equiv 3 \mod 4$. Moreover, even when E is defined over $\overline{\mathbb{F}}_q$ we always have $j(E) \in \mathbb{F}_q$ if E is supersingular, so there exists an isomorphic curve defined over $\mathbb{F}_q$. We therefore assume without loss of generality that all supersingular curves are defined over $\mathbb{F}_q$.

Isogenies. An isogeny $\phi\colon E \to E'$ over $\mathbb{F}_q$, is a non-constant rational map between elliptic curves defined over the finite field $\mathbb{F}_q$, which satisfies $\phi(\mathcal{O}) = \mathcal{O}$. It is known that two curves E and E' are isogenous over $\mathbb{F}_q$ if and only if they have the same number of points [32, Theorem 1]. Therefore, E is supersingular if and only if E' is.

Isogenies inherit the notion of degree and separability from rational maps. A separable isogeny has a kernel size equal to its degree, and is uniquely determined by its kernel up to composition with an isomorphism. The fastest known method for computing the co-domain curve of a smooth-degree isogeny is to decompose it into prime-degree components, so a degree-ℓ^T isogeny requires T isogeny evaluations of degree ℓ.

We refer to an isogeny of prime degree ℓ as an ℓ-isogeny. For any ℓ-isogeny $\phi : E \to E'$, there exists a dual $\hat{\phi} : E' \to E$ which is also an ℓ-isogeny, such that the composition $\hat{\phi} \circ \phi$ equals the multiplication-by-ℓ map.

Supersingular isogeny graphs. See [20] for a comprehensive study of supersingular isogeny graphs. For any prime ℓ, the supersingular isogeny graph $G_{\mathbb{F}_q}(\ell)$ is the directed graph having $\mathbb{F}_q$-isomorphism classes of ℓ-isogenous supersingular elliptic curves (represented by their j-invariants) as vertices, and ℓ-degree isogenies as edges. The graph $G_{\mathbb{F}_q}(\ell)$ is a Ramanujan graph [29], meaning that random walks in it have an asymptotically optimal mixing rate. Moreover, $G_{\mathbb{F}_q}(\ell)$

is $(\ell+1)$-regular whenever $\left(\frac{-p}{\ell}\right) = 1$. There are two special vertices, $j = 0$ and $j = 1728$, which represent the only curves with non-constant automorphisms. Outside of these two vertices the graph contains no self-loops, and can be taken to be undirected since edges are symmetric.

Modular polynomials. For any prime ℓ, there exists a polynomial $\Phi_\ell \in \mathbb{Z}[X, Y]$ of degree $\ell+1$, called the ℓ^{th} modular polynomial, such that two curves E and E' are $\ell-$isogenous if and only if $\Phi_\ell(j(E), j(E')) = 0$. Given a starting curve E, we can walk a step in the ℓ-isogeny graph by finding a root of $\Phi_\ell(j(E), X)$ rather than explicitly computing an isogeny.

2.2 Time-sensitive cryptography and Verifiable Delay Functions

Time-sensitive cryptography was first proposed in 1996 by Rivest, Shamir and Wagner [31]. The authors of [31], presented time-lock puzzle constructions that must be computed by performing a prescribed number of sequential squarings over an RSA modulus of unknown order. More recently, Lenstra and Wesolowski introduced in [23], a slow-timed hash function dubbed *sloth*. The evaluation of sloth is accomplished by the iterated computation of a fixed number of sequential functions. Also, the notion of Proof of Sequential Work (PoSW) was introduced by Cohen and Pietrzak in Eurocrypt 2018 [14]. Then, Boneh, Bonneau, Bünz and Fisch formalized this branch of cryptography by rigorously defining verifiable delay functions.

A Verifiable Delay Function (VDF) as defined in [23,9] is a function $f : \mathcal{X} \mapsto \mathcal{Y}$ that cannot be computed in less than a prescribed delay, regardless of the amount of parallelization available for its evaluation. At the same time, once a VDF has been computed, it can be easily verified by any third party, typically with the help of a companion proof produced during the evaluation. Moreover, the verification should be achievable with a limited amount of parallel cores, and ideally, by performing a single-core computation. Formally, a VDF is composed of three main algorithms:

- `Setup`: takes as input a security parameter λ and a delay parameter T and outputs public parameters pp.
- `Eval`: Takes a certain input x and public parameters pp and calculates an output y and a proof π.
- `Verify`: Takes as input x, y, π and pp and outputs 1 if and only if π is a valid proof for the input-output pair (x, y).

Moreover, a secure VDF satisfies the following properties:

- *Sequentiality*: The `eval` procedure can be completed in time $\mathcal{O}(T, \lambda)$ using $polylog(T)$ parallelism, but cannot be completed in time $o(T)$ even when $poly(T)$ parallelism is available
- *Completeness*: An honest evaluation always causes the verifier to accept
- *Soundness*: If y is not the output of $\texttt{Eval}(x, pp)$, then no PPT adversary can find a proof π such that the verifier accepts (x, y, π).

VDFs have important applications for Blockchain proof of work, space and stake [13], constructing a trustworthy randomness beacon [17], benchmarking of high-end servers and many more [11,33]. Several examples of VDFs proposed in the literature can be found on [9,15,28,35].

Isogeny-based VDFs The only isogeny-based VDF construction proposed as of today is the one presented by De Feo, Masson, Petit, and Sanso in Asiacrypt 2019 [15].

The authors of [15], proposed an isogeny-based VDF where the evaluator must compute the image of a point under a large smooth-degree isogeny ϕ (consisting of the composition of T isogenies each of prime degree ℓ), between two ℓ^T-isogenous supersingular curves E and E'. The order of the elliptic curves E and E' has a large prime divisor N, and their N-torsion subgroups are used for the verification via a pairing comparison using a point P with known image $\phi(P)$ (these points are public parameters and are computed at setup time).

This VDF construction has three important drawbacks. The first one is that it requires a trusted setup. This implies that a dishonest party computing the setup, can easily backdoor the function to make the evaluation much faster (as discussed in [15], knowledge of the random walk that generated E, can be used to compute the endomorphism ring of the curve E, which can be used to reduce the degree-ℓ^T isogeny to a shorter one). A second issue is that the setup computation is slow, taking as long as the delay from the evaluation itself. A third drawback is that the verification crucially depends on the computation of a pairing, which opens the door against any quantum attack targeting discrete logarithm computations.

VDFs from iterated sequential functions Given an input parameter x, the authors of [9,15] gave as an example of a naive VDF the chained computation of a one-way function as,

$$x_i = H(x_{i-1}), \text{ for } i = 1, \ldots, T,$$

with $x_0 = x$, and where the output $y = x_T$ can only be calculated sequentially independently of the amount of parallelism available for the evaluator. Notice however, that if the evaluator publishes some of the intermediate values x_i for $i = 1, \ldots, T$ (see Figure 1), then a verifier with access to many independent processors, can verify the work of the evaluator in a wall-clock delay significantly shorter than the time invested by the evaluator for producing y. This simple version of a VDF was discussed in [23, §3.1] as a *trivial design*, and later proposed by Yakovenko as a *Proof of History* consensus protocol with direct applications to blockchains [36]. Although this type of construction cannot achieve a polylogarithmic-time verification without requiring $poly(T)$ parallelism from the verifier, they are still sufficiently efficient for various applications. In particular, Yakovenko's Proof of History consensus protocol is massively used by the cryptocurrency Solana as its main consensus mechanism.

