IACR Communications in Cryptology
Vol. 2, No. 1, 31 pages.

Accelerating Isogeny Walks for VDF Evaluation

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

Graz University of Technology, Graz, Austria

Abstract. VDFs are characterized 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 accelerate the isogeny-based VDF proposed by De Feo-Mason-Petit-Sanso in 2019. It is the first work that implements a hardware accelerator for 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 both a survey of elliptic curve arithmetic to arrive at the most cost-effective curve computations and an in-depth cost analysis of the different base degree isogeny and method for the isogeny evaluation. 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 a technology-independent metric to model the delay of isogeny evaluation, which a VDF developer can use to derive secure parameters. ASIC synthesis results in 28nm are used as a baseline to estimate VDF parameters.

Keywords: Verifiable delay functions · Isogeny · Redundant representation · Accelerator

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 authorizing 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.

E-mail: david.jacquemin@student.tugraz.at (David Jacquemin), anisha.mukherjee@tugraz.at (Anisha Mukherjee), ahmet.mert@iaik.tugraz.at (Ahmet Can Mert), sujoy.sinharoy@tugraz.at (Sujoy Sinha Roy)

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 optimization and parallel computations. Thus, a VDF implementation on hardware has different aims and challenges than the implementations of conventional cryptographic primitives.

Motivation: The knowledge of the time required for a VDF under a parameter set is critical for establishing security parameters and ensuring their standardized use in the public domain. Indeed, proprietors of companies or technologies utilizing VDF typically have a good understanding of current technological constraints and their client's computational powers. Hence, they might believe that the evaluation time of the VDF aptly matches the established security standards. However, the computational capabilities of an attacker with massive resources, which could be an organization rather than an individual, possessing significant power as well as resources cannot be underestimated.

This paper examines the perspective of such an attacker with massive resources. In this context, it is crucial to note one fundamental difference between the expectations from a cryptoprocessor design of a cryptographic primitive (for example, encryption or signature schemes) and an attack hardware accelerator. The design of a cryptoprocessor is expected to fulfill the constraints of a given application such as area, energy, time, etc. On the other hand, as noted by the authors of [SHT22], the primary objective in a VDF attack implementation is to achieve the highest possible speed, whereas area or power consumption does not hold much significance as attackers' capabilities cannot be underestimated. Hence, our goal is to design a fast VDF evaluation accelerator to achieve the massive parallelization of computations possible during the VDF evaluation. Lastly, we only implemented the VDF evaluation because it is the only part that is interesting from an attacker's perspective, in a VDF, setup or verification do not need a fast hardware design.

Related work: 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. 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. [DFMPS19, CSRHT22, DMS23] are some of the recent works on isogeny-based VDFs. In this paper, we particularly focus on a new type of VDF constructed using isogenies on supersingular elliptic curves proposed in [DFMPS19]. Hereafter, we refer to this VDF construction as FMPS19. This construction offers only partial resistance to quantum attacks (quantum annoyance) because the verification step employs bilinear pairings. Isogeny VDFs are interesting because they can be constructed by combining already existing cryptographic research on isogenies with respect to efficiency and security [DFMPS19].

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. [BF21] 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 optimizations. Since their basic building blocks are closely related to FMPS19's VDF primitive, these optimizations, 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 optimized 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 optimizations within the curve arithmetic to improve performance.

We also note that isogeny-based VDF schemes remain completely unaffected by the recent attacks on SIDH/SIKE [CD23, Rob23, MMP+23]. While in the context of SIDH/SIKE, the underlying hard problem is to find the secret ℓ^T isogeny, in isogeny-based VDFs this isogeny is a part of the public setup. Their hardness assumption relies on the sequential property of point evaluation [DFMPS19, Definition 3]. SIDH/SIKE was broken because of their use of auxiliary image points being computed through isogenies that leaked sensitive information.

1.1 Main Contributions

As stated earlier, since the branch of isogeny-based VDFs is fairly recent, no work has been done to establish and verify security parameters using highly parallel implementations. Our work is the first to address this gap by providing an efficient and extremely fast hardware implementation of the ℓ^T-isogeny evaluation. Note that, hardware implementations of isogeny walks exist in the context of post-quantum cryptography (PQC) [SB23]. However, an isogeny VDF implementation would differ greatly from such a PQC implementation due to the vast difference in their respective parameter sizes (1506 vs 434 bits for SIKE [JAC+22]) as well as their constraint conditions. We identify optimization opportunities targeting various levels of the VDF construction.

We start with optimization techniques on the basic modular arithmetic that would be common to any VDF construction involving isogenies between supersingular elliptic curves. We find that using a redundant representation for integers called the Carry-Save representation (CS) [Par00] and carry-save adders (CSA) for all the isogeny modular arithmetic significantly decreases the latency of the hardware architecture. Using CS representation for isogeny arithmetic also led us to design a new method for modular reduction. This representation eventually helps us estimate the delay of low-level building blocks of our hardware using a technology-independent delay metric such as the number of Full Adders (FA). We discuss the relevance of this metric in the later part of this section when we elaborate on our hardware design.

Next, we move on to conduct a survey of the different forms of elliptic curves to identify the optimal curve that requires the least amount of resource expenditure during the isogeny evaluation. Furthermore, we show that using 4-isogenies as building blocks for evaluating the 2^T-isogeny walk gives the best performance in hardware, when compared to other powers-of-two base degree isogenies. We provide details of how this method compares to other techniques of computing large-degree isogenies such as those explored in [BFLS20, DMPR23]. In fact, computing one 4-isogeny is more efficient in terms of complexity and latency compared to computing a chain of two 2-isogenies as pointed out by [FJP14].

Finally, endowed with the aforementioned low and high-level optimization strategies, we propose two high-performance VDF evaluation architectures: FAVE and FITER. Where FAVE represents the attacker's "favorite" and stands for **F**astest **A**ccelerator for **V**DF **E**valuation. By 'Fastest', we mean that FAVE is an extreme 4^k-isogeny evaluation accelerator architecture with *near-maximum parallel processing*, assuming the availability of massive computational resources to the attacker. However, due to its extensive resource requirements, FAVE's RTL-based hardware design is too complex for our current EDA tools to synthesize using commercially available desktops and servers.

The design "FITER" resembles a homophone for "fighter" and stands for an accelerator that "fits" within current technological constraints to achieve fast VDF evaluation timing. FITER is a less parallelized 4^k-isogeny architecture, utilizing the same building blocks as FAVE, and can be synthesized using present-day EDA tools on a server with 512 GB RAM. The synthesis results of FITER are used to estimate the time and area required for FAVE. FAVE provides a lower bound on VDF evaluation time, which is crucial for setting conservative parameters for the VDF, considering the attacker's potential capabilities. We present the results for both hardware accelerators using a technology-independent delay metric, expressed in terms of the number of FA gates.

Our design is capable of evaluating an ℓ^T (with $\ell = 2$) degree isogeny with a much better throughput ($51,020/281,690$ isogenies/ms for FITER/FAVE) during evaluation compared to FMPS19 [DFMPS19] estimates (0.75 isogenies/ms in software). Hence our choice of curves and strategies, algorithmic optimizations, as well as our tweaks in the architecture design, helps us get significantly closer towards achieving the most parallelized isogeny evaluation possible. Our RTL code is available at `https://github.com/dj33-96/Isogeny-VDF`.

Real world significance of our work: As discussed in Sec. 1, VDFs enforce a minimum, sequentially irreducible computation time for evaluating a mathematical function, say $f(x)$, while allowing fast verification of $f(x) = y$. Hence, these cryptographic primitives are particularly valuable in time-sensitive applications such as blockchains or leader selection in consensus protocols, where it is essential that no participant can compute the function substantially faster than anyone else. In isogeny-based VDFs like [DFMPS19], this computational delay is determined by the degree of the isogeny to be evaluated, reflecting the inherently sequential nature of the isogeny computation in [DFMPS19]. However, existing research does not fully explore how a high-performance hardware accelerator might affect the practical real-world time for VDF evaluation.

