Parallel Hardware for Isogeny-based VDF:

Attacker's Perspective

David Jacquemin, Anisha Mukherjee, Ahmet Can Mert and Sujoy Sinha Roy

IAIK, Graz University of Technology, Austria,
{david.jacquemin,anisha.mukherjee,ahmet.mert,sujoy.sinharoy}@iaik.tugraz.at

Abstract.
The long running time of isogeny-based cryptographic constructions has proved to be a boon in disguise for one particular type of primitive called Verifiable Delay Functions (VDFs). VDFs are characterised by sequential function evaluation but an immediate output verification. In order to ensure secure use of VDFs in real-world applications, it is important to determine the fastest implementation. Considering the point of view of an attacker (say with unbounded resources), this paper aims to achieve the fastest possible hardware implementation of isogeny-based VDFs. It is the first work that implements the 2^T-isogeny walk involved in the evaluation step of an isogeny VDF. To meet our goal, we use redundant representations of integers and introduce a new lookup table-based algorithm for modular reduction. We also provide a survey of elliptic curve arithmetic to arrive at the most cost-effective curve computations and propose an improvement of the point doubling algorithm for better parallelism. The evaluation step of a VDF is defined to be sequential, which means that there is limited scope for parallelism. Nevertheless, taking this constraint into account our proposed design targets the highest levels of parallelism possible on an architectural level of an isogeny VDF implementation. We provide detailed analysis of all our arithmetic modules as well as estimates for their critical path delays and area consumption. Our 28nm ASIC design computes a $4^{100} = 2^{200}$-isogeny in $7.1\mu s$. It is the first high-performance ASIC implementation for evaluation of isogeny VDFs.

Keywords: Verifiable delay functions, Isogeny, Redundant representation, ASIC

1 Introduction

The classic adage, "Good things come to those who wait" has been made palpable in recent times by blockchains and cryptocurrencies: two of the most popular modern-day technologies. Blockchains rely on cryptographic protocols for authorising and validating digital exchanges, often aided by 'randomness' in the form of desirable time delays to avoid counterfeits. Considering block variables such as timestamps as a source of entropy or randomness have shown to be vulnerable to bias because a block miner has the potential to manipulate them. As an example, consider an on-chain lottery where the miner has to guess if the next block hash is even or odd. While betting on even, if a miner is able to generate a block comparatively 'faster' than the others and finds out that it is odd, they can discard it, thereby increasing the probability of getting an even hash the next time and hence, winning the lottery. Verifiable Delay Functions (VDF) are cryptographic primitives that came as a solution to mitigate such foul-play. They possess the ability to run for a certain fixed amount of sequential time T but their result can be verified rather quickly. In applications that need the generation of randomness beacons from public sources like stock prices, VDFs can ensure security by adding enough delay to calculate the beacon, thus preventing powerful seasoned traders to adjust the market for their gain.

Thus, VDFs are useful only when they run for more than a specific time. Determining the fastest implementation or identifying speed-ups are immensely important to set the required security level of a VDF instance.

One of the earliest attempts at construction was to compute a T-long chain of a hash function H (which would take T steps irrespective of amount of parallelism), however the verification of the output, say, $y = H^T(x)$, takes the same order of time as the only way to verify is to recompute the composition of the functions. So, although it is a delay function, it is not efficiently verifiable. Constructing delay functions that were easily verifiable as well as quantum-secure became an interesting open problem. After their introduction by [BBBF18] research around VDFs intensified. Since by virtue of their construction, VDFs need to be sequential, there is limited scope for algorithmic optimisation and parallel computations. Thus, a VDF implementation on hardware has different aims and challenges than the implementations of conventional cryptographic primitives. Like the authors of [SHT22] mention, achieving the highest performance is the main goal in a VDF attack implementation whereas area or power consumption does not hold much significance. The reason is that knowledge of the time required for a VDF under a parameter set is the key to also setting up security parameters and ensuring their standardised use in the public domain. This paper is the first work towards realising such an implementation for isogeny VDFs. Hence why, the attacker in this paper is assumed to be very powerful with almost unlimited resources, e.g., government organizations.

Several forms of VDFs have been proposed so far, such as the ones based on computing square roots in a modular field or the more recent by [Wes19] and [Pie18] on groups of unknown order. In this paper, we particularly focus on a new type of VDF constructed using isogenies on supersingular elliptic curves.

Isogenies came into limelight with the works [CLG09], who presented a collision-resistant hash function based on deterministic walks in isogeny graphs of supersingular elliptic curves. Soon they gained popularity in the cryptographic landscape because of their resistance to quantum attacks and smaller key sizes. [FMPS19, CSRHT22] are some of the recent works on isogeny-based VDFs. Isogeny VDFs are interesting because they can be constructed by combining already existing cryptographic research on isogenies with respect to efficiency and security [FMPS19].

In the literature there are several implementations (software and hardware) of VDFs based on modular square roots, time-lock puzzles [MÖS22, SHT22] but when it comes to isogeny-based VDFs, high-performance implementation works are scanty. A proof-of-concept Sage implementation of the isogeny-based VDF [FMPS19] on an Intel Core i7-8700 processor is provided by the authors of the paper. They choose a 1506-bit prime to achieve 128-bit security. These results correspond to 2-isogeny computation and evaluation during the execution of the VDF components. The following work [CSRHT22] leaves it as an open area of research to decide concrete parameters for their isogeny-based VDF construction. [BDF21] on the other hand, give a form of isogeny-based time delay primitive which they refer to as Delay Encryption and discuss certain implementation-level optimisations. Since their basic building blocks are closely related to [FMPS19]'s VDF primitive, these optimisations, in theory, could apply to the VDF too. However, the authors note that further investigations are required to test their practical advantages. Thus the only performance results for isogeny VDFs are based on software implementations. It would therefore come as no surprise that an optimised hardware implementation of isogeny computation would easily beat the existing benchmarks. We however note that [SB23] presents a design for a high-performance hardware accelerator that can aid isogeny-based cryptographic primitives such as the SIKE key exchange scheme. It employs optimisations within the curve arithmetic to improve performance. It is however worth mentioning here that isogeny-based VDF schemes remain completely unaffected by the recent attacks on SIDH/SIKE [CD22, Rob23, MMP$^+$23] because the underlying supersingular isogeny problem remains secure. These schemes were broken because of their use of auxiliary image points being computed through isogenies that leaked sensitive information.

1.1 Main Contributions

At the very outset, we present a survey of the different forms of elliptic curves as the form we choose will play a crucial role in determining the extent of efficiency in our implementation. An optimal curve would be the one that requires the least amount of resource expenditure during elliptic curve arithmetic. We end our analysis by presenting a fitting explanation for our choice of Montgomery curves and we also propose a slightly different point-doubling algorithm, whose benefits we discuss in section 4.1.

As stated earlier, since the branch of isogeny-based VDFs is fairly recent, not much work has been done towards determining the fastest implementation or providing performance results corresponding to different sets of parameters. Our work is the first to address them by providing an efficient and the fastest hardware implementation of the l^T-isogeny walk. Note that, hardware implementations of isogeny walks exist in the context of post-quantum cryptography (PQC) [SB23]. However, an isogeny VDF implementation would greatly differ from such a PQC implementation because of the vast difference in their respective parameter sizes (1506 vs 434 bits for SIKE [JAC+22]) as well as their constraint conditions.

We analytically compare the execution times of the different isogeny strategies depending on the number of parallel isogeny evaluation units, as explained in section 4.2. We settle at the fastest strategy depending on the number (1 or $k-1$) of the evaluation modules in the context of our isogeny VDF. We find that using a redundant representation for integers called the Carry-Save representation (CS) [Par00] and hence carry-save adders (CSA) for all the isogeny modular arithmetic significantly decrease the latency of the hardware architecture. Using CS representation for isogeny arithmetic also led us to design a new method for modular reduction.

We design a highly parallelized hardware architecture to efficiently compute a 4^k-isogeny. In this paper, we compute a $4^k = 2^T$-isogeny walk, using 4-isogenies as building blocks instead of 2-isogenies for better performance. Although both of these computations are mathematically equivalent, it is more efficient in terms of complexity and latency to compute one 4-isogeny instead of two 2-isogenies as pointed out by [FJP14]. Our design is also mostly parameterisable (it can accommodate varying parameter sets). It is therefore capable of handling the large parameter sizes that a VDF implementation demands.

As a result, our implementation is capable of performing a l^T (with $l = 2$) degree isogeny with a much better throughput (10069 isog/ms) during evaluation than [FMPS19] estimates (0.75 isog/ms in SW). Hence our choice of curves and strategies, algorithmic optimisations, as well as our tweaks in the architecture design, helps us get significantly closer towards achieving the fastest isogeny computation possible.

2 Mathematical background

2.1 Verifiable Delay Functions

Verifiable Delay Function or VDF is a mathematical function that takes T sequential steps for its evaluation irrespective of the processing power, however, the verification of the output of its evaluation is efficient and almost immediate. A VDF consists of the following three algorithms.

1. $\texttt{Setup}(\lambda, T) \rightarrow (\texttt{ek}, \texttt{vk})$: It takes a certain security parameter λ and a delay parameter T to set public parameters consisting of the evaluation key `ek`, and the verification key `vk` for the next steps. It should have a runtime in $poly(\lambda)$.

2. $\texttt{Eval}(\texttt{ek}, s) \rightarrow (a, \pi)$: This step involves the evaluation of the function on a given input s using `ek` to produce an output, $a = f(s)$, which is sequential in T but cannot be completed in a time less than T. It may also produce a proof π.
3. $\texttt{Verify}(\texttt{vk}, s, a, \pi) \rightarrow \{\texttt{true}, \texttt{false}\}$: It is the verification of the output a in time $poly(\lambda)$ using `vk` and the proof π, that a is indeed the correct pre-image corresponding to the input s.

Some examples of VDFs in the literature are listed as follows:
Modular square roots: Given a prime $p = 3 \bmod 4$, compute a square root $a = \sqrt{s} \bmod p$ using the formula, $a = s^{\frac{p+1}{4}}$. Clearly, evaluating the square root is sequential and the run time increases logarithmically as p grows but the verification is done in a single step; just check if $a^2 = s$. However, the computation phase actually turns out to be parallelisable. [DN93, LW17] are two well-known VDFs based on modular square roots.

Rivest-Shamir-Wagner time-lock puzzles, [RSW96]: Based on the RSA construction, it selects a modulus $N = pq$ (p, q are prime) and sets the output to $a = s^{2^T} \bmod N$. Unless someone knows the prime factorisation of N (which is secret), they would need to go through all the sequential powering steps to achieve a. The knowledge of the Euler-ϕ function for N, $\phi(N)$, will provide a shortcut to the evaluation, of course. It lacks efficient verification because the factorisation of N has to be compromised.

Wesolowski's and Pietrzak's VDF: To overcome the problem of efficient verification of time-lock puzzles, both [Wes19] and [Pie18] came up with their own versions of VDFs. [Wes19] worked with groups of unknown orders and [Pie18] introduced a new verification protocol for Rivest-Shamir-Wagner time-lock puzzles. Both these constructions, however, rely on interactive verification protocols.

Univariate permutation polynomials (UPP): This approach uses permutation polynomials of degree, say, T in a finite field $\mathbb{F}_p$ and bases the evaluation on inverting these polynomials which is sequential in time. [BBBF18] based their initial VDF discussions on such permutation polynomials.

VDFs using SNARGs: [BBBF18] and [DGMV20] independently designed a more theoretical VDF based on *succint non-interactive arguments* or SNARGs. This concept was however used in a slightly different VDF construction by [CSRHT22], which we mention in section 2.4.2.

Since the already existing VDFs had certain shortcomings, a new branch of VDFs using isogenies of supersingular elliptic curves has gained the attention of the cryptographic community [FMPS19, CSRHT22].