In order to obtain an asymptotically efficient verification without parallelism, the verification can be improved by means of verifiable computation. Verifiable

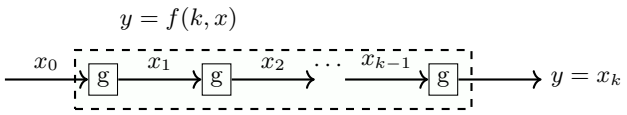

Fig. 1. Ilustration of an Iterated Sequential Function $f : \mathbb{N} \times \mathcal{X} \mapsto \mathcal{X}$, defined as $f(k, x) = g \circ g \circ \ldots \circ g$. In the figure, $x_0 = x$, and the output of the function $f = (k, x)$ is $y = x_k$.

computation can be used by the evaluator to compute a succinct non-interactive argument (SNARG), which certifies that a given computation was performed honestly. An important characteristic of a SNARG is that its verification can achieve a complexity that is logarithmic in the size of the original computation.

Both Boneh et al. [9] and Döttling et al. [16] proposed that any iterative sequential function can be augmented with a SNARG to produce an asymptotically efficient VDF. This is precisely the steps that we follow for our isogeny-based VDF, with the iterative function being instantiated by a step in the supersingular isogeny graph and the SNARG being constructed at the $\mathbb{F}_{p^2}$ arithmetic level.

3 An Isogeny-based delay function

We now present an overview for the evaluation method of our VDF, leaving the SNARG-based verification for Section 4.

Our function involves computing a walk of length T in the 2-isogeny graph of supersingular curves over $\mathbb{F}_{p^2}$, where $p^2 \equiv 9 \mod 16$ (which is required for applying Kong's square-root algorithm [22]) and $p = poly(T)$ (which is required to make field arithmetic efficient for the verifier). The walk itself is determined from a string that is derived from the input to the VDF, starting from a prescribed initial curve, and the $j-$invariant of the final curve is taken as the output of the VDF. Therefore, our output exactly matches the output from an instantiation of the Charles-Goren-Lauter hash function [12], where the isogeny at each step is determined after assigning some ordering to the outgoing isogenies. In order to make the procedure suitable for a SNARG construction, however, we develop a procedure that makes this ordering not only explicit but also verifiable using only field arithmetic.

3.1 Evaluation overview

Given a delay parameter T, we ask the evaluator to compute a walk of length T on the 2-isogeny graph, where the exact path is determined by a string s and is non-backtracking. We would not be able to specify kernel points of order ℓ^T as in the SIDH setting, since doing so would require either $p = \mathcal{O}(\ell^T)$ or working over an $\mathcal{O}(T)$ field extension, both of which would make all field arithmetic inefficient

for the verifier. Therefore, each step in the isogeny walk has to be determined "on the fly", and we chose to derive it from the modular polynomial root-finding problem since it is naturally expressed in $\mathbb{F}_{p^2}$ arithmetic.

Given two curves with j-invariants j_i and j_{i+1}, they are 2-isogenous over $\mathbb{F}_{p^2}$ if and only if the modular polynomial $\Phi_2(j_i, j_{i+1})$ vanishes. Thus, for fixed j_i, the next curve in the path can be computed by finding a root of $\Phi_2(j_i, X)$. This is a cubic polynomial, but we can exploit the fact that we already know one of the roots (namely j_{i-1}, the previous curve in the walk) to factor out a linear term: if $X = j_{i-1}$ is a known root of $\Phi_2(X) = X^3 + aX^2 + bX + c$ then we can rewrite

$$\Phi_2(X) = (X - j_{i-1})(X^2 + (a + j_{i-1})X + b + aj_{i-1} + j_{i-1}^2)$$

and focus on finding the roots of the quadratic factor. This accomplishes three distinct goals:

1. It ensures the walk is non-backtracking by discarding the $X = j_{i-1}$ root
2. It reduces the root-finding problem to a quadratic equation (reducing the size of the computation yields heavy savings on SNARG overhead)
3. It enables the step in the walk to be defined by a canonical square-root along with a bit indicating its sign

Taking into account the explicit form of Φ_2, the other two roots are given by

$$j_{i+1} = \frac{1}{2}\left(j_i^2 - 1488j_i - j_{i-1} + 162000 \pm \sqrt{D_i}\right) \tag{3}$$

where

$$\begin{aligned} D_i = & j_i^4 - 2976j_i^3 + 2j_i^2 j_{i-1} + 2532192j_i^2 - 2976j_i j_{i-1} \\ & - 645205500j_i - 3j_{i-1}^2 + 324000j_{i-1} - 8748000000 \end{aligned} \tag{4}$$

The evaluator computes a canonical square root $S_i = \sqrt{D_i}$ using Kong's algorithm [22]: first fix any quadratic nonresidue d and precompute $t = d^{\frac{p^2-9}{8}}$, then compute

$$R_i = (2D_i)^{\frac{p^2-9}{16}} \tag{5}$$

and set

$$S_i = \begin{cases} R_i D_i (2D_i R_i^2 - 1) & \text{if } (2aR_i^2)^2 = -1 \\ R_i t d D_i (2R_i^2 t^2 d^2 D_i - 1) & \text{if } (2aR_i^2)^2 = +1 \end{cases} \tag{6}$$

which can be combined into

$$S_i = \left(\frac{1 - (2aR_i^2)^2}{2}\right) R_i D_i (2D_i R_i^2 - 1) + \left(\frac{1 + (2aR_i^2)^2}{2}\right) R_i t d D_i (2R_i^2 t^2 d^2 D_i - 1) \tag{7}$$

After the square root has been calculated, the evaluator uses the input string to choose the sign, yielding a deterministic process for the walk. Note that the input string cannot have length $\mathcal{O}(T)$ (since the verifier must receive it and process it in time $polylog(T)$), so we will construct the signs pseudorandomly from a smaller string, as detailed in Section 4.1.

Note also that the evaluator could use any algorithm for computing square roots, but for the SNARG verification process it is convenient to use a procedure that is deterministic and also produces a fixed choice of sign, hence the choice of Kong's algorithm. For the verification process, the evaluator will keep track of j_i, D_i, R_i, S_i at each step and construct a SNARG that shows that equations (3), (4), (5) and (7) are satisfied.

Initial conditions. Since equation (3) requires knowledge of the $j-$invariant two steps into the past, we need to specify the first two curves as initial conditions. The 2-isogeny graph is a 3-regular graph without repeated edges or self-loops except for the two special vertices $j = 1728$ and $j = 0$. At $j = 1728$ there is one self-loop and two edges going to the vertex $j = 287496$, so we use the initial curve $j_0 = 287496$ and take $j_{-1} = 1728$ to avoid going back to the non-regular vertex. Note that the SNARG will only access values at indices 0 through $T - 1$, so we replace $j_{-1} = 1728$ by an equivalent condition on D_0 using equation (4).

4 SNARG-based Verification

In this section we deal with the verification process of the VDF, specifically how to fit the evaluation into a SNARG framework.

The notion of verifiable computation emanated from *Probabilistic Checkable Proofs* (PCP), a term first coined by Arora and Safra [2] to refer to a protocol between a prover who generates a proof of membership in a language (known as the PCP witness) for a given input and a randomized verifier which interacts with it. Both Babai et al. [3] and Feige et al. [18] independently proposed algorithms for transforming any $\mathcal{NP}$ witness into a PCP witness which, at the cost of making the verification probabilistic and the witness polynomially larger, allow for logarithmic-time verification by sampling only a logarithmic number of bits from the witness.

The results from [3] and [18] show that the history of any computation can be put into an alternate form which can be verified with an exponential speedup. In practice, however, these PCP constructions have two major drawbacks. First, they require interaction between the two parties throughout the protocol, and second, the amount of communication is still inefficient since the full PCP witness must be transmitted even if only a few bits of it are sampled. *Succinct Non-interactive Arguments* (SNARGs) are an alternative primitive which overcomes both limitations (they do not require interaction and are succinct in the sense that the witness is of logarithmic size). It was shown by Micali [26] that any PCP construction can be efficiently transform into a SNARG.