Our work addresses this gap by designing and analyzing hardware accelerator architectures and providing a technology-agnostic metric that establishes a realistic lower bound on the computation time. By understanding the fastest possible hardware evaluation, a VDF developer or an organization wanting to use a VDF can select security parameters that guarantee the required delay remains secure against well-funded adversaries. This ensures that the VDF continues to meet its time-lock property, even in the face of advanced or specialized hardware implementations.

We demonstrate the practical significance of our work in a scenario where a VDF developer, say Alice, wants a secure VDF. Alice would first check the latest silicon technology. Let us assume that the latest silicon technology is 3nm ASIC. By utilizing our technology-independent delay metrics (Sec. 4.4-5), Alice chooses conservative VDF parameters based on the delay of an elementary full-adder gate (which is 5 to 10 picoseconds (ps) in 3nm) in the state-of-the-art technology of her time.

1.2 Paper organization

In Sec. 2 we discuss all relevant mathematical concepts necessary for the paper. We give a brief description of some of the different types of VDF available in literature, discuss the basics of elliptic curve arithmetic, then describe in detail the isogeny-based VDF from [DFMPS19] and finally introduce some concepts for carry-save representation, which will serve as the backbone for integer arithmetic. The details of our contributions span Sec. 3 to Sec. 5, where we describe the different optimizations and implementation strategies we adopt for designing our attack hardware accelerator. The different subsections within Sec. 3 are organized in a bottom-up approach where we start at fundamental modular arithmetic (bottom layer-3, Sec. 3.2) and discuss our optimizations for arithmetic in carry-save representation. We then move up a layer (layer-2, Sec. 3.3-3.4) and describe elliptic-curve arithmetic built on layer-3 and also discuss all relevant optimizations, such as choice of elliptic curve representation and base isogeny-degree. Finally, we move up to the topmost layer–the design of the entire attack hardware accelerator (layer-1, Sec. 4) which encompasses all previous layers and provides design decisions based on elliptic-curve and modular arithmetic optimizations discussed in Sec. 3. In this section, we introduce two potential architecture designs of the attack accelerator, FAVE and FITER. Finally, in Sec. 5 we report all relevant implementation results and provide a technology-agnostic analysis of our obtained results. This technology-agnostic analysis aims to explain the real world use-case of our research endeavor. We conclude the paper in Sec. 6.

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 image corresponding to the input s.

Some examples of other forms 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 parallelizable. [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 factorization 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 factorization 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 *succinct non-interactive arguments* or SNARGs. This concept was however used in a slightly different VDF construction by [CSRHT22], which we mention in Sec. 2.3.2.

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 [DFMPS19, 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 $\bar{K}$-isomorphism (except for $char = 2, 3$) 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

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

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 $E_b(\mathbb{F}_{p^2})$ (or over $\mathbb{F}_p$) 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 [DFMPS19]. When the degree of the isogeny, $deg(\phi) = \ell$ is coprime to p, the isogeny is necessarily separable. Modulo composition with an isomorphism, 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 Vélu'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 $\phi \circ \hat{\phi} = [\ell]$ on E_b, and vice versa, where $[\ell]$ denotes the *multiplication-by-*ℓ map.

2.3 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 computing ℓ^T-isogenies involves a series of sequential steps whereas the verification using bilinear pairings is instant. Several constructions of isogeny VDFs have been proposed in recent years. The first one FMPS19 [DFMPS19] was introduced in 2019, followed by another in 2021 [CSRHT22], and a more recent contribution in 2023 [DMS23]. While all these approaches rely on the computation of an isogeny walk, these constructions have difference in the evaluation step and 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 $\mu_N = \{x \in K : x^N = 1\}$.

2.3.1 FMPS19 VDF construction [DFMPS19] 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: $\ell = 2$ 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, $(x, y) \to (u^2 x, u^3 y)$, where $u \in \mathbb{F}_{p^2} \setminus \mathbb{F}_p$ and $\tilde{E}$ is a quadratic twist of E over $\mathbb{F}_{p^2}$. $\tilde{E}$ 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, $Y_1 = v^{-1}(\tilde{E'}[N] \cap \tilde{E'}[\mathbb{F}_p])$ 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 $v^{-1}(\tilde{E}[N] \cap \tilde{E}[\mathbb{F}_p])$. 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 $\hat{\phi}(Q)$. A key point here is that the isogeny is fixed, so all the kernel points are known in advance for the evaluation. So given sufficient memory, this will translate the main computation of the evaluation step from a standard ℓ^T-isogeny walk like in [JAC+22] to a sequence of ℓ-isogeny point evaluations of the input point Q.

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

FMPS19 VDF construction over $\mathbb{F}_{p^2}$: The construction for this VDF follows a similar construction as the one over $\mathbb{F}_p$. Most of the the VDF setup phase is similar to the previous case, with the curve E being define over $\mathbb{F}_{p^2}$. In this construction, the authors also take 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 $E/\mathbb{F}_p$. 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 the property of 'quantum annoyance', as referred to in FMPS19.

2.3.2 Other isogeny VDF constructions

The work by [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 2023, [DMS23] proposed another quantum-resistant but "weak" VDF. Here, the term "weak" refers to the fact that parallelism may give a significant computational advantage during the VDF evaluation step. With enough parallel cores, the computational complexity of the VDF evaluation goes from $\mathcal{O}(poly(T))$ to $\mathcal{O}(T)$. Their construction involves one-dimensional isogenies as well as higher-dimensional ones for Kani's criterion. We give a brief description of their construction in the following part of the paragraph. Let $E/\mathbb{F}_p$ be a supersingular elliptic curve. The setup involves sampling and constructing the two primes ℓ and p. Such that there exist two horizontal ℓ-isogenies ϕ and ϕ' towards two others elliptic curves E_1 and E_1'. The evaluation step consist of evaluating those two horizontal ℓ-isogeny ϕ and ϕ'. The verification step makes use of Kani dimension 2 to evaluate ϕ and ϕ' over a subgroup of E of smooth order for fast verification.

In this paper, we focus on FMPS19 as a case study. Their methods of isogeny computation have been extensively studied in the context of elliptic curve cryptography. Moreover, it is the only VDF construction work which proposes some concrete parameters based on their a proof-of-concept software implementation.

2.4 Carry-save representation

The redundant binary representations (RBR) are numeral systems where integers are represented using more bits than the standard representation. The standard representation represents a positive integer a using the minimal $m = \lceil \log_2 a \rceil$ bits. In contrast, RBRs introduce redundancy by representing an integer using additional bits to gain faster arithmetic in some computational scenarios. Because of the redundancy, a number has more than one representation. 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 addition becomes significantly faster in RBR than in standard representation as the bit size grows [SKN08, MÖS22, SHT22].

One commonly used RBR is the carry-save (CS) representation [Par00]. In the CS

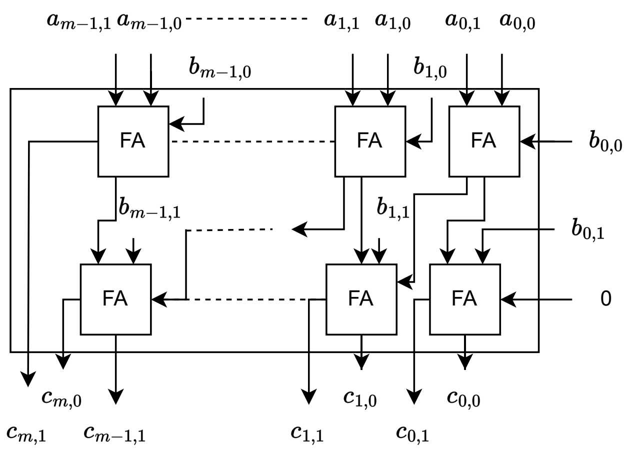

Figure 1: Addition of two numbers in CS representation

representation, an integer a is viewed as the sum of two positive integers:

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

In the CS representation, addition uses a long array of *individual* one-bit full adders (FA) called carry-save adders (CSA). Fig. 1 shows how the addition of two large-bit integers a and b is performed when they are represented in the CS using four integers a_0, a_1, b_0 and b_1 such that $a = a_0 + a_1$ and $b = b_0 + b_1$. This method is very efficient in hardware, as the critical path of addition is only two full adders [RM19]. In contrast, a ripple carry adder with the standard representation incurs a critical path proportional to the bit-length of the integers.