2.2 Elliptic curves

An elliptic curve E defined over a field K with $char \neq 2, 3$ is a smooth, projective algebraic curve of genus 1 with a special point, the unique point $\mathcal{O}$. The points of an elliptic curve form a group under addition with $\mathcal{O}$ as the identity element. The standard normal form of an elliptic curve is the Weirstrass form given by

$$E_w : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{1}$$

with $a_i \in K$. For fields of characteristic greater than 3, there is a short Weirstrass form

$$E_{sw} : y^2 = x^3 + ax + b \tag{2}$$

such that $4a^3 + 27b^2 \neq 0$. Every elliptic curve is defined uniquely up to K-isomorphism (except for $char \neq 2, 3$) through its j-invariant,

$$j(E) = 1728 \frac{4a^3}{4a^3 + 27b^2}.$$

Two frequently used forms of elliptic curve equations over the affine coordinates are the Montgomery and the Edwards, given by the respective equations:

$$E_m : by^2 = x^3 + ax^2 + x \tag{3}$$

and

$$E_{ed} : x^2 + y^2 = 1 + dx^2y^2;\ d \notin \{0, 1\} \tag{4}$$

For faster computations during implementation, the affine coordinates (x, y) are often replaced by projective coordinates, $(X, Y, Z),\ Z \neq 0$. The forward mapping is given by $(x, y) \to (xZ, yZ, Z)\ Z \neq 0$ and the reverse mapping by $(X, Y, Z) \to (X/Z, Y/Z)$.

Let E_a and E_b be two elliptic curves over $\mathbb{F}_{p^2}$. An isogeny $\phi : E_a \to E_b$ is defined as a non-constant rational map which is also a group homomorphism from $E_a(\mathbb{F}_{p^2})$ to $E_b(\mathbb{F}_{p^2})$ that preserves the identity $\mathcal{O}$. Two elliptic curves are isogenous if their orders (number of points over $\mathbb{F}_{p^2}$) are the same [Tat66]. The degree of an isogeny is its degree as a rational map [Sil09]. An isogeny is separable if it induces a separable extension of function fields [FMPS19]. When the degree of the isogeny, $deg(\phi) = l$ is coprime to p, the isogeny is necessarily separable. An isogeny that is separable has a one-to-one correspondence with its kernel, so this isogeny can be computed with the knowledge of its kernel using Velu's formula. For two isogenies $\phi : E_a \to E_b$ and $\psi : E_b \to E_c$, there exists a composite isogeny $\phi \circ \psi : E_a \to E_c$ such that, $deg(\phi \circ \psi) = deg(\phi) \cdot deg(\psi)$. If an isogeny ϕ has a degree $deg(\phi) = \prod_{i=1}^{n} p_i^{k_i}$ then it can be factored as a composition of k_i isogenies of degree p_i for all $i \in \{1, 2, \cdots, n\}$. For an l-isogeny $\phi : E_a \to E_b$, there is a unique l-isogeny $\hat{\phi} : E_b \to E_a$ such that $\phi \circ \hat{\phi} = [l]$ on E_b, and vice versa, where $[l]$ denotes the *multiplication-by-l* map.

2.2.1 Elliptic curve arithmetic

Computing isogenies requires elliptic curve arithmetic operations such as point doubling and then the actual rational map corresponding to the l-isogeny using Velu's formula. Arithmetic over affine coordinates (x, y) are usually traded with projective coordinates (X, Y, Z). Efficient explicit formulae often work with the X and Z coordinates.

We discuss optimisation techniques with respect to point doubling and algorithms for different elliptic curves later in Sec. 4.1. Here we briefly mention the expressions for Velu's formula on two popularly used elliptic curves: Montgomery and Edwards. Recall that any separable isogeny can be identified by its kernel. Given the kernel G, Velu's formula gives a method to compute the corresponding separable l-isogeny. While discussing cost estimations, we will denote multiplication by MUL and squaring by SQR.

There exist abundant discussions on efficient isogeny computations over Montgomery curves, for example in [JAC+22]. Let $(x_4, y_4) \in E_m$ be a 4-torsion point with $x_4 \neq \pm 1$ that generates the kernel $G = \langle (x_4, y_4) \rangle$. Then the curve, $E_{m'} : b'y^2 = x^3 + a'x^2 + x$ corresponding to the unique 4-isogeny, $\phi_4 : E_m \to E_{m'}$ is such that (a', b') is defined by the equation,

$$(a', b') = \left(4x_4^4 - 2, -x_4(x_4^2 + 1) \cdot B/2\right).$$

The 4-isogeny, $\phi_4 : (x_P, y_P) \to (x_{\phi_4(P)}, y_{\phi_4(P)})$ for a point $P = (x_P, y_P) \notin G$ can be described by the following two equations:

$$x_{\phi_4(P)} = \frac{-(x_P x_4^2 + x_P - 2x_4) x_P (x_P x_4 - 1)^2}{(x_P - x_4)^2 (2x_P x_4 - x_4^2 - 1)}$$

$$y_{\phi_4(P)} = y_P \cdot \frac{-2x_4^2(x_Px_4-1)(x_P^4(x_4^2+1)-4x_P^3(x_4^3+x_4)+2x_P^2(x_4^4+5x_4^2)-4x_P(x_4^3+x_4)+x_4^2+1)}{(x_P-x_4)^3(2x_Px_4-x_4^2-1)^2}.$$

In projective XZ-coordinates, we take a point $P = (X_4 : Y_4)$ of order 4 on $E_{A/C}$. First, we compute $(A_{24}^+, C_{24}) \sim (A' + 2C' : 4C')$ for projective parameters A', C' of the image curve $E_{A'/C'}$ and constants $(K_1, K_2, K_3) \in (\mathbb{F}_{p^2})^3$ such that the 4-isogeny image curve coefficients as well as the image Q' of a point $Q = (X : Z)$ can be computed as per algorithms in [JAC+22]. Both of these computations require a total of 6 MUL + 6 SQR.

In the context of Edwards curves, [KYK+20] describes an optimised 4-isogeny computation in projective YZ-coordinates. Let $(d : 1) \sim (D : C)$ in eqn. (4). Then the curve coefficients D', C' of the image curve E'_{ed} under the 4-isogeny ϕ_4 with respect to the 4-torsion point $P = (Y_4 : Z_4)$ is given by:

$$\begin{aligned} D' &= 8Y_4 \cdot Z_4 \cdot (Y_4^2 + Z_4^2) \\ C' &= (Y_4 + Z_4)^4. \end{aligned}$$

The evaluation of the 4-isogeny ϕ_4 via the image $(Y' : Z')$ of the point $P = (Y : Z)$ on E_{ed} is given by the relations:

$$\begin{aligned} Y' &= (Z^2 \cdot Y_4^2 + Y^2 \cdot Z_4^2) \cdot Y \cdot Z \cdot (Y_4 + Z_4)^2 \\ Z' &= (Z^2 \cdot Y_4^2 + Y^2 \cdot Z_4^2)^4 + 2Y^2 \cdot Z^2 \cdot Y_4 \cdot Z_4 \cdot (Y_4^2 + Z_4^2). \end{aligned}$$

2.3 Supersingular isogenies

An isogeny from an elliptic curve E to itself (or the zero map) is called an endomorphism. The set of all endomorphisms of E, denoted by $\mathrm{End}(E)$ forms a ring under addition and composition. If $\mathrm{End}(E)$ is isomorphic to an order of a quadratic imaginary field, E is called ordinary. Otherwise, if $\mathrm{End}(E)$ is isomorphic to a maximal order in a quaternion algebra then the curve is called supersingular. We work with supersingular elliptic curves in this paper. Any supersingular elliptic curve over a field of characteristic p is isomorphic to a supersingular elliptic curve over $\mathbb{F}_{p^2}$. A supersingular l-isogeny graph has as vertices the supersingular j-invariants in $\mathbb{F}_{p^2}$ and its edges are the l-isogenies. The Supersingular l-Isogeny Problem is a 'hard' problem that states the following, 'given a prime p and two supersingular elliptic curves E and E' over $\mathbb{F}_{p^2}$, find a path from E to E' in the l-isogeny graph'.

2.3.1 Strategies for computing isogenies

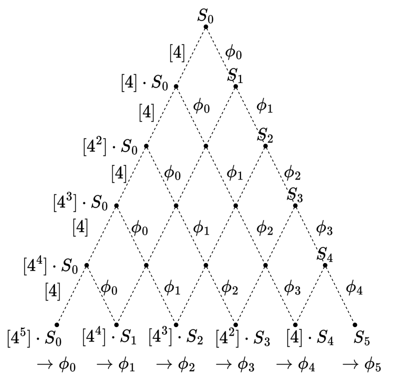


Figure 1: Computation structure for $\phi = \phi_5 \circ \phi_4 \circ \phi_3 \circ \phi_2 \circ \phi_1 \circ \phi_0$

Ever since the SIDH protocol was first proposed, a lot of work has been done to optimise the computation of smooth large-degree isogeny [FJP14, JAC+22] with different strategies.

Computing a large degree l^k isogeny ϕ is very inefficient, instead we always split it into multiple l-isogenies: $\phi = \phi_{k-1} \circ \cdots \circ \phi_2 \circ \phi_1 \circ \phi_0$. An example is provided in Fig. 1 for $k = 6$ and $l = 4$, there, computing the isogeny means starting from S_0 to reach S_5 (thus having ϕ). To compute any of the ϕ_i for $i \in [0 : 5]$, we must first find one point in the kernel: $[4^{5-i}] \cdot S_i$. In order to get ϕ, we must compute a point in the kernel of all the different ϕ_i for $i \in [0 : 5]$, which means reaching all the points at the bottom of the graph in Fig. 1. There are two main operations in isogeny "arithmetic": point quadrupling (or two point doubling) and 4-isogenies with $l = 4$. The different strategies refer to the different sequences of point doubling and 4-isogenies used to compute ϕ. There are strategies that are more efficient than others, we will present three of them.

The "basic" strategy: this strategy is straightforward. For a 4^6-isogeny of a point S_0 of order 4^6, first we start from S_0 by computing point doubling (DBL) operations until we reach a point of order 4, which is $[4^5] \cdot S_0$. We then use Vélu's formula for a 4 degree isogeny on the point of order of 4 to get the isogeny ϕ_0 and the image of S_0 through the isogeny: $S_1 = \phi_0(S_0)$. Then, we repeat this process on S_1 but this time we only compute $[4^4] \cdot S_1$ here as S_1 is now of order 4^5. We will repeat this process until we reach S_5, which is the image of S_0 through a 4^6-isogeny. In Fig. 2, the left figure shows the path to take for this strategy in the case of a 4^6-isogeny.

The Full Evaluation strategy: we will mention one where we switch point doubling for 4-isogeny evaluation, 4-iso-e. First, we start by computing $R_1 = [4] \cdot S_0$ using two DBL. We repeat this process until we reach $R_5 = [4] \cdot R_4 = [4^5] \cdot S_0$. We then proceed to compute a 4-isogeny using R_4 and compute the image of all the elements of the sequence of point $(R_i)_{i\in[0,4]}$ through this isogeny ϕ_0 (with $R_0 = S_0$). We repeat this process again by using a point in the kernel that we already have computed: $\phi_0(R_4)$ to generate the next isogeny ϕ_1. The reason is that $\phi_0(R_4)$ is a point of order of 4: $R_4 = [4^4] \cdot S_0$ has an order of 8, so $\phi(R_4)$ has an order of $8 - 4 = 4$. We repeat this process until we reach the point S_5. This strategy trades two DBL operations for a 4-iso-e compared to the basic strategy. It also has another significant advantage: it can be heavily parallelised. All of the isogeny evaluations through the same isogeny ϕ_i can be computed in parallel.