General-purpose SNARGs aim to verify a computation by working at the level of a RAM model and translating the correctness of the computation into either a circuit satisfiability or an algebraic constraint problem. For our construction we have focused on the latter approach, and use a checksum-type PCP to verify said constraint problem.

In this section we review how the different construction ingredients are adapted to our particular problem. Section 4.1 describes the process of transforming the correctness of our computation into an algebraic constraint problem (known as *arithmetization*). Section 4.2 then gives a high-level summary for the rest of the SNARG construction and the complexities of the resulting verification scheme, leaving most details for the appendix. Finally, Section 4.3 discusses the parallelization of the SNARG proof construction to obtain a concrete proof construction time close to the evaluation time.

4.1 Arithmetization

The sumcheck protocol that we work with is a PCP that verifies conditions of the form

$$\sum_{\vec{x}\in B^n} P(\vec{x}) = 0, \tag{8}$$

where P is a polynomial in n variables over some ring $R \supset B$.

To turn the verification of our VDF into an instance of this problem we *arithmetize* by storing the intermediate values into polynomials that act as lookup tables. Specifically, for the computation with T steps, we pick a base b and integer n such that $T \approx b^n$, and let $B = \{0, 1, \ldots, b-1\}$. The time steps $t \in \{0, 1, \ldots, T-1\}$ can then be expressed as $b-$ary strings of length n, where we refer with $t_{\vec{x}}$ to the integer represented by string $\vec{x} \in B^n$.

For $\vec{y} \in B^n$ we define the polynomial

$$\delta_{\vec{y}}(\vec{x}) = \prod_{j=0}^{n-1} \prod_{z\in B-\{y_j\}} \frac{x_j - z}{y_j - z} \tag{9}$$

which maps $\vec{x} \in B^n$ to 1 if $\vec{x} = \vec{y}$ and 0 if $\vec{x} \neq \vec{y}$. Regarding B as a subset of $\mathbb{F}_{p^2}$, the above formula can be seen as a polynomial over $(\mathbb{F}_{p^2})^n$, and is in fact the unique degree-$(b-1)$ polynomial that agrees with the δ function on B^n (note that throughout this paper, the "degree" of a multivariate polynomial refers to the maximum degree of any individual variable).

The δ polynomial can be used as an auxiliary tool to select a specific index when summing over all indices. Given the sequence j_i of j-invariants at each step, we can define the polynomial

$$j(\vec{x}) = \sum_{y\in B^n} j_{t_{\vec{y}}} \delta_{\vec{y}}(\vec{x}). \tag{10}$$

This polynomial encodes the history of the computation since it maps $\vec{x} \in B^n$ to $j_{t_{\vec{x}}}$, but can also be evaluated (with less predictable outcome) over all of $(\mathbb{F}_{p^2})^n$.

We can then define similar polynomials $D(\vec{x})$, $R(\vec{x})$, $S(\vec{x})$ for the quantities D_i, R_i, S_i defined in Section 3, and also use the constants

$$L_{t_1,t_2} = \begin{cases} 1 & \text{if } \ t_2 = t_1 + 1 \\ 0 & \text{else} \end{cases}$$

to define the polynomial

$$L(\vec{x},\vec{y}) = \sum_{\vec{x}',\vec{y}' \in B^n} L_{t_{\vec{x}'},t_{\vec{y}'}} \delta_{\vec{x}'}(\vec{x}) \delta_{\vec{y}'}(\vec{y}) \tag{11}$$

which vanishes in B^{2n} unless $\vec{x}$ and $\vec{y}$ represent consecutive integers. Similarly to the δ polynomial, this polynomial will be used as a sequential counter that selects only consecutive indices when summing over all pairs of indices.

With this in hand, equations (3), (4), (5) and (7) can be represented as polynomial conditions. For instance, (3) becomes

$$P^j(\vec{x},\vec{y},\vec{z}) = 0 \quad \forall\ (\vec{x},\vec{y},\vec{z}) \in B^{3n}, \tag{12}$$

where

$$P^j(\vec{x},\vec{y},\vec{z}) := \\ \left[2j(\vec{z}) - j(\vec{y})^2 + 1488j(\vec{y}) + j(\vec{x}) - 162000 - s(\vec{y})S(\vec{y})\right] L(\vec{x},\vec{y})L(\vec{y},\vec{z}).$$

Here, $s(\vec{x})$ represents the choice of sign at step $t_{\vec{x}}$, which we have also represented as a polynomial. We have not specified how the sign is derived from the input, but it will be necessary for the verifier to be able to efficiently evaluate $s(\vec{x})$. Therefore, we will take

$$s(\vec{x}) = \prod_{i=0}^{n-1} s_i(x_i), \tag{13}$$

where s_i are single-variable polynomials of degree $b-1$ mapping B to $\{+1,-1\}$. Since a polynomial of degree $b-1$ is determined by its values at any b points, we only need b bits to uniquely specify each s_i. This means that we use a total of nb bits to define $s(\vec{x})$, and we now define these bits as the input to the VDF (the input bits can be passed through a hash function first to enforce pseudorandomness of the resulting polynomial).

We then turn equations (4) and (7) into polynomial conditions

$$P^D(\vec{x},\vec{y}) = 0 \quad \forall\ (\vec{x},\vec{y}) \in B^{2n}$$

and

$$P^S(\vec{x}) = 0 \quad \forall\ \vec{x} \in B^n$$

in an analogous way.

As for equation (5), the polynomial that we would obtain by arithmetizing it directly would be of large degree, which is undesirable for the SNARG construction (as described in Appendix A, the sumcheck protocol verification is linear in

the degree of the polynomial). Therefore, we introduce additional state and break the exponentiation down into a right-to-left strategy: let $K = \lceil \log((p^2-9)/16) \rceil$ and $e_0, e_1, \ldots, e_{K-1}$ be the bits of $(p^2-9)/16$. We define $R_i^{(0)} = 1$, $D_i^{(0)} = D_i$ and for $0 < k < K$,

$$D_i^{(k)} = (D_i^{(k-1)})^2, \tag{14}$$

and

$$R_i^{(k)} = R_i^{(k-1)} (D_i^{(k-1)})^{e_k} \tag{15}$$

so that $R_i^{(k-1)} = R_i$. For each $0 \leq k < K$ we define polynomials $D^{(k)}(\vec{x})$ and $R^{(k)}(\vec{x})$ from the values of $D_i^{(k)}$ and $R_i^{(k)}$, respectively, and use them to write polynomial conditions

$$P^{R,k}(\vec{x}) = 0 \quad \forall\ \vec{x} \in B^n$$

and

$$P^{D,k}(\vec{x}) = 0 \quad \forall\ \vec{x} \in B^n.$$

Note that we now have $K = \log p$ pairs of polynomials to work with, but each being of degree $d = \mathcal{O}(b)$ just as P^j, P^D and P^S (this is inherited from the degree of the δ polynomial).

Finally, to wrap the whole verification into the form of (8), we use the weighted sum

$$\begin{aligned} P(\vec{x}, \vec{y}, \vec{z}) = {} & w^J(\vec{x}, \vec{y}, \vec{z}) P^J(\vec{x}, \vec{y}, \vec{z}) \\ & + w^D(\vec{x}, \vec{y}) P^D(\vec{x}, \vec{y}) + w^S(\vec{x}) P^S(\vec{x}) \\ & + \sum_k \left(w^{D,k}(\vec{x}) P^{D,k} + w^{S,k}(\vec{x}) P^{S,k} \right), \end{aligned} \tag{16}$$

where each w is a weight polynomial. These polynomials are defined in the same way as $s(\vec{x})$ in equation (13), but are chosen randomly by the verifier so that proving that

$$\sum_{\vec{x},\vec{y},\vec{z} \in B^n} P(\vec{x}, \vec{y}, \vec{z}) = 0$$

is enough to convince the verifier that $P^J, P^D, P^S, P^{k,D}, P^{k,S}$ vanish at all points, and hence that the whole computation is correct.