For example, we want to add $a = 12$ and $b = 11$. Let, $a_0 = 10$, $a_1 = 2$, $b_0 = 0$, and $b_1 = 11$ (or any other combinations). Following fig. 1, first a_0, a_1 and b_0 are added using one CSA array, which gives 4 and 8 as outputs. Next, the two outputs 4, 8 and b_1 are added together using another CSA array. The outputs are $c = 16$ and $s = 7$. We never recombine c and s because the combination will introduce a carry propagation chain, thus severely increasing the critical path. A longer critical path means lower clock frequency and slower design.

3 Optimizations

In this section, we first list certain VDF-specific challenges that motivate the optimization strategies adopted in the proposed attack accelerators.

3.1 Challenges in accelerating VDF evaluation

Various VDF constructions [DFMPS19, CSRHT22, DMS23] use distinct evaluation procedures, each presenting its own set of challenges. The VDF evaluation in [CSRHT22] relies solely on modular arithmetic as the isogeny walk is computed on the j-invariants and is determined by the j-invariants of the two previous curves. The primary computational bottleneck in this VDF is calculating a modular square root, and the VDF parameters are set to facilitate this computation as a series of modular multiplications, using Kong's algorithm [KCYL06]. The VDF evaluation in FMPS19 involves evaluating a point through an ℓ^T-isogeny. With sufficient memory, this becomes a sequence of ℓ-isogeny evaluations of the point since the isogeny is fixed and known. Hence, fast evaluation requires fast modular arithmetic and optimal setup (identifying the best curve, formula, and degree). The VDF presented in [DMS23] involves computing two large prime and horizontal isogenies during the evaluation step. The evaluation is mostly similar to [DFMPS19], but with two main differences: the isogeny is neither pre-defined nor smooth.

We observe that in all the three aforementioned VDF constructions [DFMPS19, CSRHT22, DMS23], fast modular arithmetic is critical for speeding up the VDF evaluation. Beyond the modular arithmetic layer, algorithmic and arithmetic optimizations specific to each scheme further accelerate the process. Therefore, in this paper, we focus on parallelizing modular arithmetic. As a case study, we examine the first isogeny-based VDF FMPS19 and explore high-level optimizations.

1. FMPS19 construction suggests a field prime of 1506-bits to achieve 128-bit security [DFMPS19]. Large integer arithmetic is problematic due to carry propagation issues. Hence, efficient design strategies are crucial to achieve a highly parallel implementation with a low latency.

2. Various prior works have used CS form to speed up certain operations or algorithms [Pur83, MMM03, SHT22]. One such optimization was also used in 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. 3.2.1.
   - 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. 3.2.2 and those with modular subtraction in Sec. 4.1.

3. 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 elliptic curve arithmetic operations 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 almost 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. This step can be done very efficiently: switching from a Montgomery curve to an Edwards 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 settle for the best curve in Sec. 3.3 and Sec. 3.4.

4. Most isogeny-based cryptographic primitives in the literature [JAC$^+$22, SB23] use an optimized strategy which is well suited for efficient isogeny computations when

resources are limited. However, an adversary with extensive memory and parallel processing capabilities could adopt a different and much faster approach for evaluation. Therefore, it is essential to identify the most efficient isogeny evaluation strategy that such a powerful adversary could use for rapid VDF evaluation. More information on the different strategies can be found in appendix B.

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

3.2 CS representation for a fast design

The solution to challenge 3 in Sec. 3.1 lies in the use of a carry-save representation for integers as working with large parameters is made easy with CS representation, see Sec. 2.4. In challenge 4, we presented two issues with using CS representation in isogeny-based cryptography: modular reduction and sign checking. We mitigate these issues in the following part of the section. Let, p be an m-bit prime.

For modular reduction, we use two distinct algorithms. The first is adapting the Montgomery algorithm for CS representation, as 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 $b \equiv a \cdot R^{-1} \mod p$, $b < 2p$, $R = 2^{m+3}$. The algorithm utilizes $m \cdot (3m+7)$ logical-AND gates and three adder trees, making it highly efficient for reducing the output after a multiplication or squaring operation. However, the algorithm has one major shortcomings: it is inefficient for reducing the result in a modular addition or subtraction. To address this, we have developed a new alternative algorithm (see Alg. 1), which is primarily used for the modular reduction following an addition and a subtraction.

3.2.1 The sign issue in CS representation

To perform modular operations in the CS representation, we need to reduce the result modulo p. In the standard integer representation, modular reduction after addition or subtraction is performed as an inequality test $(a+b>p)$ or $(a-b<0)$ followed by a conditional subtraction or addition of p. While addition (or subtraction) is very easy in CS form, 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 overflow just by looking at the two uncombined 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 be 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 (Most Significant Bit). 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 change it from 1 to 0, making this integer positive. Hence, correctly guessing the sign requires adding the carry and the save together, defeating the purpose of using CS representation.

3.2.2 CS modular reduction for addition and subtraction

We propose a method for performing modular reduction in CS representation, as outlined in Alg. 1. For $i \in \mathbb{N}$, this approach takes a $(m+i)$-bit integer in CS form and reduces it modulo p to an 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 $(M \cdot 2^{m-1} \pmod p)$ 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. In this way, we can guarantee that the output will be m-bit long. In fig. 2, two of the three inputs are $m-1$ bits such 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 both the outputs are m-bit long. Fig. 3 shows the architecture diagram of the proposed reduction technique. Using the same example as Sec.3.2.1, 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 $S = 3 \cdot 2^5 \mod 61 = 35$. We then add in a CSA: $24(0b11000) + 11(0b1011) + 35(0b100011) = 22(0b10110) + 48(0b110000) = 70 = 9 \mod 61$.

Algorithm 1 Modular reduction 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

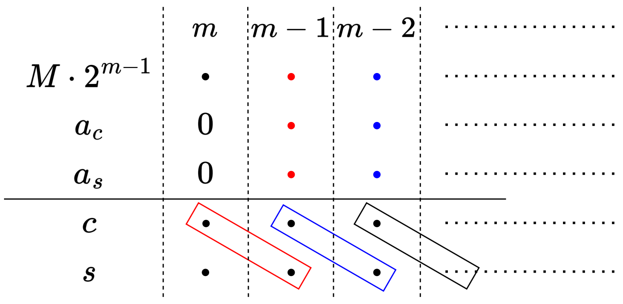

Figure 2: The last CSA addition of our modular reduction

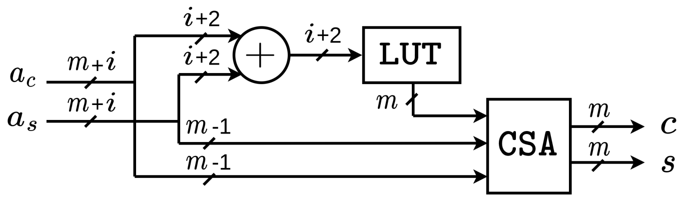

Figure 3: Architecture diagram for a small reduction

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 (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 $[0 : 2^{m+1} - 2]$.