The Optimized strategy: first introduced by [FJP14, JAC+22], this strategy is done by finding an optimum computation strategy. Those strategies, as shown in the right figure of Fig 2, are well-balanced strategies because they tend to have a similar cost of DBL and 4-iso-e. Those strategies also avoid going through some of the internal points (eg. $\phi_0(R_1)$) in the isogeny tree which lowers the complexity: every node not reached in the graph is one less DBL or 4-iso-e. First, a linear representation of the strategy is generated (usually hard-coded in the implementation). In Fig. 2, the representation of an optimum strategy used is $[3, 1, 1, 1, 1]$. We first compute $T_1 = [4^3] \cdot S_0$, $T_2 = [4^1] \cdot T_1$, $T_3 = [4^1] \cdot T_2$. The order of T_3 is 4, so we use this point to compute the first isogeny ϕ_0. We then evaluate all the point in $(T_i)_{i\in[0,2]}$ through ϕ_0. Like in the Full Evaluation strategy, $\phi_0(T_2)$ is already of order 4, meaning we can already compute ϕ_1. This step is repeated to get ϕ_2 and to compute S_3. Then we calculate $T_4 = [4] \cdot S_3$ and $T_5 = [4] \cdot T_4$, and finish computing ϕ by getting ϕ_3 from T_4 and ϕ_4 from $\phi_3(T_5)$. We are able to reach S_5 using 14 DBL and 9 4-iso-e, which is lower than with the other strategy: compared to 30 DBL and 5 4-iso-e for the basic one; 10 DBL and 15 4-iso-e for the second strategy.

2.4 Isogeny-based VDFs

Unlike other VDFs that rely on ad-hoc assumptions for proving their security, isogeny-based VDFs enjoy the property of being cryptographically secure due to the underlying supersingular isogeny 'hard problem'. Supersingular isogeny VDFs make use of the fact that

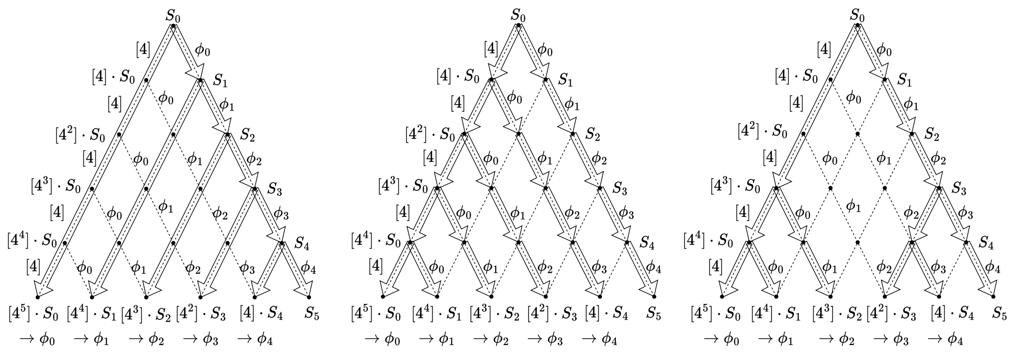


Figure 2: Three isogeny computation strategies

computing l^T-isogenies involves a series of sequential steps whereas the verification using bilinear pairings is instant. There are two constructions of isogeny VDFs, one [FMPS19] introduced in 2019 and the other [CSRHT22] in 2021. While both involve computing isogeny walks during the evaluation, the methods used for verification differ greatly.

To begin with, we give a brief description of the VDF instances discussed in [FMPS19]. They are non-interactive, and by virtue of their design, the proof is empty, meaning that no additional resources are consumed in obtaining the proof; it is a part of the output itself. They need a trusted setup to establish all public parameters. The evaluation is a T-sequential walk on a l-isogeny graph of a supersingular curve E. The verification uses the output isogeny to evaluate a Weil (or a Tate) pairing. The Weil pairing e_N is a form of bilinear pairing over supersingular elliptic curves E and E', $e_N : E[N] \times E'[N] \to \mu_N$ where N is a prime, $E[N]$ and $E'[N]$ are the subgroups of order N containing points in E and E' respectively of order N, and $\mu_N = \{x \in K : x^N = 1\}$.

2.4.1 VDF over $\mathbb{F}_p$

Consider a prime p such that $p+1$ contains a large prime factor N, and a supersingular elliptic curve E over $\mathbb{F}_p$. The choice of the starting degree l has two options: $l = 2$ only if $p = 7 \mod 8$, or, l is a small prime such that $(\frac{-p}{l}) = 1$. For a supersingular elliptic curve E over $\mathbb{F}_p$, let $E[N]$ be its subgroup of N-torsion points and e_N be the Weil pairing defined over $E[N]$. By virtue of its construction [FMPS19], $|E(\mathbb{F}_p)| = p+1$ and E has a unique cyclic subgroup of order N. Let $X_2 = E[N] \cap E(\mathbb{F}_p)$. Define a map $v : E \to \tilde{E}$, such that, $(x, y) \to (u^2x, u^3y)$, where $u \in \mathbb{F}_{p^2} \setminus \mathbb{F}_p$ and $\tilde{E}$ is a quadratic twist of E over $\mathbb{F}_{p^2}$. $\tilde{E}$ has the same order $p+1$ and hence also contains a unique cyclic subgroup $\tilde{X} = \tilde{E}[N] \cap \tilde{E}[\mathbb{F}_p]$. The isogenous image curve E' has the same group structure as E and so contains cyclic groups, $Y_1 = v^{-1}(E'[N] \cap E'[\mathbb{F}_p])$ and $Y_2 = E'[N] \cap E'[\mathbb{F}_p]$, with $\tilde{E}' = v(E')$ as the quadratic twist of E'. The three main steps of the VDF are given below.

- `Setup(`λ`, T)`: For a security parameter λ, choose primes N and p with the properties stated above. Next, choose a supersingular elliptic curve E over $\mathbb{F}_p$ and a suitable degree l of the isogeny to compute the l^T-isogeny $\phi : E \to E'$ and its dual $\hat{\phi}$. Also compute $\phi(P)$ for a choice of generator P of $v^{-1}(\tilde{E}[N] \cap E[\tilde{\mathbb{F}}_p])$. The output is the pair, $(\mathtt{ek}, \mathtt{vk}) = (\phi, (E, E', P, \phi(P)))$.

- `Evaluation(ek,`$Q \in Y_2$`):` For $Q \in Y_2$, compute $\hat{\phi}(Q)$.

- `Verification(vk,`$Q, \hat{\phi}(Q)$`):` Verify, $\hat{\phi}(Q) \in X_2$ and, $e_N(P, \hat{\phi}(Q)) = e_N(\phi(P), Q)$.

2.4.2 Other forms of isogeny VDFs

A construction for VDF over $\mathbb{F}_{p^2}$ follows a similar construction as the one over $\mathbb{F}_p$. The primes N, p are chosen as before, along with a supersingular elliptic curve $E/\mathbb{F}_p$ regarded as a curve over $\mathbb{F}_{p^2}$. A small prime l is chosen as the base isogeny degree. The sets X_1, X_2 and the quadratic twist $\tilde{E}$ also have the same definitions as the previous construction. ϕ is the cyclic l^T-isogeny. One out of the $(l+1)^{T-1}$ choices for the dual $\hat{\phi}$ over $\mathbb{F}_{p^2}$ is chosen through a random non-backtracking walk in the l-isogeny graph. Define the sets, $Y_1 = \phi(X_1)$ and $Y_2 = \phi(X_2)$. However, there is no known efficient way to sample from Y_2 or Y_1, indeed the image curve is generally defined over $\mathbb{F}_{p^2}$ and it has therefore no $\mathbb{F}_p$-twists [FMPS19]. To overcome this problem, instead of a bijection [FMPS19] takes into account the $N - to - 1$ trace map defined as, $Tr : E/\mathbb{F}_{p^2} \to E/\mathbb{F}_p$, $P \mapsto P + \pi(P)$, where π is the Frobenius endomorphism on $E/\mathbb{F}_p$. So for $P \in X_1$ and $R \in E[N]$ the following equation holds: $e_N(P, Tr(R)) = e_N(P, R)^2$. The VDF is defined as $f : E'[N] \to X_2$ such that, $Q \mapsto (Tr \circ \hat{\phi})(Q)$. Hence, in the evaluation step, one needs to compute $(Tr \circ \hat{\phi})(Q)$. Verification involves checking if the following equality is true: $(Tr \circ \hat{\phi})(Q) \in X_2$ and, $e_N(P, (Tr \circ \hat{\phi})(Q)) = e_N(\phi(P), Q)^2$.

Although the use of bilinear pairings means that the aforementioned VDFs are not entirely quantum secure, they can still possess what [FMPS19] call 'quantum annoyance'. [CSRHT22] proposed a quantum-safe version in 2022 by addressing most of the shortcomings of the previous construction by [FMPS19]. The Setup involves selecting a delay parameter T and a prime p such that $p = poly(T)$ and $p^2 \equiv 9 \mod 16$. Since the isogeny walk in the evaluation step is computed only as a function of the j-invariants of the two previous curves, the setup only considers two specific vertices in the 2-isogeny graph corresponding to $j_{-1} = 1728$ and $j_0 = 287496$ respectively. Evaluation is computing an isogeny walk-in $\mathbb{F}_{p^2}$ of length T on a 2-isogeny graph wherein the exact path is determined by an input string s. Since bilinear pairings can be solved using quantum algorithms for solving discrete logarithms, [CSRHT22] replaced them with SNARGs for the verification.

In this paper, we focus on [FMPS19] because their methods of isogeny computation have been extensively studied in the context of elliptic curve cryptography. There exists many implementation works [FJP14, SB23, CLN16] related to isogeny walk computations. [CSRHT22] however presents a theoretical approach where they evaluate the j-invariants and no actual implementation of their methods exist.

2.5 Carry-save representation

The redundant binary representation (RBR) is a binary numeral system that uses more than the minimum amount of bits required to represent an integer. Let a be a positive integer, we define that minimum as $m = \lceil \log_2 a \rceil$. The most common forms of RBRs are the carry-save (CS) and the redundant signed-digit representation (RSD) [Par00]. In the CS representation, an integer a is viewed as the sum of two positive integers:

$$a = a_0 + a_1, \text{ with } a_0 = \sum_{i=0}^{m-1} a_{i,0} \cdot 2^i \text{ and } a_1 = \sum_{i=0}^{m-1} a_{i,1} \cdot 2^i.$$

The most interesting property of RBR is its ability to perform addition (and subtraction) without using any carry chain propagation. This makes addition constant time regardless of the bit size, thus it becomes significantly faster in RBR than in standard representation as the bit size grows [SKN08, MÖS22, SHT22]. In CS representation, addition is done using a long array of one-bit full adders (FA) called carry-save adders (CSA). These enable us to turn a three integers addition into one in CS format (so two integers). So, the addition of two large bit numbers a and b in CS (which is represented by four integers a_0, a_1, b_0, b_1 with $a = a_0 + a_1$ and $b = b_0 + b_1$) is done using two arrays of full adders, see

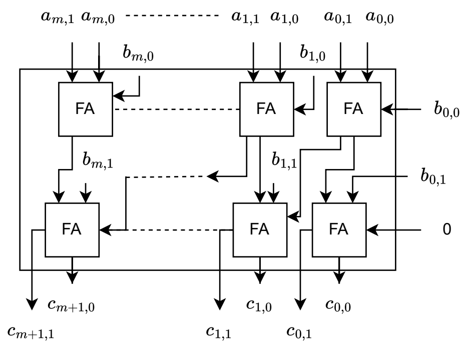


Figure 3: Addition of two numbers in CS representation

Fig. 3. This method for adders is very efficient in terms of hardware, as the critical path of addition is only two full adders [RM19].

As an example, we want to add $a = 12$ and $b = 11$. We can initiate $a_0 = 10, a_1 = 2$ and $b_0 = 0, b_1 = 11$ (or any other combinations). Following Fig. 3, we will add, using a CSA, a_0, a_1 and b_0, the output will be 4 and 8. We then add our two outputs 4,8 and b_1 together in another CSA, the output will be $c = 16$ and $s = 7$. We never recombine the two CS outputs c and s together, because the only way to compute it is to use a multi-bit adder. Those are very expensive, for example, the "classic" ripple-carry adder has a very long carry-chain propagation, making the delay proportional to m-FA, the size of our data. This severely increases the critical path, meaning we would have to in either add more clock cycles thus increasing the latency of our design or lower the frequency (with the same effect). Therefore, when using CS representation, we cannot use any large multi-bit adder to perform any operations.