We stress that general-purpose SNARGs exist that can be applied to any program (see for example [6] and [5]), but they usually perform arithmetization at the ALU level, which increases the overhead cost. For instance, they may have lookup polynomials that encode the values of CPU registers at each time step and obtain polynomial conditions that represent the correctness of ALU operations (which in our case would include even the breakdown of all $\mathbb{F}_{p^2}$ arithmetic into basic ALU operations). The arithmetization that we propose is performed directly at the field arithmetic level, meaning that the values encoded into our polynomials are $\mathbb{F}_{p^2}$ elements and our polynomial conditions represent equality over this field, without actually including the correctness of $\mathbb{F}_{p^2}$ arithmetic procedures in the SNARG proof since it is within the verifier's capabilities to perform them directly.

4.2 Overview of the SNARG construction

We now present a high-level summary of the steps required to complete the SNARG construction, with a more detailed account presented in the appendix.

We have reduced the verification process to the verification of a condition of the form[4]

$$\sum_{\vec{x}\in B^n} P(\vec{x}) = 0,$$

where P is of degree $d = \mathcal{O}(b)$ in n variables, and $T = n^b$. This verification is handled by the sumcheck protocol of Lund, Fortnow, Karloff and Nisan [25], detailed in Appendix A. The sumcheck protocol is a PCP that reduces the problem to the verifier's ability to evaluate P at a random point $\vec{x} \in (\mathbb{F}_{p^2})^n$. To enable this, the prover publishes a PCP witness that contains the table of values for each of the polynomials $j(\vec{x}), D(\vec{x}), S(\vec{x}), R^{(k)}(\vec{x}), S^{(k)}(\vec{x})$ in all of $(\mathbb{F}_{p^2})^n$. The table of values for $L(\vec{x}, \vec{y})$ is assumed to be precomputed and publicly available (since it is independent of the input), while the polynomials for the sign and the random weights are directly evaluated by the verifier.

The next step is to apply Micali's transform [26], described in detail in Appendix B, which both eliminates interactiveness and shortens the proof to a polylogarithmic size. The core idea is to encode the tables of values into a Merkle tree and publish only the root of the tree as a commitment, answering specific queries with a value along with its verification path in the Merkle tree. We then apply the Fiat-Shamir transform [19] to replace all random choices from the verifier (including the choice of the weight polynomials in (16)).

The resulting verification scheme has a prover time complexity of $\mathcal{O}((nb)^n b \log b)$ with $\mathcal{O}(n^2 \log p)$ parallelism and space complexity of $\mathcal{O}((n^2 b)^n \log p)$ field elements (from computing and storing the tables of values, as detailed in Appendix A.1), and a verifier time complexity of $\mathcal{O}(n^3 b \log(nb) \log p)$ with no parallelization nor significant storage requirements (from the *degree test*, detailed in Appendix A.2). We argue in Section 5 that a choice of $p = poly(T)$ is natural, so the evaluator parallelism is $polylog(T)$.

There is still some freedom regarding the choice of parameters n and b, since they only need to satisfy $n^b \approx T$. Choosing $b \approx (\log T)^{1/\epsilon}$ for some $\epsilon > 0$ means that

$$n \approx \frac{\log T}{\log b} = \frac{\epsilon \log T}{\log \log T},$$

and

$$n^n \approx \left(\frac{\epsilon \log T}{\log \log T}\right)^{\frac{\epsilon \log T}{\log \log T}} < \log T^{\frac{\epsilon \log T}{\log \log T}} = T^\epsilon,$$

so the prover complexity becomes $\tilde{\mathcal{O}}(T^{1+\epsilon})$ time (which can be made arbitrarily close to linear) and $\mathcal{O}(T^{1+2\epsilon} \log p)$ space, while the verification complexity becomes $\tilde{\mathcal{O}}((\log T)^{4+1/\epsilon})$.

[4] For ease of notation we collapse $\vec{x}, \vec{y}, \vec{z}$ into a single vector, implicitly substituting $n \mapsto 3n$ throughout.

Another option is to chose a slowly decreasing function $\epsilon = 1/\log\log T$. This causes the proof construction to be strictly quasi-linear, but at the cost of making the verification only quasi-polylogarithmic.

4.3 Parallelization of the Proof Construction

It is also desirable for the evaluation algorithm of a VDF to take time that is concretely (as opposed to asymptotically) close to T. Both Boneh et. al. [9] and Döttling et. al. [16] independently proposed similar methods for exploiting parallelism to finish evaluation of an iterative function and its SNARG proof construction at the same time by working with subsegments of geometrically decreasing size. Assuming that computing the proof is slower than the function evaluation by a factor of α, the evaluator stops after completing $T/(1+\alpha)$ iterations of the computation and then starts a proof for this partial computation in parallel with the remaining $\alpha T/(1+\alpha)$ steps. This is repeated recursively, so the evaluator does proof constructions of size $T\left(\frac{\alpha}{1+\alpha}\right)^i$ for $i = 1, 2, 3, \ldots$ until $i \approx \log(T)/\log(1+\frac{1}{\alpha})$ when approximately a single step remains and it can be computed directly by the verifier without proof. Since $\log(1+\frac{1}{\alpha}) = \frac{1}{\alpha} + \mathcal{O}(\frac{1}{\alpha^2})$, this increases the parallelization requirement by a factor of $\alpha \log T$.

Note that in the case when $b = (\log T)^{1/\epsilon}$ for constant ϵ this results in an amount of parallelism polynomial in T, which is coined a *weak VDF* by Boneh et al. [9, Definition 5]. We favor the case when $\epsilon = 1/\log\log T$ instead, which means the parallelism is strictly logarithmic at the cost of making the verification slightly slower (quasi-polylogarithmic).

5 Security Analysis

The soundness of the VDF relies entirely on that of the SNARG proof, which is discussed throughout the appendix. In this section we discuss the other crucial security property of a VDF, namely its sequentiality.

As a side note, we point out that any protocol where the isogeny walk is not prescribed in some way is insecure in terms of sequentiality. For instance, one could have asked the evaluator for a SNARG proof of *any* large-degree isogeny and naively hope that this makes for a good proof of sequential work even if the output is not unique. However, much like the proof of isogeny knowledge proposed by Leroux [24], this does not constitute proof of a sequential computation if the evaluator is free to choose the path. Indeed, even if backtracking is avoided it has been shown by Adj, Ahmadi and Menezes [1] that it is easy to find cycles of any length in the $\ell-$isogeny graph, which allows a cheating evaluator to construct a SNARG for a "long" walk by repeating a short cycle as many times as necessary. Of course, the prescribed walks in our construction may still contain cycles, but this is not a problem so long as they cannot be forced nor predicted by the evaluator.

In our context, sequentiality relies on a similar version of the *Isogeny Shortcut Problem* defined by De Feo et al. [15] :

Problem 1 (Isogeny Shortcut Problem). Let $E/\mathbb{F}_p$ be a random supersingular elliptic curve and $\phi : E \to E'$ an isogeny of degree ℓ^T to a curve $E'/\mathbb{F}_{p^2}$. After a precomputation time $poly(T, \lambda)$, find the image of a given point whose order is coprime to ℓ in time $o(T)$.