3.3 Choice of curve and strategy

In the following part, we explain the choice of the elliptic curve form and the isogeny computation strategy from an attacker's perspective.

Choice of curve: We first present Table 1 outlining the various choices of elliptic curves and the analysis cost for a 2-isogeny evaluation in terms of MUL, DIV and ADD representing the number of multiplications, divisions and additions respectively. In this work, we assume that one multiplication is equivalent to one squaring. The Table 1 considers three forms of elliptic curves, namely Edwards, Montgomery and Weierstrass. In [Was08], the author describes the formula for isogeny evaluation in the case of a general Weierstrass curve. The figures for Montgomery curves are taken from [JAC+22], while the ones for Edwards are from [KYK+18, MS16] where the authors first use the birational relation between twisted Edwards curves and Montgomery curves, and then apply the isogeny evaluation formula for Montgomery curves. From Table 1, we conclude that Montgomery curves have an advantage over the other curve forms due to the least amount of multiplication and addition required for a 2-isogeny evaluation. Hence, we choose Montgomery curves.

Choice of strategy: Implementations of isogeny walks have employed various computational strategies to improve the efficiency of computing large smooth-degree isogenies. One widely used strategy, known as an "optimized strategy" (see Sec. B), is always used in literature [JAC+22, SB23] for computing smooth-degree isogenies due to its lower

Table 1: Operation count comparison

Curve shape	**Point doubling**		
	MUL	**ADD**	**DIV**
Edwards-XZ, from [KYK+18, MS16]	3	6	1
Montgomery-XZ, from [JAC+22]	3	2	1
Weierstrass, from [Was08]	6	3	1

complexity of $\mathcal{O}(T \cdot \log(T))$ in the case of a 2^T-isogeny, when compared to other strategies. However, in FMPS19 VDF setting, the isogeny is fixed during the setup phase. Fixing the isogeny also fixes the isogeny walk, which determines the kernels of each small ℓ-isogeny step along the walk. As a result, all the kernel points for the isogeny walk are available before the VDF evaluation. These precomputed kernel points can be stored in an accelerator, significantly reducing the amount of computation for isogeny evaluation. With this approach, the computation becomes an iterative application of an ℓ-isogeny evaluation on the input point P, using the stored kernel points. The complexity of this "strategy" for an ℓ^T-isogeny is $\mathcal{O}(T)$, making it much faster than the $\mathcal{O}(T \cdot \log(T))$ complexity of an optimized strategy. This "precomputation strategy" is only applicable in FMPS19's evaluation step or whenever the isogeny is predetermined in the setup phase.

3.4 Choice of isogeny degree

It is well known that a smooth-degree isogeny can be computed as a chain of smaller degree isogenies, which we refer to as 'base degree' isogenies. We choose 4-isogenies as the base degree for FMPS19's evaluation step. Below, we explain the various factors that influenced this choice.

Since FMPS19's evaluation step involves computing a large known 2^T-degree isogeny with respect to a curve point P, our choice of the base isogeny degree is limited only to powers of two. For computing smooth higher-degree isogenies, one common approach in literature [FJP14, EKA22, CH17] is to decompose the large-degree isogeny into multiple small-degree isogenies and evaluate them instead of a direct large-degree isogeny using Vélu's formula [Vél71]. This is because directly computing large degree isogenies (such as 8, 16 or higher) using Vélu's formula [Vél71] tends to get more expensive with increasing isogeny degree, for example, going from 4 to 16 and then to 64-isogenies results in an increase of operations from 10 to 34 and then to 130 multiplications, respectively. Following Vélu's formula 5, there are two steps to compute the isogeny evaluation ϕ of a point $P(x, y)$: first, one must compute all the points in the kernel K of the isogeny, and then the second step is computing the rational function,

$$\phi(x) = x \cdot \prod_{\omega \in \mathrm{K}^*} \frac{x \cdot \omega - 1}{x - \omega}, \text{ from [CH17].} \tag{5}$$

Finding all the kernel points is computationally less demanding for small-degree isogenies such as 2 or 4. For example, in evaluating a 2-isogeny on a generic elliptic curve, we only need to compute one kernel element (using a few point-doubling operations) as the other element is the point at infinity. However, in the case of larger isogenies, computing the kernel becomes significantly costlier as the number of kernel points increases. In FMPS19 isogeny evaluation, the walk is fixed at the setup phase and hence the kernel points can be pre-computed. The next step is evaluating the rational function in Eqn. 5, for which the projective coordinate system ($x = X/Z$) is used to avoid expensive modular divisions. The modular division between the numerator and denominator is not performed directly;

instead, both terms are retained for use in later computations.

$$\phi(x) = \phi(X, Z) = \frac{X \cdot \prod_{\omega \in \mathrm{K}^*} X \cdot \omega - Z}{Z \cdot \prod_{\omega \in \mathrm{K}^*} X - \omega \cdot Z}. \tag{6}$$

The equation above is generic and can evaluate any power-of-two degree isogeny. In the following, we compare the costs of evaluating an ℓ-degree isogeny where $\ell \in \{2^1, 2^2, 2^3, 2^4\}$.

The computation of numerator requires $\ell - 1$ modular multiplications involving the different points ω and an additional $\ell - 1$ modular multiplications to compute the big product, including the multiplication by X, resulting in a total of $2 \cdot (\ell - 1)$ multiplications. A similar approach can be applied for the denominator, which also requires $2 \cdot (\ell - 1)$ modular multiplications. Thus, one ℓ-isogeny evaluation requires $4 \cdot (\ell - 1)$ modular multiplications, assuming the kernel is precomputed.

The number of multiplications can be optimized further by considering the fact that the kernel of an isogeny is a group where each point $P(x, y)$ has its inverse $Q(x, -y)$ in the group, excluding the point at infinity and the point of order two. Thus, for our cost analysis, we need to consider only $(\ell - 2)/2$ distinct points in the kernel. For $\ell = 2$, the cost (number of multiplications) remains unchanged. For $\ell = 2^m$ with $m > 1$, the cost of evaluating the numerator or denominator is $= (\ell - 2)/2 + ((\ell - 2)/2 - 1) + 1 + 1 + 2 = \ell + 1$. Thus, evaluating $\phi(x)$ or ℓ-degree isogeny requires $2 \cdot \ell + 2$ modular multiplications.

From an attacker perspective, the most important metric is not the total number of operations but the minimum latency required to compute the isogeny evaluation. Based on Eqn. 6 and the preceding analysis, the minimum latency can be calculated for each base power-of-two isogeny degree. A key observation is that both the numerator and the denominator can be computed in parallel due to the absence of data dependency and the symmetry of their operations. For $\ell > 2$, the computation begins by evaluating all inner products of the rational function in Eqn. 6 between w_i (w in the equation) and X in parallel (excluding redundant elements in the kernel). The resulting products can then be aggregated using a product tree, followed by a squaring operation. The product between X and $X \cdot w_j - Z$ is computed in parallel, where w_j is the kernel point of order 2. Finally, the result of the squaring operation is multiplied with the resultant of the product, $(X \cdot (X \cdot w_j - Z))$. The total latency for the evaluation is given by $\text{lat} = 1 + \log_2(\ell/2) + 1 + 1 = 3 + \log_2(\ell/2)$. The case of $\ell = 2$ is trivial with $\text{lat} = 2$ multiplications.

Now, we will consider a positive integer s and a large degree 2^s isogeny. Table 2 presents the cost of evaluating this isogeny in number of multiplications (we consider the cost of one squaring to be equal to one multiplication) to calculate a 2^s-isogeny using different ℓ-isogenies.