3 Challenges in our VDF implementation

1. In Sec. 2.2, we discussed elliptic curve arithmetic over its different forms. The first challenge in our isogeny VDF implementation is to choose the right form of elliptic curve that needs the least point doubling and isogeny computations and an appropriate base isogeny degree. The reason why the choice of curve is one of the deciding factors is stated as follows: there exist transformation maps between all forms of elliptic curves. So in theory, an attacker can port the evaluation isogeny to the optimal curve-form to gain speedup. This threat model is valid since the transformation is just a one-time operation done at the beginning and at the end of the VDF evaluation. The worst is that this step can be very efficient: switching from a Montgomery curve to an Edward curve in projective coordinates takes only two additions as explained in [KYK$^+$18, MS16]. We show how we narrow down our search to the starting isogeny degree as 4 and then finally settle for the best curve in Sec. 4.1.

2. Most isogeny implementations in the literature [JAC$^+$22, SB23] use the Optimized strategy as mentioned in Sec. 2.3.1 but is it also the best strategy for our VDF acceleration? This question is studied in Sec. 4.2.

3. [FMPS19]'s construction (given in 2.4.1) uses a 1506-bit prime to achieve 128-bit security. It is highly imperative that we apply carefully-chosen design strategies so as to achieve a fast and efficient implementation.

4. Various prior works have used CS form to speed up certain operations or algorithms [Pur83, MMM03, SHT22]. One such optimisation was also used on elliptic curve cryptography [RM19], only to speed up the Montgomery multiplication. No previous work has tried a full CS form for isogeny-based cryptography. We observe that using CSAs in practice for an isogeny hardware design brings up challenges that have not been previously addressed, such as:
    - Checking the sign. It is well-known that identifying the sign of an integer with utmost certainty in CS representation is not straightforward. Most previous works on CSA have avoided this issue by converting it back to standard representation [SHT22]. Only [KH98] has tried to address this and has managed to narrow down the uncertainty range, which unfortunately, is not enough for isogeny-based cryptography. This problem is addressed in Sec. 4.3.1.
    - Modular addition and subtraction. Various works have addressed the issue of how to do a Montgomery reduction in CS representation [RM19, MÖS22], which is useful following a squaring or multiplication. It is not efficient to use such an algorithm after addition or subtraction. We address the issues with reduction in Sec. 4.3.2 and those with modular subtraction in Sec. 5.2.

4 Algorithmic optimisations

In this section, we explain our solutions for overcoming all the aforementioned challenges one-by-one. We also describe our algorithmic and design optimisations to achieve a massively parallel hardware accelerator for isogeny-based VDF evaluation.

4.1 Choice of curve and isogeny

We first present a table (Table 1) outlining the various choices of elliptic curves and the cost estimations for point doubling. MUL, SQR and ADD represent the number of multiplication, squaring and addition operations respectively.

A Montgomery curve in affine coordinates is given by Eqn. (3). Let $P = (x, y)$ be a point in E_m whose order is not a multiple of 2. Then, the point $[2] \cdot P$ is given by the equation:

$$[2] \cdot P = \left(\frac{(x^2-1)^2}{4x(x^2+ax+1)}, \frac{(x^2-1)(x^4+2ax^3+6x^2+2ax+1)}{8x^2(x^2+ax+1)^2} \right).$$

Using the equivalent projective coordinates, $(A : B : C) \sim (a : b : 1)$ in $\mathbb{P}^2(\mathbb{F}_{p^2})$, Eqn. (3) will take the following form:

$$E_{A/C} : By^2 = Cx^3 + Ax^2 + Cx. \tag{5}$$

Table 1: Operation count comparison

Curve shape	**Point doubling**		
	MUL	**SQR**	**ADD**
Edwards-XZ	2	6	6
Montgomery-XZ	4	2	4
Weierstrass	6	3	6
Jacobi	3	4	8
Hessian	3	6	3
Twisted Hessian	3	6	3

Next, to simplify Eqn. 5, consider $(A : C) \sim (a : 1) \in \mathbb{P}^1(\mathbb{F}_{p^2})$ such that one can define $(A_{24}^{+} : C_{24}) \sim (A+2C : 4C)$. Alg. 2 shows the point doubling in projective XZ-coordinates starting with the point $(X_p : Z_p)$ and using the representation $(A_{24}^{+} : C_{24})$. This algorithm is a slightly modified version of the original point doubling algorithm given in Alg. 1 and provides more opportunities for parallelism in hardware.

Algorithm 1 Original point doubling on Montgomery curves

```
Input: $X_p, Z_p, A_{24}, C_{24}$, $P(X, Z)$ a point
    on the curves $E_a$
Output: $X_{[2]\cdot P}, Z_{[2]\cdot P}$ with $[2] \cdot P$
 1: $t_0 \leftarrow X_p - Z_p$
 2: $t_1 \leftarrow X_p + Z_p$
 3: $t_0 \leftarrow t_0^2$
 4: $t_1 \leftarrow t_1^2$
 5: $Z_{[2]\cdot P} \leftarrow C_{24} \cdot t_0$
 6: $X_{[2]\cdot P} \leftarrow Z_{[2]\cdot P} \cdot t_1$
 7: $t_1 \leftarrow t_1 - t_0$
 8: $t_0 \leftarrow A_{24} \cdot t_1$
 9: $Z_{[2]\cdot P} \leftarrow Z_{[2]\cdot P} + t_0$
10: $Z_{[2]\cdot P} \leftarrow Z_{[2]\cdot P} \cdot t_1$
11: return $(X_{[2]\cdot P}, Z_{[2]\cdot P})$
```

Algorithm 2 Modified point doubling on Montgomery curves

```
Input: $X_p, Z_p, A_{24}, C_{24}$, $P(X : Z)$ a point
    on the curve $E_{A/C}$
Output: $X_{[2]\cdot P}, Z_{[2]\cdot P}$ with $[2] \cdot P$
 1: $t_0 \leftarrow X_p - Z_p$
 2: $t_1 \leftarrow X_p + Z_p$
 3: $t_0 \leftarrow t_0^2$
 4: $t_1 \leftarrow t_1^2$
 5: $Z_{[2]\cdot P} \leftarrow C_{24} \cdot t_0$
 6: $t_2 \leftarrow t_1 - t_0$          ▷ Modification starts.
 7: $X_{[2]\cdot P} \leftarrow Z_{[2]\cdot P} \cdot t_1$
 8: $t_0 \leftarrow A_{24} \cdot t_2$
 9: $Z_{[2]\cdot P} \leftarrow Z_{[2]\cdot P} + t_0$
10: $Z_{[2]\cdot P} \leftarrow Z_{[2]\cdot P} \cdot t_2$
11: return $(X_{[2]\cdot P}, Z_{[2]\cdot P})$
```

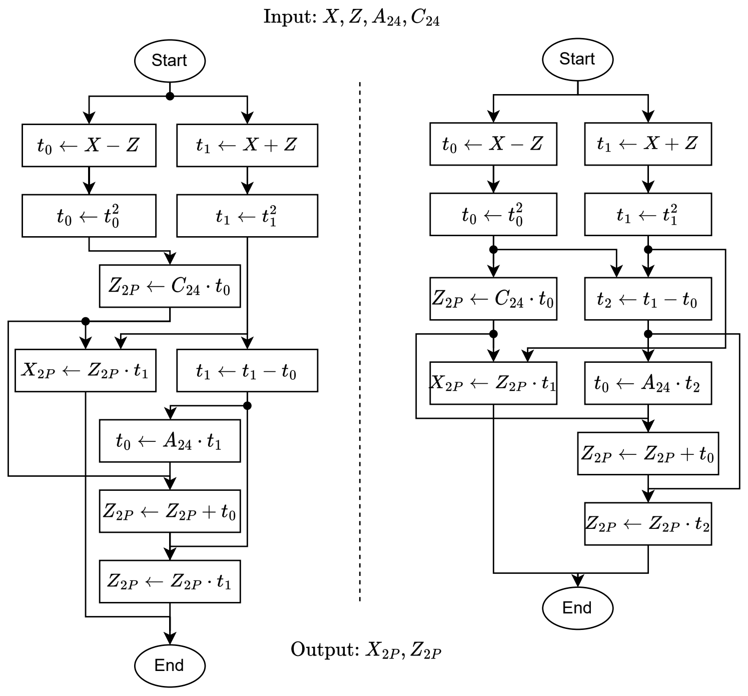

Input: X, Z, A_{24}, C_{24}
Start
$t_0 \leftarrow X - Z$
$t_1 \leftarrow X + Z$
$t_0 \leftarrow t_0^2$
$t_1 \leftarrow t_1^2$
$Z_{2P} \leftarrow C_{24} \cdot t_0$
$X_{2P} \leftarrow Z_{2P} \cdot t_1$
$t_1 \leftarrow t_1 - t_0$
$t_0 \leftarrow A_{24} \cdot t_1$
$Z_{2P} \leftarrow Z_{2P} + t_0$
$Z_{2P} \leftarrow Z_{2P} \cdot t_1$
End
Start
$t_0 \leftarrow X - Z$
$t_1 \leftarrow X + Z$
$t_0 \leftarrow t_0^2$
$t_1 \leftarrow t_1^2$
$Z_{2P} \leftarrow C_{24} \cdot t_0$
$t_2 \leftarrow t_1 - t_0$
$X_{2P} \leftarrow Z_{2P} \cdot t_1$
$t_0 \leftarrow A_{24} \cdot t_2$
$Z_{2P} \leftarrow Z_{2P} + t_0$
$Z_{2P} \leftarrow Z_{2P} \cdot t_2$
End
Output: X_{2P}, Z_{2P}

Figure 4: Flow diagram for the Alg. 1 and Alg. 2

The point doubling formula for a point $P = (x, y)$ on an Edwards curve (given by

Eqn. 4) over the affine coordinates is given in Eqn. 6:

$$[2]\cdot P = \Big(\frac{2xy}{x^2+y^2}, \frac{y^2-x^2}{2-x^2-y^2}\Big). \tag{6}$$

The first obvious replacement in Eqn. (6) can be done using Eqn. (4). We replace x^2 with $\frac{1-y^2}{1-dy^2}$ in Eqn. (6). In projective coordinates, for a point $P = (Y : Z)$, the second coordinate of Eqn. 6 can be written as:

$$\begin{aligned}\frac{y^2-x^2}{2-x^2-y^2} &= \frac{y^2(1-dy^2)-(1-y^2)}{2(1-dy^2)-(1-y^2)-y^2(1-dy^2)}\\ &= -\frac{1+2y^2+dy^4}{1-2dy^2+dy^4}\\ &= -\frac{Z^4+dY^4-2Y^2\cdot Z^2}{Z^4-2dY^2\cdot Z^2+dY^4}.\end{aligned}$$

Simplifying the above equation gives us the following two relations:

$$\begin{aligned}Y_{[2]P} &= -(Z^2-Y^2)^2 - dY^4 + Y^4\\ Z_{[2]P} &= (Z^2-dY^2)^2 - d^2Y^4 + dY^4.\end{aligned}$$

Notice from Table 1 that point doubling over Edwards curve involves the least number of multiplications. For comparison purposes let us assume that during implementation the relationship 1 SQR = 1/2 MUL holds. Then, the total cost of multiplications and squaring over both Montgomery and Edwards turns out to be the same, but additions make Edwards curves less efficient. In other estimations one squaring operation is assumed to be more than half of a multiplication, as shown in [KYK+20] (one SQR is one MUL). The Montgomery curves have significant advantages over others due to the lowest amount of multiplication and squaring. Hence, we have selected Montgomery curves.