Our setting differs from the one in this problem in three important ways:

1. Our problem is not to find images of points, but codomain curves. Since the codomain curve can be computed from any three point evaluations, the problem in our setting could be considered more general. However, all known point-evaluation methods have complexities asymptotically equal to those of codomain-evaluation, so we do not expect our security assumption to be significantly stronger.
2. The precomputation time that we allow in our setting is granted before learning the isogeny to be evaluated, which reflects the fact that our VDF uses a different isogeny for each input as opposed to fixing the isogeny at setup time. In this regard, our security is stronger since it relies on a much weaker assumption.
3. We do not assume that the starting curve was randomly sampled, which is done in [15] to prevent shortcut attacks (see below) when the endomorphism ring is known. However, we argue that such attacks are unimportant in our setting precisely due to the previous point. Starting from a public curve means we do not need a trusted setup.

Despite these differences, our security analysis is very similar to De Feo's, as we still distinguish two types of attacks: either finding a way to perform the isogeny walk faster (possibly exploiting parallelism), or attempting to find an equivalent isogeny walk of smaller degree (which we call *isogeny shortcuts*).

Faster Isogenies The best known method for computing a degree-ℓ^T isogeny is by sequentially performing the composition of T consecutive ℓ-isogenies, where each ℓ-isogeny is computed using Vélu's formulas. We consider the time of this computation as a lower bound for the delay parameter offered by our VDF, even though our specification of the evaluation algorithm does not use kernel points. While Vélu's formulas parallelize almost perfectly, attempting to directly evaluate an isogeny of degree ℓ^T in time $O(T)$, would require an amount of parallelism exponential in T which exceeds the evaluator's capabilities. This is leaving aside the fact that our isogeny is not readily presented as a kernel that could be plugged directly into these formulas, and even if such a kernel could be obtained it would be defined over a degree-T field extension. Assuming then that the best strategy is to decompose into small-degree isogenies, it is unlikely that any algorithm would be able to compute the composition of all such isogenies without going through each intermediate curve.

While none of the above premises are often studied as security assumptions, they are long-standing conjectures that have endured the test of time, despite considerable incentives to optimize isogeny evaluations. For instance, finding a way to compute an isogeny of degree ℓ^T in linear time without exploiting the

smooth decomposition would be equivalent to computing an arbitrary-degree isogeny in time logarithmic in the degree, which would be ground-breaking for all isogeny-based cryptography.

It should be noted that recent improvements to the evaluation of $\ell-$isogenies do exist, most notably the algorithm of Bernstein et al. [7], which achieves a square-root time complexity improvement over the previous state-of-the-art. Although this does not affect the asymptotic complexity of computing T isogenies in series, having the possibility for variations in the concrete cost is still problematic for a VDF. However, the gains in these new formulas are asymptotic in ℓ and we only use $2-$isogenies for which the formulas are so simple that they can be conjectured to be already optimal.

Isogeny Shortcuts The second possibility is for the evaluator to produce a different isogeny path between the two end curves, and hope that it is shorter than the original. Because construction of the proof requires the evaluator to know each of the j-invariants in the original isogeny path, one might mistakenly assume that such an attack would be useless. However, an attacker that is able to predict the output in a shorter time, even if unable to produce a proof for it, would still violate the security of the VDF.

A shorter path always exists because the isogeny graph is of size $\mathcal{O}(p)$ and the optimal expander property of Ramanujan graphs [29] implies the distance between any two curves is bounded by $\mathcal{O}(\log p)$ (which is necessarily logarithmic in T, otherwise all field arithmetic would be inefficient). However, the sequentiality property only requires that such path cannot be found in time $o(T)$. In the case of arbitrary curves, when the endomorphism ring is not known, the best one can do is to try to solve the isogeny problem between the curves, disregarding the already known isogeny. The best algorithm for this is a birthday attack, which takes $\mathcal{O}(p^{1/4})$ time using a quantum Grover search [8]. This sets a theoretic limit on the admissible delay parameters of $T = \mathcal{O}(p^{1/4})$. However, it should be noted that even this attack would not apply directly since it requires previous knowledge of the codomain curve.

In our construction, we take an additional risk of fixing the starting curve $j = 1728$ where not only is the endomorphism ring known, but also elements of norm ℓ^n can be easily found with the KLPT algorithm [21]. This leads to an attack analogous to the CGL hash collision-finding algorithm that was described in [27] and optimized by [15], which computes a shorter isogeny in time $poly(T, \log p)$. One could always start from a curve of unknown endomorphism ring to avoid this attack at the cost of requiring a trusted setup, but in our case even this attack is admissible because there is a different isogeny used in each input and computing the shortcut is still slower (in the VDF from De Feo et al. [15], this is more of a problem because the same isogeny is always used, so the shortcut breaks the VDF for all inputs after being computed once).

More generally, it is unlikely that any kind of reduction could be computed fast enough since our isogeny is not readily represented as an ideal in the quaternion algebra. Any reduction that works over the endomorphism ring would have

to translate the isogeny at each step into the language of quaternion ideals, necessarily resulting in a $\Omega(T)$ complexity, and the concrete time of whatever parsing need to be performed is unlikely to be much faster than a simple degree-2 isogeny.

6 Conclusions and Future Work

We have presented a framework for applying a SNARG at the field-arithmetic level to verify an isogeny walk, and used it to obtain a postquantum isogeny-based VDF that does not require a trusted setup and is less susceptible to isogeny shortcut attacks since it uses a different isogeny walk for each input.

In terms of asymptotic complexity, our VDF is less efficient relative to other constructions: for example, the VDF from De Feo et al. [15] has $\mathcal{O}(T)$ evaluator-space complexity and verification time constant in T. However, no previous VDF construction achieves post-quantum security.

Although SNARG-based VDFs have been deemed mainly of theoretical interest by Boneh et al. [9], alluding to the fact that current SNARG constructions have concrete costs about 100,000 times larger than the original computation, it should be noted that this is the case for general-purpose constructions which verify every step of the computation at the bit-operation level. Since our construction verifies steps of the computation at the field-arithmetic level, it has the potential to save significantly on overhead. Therefore, it could prove an interesting future work to implement and benchmark our construction.

We leave it also as future work to propose concrete parameters for our construction. We stress that definitions such as 128-bit security are not meaningful for the sequentiality property of a VDF, whereas for soundness our security is completely derived from Micali's work [26] and based on symmetric cryptography. This means that the choice of parameters should be based only on the desired delay time, which is impossible to fine-tune until a working implementation is obtained.

We also point out that there is a factor of $\log p$ in both the evaluator's parallelism complexity and the verifier's total complexity that emanates from the breakdown of an exponentiation to verify a square-root computation, as represented by equations (14) and (15). If the square-root computation was verified via a squaring rather than repeating the computation, this factor could be eliminated. However, the longer computation is required due to the fact that we perform our arithmetization at the field arithmetic level, meaning that we need a deterministic way of picking a sign that uses only field operations. We leave it as an open question whether one can design a method to, given both roots of a quadratic polynomial, choose one of them deterministically using only field arithmetic and without resorting to a large exponentiation.

Finally, it should also be noted that the SNARG mechanism we have described is fairly rudimentary and there are various recent developments that achieve slight optimizations. However, most of these improvements rely on reducing the arithmetic over ad hoc fields (such as [5], which uses a binary field)

whereas our SNARG is constrained to work in the field of the elliptic curve. We note that this problem is likely to be ubiquitous when adapting SNARGs to work with existing cryptographic frameworks, since such frameworks usually include arithmetic over a prescribed field. Working directly over this prescribed field is bound to save significantly on overhead when constructing SNARG proofs, so we also encourage further optimizations of SNARG constructions over arbitrary fields.