Table 2: Number of modular multiplications 'MUL' in computing 2^s-isogeny using $\ell \in \{2^1, 2^2, 2^3, 2^4\}$-isogenies

Isogeny degree	$\ell = 2^1$	$\ell = 2^2$	$\ell = 2^3$	$\ell = 2^4$
Number of isogeny evaluations	$s/2$	$s/4$	$s/6$	$s/8$
Latency per evaluation	2	3	5	6
Latency in MUL in total	s	$0.75 \cdot s$	$0.83 \cdot s$	$0.75 \cdot s$

The final choice of 4-isogenies: From Table 2, both $\ell = 4$ and $\ell = 16$ appear to be optimal choices, as they show the lowest overall latency. However, 4-isogenies also provide a significant advantage in terms of area utilization compared to 16-isogenies. As noted previously, the number of operations scales exponentially with the degree of isogeny. This exponential growth makes higher-degree isogeny evaluation absurdly costly in our setting. For this reason, we selected 4-isogenies over 16-isogenies, specifically since the latter requires 34 parallel multiplication cores, compared to only 10 for 4-isogenies. Thus, given that both 4-isogenies and 16-isogenies have the same latency, we opted for 4-isogenies as more practical. Furthermore, isogeny evaluations for degrees above $\ell = 16$ are not considered because, although higher-degree isogenies might offer a theoretical latency advantage, our choice of 4-isogenies represents a realistically optimal balance between efficiency and feasibility, even when considering an attacker with massive resources. Despite using a base degree of 4-isogeny, synthesizing such a large design remains highly challenging with current silicon technology. With a higher base degree isogeny, the area cost would increase exponentially and this exponential increase may lead to multiple issues such as unavailability of enough bandwidth for transferring all the kernel points to the accelerator, which will in-turn cap the possible speedup.

Nevertheless, foreseeing a scenario where futuristic advancements in silicon technology may allow attackers to utilize 64 or even higher base isogenies degrees, we demonstrate our (technology-agnostic) critical path analysis that can be followed by the future attacker to also estimate feasibility of these higher-degree isogenies later, in Sec. 5.

Other methods for evaluating isogenies: Recent works, such as [BFLS20, DMPR23] have proposed more efficient methods for computing large-degree isogenies than Vélu's formula 5, but only on non-smooth degree isogeny of degree q. The work [BFLS20] improves the evaluation complexity of the rational function given by eqn. 5 to $\tilde{\mathcal{O}}(\sqrt{q})$ where the notation $\tilde{\mathcal{O}}$ does not take into account the logarithmic factors in $q > 0$, by applying the baby-step giant-step algorithm. Two key conclusions can be drawn from this work:

- In [BFLS20], the authors compare their square-root Vélu's formula with the original Vélu's formula and conclude that directly using their formula for $q < 100$-isogenies does not provide any speedup. Therefore, the original Vélu's formula remains more efficient for smaller base degrees.

- Precomputing kernel points does not lead to a significant speedup, as the algorithm's complexity is still $\tilde{\mathcal{O}}(\sqrt{q})$.

Additionally, we explore the possibility that, *can the square root Vélu's formula be used to reduce the cost of evaluating powers-of-two base degree isogenies such as* $(2^8 > 100)$ *or higher compared to our aforementioned analysis*?

A direct approach using square-root Vélu [BFLS20] for a large degree 2^T-isogeny results in a complexity of $\tilde{\mathcal{O}}(\sqrt{2^T}) = \tilde{\mathcal{O}}(2^{T/2})$ multiplications, which is significantly more expensive than using 4-isogenies (see Table 2). Another approach would be to split the 2^T-isogeny into steps of q-isogeny with $q = 2^s > 100$, and apply the square root Vélu's [BFLS20] formula to the q-isogenies. The complexity of $q = 2^s$-isogeny is $\tilde{\mathcal{O}}(2^{s/2})$, and we will need T/s evaluation steps to compute a 2^T-isogeny. Therefore, the overall complexity will be in $\tilde{\mathcal{O}}(T/s \cdot 2^{s/2})$ modular multiplications. Let us further refine this cost analysis to compare it with 4-isogenies. We can consider $\tilde{\mathcal{O}}(\sqrt{q}) = K \cdot \sqrt{q} \cdot (\log q)^L$ with $K > 2$ and $L \geq 0$. We apply this refined formula on the cost analysis of a 2^T-isogeny using 2^s base degree isogenies, the cost becomes $K \cdot T/s \cdot 2^{s/2} \cdot (\log 2^s)^L = K/s \cdot 2^{s/2} \cdot (s \cdot \log 2)^L \cdot T$ modular multiplications. This is superior to the cost of using 4-isogenies as base degree isogenies.

Further, the authors of [DMPR23] use higher dimensional isogenies to compute non-smooth degree isogenies. This concept of embedding one dimensional isogeny into higher dimensional isogenies has resulted from the lack of an efficient representation of non-smooth-degree isogenies. However, we do not use higher dimensional isogenies in the case of FMPS19 as smooth one-dimensional isogenies can be efficiently represented as a chain of smaller-degree isogenies.

4 Hardware architecture of accelerator

In this work, we adopt an attacker's perspective to evaluate the 2^T-degree isogeny in the least amount of time given specific fixed parameter sizes (p as a 1506-bit prime) and assuming all the kernel points are precomputed as in FMPS19 [DFMPS19]. A hardware accelerator is designed to achieve near-maximum computational parallelism to reduce latency, assuming the attacker has unlimited resources. However, the large-scale design of our parallel attack accelerator, FAVE, surpasses the synthesizing capabilities of the EDA tools available in our lab. This does not imply that a well-resourced adversary would be unable to realize FAVE. To estimate FAVE's time and area requirements, we use a secondary, scaled-down architecture called FITER, which is synthesizable with current EDA tools. We describe the design strategy for arithmetic operations using carry-save representation and then move on to FAVE and FITER's design descriptions. The FAVE hardware acceleration is for the evaluation of FMPS19's VDF.

4.1 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. 2.4), 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. 1. This is very efficient in terms of timing since the critical path of addition consists of only of two full adders. Accumulations can be done in CS by a large adder tree circuit called Wallace tree [Wal64], or its more compact variant, the Dadda tree [Dad65]. The Dadda tree minimizes 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.

Efficient modular subtraction is more complicated in CS representation. We decided to apply the classic two's complement method in the CS representation to compute the subtraction since those two can be applied at the same time. 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. All the additions here are computed using carry-save adders (CSA).

The main challenge in modular subtraction is managing the reduction. As mentioned in Sec. 3.2.1, CS representation does not allow determining the sign of a result. When a subtraction causes an overflow (i.e., $b > a$), we cannot directly apply the reduction algorithm 1 as it cannot deal with the overflow. The solution here is to make sure there is no overflow. 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 computes $c = a - b \mod p = a_0 + a_1 + \bar{b}_0 + \bar{b}_1 + 3 \cdot p + 2$, and accumulates all integers with CSAs.

4.2 The attack accelerator: FAVE

FAVE's design aims to achieve the maximum parallel processing possible for the VDF evaluation step, which in this case is a 4^k-degree isogeny evaluation. A very powerful attacker with such parallel computation capability backed by immense resources 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 operation for 4-isogeny evaluation (4-iso-e) requires tens of modular additions, subtractions, multiplications, and squaring on $\mathbb{F}_p$ or $\mathbb{F}_{p^2}$. This provides us with the possibility of unrolling these operations. Thus, instead of having an arithmetic module that computes modular arithmetic operations, we have one module that computes higher-order elliptic curve arithmetic corresponding to 4-iso-e with unrolled modular arithmetic. Using carry-save (CS) representation effectively mitigates critical path issues due to unrolling. Going one step higher in the function hierarchy is not a viable option due to the serial nature of the computation: the evaluation step requires us to compute one 4-isogeny evaluation after the other using the output of the previous evaluation and two new kernel points as inputs. Connecting multiple 4-isogeny evaluation processors in a series does not decrease the VDF evaluation time, as any reduction in cycle count is counterbalanced by a corresponding increase in the clock period.