Selection of 4-isogenies: We have selected 4-isogenies for our design. The reason for using 4-isogenies instead of 2 or 8-isogenies is that 4-isogenies offer superior efficiency. Table 2 presents the costs for the different operations needed to compute large power-of-two isogenies using Montgomery curves in projective coordinates. The most expensive operations in this table are multiplication and squaring, with the former being slightly more expensive than the latter (see Sec 5.3.1). For our VDF, 4-isogenies necessitate 6 multiplications and 2 squaring compared to the 8 multiplications required by two 2-isogenies. It is evident from Table 5 that both point-doubling and 4-isogenies have similar critical paths, unlike 2-isogenies which have a shorter path. Our hardware design (see Sec.5) is able to compute both one point-doubling and one 4-isogeny in one clock cycle for a frequency set by the longest critical path: the frequency in any hardware platform is defined as the inverse of the longest critical path in the design. Replacing 4-isogenies with 2-isogenies in our design will need twice the latency to compute one 4-isogeny (since, one 4-isogeny = two 2-isogenies). We will also not get any frequency increase, as the critical path will anyway be set by the point-doubling operation. This makes 4-isogenies twice as fast as 2-isogenies, hence it is a must-have feature in our VDF implementation. 8-isogenies are never considered for computation as the cost of computing a larger base degree isogenies scales very badly. This conclusion is supported by both [EKA22, CLN16], albeit with another argument (squaring has one less field multiplication than multiplication in $\mathbb{F}_{p^2}$) as both target a different application: post-quantum cryptography.

4.2 Choice of computational strategy

In Sec. 2, we presented three different strategies to compute an l^k-isogeny. In this section, we present the cost of these strategies in terms of the number of DBL, 4-iso-c and 4-iso-e

Table 2: Cost comparison between 2-iso and 4-iso from [EKA22]

Operation	**Add+Sub**	**Multiplication**	**Squaring**
Point doubling	4	4	2
Compute 2-iso	1	0	2
Evaluation 2-iso	6	4	0
Point Quadrupling	8	8	4
Compute 4-iso	5	0	4
Evaluation 4-iso	6	6	2

Table 3: Cost of the different strategies for a 4^k-isogeny

Strategy	**# of DBL**	**# of 4-iso computation**	**# of 4-iso evaluation**
Basic	$(k-1)\cdot k$	k	$k-1$
Full Eval.	$2\cdot(k-1)$	k	$(k-1)\cdot k/2$
Optimized	$\approx\lfloor 1.4\cdot k\cdot \log(k)\rfloor$	k	$\approx\lfloor 1.3\cdot(k-2)\cdot\log(k-2)\rfloor$

operations for $l = 4$. The basic strategy costs $2\cdot(k-1)+2\cdot(k-2)+\cdots+2\cdot 2 = k\cdot(k-1)$ DBL and k 4-iso-c and $(k-1)$ 4-iso-e computations. So, the complexity is $\mathcal{O}(k^2)$ for DBL and $\mathcal{O}(k)$ for 4-iso-e. The Full Evaluation strategy costs $2(k-1)$ DBL, k 4-iso-c and $(k-1)k/2$ 4-iso-e. Thus, the complexity is $\mathcal{O}(k)$ for DBL and $\mathcal{O}(k^2)$ for 4-iso-e. This method also has a parallelization advantage: all the 4-isogeny evaluations (for the same 4-isogeny) can be computed simultaneously (see Sec. 2.3.1). The Optimized strategy costs $\approx\lfloor 1.4\cdot k\cdot\log(k)\rfloor$ DBL, k 4-iso-c and $\approx\lfloor 1.3\cdot(k-2)\cdot\log(k-2)\rfloor$ 4-iso-e. The complexity is $\mathcal{O}(k\cdot\log k)$ for DBL and 4-iso-e.

Table 3 summarises the isogeny strategies and their costs. It clearly shows that using the Optimized strategy has the least amount of computation, thus the lowest latency. However, to compute the fastest possible isogeny implementation for our VDF, we must consider a hardware platform's ability to compute operations in parallel. With multiple isogeny evaluation modules, we can compute point evaluation in parallel, which in some strategies (the second and third) decreases the latency. Table 4 summarises the latency of all three presented strategies depending on the number of isogeny evaluation modules instantiated. The optimized strategy is still the fastest even with multiple evaluation modules, except with $k-1$ modules, where the Full Evaluation strategy beats the optimized one. This means that there is a threshold on the number of point evaluation modules instantiated (that depend on k) after which the Full Evaluation strategy becomes the fastest. That threshold is not always available: if k is really large, it would be impractical to instantiate k 4-iso-e modules. To conclude, adding more 4-iso-e modules in parallel will decrease the latency of the 4^k-isogeny. Past a certain number of them, the full Evaluation strategy will be faster than the Optimized one, only if k is small enough such that all of them can be instantiated. In all other cases, the Optimized strategy is the best choice.

This conclusion is validated by previous works on improving isogeny strategy [CVOJRH20, CFGR22]. This analysis is not novel as the full Evaluation strategy is considered one of the basic isogeny computation strategies. It is never used in practice due to high area consumption. Nevertheless, in the context of an isogeny VDF we need this parallel isogeny evaluation to obtain the smallest latency for the evaluation step.

For our design, we present two cases one with only one isogeny evaluation core, where we have selected an optimized strategy. The second case features $k-1$ evaluation modules, where we use the Full Evaluation strategy.

4.3 CS representation for a fast design

The solution to challenge 3 (see Sec. 3) lies in the use of a carry-save representation for integers as working with large parameters is made easy with CS representation, see Sec. 2.5. In challenge 4, we presented two issues with using CS representation in isogeny cryptography: modular reduction and sign checking. We define m as the bit-size of our prime p.

Table 4: Latency (in cc) of different strategies for a 4^k-isogeny with different parallelization levels of *4-iso-e* module

Strategy	**1× *4-iso-e***	**2× *4-iso-e***	**k× *4-iso-e***
Basic	$(k-1)\cdot(k+2)$	$(k-1)\cdot(k+2)$	$(k-1)\cdot(k+2)$
Full Eval. even	$(k-1)(k+6)/2$	$3\cdot(k-1)+k(k+2)/4$	$4\cdot(k-2)$
Full Eval. odd	$(k-1)(k+6)/2$	$3\cdot(k-1)+(k+1)^2/4$	$4\cdot(k-2)$
Optimized	$\approx\lfloor 2.53\cdot k\cdot\log(k)\rfloor$	$\approx\lfloor 1.95\cdot k\cdot\log(k)\rfloor$	$\approx\lfloor 1.47\cdot k\cdot\log(k)\rfloor$

For modular reduction, we use two different algorithms. The first one is an adaptation of the Montgomery algorithm for CS representation proposed by [MÖS22]. This algorithm takes a $(2m+2)$-bit integer a in CS form and returns an $(m+1)$-bit integer b in CS form, where $b \equiv a \cdot R^{-1} \mod p$, $b < 2p$, $R = 2^{m+3}$. This algorithm uses $m \cdot (3m+7)$ logical-AND gates combined with three adder trees. This algorithm is very efficient in reducing the output following a multiplication or a squaring but has two shortcomings for our VDF implementation: the first is that it is highly inefficient to use to reduce after an addition or subtraction. The second one is that the output is one bit longer than the input: p is m-bit long, so we want the output of our reduction to be of the same size and not $(m+1)$-bit as discussed in Sec. 4.3.2. This other algorithm will be primarily used the modular reduction following an addition and a subtraction.

4.3.1 The sign issue in CS representation

To perform modular operations in CS representation, we need to perform reduction modulo p. In normal integer representation, the addition or subtraction of a and b, the modular reduction becomes an inequality test $(a+b>p)$ or $(a-b<0)$ followed by a conditional addition or subtraction of p, to put the result in the range $[0, p-1]$. While addition (or subtraction) is very easy in CS, testing the two aforementioned inequalities is impossible without implementing a large degree adder. To test $a+b>p$, we compute $a+b-p$ and check for an overflow (i.e., if the $m+1$ th bit of the output is 1 or 0). To correctly do this, we need to add the carry and the save of $d = a+b-p$, which will result in using a large-sized adder. We have to combine the two "shares", as it is not possible to guarantee the presence of an overflown just by looking at the two shares: as an example, let us consider $p = 61$ prime, and two integers $a = 40$ and $b = 24$ with CS form $a : a_0 = 32(0b00100000), a_1 = 8(0b0001000)$ and $b : b_0 = 16(0b0010000), b_1 = 8(0b0001000)$. When performing a modular addition in normal representation, we compute $d = a+b-p = 40+24-61 = 3$. We change the subtraction of p by an addition by its two's complement $-p = \bar{p}+1$: in our example, $-p = 61$ XOR $127+1 = 67$. So $d = a+b-p = 40+24+67 = 131 = 3 \mod 64$. In CS, $e = a+b-p = 32(0b00100000)+8(0b0001000)+16(0b0010000)+8(0b0001000)+67(0b1000011)$ will represented by e_0 and e_1 with $e = e_0+e_1$, $e_0 = 56(0b0111000)$ and $e_1 = 75(0b1001011)$. We still have $e_0+e_1 = 131 = 3 \mod 64$. We then need to select the correct output. This is done easily in normal representation by checking the $m+1$-bit of e, with $e\,[m+1] = 1$ meaning an overflow, so the correct output is $a+b$. Instead, if $e\,[m+1] = 0$, then there is no overflow and the correct output is $a+b-p$. Here $a+b = 64 > p = 61$, so we should choose $a+b-p$ as our output (and not $a+b$). How does one check this in CS form without adding the carry and the save together?

The answer is we cannot, as the above example shows. Checking $m+1$ bits of an integer in CS representation is not enough: the sign of the integer cannot be determined by just checking the MSb. As carry propagation from the lower bits can change the result of our test as we see in our example: $e = 131 = 56(0b0111000)+75(0b1001011)$. The fourth bit creates a carry that will propagate until it reaches the MSb and will change it from 1 to 0, making this integer positive. Hence, correctly guessing the sign requires adding the carry and the save together, which will essentially defeat the purpose of using CS representation.

4.3.2 CS modular reduction for addition and subtraction

In this section, we present a new approach to perform a modular reduction in CS representation, see Alg. 3. Let $i \in \mathbf{N}$, this approach takes a $(m+i)$-bit integer in CS form and reduces it modulo p to a m-bit integer in CS representation. First, we take the $i+1$ most significant bits of our inputs and add them together with a $(i+1)$-bit ripple-carry adder. The $(i+1)$-bit output of the previous adder: M then goes into a lookup table that stores $M \cdot 2^{m-1} \mod p$ for $M \in [0 : 2^{i+1}-1]$. In the last step, we add $M \cdot 2^{m-1}$ and the $(m-1)$ remaining bits from our input together via a carry-save adder. Thanks to this, we can guarantee that the output will be m-bit long. In Fig 5 two of the three inputs are $m-1$ bits, meaning that in the CSA, the operations on the m-bit will always be the addition of three bits, with two of them set to 0. A full adder has two outputs: the carry and the save. The save bit will be set by the m-bit of the third input $(M \cdot 2^{m-1})$, while the carry bit is always set at 0. This ensures that our outputs are both m-bit long. Fig. 6 shows the architecture diagram of our new reduction. Using the same example as Sec.4.3.1, we have $e = e_0 + e_1 = 56(0b0111000) + 75(0b1001011)$ that we want to reduce mod $p = 61$ to 6-bit CS form. First we generate $M = 1(0b01) + 2(0b10) = 3$, and $S = 3 \cdot 2^5 \mod 61 = 6$. We then add in a CSA: $24(0b11000) + 11(0b1011) + 35(0b100011) = 22(0b10110) + 48(0b110000) = 70 = 9 \mod 61$.