References

1. Adj, G., Ahmadi, O., Menezes, A.: On isogeny graphs of supersingular elliptic curves over finite fields. Finite Fields and Their Appl. **55**, 268–283 (2019). https://doi.org/10.1016/j.ffa.2018.10.002, https://doi.org/10.1016/j.ffa.2018.10.002
2. Arora, S., Safra, S.: Probabilistic checking of proofs: A new characterization of NP. J. ACM **45**(1), 70–122 (1998). https://doi.org/10.1145/273865.273901, https://doi.org/10.1145/273865.273901
3. Babai, L., Fortnow, L., Levin, L.A., Szegedy, M.: Checking computations in polylogarithmic time. In: Proceedings of the Twenty-Third Annual ACM Symposium on Theory of Computing. p. 21–32. STOC '91, Association for Computing Machinery, New York, NY, USA (1991). https://doi.org/10.1145/103418.103428, https://doi.org/10.1145/103418.103428
4. Babai, L., Fortnow, L., Lund, C.: Non-deterministic exponential time has two-prover interactive protocols. Comput. Complex. **1**, 3–40 (1991)
5. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable, transparent, and post-quantum secure computational integrity. IACR Cryptol. ePrint Arch. **2018**, 46 (2018), http://eprint.iacr.org/2018/046
6. Ben-Sasson, E., Chiesa, A., Genkin, D., Tromer, E., Virza, M.: Snarks for C: verifying program executions succinctly and in zero knowledge. In: Canetti, R., Garay, J.A. (eds.) Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II. Lecture Notes in Computer Science, vol. 8043, pp. 90–108. Springer (2013). https://doi.org/10.1007/978-3-642-40084-1_6, https://doi.org/10.1007/978-3-642-40084-1_6
7. Bernstein, D.J., De Feo, L., Leroux, A., Smith, B.: Faster computation of isogenies of large prime degree. In: Galbraith, S. (ed.) ANTS-XIV - 14th Algorithmic Number Theory Symposium. Proceedings of the Fourteenth Algorithmic Number Theory Symposium (ANTS-XIV), vol. 4, pp. 39–55. Mathematical Sciences Publishers, Auckland, New Zealand (Jun 2020). https://doi.org/10.2140/obs.2020.4.39, https://hal.inria.fr/hal-02514201
8. Biasse, J., Jao, D., Sankar, A.: A quantum algorithm for computing isogenies between supersingular elliptic curves. In: Meier, W., Mukhopadhyay, D. (eds.) Progress in Cryptology - INDOCRYPT 2014 - 15th International Conference on Cryptology in India, New Delhi, India, December 14-17, 2014, Proceedings. Lecture Notes in Computer Science, vol. 8885, pp. 428–442. Springer (2014). https://doi.org/10.1007/978-3-319-13039-2_25, https://doi.org/10.1007/978-3-319-13039-2_25
9. 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 Proceedings, Part I. Lecture Notes in Computer Science, vol. 10991, pp. 757–788. Springer (2018)
10. Bostan, A., Éric Schost: Polynomial evaluation and interpolation on special sets of points. Journal of Complexity **21**(4), 420–446 (2005), festschrift for the 70th Birthday of Arnold Schonhage
11. Cai, J., Lipton, R.J., Sedgewick, R., Yao, A.C.: Towards uncheatable benchmarks. In: Proceedings of the Eigth Annual Structure in Complexity Theory Conference. pp. 2–11. IEEE Computer Society (1993)
12. Charles, D.X., Lauter, K.E., Goren, E.Z.: Cryptographic hash functions from expander graphs. Journal of Cryptology **22**(1), 93–113 (Jan 2009). https://doi.org/10.1007/s00145-007-9002-x, https://doi.org/10.1007/s00145-007-9002-x
13. Chia Network Collaboration: Chia DAQ. Chia network (2021), available at: https://www.chia.net/faq/
14. Cohen, B., Pietrzak, K.: Simple proofs of sequential work. In: Nielsen, J.B., Rijmen, V. (eds.) Advances in Cryptology - EUROCRYPT 2018, Part II. Lecture Notes in Computer Science, vol. 10821, pp. 451–467. Springer (2018)
15. De Feo, L., 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, Part I. Lecture Notes in Computer Science, vol. 11921, pp. 248–277. Springer (2019)
16. Döttling, N., Garg, S., Malavolta, G., Vasudevan, P.N.: Tight verifiable delay functions. In: Galdi, C., Kolesnikov, V. (eds.) Security and Cryptography for Networks - 12th International Conference, SCN 2020. Lecture Notes in Computer Science, vol. 12238, pp. 65–84. Springer (2020)
17. Drake, J.: Minimal VDF randomness beacon. ETH Research (2018), available at: https://ethresear.ch/t/minimal-vdf-randomness-beacon/3566
18. Feige, U., Goldwasser, S., Lovász, L., Safra, S., Szegedy, M.: Interactive proofs and the hardness of approximating cliques. J. ACM **43**(2), 268–292 (1996). https://doi.org/10.1145/226643.226652, https://doi.org/10.1145/226643.226652
19. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) Advances in Cryptology - CRYPTO '86. Lecture Notes in Computer Science, vol. 263, pp. 186–194. Springer (1986)
20. Kohel, D.: Endomorphism rings of elliptic curves over finite fields. Ph.D. thesis, UC Berkeley (December 1996)
21. Kohel, D., Lauter, K.E., Petit, C., Tignol, J.: On the quaternion l-isogeny path problem. IACR Cryptol. ePrint Arch. **2014**, 505 (2014), http://eprint.iacr.org/2014/505
22. Kong, F., Cai, Z., Yu, J., Li, D.: Improved generalized Atkin algorithm for computing square roots in finite fields. Inf. Process. Lett. **98**(1), 1–5 (2006). https://doi.org/10.1016/j.ipl.2005.11.015, https://doi.org/10.1016/j.ipl.2005.11.015
23. Lenstra, A.K., Wesolowski, B.: Trustworthy public randomness with sloth, unicorn, and trx. Int. J. Appl. Cryptogr. **3**(4), 330–343 (2017). https://doi.org/10.1504/IJACT.2017.10010315, https://doi.org/10.1504/IJACT.2017.10010315
24. Leroux, A.: Proofs of isogeny knowledge and application to post-quantum one-time verifiable random function. IACR Cryptol. ePrint Arch. **2021**, 744 (2021), https://eprint.iacr.org/2021/744

25. Lund, C., Fortnow, L., Karloff, H.J., Nisan, N.: Algebraic methods for interactive proof systems. J. ACM **39**(4), 859–868 (1992)
26. Micali, S.: Computationally sound proofs. SIAM J. Comput. **30**(4), 1253–1298 (Oct 2000)
27. Petit, C., Lauter, K.E.: Hard and easy problems for supersingular isogeny graphs. IACR Cryptol. ePrint Arch. **2017**, 962 (2017), http://eprint.iacr.org/2017/962
28. Pietrzak, K.: Simple verifiable delay functions. In: Blum, A. (ed.) 10th Innovations in Theoretical Computer Science Conference, ITCS 2019. LIPIcs, vol. 124, pp. 60:1–60:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2019)
29. Pizer, A.K.: Ramanujan graphs and Hecke operators. Bouletin of the American Mathematical Society **23**, 127–137 (1990). https://doi.org/https://doi.org/10.1090/S0273-0979-1990-15918-X
30. Reed, I.S., Solomon, G.: Polynomial codes over certain finite fields. Journal of the Society for Industrial and Applied Mathematics **8**(2), 300–304 (1960). https://doi.org/10.1137/0108018, https://doi.org/10.1137/0108018
31. Rivest, R.L., Shamir, A., Wagner, D.A.: Time-lock puzzles and timed-release crypto. Tech. rep., MIT (1996), available at: https://tinyurl.com/time-lock-puzzles
32. Tate, J.: Endomorphisms of abelian varieties over finite fields. Inventiones Mathematicae **22**, 134—-144 (1966)
33. VDF Alliance: VDF research. VDF Alliance (2021), available at: https://vdfresearch.org/
34. Wahby, R.S., Setty, S., Howald, M., Ren, Z., Blumberg, A.J., Walfish, M.: Efficient RAM and control flow in verifiable outsourced computation. Cryptology ePrint Archive, Report 2014/674 (2014), https://eprint.iacr.org/2014/674
35. Wesolowski, B.: Efficient verifiable delay functions. J. Cryptol. **33**(4), 2113–2147 (2020)
36. Yakovenko, A.: Solana: A new architecture for a high performance blockchain v0.8.13. Solana cryptocurrency whitepaper (2020), available at: https://tinyurl.com/solana-whitepaper