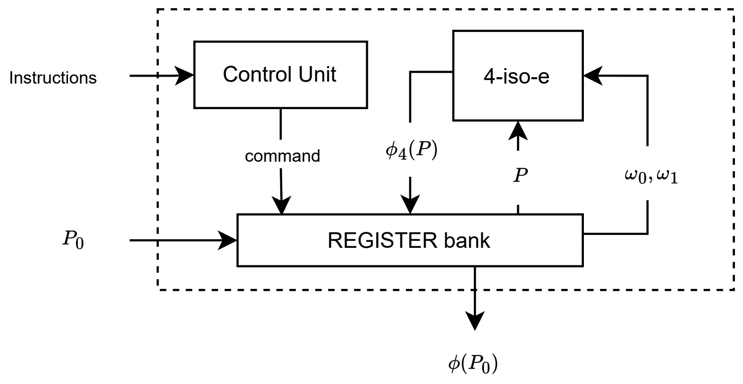

Figure 4: Architecture diagram of FAVE.

FAVE is an instruction set architecture (ISA) equipped with instructions for computing isogeny evaluation, uploading and downloading data. The high-level block diagram of the FAVE cryptoprocessor architecture is shown in fig. 4. It consists of three modules: one register bank which serves as the memory section of the accelerator, one control unit, and 4-iso-e. We translate the large-degree isogeny computation into a sequence of instructions. When an instruction is sent, the control unit translates that instruction into various control signals and multiplexers.

We present in fig. 5 the computation flow diagram of the 4-iso-e module. It computes the image of a point $P(X_P : Z_P)$ through a 4-isogeny, following eqn. 6. As explained in Sec. 3.4, w_1 represent the kernel of order 4, thus is squared to take into account the second kernel point with the same value and w_0 represent the kernel point of order 2. In this module, all modular operands are represented in CS form and computed combinatorially while trying to minimize the critical path. This design enables the module to execute one 4-isogeny evaluation in one cycle, and the critical path is approximately 3 multiplications.

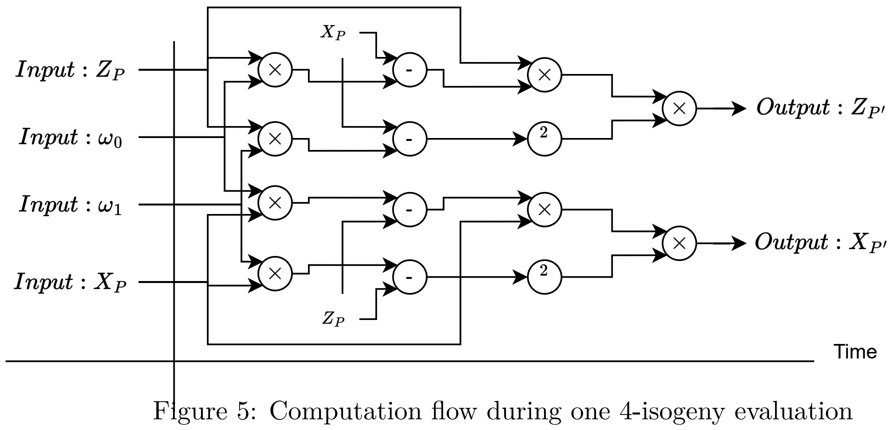

Figure 5: Computation flow during one 4-isogeny evaluation

4.3 Modeling FAVE using FITER

FAVE has the highest possible parallelism for 4-isogeny evaluation but demands a substantial area. Due to the limitations of the commercial EDA tools and the processing capabilities of the server computers in our lab, we were unable to complete the synthesis of the full FAVE design. To estimate FAVE's area and time requirements, we modelled it using a scaled-down version called FITER. The primary distinction between FITER and FAVE is that FITER employs modular arithmetic operations as its fundamental components (fig. 6) instead of computing an entire isogeny evaluation in one cycle like in FAVE (fig. 4). This reduces the area usage of FITER by a factor of ≈ 10. We elaborate on its design in the following paragraph. The modular arithmetic components remain the same in FAVE and FITER.

The main idea behind isogeny VDF evaluation is a long and predefined sequence of modular operations. This evaluation, given by [FJP14], involves following an isogeny computing strategy and performing a sequence of elliptic curves arithmetic operations: 4-isogeny evaluation (as we choose $\ell = 4$). All of these elliptic curve operations consist of a sequence of modular additions, subtractions, multiplications, and squaring on $\mathbb{F}_p$ or $\mathbb{F}_{p^2}$ depending on the setup. This configuration is very favorable for an instruction set architecture (ISA) framework, which we have selected for the FITER design. We have presented our architecture in fig. 6 that is split into three parts. The first part is the memory section (REG bank) consisting of registers and multiplexers that store all inputs and data during the protocol. We choose registers over SRAMs to keep clock cycles as low as possible during memory access. The second section consists of all the modular arithmetic units (adder, multiplier, subtraction and modular reduction). All of these units use CS representation as described in Sec. 4.1. The last part is the control unit that generates signals during the protocol to control the memory management and the operations selections from the instruction it has received. The data selection is performed at the end (right before the data is stored in the REG bank) using a multiplexer. This processor uses a serialized approach for the arithmetic operations, as it executes only one modular operation per clock cycle, thus taking multiple clock cycles to perform higher elliptic curves arithmetic operations. In this design, an instruction refers to a modular operation (addition, subtraction or multiplication) and is performed in one clock cycle.

4.4 Critical path analysis

Here, we estimate the delay δ of every arithmetic and elliptic curve function in our design, we aim to present our cost analysis results in technology-independent delay metric. This metric allows any VDF developer to estimate the potential maximum computation speed

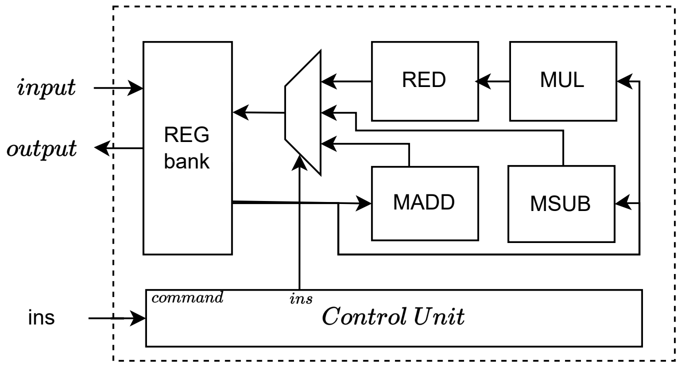

Figure 6: FITER cryptoprocessor. RED, MUL, MADD and MSUB blocks represent units that can perform modular reduction, multiplication, modular addition and modular subtraction in CS representation, respectively.

of the VDF while considering advancements in silicon technology, an example of which we provide later in Sec. 5. 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 note that 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:

- **Integer addition (ADD)**: $\delta_{\text{add}} \approx 2 \cdot \tau_{FA}$, see fig. 1.
- **Integer 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.
- **Integer 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.
- **Integer 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 now provide details of the estimated delays of various operations in term of full adders for the two architectures. The delays of FITER's building blocks comprising of modular arithmetic (Sec. 4.3) are given below:

- **Modular addition (MADD)**: $\delta_{\text{M. ADD}} \approx 4 \cdot \tau_{FA} + \tau_{HA} + \tau_{MUX}$. A modular addition is composed of one addition (ADD) and one 2-bit new reduction.
- **Modular subtraction (MSUB)**: $\delta_{\text{M. SUB}} \approx 7 \cdot \tau_{FA} + \tau_{XOR} + \tau_{HA} + \tau_{MUX}$. A modular subtraction is made of one subtraction and one reduction.
- **Modular multiplication (MMUL)**: $\delta_{\text{M. MUL}} \approx 4 \cdot \tau_{AND} + (f(m+3) + 2 \cdot f(m) + 11) \cdot \tau_{FA} + \tau_{XOR} + \tau_{HA} + \tau_{MUX}$.