Algorithm 3 Reduction Algorithm in CS form

Input: a in CS form $a = a_0 + a_1$, where a_0 and a_1 are $m+i$-bit integers. i is a small integer. p is an m-bit prime.
Output: b in CS form: $b = b_0 + b_1 \equiv a \mod p$ with b_0, b_1 m-bit long integers

```
1: $M \leftarrow a_0[m+i-1 : m-1] + a_1[m+i-1 : m-1]$      ▷ Using an $i$-bit adder
2: $S \leftarrow (M \cdot 2^{m-1}) \mod p$                  ▷ Using an LUT table
3: $b_0, b_1 \leftarrow a_0[m-1 : 0] + a_1[m-1 : 0] + S$    ▷ Using a CSA
4: return $b_0, b_1$
```

Our approach can perform a reduction $\pmod p$ as well as a reduction in the bit size of the two carry-save shares simultaneously. This is very useful in the following two cases: addition or subtraction, as we are dealing with $m+1$-bit $(i=2)$ integers in CS form. The lookup table will be small: $2^{i+1} - 1 = 2^3 - 1 = 7$ possible values. The second advantage is that the adder for M will be small too, meaning a very small increase in the critical path. In the case of $i = 2$, a 2-bit ripple-carry adder can be done using one full adder and a half adder.

The other feature of this algorithm is that it allows to set the output size at m-bit long after a Montgomery reduction. As p is m-bit long, we would want to keep working with m-bit long integers for the two parts of the CS representation. The output of a square

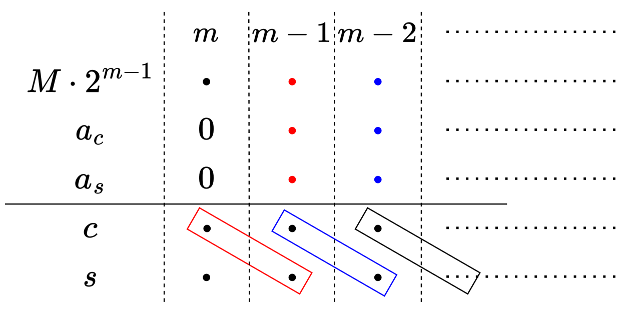


Figure 5: The last CSA addition of our modular reduction

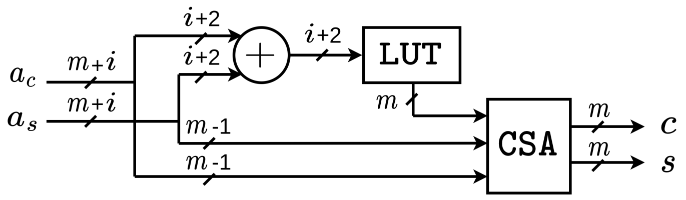


Figure 6: Architecture diagram for a small reduction

(it also applies to multiplication) of an integer a in CS form will be a $2m+2$-bit integer again in CS form: $a^2 = (a_0 + a_1)^2 = a_0^2 + a_1^2 + 2 \cdot a_0 \cdot a_1$, as both a_0 and a_1 are m-bit long, so a_0^2, a_1^2 will be $2m$-bit long and $2 \cdot a_0 \cdot a_1$ will be $2m+1$. The addition of all three is $2m+2$-bit long. The Montgomery algorithm [MÖS22] only reduces a $2m+2$-bit long integer into a $m+1$-bit integer, our small reduction unit can further reduce it to an m-bit integer modulo p. Indeed, our reduction unit is very efficient in hardware design as all three of its components (ripple-carry adder, CSA adder and LUT) are very well suited for hardware platforms.

Modular addition is done by combining a normal CSA addition and this new reduction algorithm to reduce each output back into m bits modulo p. This means we don't use full modular arithmetic in this design, instead, we allow each share of an integer (c and s in CS form) to take values in the range $[0 : 2^m - 1]$. So, $d = d_0 + d_1$ is in range $[0 : 2^{m+1} - 2]$.

5 Hardware architecture

In this section, we present a highly parallelized hardware architecture for implementing the evaluation step of isogeny-based VDFs and provide its critical path analysis. This accelerator is designed to quest for the fastest implementation possible in an attack scenario using massive resources.

5.1 Overall design

For the overall cryptoprocessor, we choose an instruction set architecture (ISA) framework. The main idea behind isogeny VDFs evaluation is the sequential computation of a large-degree isogeny and two isomorphism transformations, with the main operation being the computation of a large-degree isogeny. The fastest way to compute a large degree isogeny, given by [FJP14], is to use an isogeny computing strategy and perform a sequence of elliptic curves arithmetic operations: point doubling, 4-isogeny curve and 4-isogeny evaluation (as we choose $l = 2$). Most of these operations must be done sequentially and thus cannot be parallelised, as discussed in Sec. 4.2. Hereafter we will refer to the operations point doubling as **DBL**, 4-isogeny curve computation as **4-iso-c** and 4-isogeny evaluation as **4-iso-e** respectively. All three operations also consist of a sequence of modular additions, subtractions, multiplications, and squaring on $\mathbb{F}_p$ or $\mathbb{F}_{p^2}$.

The main goal of our work is to provide the fastest implementation of the VDF evaluation step, which in this case is a 4^k-degree isogeny evaluation. A powerful attacker with the fastest evaluation will be able to cheat if the parameters of the VDF are not large enough for an expected delay. A fast accelerator naturally demands that we unroll as many arithmetic operations as possible within the sequential VDF evaluation algorithm. Most hardware implementations of isogeny-based post-quantum cryptography in the literature [SB23, KAK+20] use a serialized core that is only capable of one modular operation at a time. However, we noticed that the higher-order operations (DBL, 4-iso-c and 4-iso-e) require a dozen modular operations. Implying it was possible to unroll those

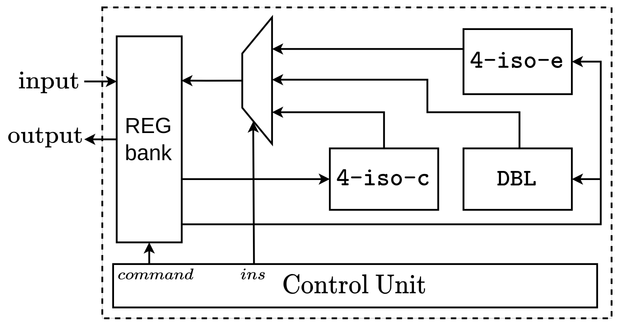


Figure 7: Architecture diagram. 4-iso-c/e stand for isogeny computation/evaluation.

operations: instead of having an arithmetic module that computes modular arithmetic, we have modules that compute higher-order elliptic curves arithmetic (and 4-isogeny). Going one step higher in function hierarchy is not a viable option since the complexity of computing isogenies grows exponentially with their size. So, DBL, 4-iso-c and 4-iso-e are the highest order of functions that we have in our design.

The high-level block diagram of the cryptoprocessor architecture is shown in Fig. 7 and focuses on performing a 4^k degree isogeny in the shortest amount of time. Our cryptoprocessor consists of five modules: one register bank, one control unit, DBL, 4-iso-c and 4-iso-e. We translate the large-degree isogeny computation into a sequence of instructions using the three units. The sequence itself will depend on the number of parallel 4-iso-e modules present in the design following Sec. 4.2. The register bank acts as our system memory that stores all inputs and data during the protocol. It consists of registers and multiplexers, we choose these over BRAMs to keep clock cycles as low as possible during memory access. The control unit generates signals during the protocol to control the memory management and the operations selections.

5.2 Design of arithmetic in CS representation

This section covers how we perform modular arithmetic in CS representation and how we designed our modular subtraction. We cannot use multi-bit adders to perform additions in CS representation (see Sec. 2.5), so instead, additions are done using one-bit Full Adders via carry-save adders (CSA). The addition of two large bit numbers a and b in CS (which is represented by four integers a_0, a_1, b_0, b_1 such as $a = a_0 + a_1$ and $b = b_0 + b_1$) is done using two arrays of Full Adders, see Fig. 3. This is very efficient in terms of timing since the critical path of addition consists of only of two Full Adders. Accumulations are done in CS representation by a large adder tree circuit called Wallace tree [Wal64], or its more compact variant, the Dadda tree [Dad65]. The Dadda tree minimises the number of operands needed to reduce an adder tree but has the same latency as the Wallace tree. The multiplication, in CS representation, is split into two phases. First, we compute the partial products using m^2 logical-AND gates into a large adder tree. As our inputs are in CS form, we need to multiply all the parts together, leading to four different adder trees: $c = a \cdot b = a_0 \cdot b_0 + a_0 \cdot b_1 + a_1 \cdot b_0 + a_1 \cdot b_1$. In the second phase, initially, we reduce individual partial products using four Wallace or Dadda adder trees. A CSA tree then combines all the reduced partial products in CS representation. The squaring in CS representation is as in [MÖS22], by using $(m + 1) \cdot (2m + 3)$ logical AND gates.

A modular subtraction is quite complicated to perform in CS representation. We decided to use the classic two's complement method to compute the subtraction since it works well with CS representation. To turn a CS integer into its two's complement we only have to change both the carry and the save. Another advantage is that the addition by one (in the two's complement) is not problematic at all as we do not need any multi-bit adder to compute it. Instead, we can use a simple CSA for it. Let $a, b \in \mathbb{N}$ in CS form, to compute $c = a - b = a_0 + a_1 - b_0 - b_1 = a_0 + a_1 + \bar{b}_0 + 1 + \bar{b}_1 + 1 = a_0 + a_1 + \bar{b}_0 + \bar{b}_1 + 2$. The notations $\bar{b}_0$ and $\bar{b}_1$ represent bit-wise negation of b_0 and b_1. The rest is a simple accumulation done with CSAs of all five integers. The biggest issue with subtraction is the reduction. As mentioned in Sec. 4.3.1, it is not possible to determine with certainty the sign of an integer in carry-save. This is even more problematic in the case of a modular subtraction than addition, due to overflow (when $b > a$), thus we cannot directly apply our new algorithm 3 for the modular subtraction as our new algorithm cannot deal with the overflow that will appear in subtraction. The solution here is to make sure there is no overflow, since we cannot handle them efficiently. We preemptively add $3p$ before the subtraction to avoid dealing with overflows and negative numbers: $3p - b > 0$, we can safely add with a and then compute a partial reduction to have both output (c and s) as m bit integers. Thus, our modular subtraction compute $c = a - b \mod p = a_0 + a_1 + \bar{b}_0 + \bar{b}_1 + 3 \cdot p + 2$, and accumulates all integers with CSAs.

5.3 Elliptic Curves Units

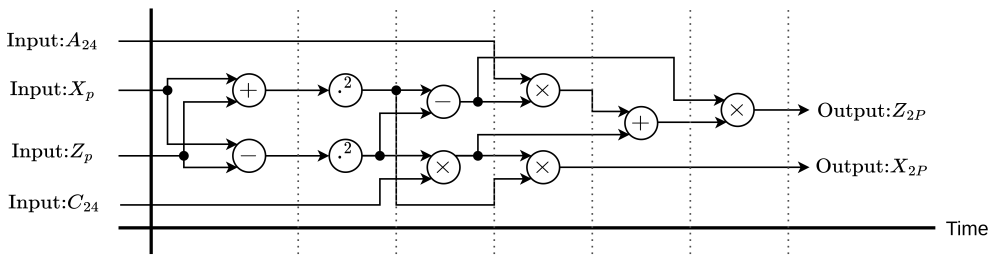


Figure 8: Computation flow for point doubling

This section describes our design decisions for the three modules: **DBL**, **4-iso-c** and **4-iso-e**. Fig. 8 describes the computation flow diagram of the DBL module, which computes the elliptic curve point doubling following the formula given in Sec. 4.1. We have slightly modified the order of the operations in the point-doubling algorithm to allow more parallelism while also reducing the critical path, see Alg. 2. Due to all of these adaptations, we are able to perform this whole computation in one clock cycle.