A The Sumcheck Protocol

We base our verification process on the sumcheck protocol first described by Lund, Fortnow, Karloff and Nisan [25], which was first applied to verifiable computation in [3]. The sumcheck protocol is a probabilistic checkable proof where the prover demonstrates an arithmetic condition over a given ring of the form

$$\sum_{\vec{x}\in B^n} P(\vec{x}) = c_0 \tag{17}$$

to a verifier with computational power $\mathrm{Poly}(n, \deg P)$, by first producing a witness W_P and then engaging in an $n-$round interactive protocol:

1. The prover computes and publishes the partial sum

$$P_0(x_0) = \sum_{(x_1,\ldots,x_{n-1})\in B^{n-1}} P(x_0, \ldots x_{n-1}) \tag{18}$$

2. The verifier checks that
$$\sum_{x_0 \in B} P_0(x_0) = c_0, \tag{19}$$
then picks $r_0 \xleftarrow{\$} I$ from a subset $B \subset I \subset \mathbb{F}_{p^2}$ and computes $c_1 = P_0(r_0)$.
3. The prover now shows that
$$\sum_{(x_1,\ldots,x_{n-1}) \in B^{n-1}} P(r_0, x_1, \ldots, x_{n-1}) = c_1 \tag{20}$$
by applying the protocol recursively, since this is of the same form as (8) but in one less variable.
4. After n rounds, the verifier has picked random $r_0, \ldots, r_{n-1}$ and it must be shown that
$$P(r_0, \ldots, r_{n-1}) = c_n, \tag{21}$$
which is done through direct evaluation by the verifier.

Note that verifying (17) directly would take the verifier $(\#B)^n$ evaluations of P, but the sumcheck protocol reduces this to a single evaluation at a random point outside of B^n. Since a single evaluation is still outside the verifier's capabilities, the witness W_P that the prover publishes beforehand is a list of values of all the underlying polynomials required for the evaluation of $P(\vec{x})$ (e.j. the polynomials $j, D, S, D^{(k)}, R^{(k)}$ defined in Section 4.1), at each point of I^n.

The security of the sumcheck protocol is reflected by the following theorem:

Theorem 1 (BFL [4]). *If the verifier can query $P(\vec{r})$ reliably and $\sum_{\vec{x} \in B^n} P(\vec{x}) \neq c_0$, then any cheating prover has a probability negligible in $\lambda \equiv nd/\#I$ of causing the verifier to accept, assuming that $d \equiv deg(P) < \#I$.*

Proof. If the value of the sum is incorrect but P_0 is computed correctly as defined in (18) then equation (19) will be false and cause the verifier to reject. However, the cheating prover may lie and return a different polynomial $\tilde{P}_0(x_0)$ which satisfies (19) but was not computed according to (18). The recursive step is then to check that P_0 and $\tilde{P}_0$ agree on a random point r_0. Since distinct polynomials of degree d can only agree on up to d points, this means that there is a probability of at least $(d/\#I)$ that the value c_1 of the next sum is also false, and the same logic applies for each iteration. In the final step the verifier catches any false values via its direct evaluation, so the verification passes only if $P_i(r_i) = \tilde{P}_i(r_i)$ at any of the previous steps, which happens with probability at most $(1 - d/\#I)^n < \exp(-\lambda)$ where we have used the inequality $1 - x < e^{-x}$ for $x \in (0, 1)$.

The above suggests picking $\#I = \mathcal{O}(nd)$ to achieve constant soundness, but note that we also assume that the verifier can query $P(\vec{r})$ reliably while in reality it only receives a list of values which may not even represent a degree-d polynomial. This problem is addressed in Section A.2 and actually requires increasing the size of I to $\mathcal{O}(n^2 d)$, which means the size of the witness is $\mathcal{O}(n^{2n} d^n)$ field elements.

A.1 Computation of the PCP witness via multievaluation

Let $\{f^i(\vec{x})\}_i$ be the set of lookup polynomials used in our arithmetization. The heaviest part of the prover's work is the construction of the witness, which involves evaluating each $f^i(\vec{x})$ (which are of degree $b-1$ in n variables, as defined in (10)) at all points in I^n for some subset $I \subset \mathbb{F}_{p^2}$ of size $\#I = \mathcal{O}(n^2 d)$ where $d = \mathcal{O}(b)$ is the degree of the polynomial in (17).

The problem of batching evaluations of a polynomial to achieve a higher efficiency is known as polynomial multievaluation, and the multivariate case can be easily reduced to the single-variable case. Indeed, for a multivariate polynomial

$$f(x_0, \dots x_{n-1}) = \sum_{i_0, i_1, \dots, i_{n-1}} f_{(i_0, \dots, i_{n-1})} x_0^{i_0} \cdots x_{n-1}^{i_{n-1}},$$

we may rewrite

$$f(x_0, \dots, x_{n-1}) = \sum_{i_1, \dots, i_{n-1}} \hat{f}_{i_1, \dots, i_{n-1}}(x_0) x_1^{i_1} \cdots x_{n-1}^{i_{n-1}}, \tag{22}$$

where

$$\hat{f}_{i_1, \dots, i_{n-1}}(x_0) = \sum_{i_0} f_{i_0, i_1, \dots, i_{n-1}} x^{i_0},$$

which suggests the following procedure:

1. For each $(i_1, \dots, i_{n-1}) \in B^{n-1}$, evaluate the single-variable polynomial $\hat{f}_{i_1, \dots, i_{n-1}}(x_0)$ at each point $x_0 \in I$
2. Use (22) to solve the problem recursively, which now requires evaluations at all combinations of only $n-1$ variables, repeated $\mathcal{O}(nb)$ times (once for each value of x_0)

Let $T(n)$ be the complexity of obtaining the evaluations at all combinations of n variables. The above procedure yields the recursion

$$T(n) = b^{n-1} T(1) + nbT(n-1),$$

which resolves to

$$T(n) = \mathcal{O}\left((nb)^n T(1)\right) \tag{23}$$

For the single variable case we make use of the Newton basis representation: given a set of points $x_0, x_1, \dots, x_{b-1}$ we define $x^{\underline{i}} := (x-x_0)(x-x_1)\cdots(x-x_{i-1})$, and then every polynomial of degree $b-1$ has a representation $f = \sum_i \tilde{f}_i x^{\underline{i}}$, where $\tilde{f}_i$ are known as the coefficients of f in the Newton basis associated to the points $\{x_i\}$. Note that equation (22) takes the same form in the Newton basis, so the same reduction from multi-variable to single-variable applies.

Assuming that we know the values of f at the points $\{0, 1, \dots, b-1\}$, we can extrapolate to find the values at $\{hb, hb+1, \dots, hb+b-1\}$ for any $h \in \mathbb{N}$ via the following procedure:

1. Construct the coefficients of f in the $\{0, 1, \ldots, b-1\}$ Newton basis
2. Obtain new coefficients by shifting the basis to $\{hb, hb+1, \ldots, hb+b-1\}$
3. Use the new coefficients to evaluate $f(hb), f(hb+1), f(hb+b-1)$

Step 1 is accomplished easily by setting $\tilde{f}_0 = f(0)$ and then

$$\tilde{f}_i = f(i) - \sum_{j=0}^{i-1} \frac{\tilde{f}_j}{(i-j-1)!},$$

for each $0 < i \leq b-1$, which runs in $\mathcal{O}(b)$.