Since FAVE's design uses elliptic curves operations as building blocks (Sec. 4.2), the critical path of isogeny evaluation, which is the only operation interesting to the attacker, is listed below:

- **4-iso-e**: $\delta_{\text{4-iso-e}} \approx 12 \cdot \tau_{AND} + (3 \cdot f(m+3) + 6 \cdot f(m) + 40) \cdot \tau_{FA} + 4 \cdot \tau_{XOR} + 4 \cdot \tau_{HA} + 4 \cdot \tau_{MUX}$. The critical path, given in fig. 5, is one modular subtraction, three multiplications and three Montgomery reductions.

5 Results

In this work, we set out to design an attack accelerator with near-maximum parallel processing capabilities to evaluate a 2^T isogeny in the scenario of a VDF. This led us to design FAVE that achieves this feat, but requires massive amount of resources. Hence, we introduced a second design FITER to help us estimate FAVE's area and time requirements. In this section we first provide a detailed analysis of the advantage of our design choice to use CS representation for integer arithmetic. Then, we provide synthesis results for the building blocks of FITER, the challenges we faced and the methods we adopted to overcome these challenges. Once equipped with these results for FITER, we move on to estimate the area and timing requirements of FAVE. Finally, we also discuss technology-independent metrics in order to showcase the relevance of our analysis even in the fast-changing technological landscape.

CS vs non-redundant representation: In non-redundant or standard 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 will still be longer than 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 generate partial products before adding them together. For high performance, the small multiplications can be computed in parallel by using multiple small multipliers and 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 $\approx 3000\times$ compared to a non-redundant multiplier (note that the speedup value might be lower depending on how the design approaches).

Synthesis exploration: Synthesizing large designs using EDA tools is not a trivial task. In this work, we used Cadence Genus 2019 and targeted a 28nm library for ASIC synthesis. The relatively old but commonly available 28nm ASIC technology is a baseline for estimating the area and time of VDF evaluation in more advanced technologies, such as 3nm, using standard technology scaling principles. We worked on a CPU node equipped with one AMD EPYC 9754 128-Cores Processor running at 2.25GHz with 512 GB RAM. Even though we used a very powerful CPU, we faced several problems to synthesize for large m (e.g., $m = 1506$) in ASIC. Specifically, the synthesis tool at first could not meet the

Table 3: Area cost of the arithmetic modules.

Size	**Critical path (ns)/Area (mm^2)**				
(m)	$\mathbb{F}_p$ 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	0.76/0.210
1506	0.66/0.0250	0.70/0.0130	1.3/42	1.1/58.6	0.77/44.1

heavy requirements of different elements in the design for large m and failed to complete the synthesis.

The standard approach for synthesis is to synthesize the design using the default configuration. By default, the tool tries to unify the different instances of the same module and ungroup smaller modules (e.g., flattening the design) to achieve better area/timing results for the synthesis. However, this approach complicates the synthesis operation and increases the run time significantly and caused it to get stuck while showing no process after a few days. The standard synthesis approach worked only for small m values while it failed to generate a synthesized netlist for large m values.

We tried several approaches to find the optimal way to synthesize our design. We followed a divide-and-conquer approach for synthesis. We first divided a target design into smaller modules and generated a synthesized netlist for each module separately. Then, we used the synthesized netlists of these modules to construct and synthesize the target design. In order to improve the run time, we used `read_netlist` synthesis command for reading already synthesized netlist design files for small modules. This enables us to simplify design elaboration and mapping steps. This approach significantly improved the run time and enabled the tool to finish the runs.

Besides, we used synthesis commands to minimize "ungrouping" and "unification" of sub modules. Further, we also used "preserve" command to eliminate any extra optimization effort to improve the run time. Using all of this, we were able to synthesize the multiplication, squaring and reduction modules for the largest parameter set $m = 1506$ which we were previously failing to do so. Finally, we successfully synthesized the entire FITER architecture.

Area and timing results for FITER: In this paragraph, we provide the implementation results of the proposed designs and analyze them to derive useful conclusions. 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. Table 3 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 in both FITER and FAVE. The tool reported that the FITER design has an area of 99 mm^2 and a frequency of 360 MHz for $m = 1506$. The overall area of FITER in gate equivalent (GE) units is $71,048,509$ GE. In FITER, the area breakdown is 96% for logic and 0.5% for memory.

Area and timing estimation of FAVE using FITER: We now provide estimations for the FAVE design using FITER as a reference. For the FAVE design, recall from Sec. 4.4 that the critical path is defined by the 4-iso-e unit and is characterized by the following: one modular subtractions, three multiplications, and three Montgomery reductions. Following Sec. 4.4 and Table 3, we calculate the critical path of FAVE for $m = 1506$ to be approximately $1 \cdot 0.7 + 3 \cdot 1.1 + 3 \cdot 0.7 = 6.1$ ns. We also need to added an extra 1 ns to take into account the delay of the memory and routing, thus the delay is of 7.1 ns, which translates to a clock frequency of $1/((7.1) \cdot 10^{-9}) = 140$ MHz. Following Sec. 4.4 and Table 3, the FAVE design uses four modular subtractions, eight multiplications, two squaring and ten large reduction units. Thus, based on the the analysis above, the area for FAVE is estimated to be,

$$A = 4 \cdot 0.0130 + 8 \cdot 58.6 + 2 \cdot 42 + 10 \cdot 44.1 = 994 \text{ mm}^2.$$

Table 4 shows the speed-up of FAVE over FITER. The last column represent the metrics for FAVE relative to FITER (FAVE/FITER). Overall for elliptic curves operations, the FAVE architecture is $14 \cdot 0.394 = 5.5\times$ faster than FITER. These results translate into a throughput of $(1/7.1 \cdot 10^6) = 140,845$ isogenies of degree 4 per milliseconds for FAVE and $(1/(2.8 \cdot 14) \cdot 10^6) = 25,510$ isogenies of degree 4 per milliseconds for FITER. Both designs outperform the software implementation of [DFMPS19], which achieves 0.75 2-isogenies per ms, which is a speedup of $68,026\times$ for FITER compared to the software implementation.

Table 4: Comparison between FAVE and FITER

Design / Metrics	**FAVE**	**FITER**	**FAVE/FITER**
Critical path	7.1 ns	2.8 ns	0.394
Latency for ec operations	1 cc	14 cc	14

Bandwidth discussion: The accelerator needs $2 \cdot 1506 = 3012$ bits per cycle for the two kernel points associated with each 4-isogeny, given $m = 1506$. This translates to a bandwidth requirement of around 53 GB/s, which is well within the capacity of current-generation HBM3E memory, capable of delivering up to 1.2 TB/s. Furthermore, a resourceful adversary could potentially integrate multiple memory IPs to enable parallel access, further mitigating any potential bandwidth limitations.
Technology-agnostic analysis and a use-case for VDF: Table 5 presents the critical path delay for all the modules presented in Sec. 4 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. We have noticed a logarithmic relationship between the number of full adders (FA) in the critical path and the bit size of the prime. In Table 5, we highlight the critical path delay in the number of full adders on critical path as the technology-agnostic parameter because this can be used to determine the latency of the VDF over time even with improving technology.

Table 5: Critical path delay of the different modules.