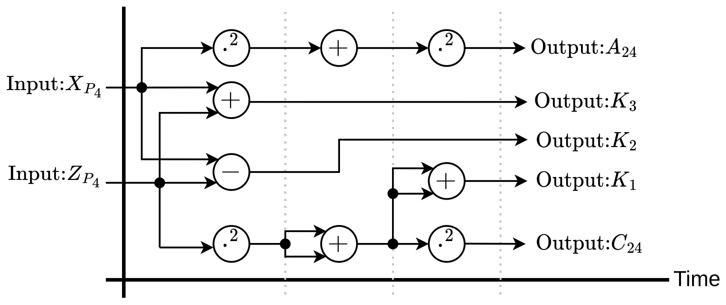


Figure 9: Computation flow for 4-isogeny curve

Fig. 9 presents the computation flow diagram for the 4-iso-c module. Given a point of order 4, it computes the image curve of a degree 4-isogeny. This module parallelizes the steps in the 4-iso-curve algorithm given in [JAC+22] as much as possible to reduce the critical path. We managed to compute this step in one clock cycle.

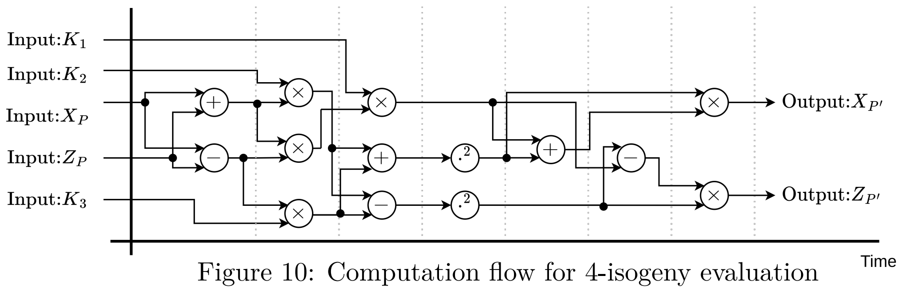


Figure 10: Computation flow for 4-isogeny evaluation

We present in Fig 10 the computation flow diagram of the 4-iso-e module. It computes the image of a point through the 4-isogeny calculated before (4-iso-curve algorithm) following the 4-iso-eval algorithm in [JAC+22]. We are able to compute this algorithm in one cycle.

5.3.1 Critical path analysis

Here, we estimate the delay δ of every arithmetic and elliptic curve function in our design. We will denote τ_{FA} as the delay of a Full Adder, τ_{HA} as the delay of a Half Adder, τ_{AND} as the delay of an AND gate, τ_{XOR} as the delay of an XOR gate and τ_{MUX} as the delay of a multiplexer. We will also note for $s \in \mathbb{N}$, $f(s) \approx \lfloor \frac{\ln s}{\ln 3/2} \rfloor$. We present the estimation of the delay for the arithmetic operations:
Normal addition (ADD): $\delta_{\text{add}} \approx 2 \cdot \tau_{FA}$, see Fig. 3.
Normal subtraction (SUB): $\delta_{\text{sub}} \approx \tau_{XOR} + 3 \cdot \tau_{FA}$. We perform subtraction using the 2's complements method. Which adds an extra XOR operation before the addition. We need three CSA in succession to add six integers to add together, leading to an extra delay of three Full Adders.
Normal Multiplication (MUL): $\delta_{\text{mul}} \approx \tau_{AND} + (f(m) + 4) \cdot \tau_{FA}$. The delay is one AND-gate to compute the partial products. To reduce these partial products into CS form, we use a large adder tree that has a delay of $f(m)$ Full Adders and four extra Full Adders.
Normal squaring (SQR): $\delta_{\text{sqr}} \approx \tau_{AND} + (f(m) + 2) \cdot \tau_{FA}$, from [MÖS22].
New Reduction of *i*-bit: $\delta_{\text{nre}} \approx (i + 1) \cdot \tau_{FA} + \tau_{HA} + \tau_{MUX}$. The initial adder used to compute M, increases the critical path i FA and one HA (i-bit adders). The next step is the LUT table, which is equivalent to a multiplexer in terms of delay.
Montgomery Reduction: $\delta_{\text{mre}} \approx 3 \cdot \tau_{AND} + (f(m+3) + f(m) + 7) \cdot \tau_{FA} + \tau_{XOR} + \tau_{HA} + \tau_{MUX}$. From the Montgomery algorithm in [MÖS22], we have calculated the delay for it.

We will now present our estimations of the delay of our three hardware modules:
Point doubling: $\delta_{\text{DBL}} \approx 12 \cdot \tau_{AND} + 5 \cdot \tau_{XOR} + (3 \cdot (f(m+3) + 2 \cdot f(m)) + 50) \cdot \tau_{FA} + 6 \cdot \tau_{HA} + 6 \cdot \tau_{MUX}$. From the Alg. 2 and Fig. 8, the critical path in that module is the following instruction: SUB $\rightarrow$ SQR $\rightarrow$ SUB $\rightarrow$ MUL $\rightarrow$ ADD $\rightarrow$ MUL. So two multiplication, one squaring, one addition, two subtractions, three reductions and three Montgomery Reduction.
4-iso-c: $\delta_{\text{4-iso-c}} \approx 8 \cdot \tau_{AND} + 2 \cdot (2 \cdot f(m) + f(m+3) + 11) \cdot \tau_{FA} + 2 \cdot \tau_{XOR} + 3 \cdot \tau_{HA} + 3 \cdot \tau_{MUX}$. The critical path, given in Fig. 9, is one modular addition and two modular squaring.
4-iso-e: $\delta_{\text{4-ido-e}} \approx 12 \cdot \tau_{AND} + 6 \cdot \tau_{XOR} + (3 \cdot (f(m+3) + 2 \cdot f(m)) + 52) \cdot \tau_{FA} + 6 \cdot \tau_{HA} + 6 \cdot \tau_{MUX}$. The critical path, given in Fig. 9, is 3 modular subtractions, 2 multiplications, 1 squaring and 3 Montgomery reductions.

Table 5: Critical path delay of the different modules.

Module	without FA	$m = 89$ (FA)	$m = 1506$ (FA)
DBL	$12 \cdot \tau_{AND} + 5 \cdot \tau_{XOR} + 6 \cdot \tau_{HA} + 6 \cdot \tau_{MUX}$	$102 \cdot \tau_{FA}$	$212 \cdot \tau_{FA}$
4-iso-c	$8 \cdot \tau_{AND} + 2 \cdot \tau_{XOR} + 3 \cdot \tau_{HA} + 3 \cdot \tau_{MUX}$	$82 \cdot \tau_{FA}$	$130 \cdot \tau_{FA}$
4-iso-e	$12 \cdot \tau_{AND} + 6 \cdot \tau_{XOR} + 6 \cdot \tau_{HA} + 6 \cdot \tau_{MUX}$	$105 \cdot \tau_{FA}$	$214 \cdot \tau_{FA}$

6 Results

In this section, we provide the implementation results of the proposed design and present an analysis of the results. The proposed arithmetic units are coded using Verilog RTL and they are fully parameterized, meaning that the bit width of the datapath can be set before the implementation. All units are implemented with a 28nm ASIC library using the Cadence Genus tool. We evaluate our design for several bit widths ($m = \log_2(p)$) and this provides us a general trend of area, performance, and the critical path depending on the prime p.

Table 5 presents the critical path delay for all the modules presented in Sec. 5 for a field prime, p, of size $m = \{89, 1506\}$ ($m = 89$ is a toy example here). Since we adopt CS representation, a change in the bit size of p only affects the CS adder tree depth. Thus, as shown in Table 5, increasing the bit size of p only adds full adders to the critical path of our design involving CS adder trees, and the critical path of the remaining part of the design is not affected by the bit width of p. Fig. 11a shows a logarithmic relationship between the number of full adders (FA) in the critical path and the bit size of the prime.

Table 6 shows the critical path delay and area of different bit sizes (m) reported by the Cadence Genus tool for every arithmetic unit that is used as a sub-module by three main modules, DBL, 4-iso-c and 4-iso-e. As we increase the bit size, there was a significant increase in the synthesis time of the design. We therefore report actual critical path and area results for addition, subtraction, multiplication, squaring, and reduction units up to 511-bit. Then, we use the results for eight different bit sizes ($m = 40$ to $m = 511$) to extrapolate the results for $m = 1506$. The extrapolation curve is shown in Fig. 11b. From section 5.3.1, the critical path of our design is defined by the 4-iso-e unit and it is characterized by: 3 modular subtractions, 2 multiplications, 1 squaring, and 3 Montgomery reductions. Following section 5.3.1 and Table 6, we calculate the critical path of our design as $\approx 3 \cdot 0.4 + (1+3) \cdot 0.8 + 3 \cdot 1.35 = 8.45$ns. Note that the result of our overall design includes delay information of addition, subtraction, multiplication, squaring and reduction units reported by the tool, excluding memory and the cost of routing and connecting different units. Thus, this result presents a lower bound on delay value for ASIC implementation.

Table 7 presents the latency and timing results of our design for different k values. With one evaluation unit, we use the optimized strategy, while with $k-1$ evaluation units, we use the full-evaluation strategy. This highlights the effect of parallel isogeny evaluation units, we gain a speedup of around 60%.

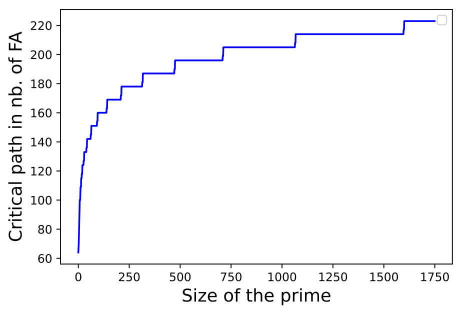
CS vs non-redundant representation: In non-redundant representation, a ripple-carry adder performs a 1506-bit addition using 1506 full adders with critical path $\delta_{1506-\text{add}} = 1506 \cdot \tau_{FA}$. It is possible to use more sophisticated adder architectures like carry-lookahead adder (CLA) or carry-select adder to reduce the critical path; however, their critical path still will be longer compared to redundant CS representation. For example, a 1506-bit adder with 8-bit CLA has a critical path of $\approx \tau_{8-CLA} \cdot \frac{1506}{8}$ while carry-save adder has a critical path of only τ_{FA}. The addition with CS representation is $\approx 1500\times$ faster than ripple-carry adder-based addition. Multiplication implementations with non-redundant representation follow a divide-and-conquer approach [ZZO+23] where small multipliers are used to generate partial products before adding them together. For high performance, the small multiplications can be computed in parallel by using multiple small multipliers and

Table 6: Area cost of the arithmetic modules. Results in bracket are extrapolated.