Step 2 can be done by first transforming from the Newton basis to the monomial basis, and then from the monomial basis to the new Newton basis. Both procedures can be done in $\mathcal{O}(b \log b)$ (see [10, Section 3.1]).

Finally, step 3 is accomplished by inverting step 1:

$$f(hb+i) = \tilde{f}_i + \sum_{j=0}^{i-1} \frac{\tilde{f}_j}{(i-j-1)!},$$

which also runs in $\mathcal{O}(b)$.

Going back to (23), this means the evaluation of an $n-$variable polynomial of degree $b-1$ at $b-1$ points can be done in time $\mathcal{O}((nb)^n b \log b)$. Recall that the sumcheck protocol requires evaluation of f in a set of size $\#I = \mathcal{O}(n^2 d)$ where $d = \mathcal{O}(b)$, so it is natural to split I into $\mathcal{O}(n^2)$ subsets I_k of size $b-1$, and perform the multievaluation in batches. We assume the prover processes all batches in parallel, and also simultaneously for each polynomial f, so the whole witness construction takes parallel time $\mathcal{O}((nb)^n b \log b)$ with $\mathcal{O}(n^2 \log p)$ parallelism (where the factor of $\log p$ comes from the number of polynomials in our arithmetization, namely the $D^k(\vec{x})$ and $R^k(\vec{x})$ defined in (14) and (15)).

A.2 The degree test

As previously mentioned, the sumcheck protocol depends on the assumption that the verifier is able to query each $f_i(\vec{r})$ reliably, but the list of values in the witness may be arbitrary. Note that a polynomial of degree $b-1$ in n variables is uniquely determined by its values at any b^n points, meaning that every such polynomial corresponds to some computation history, and it suffices to show that the table of values is consistent with some degree-$(b-1)$ polynomial.

Since there is large redundancy in representing a polynomial by its table of values, we can treat said representation as an error correcting code, which corresponds to the Reed-Solomon code [30], where the codeword can be inferred by looking at only a small portion of itself and another portion is used as a consistency check. The idea is that we reduce to a single-variable polynomial by fixing random values to all but one coordinate, at which point it is within the verifier's capabilities to check that the values along this subspace are consistent with a single-variable degree$-(b-1)$ polynomial.

Algorithm 1 Degree test

Input: A table of values for $f : I^n \to \mathbb{F}_{p^2}$ and an integer c
Output: $True$ if f is largely consistent with a degree-$(b-1)$ polynomial, $False$ otherwise

$\vec{r} \xleftarrow{\$} I^n$
$j \xleftarrow{\$} \{0, \ldots, n-1\}$
for $i = 0$ to $b-1$ **do**
    $\vec{r}_i \leftarrow (\vec{r}$ with its j^{th} coordinate replaced by $i)$
    `Query` $f(\vec{r}_i)$
end for
$g \leftarrow$ `POLY_EXTRAPOLATE`$(f(\vec{r}_0), \ldots, f(\vec{r}_{n-1}))$.
for $i = 1$ to c **do**
    $t \xleftarrow{\$} I$
    $\vec{r}_{test} \leftarrow (\vec{r}$ with its j^{th} coordinate replaced by t)
    `Query` $f(\vec{r}_{test})$
    if $f(\vec{r}_{test}) \neq g(t)$: **return** $False$
end for
return $True$

The degree test, described in Algorithm 1, first constructs a degree-$(b-1)$ polynomial by running the extrapolation subroutine from last section on the first b values, and then checks that this polynomial agrees with f in another c randomly chosen points. Taking $c > n$, it has a complexity of $\mathcal{O}(c \log b)$ due to the multievaluations and makes $\mathcal{O}(c)$ queries to f.

The soundness of the test follows from the next theorem:

Theorem 2. *Let ρ be the smallest number such that f differs from a degree-$(b-1)$ polynomial only in a fraction ρ of the points in I^n. If $\rho > 20nb^2/\#I$ then the probability of the degree test with $c > n^2(b+1)$ passing is bounded above by $1-\delta$ for some constant $\delta > 0$ that is independent of n, b.*

Proof. See Proposition 5.6, Theorem 5.13 and Remark 5.15 from [4].

By picking $\#I = \mathcal{O}(nb^2)$, a successful degree test with $c = \mathcal{O}(n^2 b)$ gives us a constant bound for ρ with constant confidence. This means that the probability of a "false" value coming up in the final step of the checksum protocol is also bounded by a constant, and we can achieve arbitrarily high soundness by repeating $\mathcal{O}(1)$ times. Since the number of lookup polynomials that need to be tested in our isogeny construction is logarithmic in p, this means that the degree tests need to query a total of $\mathcal{O}(n^2 b \log p)$ values.

B SNARGs from PCPs

Formally, a SNARG for an $\mathcal{NP}$ language L consists of two algorithms,

- **SNARG-Prove:** on input (x, w) such that $x \in L$ and w is an $\mathcal{NP}$ witness for x, outputs a proof π

– **SNARG-Verify:** on input (x, π), outputs either $true$ or $false$,

which satisfy three properties:

1. **Succinctness:** The new proof π is of length $polylog(|xw|)$.
2. **Completeness:** SNARG-Verify$(x, \pi) = true$ whenever $\pi =$ SNARG-Prove(x, w) for some w such that (x, w) is a valid member-witness pair for L.
3. **Computational Soundness:** If $x \notin L$, any PPT algorithm has negligible probability of producing a proof π such that SNARG-Verify$(x, \pi) = true$.

In addition, we set efficiency constraints: the proving algorithm must run in time $poly(|xw|)$, while the verification algorithm runs in $polylog(|xw|)$.

Probabilistic Checkable Proofs achieve similar goals, but they require communicating a witness that is much larger than the original. Moreover, the interactiveness of the protocol means that it cannot be directly dropped in for various primitives like VDFs. Micali [26] solved both problems by proposing a transform that acts on any PCP to produce a SNARG. That is, it achieves succinctness and non-interactiveness, at the cost of replacing soundness by computational soundness (i.e. a computationally unbound attacker may be able to fool the verifier, and soundness is defined in terms of a security parameter).

To achieve succinctness, the Micali transform encodes the witness W into a Merkle tree, and communicates only the root of the tree as a commitment rather than the whole witness. In our case, one Merkle tree of heigth $\mathcal{O}(n \log(nb))$ is constructed for each polynomial f, and each leaf contains the value of $f(\vec{x})$ for one $\vec{x} \in I^n$. Whenever a query for $f(\vec{r})$ is made, the evaluator responds with the value $f(\vec{r})$ along with a verification path from the corresponding leaf to the root, and the verifier computes the hashes in the verification path to check that it agrees with the Merkle root commitment. Therefore, each query to the witness now comes at a cost of $\mathcal{O}(n \log(nb))$ for both verifier and prover, which increases the verifier's complexity to $\mathcal{O}(n^3 b \log(nb) \log p)$ due to the queries in the degree test (Appendix A.2).

Since the resulting protocol depends only on public random coins, it can easily be made non-interactive via a Fiat-Shamir transform [19]. For our sumcheck protocol this can be achieved by obtaining each value r_i from a cryptographic hash of the previous c_i value and the Merkle root commitment, whereas for the degree test it suffices to pick the evaluation points from a hash of the Merkle root along with a counter value. As long as the hash function is collision-resistant, it is straightforward to show that the soundness of this construction reduces to the soundness of the original PCP (see [26] for concrete proofs of security).