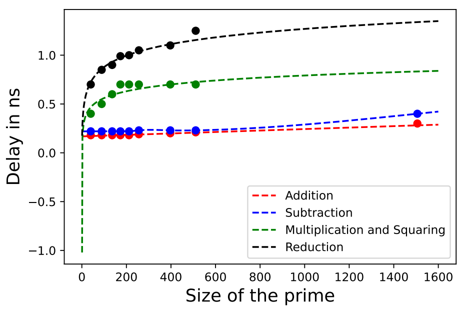
Size (m)	**Critical path (ns)/Area (mm^2)**				
	$\mathbb{F}_p$ Add.	$\mathbb{F}_p$ Sub.	$\mathbb{F}_p$ Sqr.	$\mathbb{F}_p$ Mult.	$\mathbb{F}_p$ Red.
40	0.18/0.001	0.22/0.002	0.40/0.033	0.40/0.070	0.70/0.058
89	0.18/0.004	0.22/0.004	0.50/0.133	0.50/0.320	0.85/0.210
136	0.18/0.007	0.22/0.007	0.60/0.310	0.60/0.702	0.90/0.558
173	0.18/0.008	0.22/0.009	0.70/0.549	0.70/1.148	1.00/0.864
212	0.18/0.010	0.22/0.011	0.70/0.822	0.70/1.453	1.04/1.387
256	0.19/0.012	0.23/0.011	0.70/1.258	0.70/2.184	1.06/1.909
397	0.20/0.014	0.23/0.017	0.70/2.938	0.70/6.682	1.25/3.902
511	0.21/0.020	0.23/0.023	0.70/5.271	0.70/11.80	1.25/6.742
1506	0.30/0.034	0.40/0.093	(0.80/49)	(0.80/85)	(1.35/52)

Table 7: Timing results for isogeny VDF. Clock frequency is 118 MHz.

k	1× Eval. unit		$(k-1)\times$ Eval. unit	
	Latency (in cc)	**Latency (in ms)**	**Latency (in cc)**	**Latency (in ms)**
100	1316	0.01119	836	0.00710
1000	19861	0.16882	11874	0.10093
10000	262855	2.23427	149224	1.26840



(a) Critical path (in terms of number of FAs) vs. bit size of prime



(b) Critical path (in terms of delay in ns) vs. bit size of prime

the additions of partial products can be performed using CSA and one 3012-bit addition. For an 8-bit small multiplier (the choice of 8-bit is from [ZZO+23]), the critical path of a 1506-bit multiplier is $\delta_{1506-\text{mul}} \approx \tau_{\text{8-mul}} + (\log_{1.5}(1506/16)) \cdot \tau_{FA} + \tau_{\text{3012-add}}$. For the multiplication, the delay of the non-redundant 8-bit multiplier is hard to estimate due to different possible design approaches; however, it will always be longer than the delay of partial product multiplier in CS form, one AND-gate [MÖS22]. The depth of the CSA adder tree (in the reduction of the partial products) is lower in non-redundant representation compared to the CS form; thus, it has a lower critical path for adder tree implementation. However, the large integer (3012-bit) addition at the end of the non-redundant representation negates any advantages it had, making CS form significantly faster. The 1506-bit multiplier with CS form can have a speedup of up to ≈ 3000× compared to a non-redundant multiplier (note that the speedup value might be lower depending on how the design approaches).

7 Conclusion

Isogeny-based VDF constructions are becoming popular because of their well-studied cryptographic properties. Apart from conceptual isogeny VDF constructions and their unoptimised software implementations, no efficient implementation suitable enough for setting realistic security parameters exists. This paper is the first work and realises a parallel hardware implementation of isogeny-based VDFs for ASIC platform. Our design is made with the highest level of parallelism possible within the sequential constraints of a VDF evaluation. It aims at accelerating large-degree isogeny computations, which is the core operation of isogeny VDF evaluation. Our design performs full elliptic curves arithmetic operations in one clock cycle (a task that takes a processor dozens of instructions) at a decent clock frequency. In a 28nm technology, our design achieves 118 MHz clock frequency.

The VDF scheme described in [CSRHT22] uses another type of evaluation step where it only computes the co-domain curves and not the isogeny walk. However, we can still apply our work on [CSRHT22] VDF by adding a module that computes the j-invariant for elliptic curves. Our work can help to answer the open question that remains from an attacker's perceptive: *does using co-domain curves instead of isogeny walks provide an evaluation time advantage?* This serves as an interesting future research direction to explore. We hope that our work can be used in standardising isogeny VDFs for real-world applications.

Acknowledgement

This work was supported in part by the State Government of Styria, Austria – Department Zukunftsfonds Steiermark. We are thankful to Samuel Pagliarini for helpful discussions. We also would like to thank the anonymous reviewers for their useful suggestions and comments.

References

[BBBF18] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018*, pages 757–788, Cham, 2018. Springer International Publishing.

[BDF21] Jeffrey Burdges and Luca De Feo. Delay encryption. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021*, pages 302–326, Cham, 2021. Springer International Publishing.

[CD22] Wouter Castryck and Thomas Decru. An efficient key recovery attack on sidh (preliminary version). *Cryptology ePrint Archive*, pages Paper–2022, 2022.

[CFGR22] Hao Cheng, Georgios Fotiadis, Johann Großschädl, and Peter Y. A. Ryan. Highly vectorized SIKE for AVX-512. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(2):41–68, 2022.

[CLG09] Denis Xavier Charles, Kristin E. Lauter, and Eyal Z. Goren. Cryptographic hash functions from expander graphs. *J. Cryptol.*, 22(1):93–113, 2009.

[CLN16] Craig Costello, Patrick Longa, and Michael Naehrig. Efficient algorithms for supersingular isogeny diffie-hellman. Cryptology ePrint Archive, Paper 2016/413, 2016. https://eprint.iacr.org/2016/413.

[CSRHT22] Jorge Chavez-Saab, Francisco Rodríguez-Henríquez, and Mehdi Tibouchi. Verifiable isogeny walks: Towards an isogeny-based postquantum vdf. In Riham AlTawy and Andreas Hülsing, editors, *Selected Areas in Cryptography*, pages 441–460, Cham, 2022. Springer International Publishing.

[CVOJRH20] Daniel Cervantes-Vázquez, Eduardo Ochoa-Jiménez, and Francisco Rodríguez-Henríquez. Parallel strategies for sidh: Towards computing sidh twice as fast. Cryptology ePrint Archive, Paper 2020/383, 2020. https://eprint.iacr.org/2020/383.

[Dad65] L. Dadda. Some schemes for parallel multipliers. *Alta Frequenza*, 34:349–356, 1965.

[DGMV20] Nico Döttling, Sanjam Garg, Giulio Malavolta, and Prashant Nalini Vasudevan. Tight verifiable delay functions. In Clemente Galdi and Vladimir Kolesnikov, editors, *Security and Cryptography for Networks - 12th International Conference, SCN 2020, Amalfi, Italy, September 14-16, 2020, Proceedings*, volume 12238 of *Lecture Notes in Computer Science*, pages 65–84. Springer, 2020.

[DN93] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In Ernest F. Brickell, editor, *Advances in Cryptology — CRYPTO' 92*, pages 139–147, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.

[EKA22] Rami Elkhatib, Brian Koziel, and Reza Azarderakhsh. Faster isogenies for post-quantum cryptography: SIKE. In Steven D. Galbraith, editor, *Topics in Cryptology - CT-RSA 2022 - Cryptographers' Track at the RSA Conference 2022, Virtual Event, March 1-2, 2022, Proceedings*, volume 13161 of *Lecture Notes in Computer Science*, pages 49–72. Springer, 2022.

[FJP14] Luca De Feo, David Jao, and Jérôme Plût. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. *J. Math. Cryptol.*, 8(3):209–247, 2014.

[FMPS19] Luca De Feo, Simon Masson, Christophe Petit, and Antonio Sanso. Verifiable delay functions from supersingular isogenies and pairings. *IACR Cryptol. ePrint Arch.*, page 166, 2019.

[JAC$^+$22] David Jao, Reza Azarderakhsh, Matthew Campagna, Craig Costello, Luca De Feo, Basil Hess, Aaron Hutchinson, Amir Jalali, Koray Karabina, Brian Koziel, Brian LaMacchia, Patrick Long, Michael Naehrig, Geovandro Pereira, Joost Renes, Vladimir Soukharev, and David Urbanik. Sidh-spec, 2022.

[KAK$^+$20] Brian Koziel, A-Bon Ackie, Rami El Khatib, Reza Azarderakhsh, and Mehran Mozaffari Kermani. Sike'd up: Fast hardware architectures for supersingular isogeny key encapsulation. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 67(12):4842–4854, 2020.

[KH98] Q. K. Kop and Ching-Yu Hung. Fast algorithm for modular reduction. 1998.

[KYK$^+$18] Suhri Kim, Kisoon Yoon, Jihoon Kwon, Seokhie Hong, and Young-Ho Park. Efficient isogeny computations on twisted edwards curves. *Secur. Commun. Networks*, 2018:5747642:1–5747642:11, 2018.

[KYK$^+$20] Suhri Kim, Kisoon Yoon, Jihoon Kwon, Young-Ho Park, and Seokhie Hong. New hybrid method for isogeny-based cryptosystems using edwards curves. *IEEE Trans. Inf. Theory*, 66(3):1934–1943, 2020.

[LW17] Arjen K. Lenstra and Benjamin Wesolowski. Trustworthy public randomness with sloth, unicorn, and trx. *Int. J. Appl. Cryptogr.*, 3(4):330–343, 2017.

[MMM03] C. Mclvor, M. McLoone, and J.V. McCanny. Fast montgomery modular multiplication and rsa cryptographic processor architectures. In *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003*, volume 1, pages 379–384 Vol.1, 2003.

[MMP$^+$23] Luciano Maino, Chloe Martindale, Lorenz Panny, Giacomo Pope, and Benjamin Wesolowski. A direct key recovery attack on sidh. In *Advances in Cryptology–EUROCRYPT 2023: 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part V*, pages 448–471. Springer, 2023.

[MÖS22] Ahmet Can Mert, Erdinç Öztürk, and Erkay Savas. Low-latency ASIC algorithms of modular squaring of large integers for VDF evaluation. *IEEE Trans. Computers*, 71(1):107–120, 2022.

[MS16] Dustin Moody and Daniel Shumow. Analogues of vélu's formulas for isogenies on alternate models of elliptic curves. *Math. Comput.*, 85(300):1929–1951, 2016.

[Par00] Behrooz Parhami. *Computer arithmetic - algorithms and hardware designs.* Oxford University Press, 2000.

[Pie18] Krzysztof Pietrzak. Simple Verifiable Delay Functions. In Avrim Blum, editor, *10th Innovations in Theoretical Computer Science Conference (ITCS 2019)*, volume 124 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 60:1–60:15, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[Pur83] George B. Purdy. A carry-free algorithm for finding the greatest common divisor of two integers. *Computers & Mathematics with Applications*, 9(2):311–316, 1983.

[RM19] Debapriya Basu Roy and Debdeep Mukhopadhyay. High-speed implementation of ECC scalar multiplication in gf(p) for generic montgomery curves. *IEEE Trans. Very Large Scale Integr. Syst.*, 27(7):1587–1600, 2019.

[Rob23] Damien Robert. Breaking sidh in polynomial time. In *Advances in Cryptology–EUROCRYPT 2023: 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part V*, pages 472–503. Springer, 2023.

[RSW96] R. L. Rivest, A. Shamir, and D. A. Wagner. Time-lock puzzles and timed-release crypto. Technical report, USA, 1996.

[SB23] Guantong Su and Guoqiang Bai. Towards high-performance supersingular isogeny cryptographic hardware accelerator design. *Electronics*, 12(5), 2023.

[SHT22] Kavya Sreedhar, Mark Horowitz, and Christopher Torng. A fast large-integer extended GCD algorithm and hardware design for verifiable delay functions and modular inversion. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(4):163–187, 2022.

[Sil09] J.H. Silverman. *The Arithmetic of Elliptic Curves.* Graduate Texts in Mathematics. Springer New York, 2009.

[SKN08] Koji Shigemoto, Kensuke Kawakami, and Koji Nakano. Accelerating montgomery modulo multiplication for redundant radix-64k number system on the FPGA using dual-port block rams. In Cheng-Zhong Xu and Minyi Guo, editors, *2008 IEEE/IPIP International Conference on Embedded and Ubiquitous Computing (EUC 2008), Shanghai, China, December 17-20, 2008, Volume I*, pages 44–51. IEEE Computer Society, 2008.

[Tat66] J. Tate. Endomorphisms of Abelian Varieties over Finite Fields. *Inventiones Mathematicae*, 2:134, January 1966.

[Wal64] Christopher S. Wallace. A suggestion for a fast multiplier. *IEEE Trans. Electron. Comput.*, 13(1):14–17, 1964.

[Wes19] Benjamin Wesolowski. Efficient verifiable delay functions. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019*, pages 379–407, Cham, 2019. Springer International Publishing.

[ZZO$^+$23] Danyang Zhu, Rongrong Zhang, Lun Ou, Jing Tian, and Zhongfeng Wang. Low-latency design and implementation of the squaring in class groups for verifiable delay function using redundant representation. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2023(1):438–462, 2023.