Module	**without FA**	$m = 89$ **(FA)**	$m = 1506$ **(FA)**
MADD	$\tau_{HA} + \tau_{MUX}$	$4 \cdot \tau_{FA}$	$4 \cdot \tau_{FA}$
MSUB	$\tau_{XOR} + \tau_{HA} + \tau_{MUX}$	$7 \cdot \tau_{FA}$	$7 \cdot \tau_{FA}$
MMUL	$4 \cdot \tau_{AND} + \tau_{XOR} + \tau_{HA} + \tau_{MUX}$	$32 \cdot \tau_{FA}$	$59 \cdot \tau_{FA}$
4-iso-e	$12 \cdot \tau_{AND} + 4 \cdot \tau_{XOR} + 4 \cdot \tau_{HA} + 4 \cdot \tau_{MUX}$	$130 \cdot \tau_{FA}$	$193 \cdot \tau_{FA}$

Let us revisit the example where Alice wants to determine secure VDF parameters in Sec. 1.1. Alice needs to determine the smallest safe parameter k (w.r.t 4^k-isogeny) and T (w.r.t 2^T-isogeny) for a chosen prime, such that the VDF evaluation takes at least t seconds. We will now show how to calculate k and T based on the results we have obtained using τ_{FA}. The time t for the VDF evaluation is given by:

$$t = l \cdot \tau_{\text{FA}} \cdot f(k) \;\Leftrightarrow\; f(k) = \frac{t}{l \cdot \tau_{\text{FA}}}, \tag{7}$$

where l is the number of full adders used in a 4-isogeny evaluation, and $f(k)$ the number of 4-isogeny evaluation in the VDF. Alice followed the FMPS19 VDF and chooses the same 1506-bit prime p. In her analysis, she assumes that the latest 3nm silicon technology will remain optimal for a few more years, allowing her to estimate the delay of a full adder $\tau_{\text{FA}} = 5$ to 10 ps. From Table 5, she finds $l \approx 200$ by approximating the other gate-level elements as full adder equivalents. Alice wants to set the minimum delay for her VDF to be 1 minute, so $t = 60$ seconds. She can now estimate the security parameter T using $T = 2 \cdot k$ and calculates $f(k)$ as $f(k) = 60/(200 \times 5 \times 10^{-12}) \approx 60.0 \times 10^9$, assuming the lower bound of the full adder delay. From Sec.2.2, we know that $f(k) = k$, so $k \approx 60.0 \times 10^9$, and hence $T = 2 \cdot k = (60.0 \times 10^9) \cdot 2 = 120.0 \times 10^9$. Alice can set the VDF evaluation step to a $2^{120 \times 10^9}$-degree isogeny. In comparison, using 28nm silicon technology, where the delay of a full adder typically ranges from 40 to 70 ps, depending on the library, optimization techniques, and design constraints such as power and area trade-offs. In our accelerator we found that the delay was 46 ps. Using the same estimation methodology as before, the VDF parameter T in 28nm would be set to 13.0×10^9.

6 Conclusion

Isogeny-based VDF constructions are becoming popular because of their well-studied cryptographic properties. Apart from conceptual isogeny VDF constructions and their unoptimized software implementations, no efficient implementation suitable enough for setting realistic security parameters exists. The time required for a VDF evaluation is crucial for setting security parameters. An attacker could cheat if they are able to compute the VDF faster by using immense resources. This paper considered such an attacker with unbounded resources and aimed to design an attack accelerator that utilizes massive parallel computational capabilities for VDF evaluation for the isogeny-based VDF in FMPS19 [DFMPS19]. We proposed two low-latency hardware implementations, FAVE and FITER of isogeny-based VDFs for ASIC platforms. Both of our designs have been made using CS representation to greatly increase computational speed for a VDF evaluation. While FAVE is an extreme accelerator with near-maximum parallel processing capabilities, FITER is a scaled-down, less parallel accelerator that is synthesizable using present-day EDA tools. Using synthesis results from FITER, we estimated the area and time required by the large and unrolled design of FAVE. We further provided comparison parameters which are independent of changing technologies. We illustrated with an example that using these results and estimates, it is possible to realistically calculate a lower bound for the time required to evaluate a 2^T-isogeny, thus realizing the goal of our work. We hope that our work can be used in standardizing isogeny VDFs for real-world applications.

Acknowledgement

This work was supported in part by the FWF (Austrian Science Fund) grant PAT6402023. We are thankful to Aikata Aikata, Florian Krieger and Florian Hirner for helping with the synthesis of the large design. We are also grateful to Dr. Christophe Petit for helpful discussions. Finally, we extend our thanks to the anonymous reviewers of IACR CiC 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 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I*, volume 10991 of *Lecture Notes in Computer Science*, pages 757–788. Springer, 2018. doi:10.1007/978-3-319-96884-1_25.

[BF21] Jeffrey Burdges and Luca De Feo. Delay encryption. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part I*, volume 12696 of *Lecture Notes in Computer Science*, pages 302–326. Springer, 2021. doi:10.1007/978-3-030-77870-5_11.

[BFLS20] Daniel J. Bernstein, Luca De Feo, Antonin Leroux, and Benjamin Smith. Faster computation of isogenies of large prime degree. *CoRR*, abs/2003.10118, 2020. URL: https://arxiv.org/abs/2003.10118, arXiv:2003.10118.

[CD23] Wouter Castryck and Thomas Decru. An efficient 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*, page 423–447, Berlin, Heidelberg, 2023. Springer-Verlag. doi:10.1007/978-3-031-30589-4_15.

[CH17] Craig Costello and Huseyin Hisil. A simple and compact algorithm for sidh with arbitrary degree isogenies. Cryptology ePrint Archive, Paper 2017/504, 2017. https://eprint.iacr.org/2017/504. URL: https://eprint.iacr.org/2017/504.

[CLG09] Denis Xavier Charles, Kristin E. Lauter, and Eyal Z. Goren. Cryptographic hash functions from expander graphs. *J. Cryptol.*, 22(1):93–113, 2009. doi:10.1007/s00145-007-9002-x.

[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. URL: https://doi.org/10.1007/978-3-030-99277-4_21.

[Dad65] L. Dadda. Some schemes for parallel multipliers. *Alta Frequenza*, 34:349–356, 1965. URL: http://bwrcs.eecs.berkeley.edu/Classes/icdesign/ee241_s01/PAPERS/archive/dadda65.pdf.

[DFMPS19] Luca De Feo, Simon Masson, Christophe Petit, and Antonio Sanso. Verifiable delay functions from supersingular isogenies and pairings. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology – ASIACRYPT 2019*, pages 248–277, Cham, 2019. Springer International Publishing. URL: https://doi.org/10.1007/978-3-030-34578-5_10.

[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. doi:10.1007/978-3-030-57990-6_4.

[DMPR23] Pierrick Dartois, Luciano Maino, Giacomo Pope, and Damien Robert. An algorithmic approach to (2,2)-isogenies in the theta model and applications to isogeny-based cryptography. *IACR Cryptol. ePrint Arch.*, page 1747, 2023. URL: https://eprint.iacr.org/2023/1747.

[DMS23] Thomas Decru, Luciano Maino, and Antonio Sanso. Towards a quantum-resistant weak verifiable delay function. In Abdelrahaman Aly and Mehdi Tibouchi, editors, *Progress in Cryptology - LATINCRYPT 2023 - 8th International Conference on Cryptology and Information Security in Latin America, LATINCRYPT 2023, Quito, Ecuador, October 3-6, 2023, Proceedings*, volume 14168 of *Lecture Notes in Computer Science*, pages 149–168. Springer, 2023. doi:10.1007/978-3-031-44469-2_8.

[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. URL: https://doi.org/10.1007/3-540-48071-4_10.

[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. doi:10.1007/978-3-030-95312-6_3.

[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. doi:10.1515/jmc-2012-0015.

[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. *NIST*, 2022. URL: https://sike.org/files/SIDH-spec.pdf.

[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. doi:10.1109/TCSI.2020.2992747.

[KCYL06] Fanyu Kong, Zhun Cai, Jia Yu, and Daxing Li. Improved generalized atkin algorithm for computing square roots in finite fields. *Information Processing Letters*, 98(1):1–5, 2006. URL: https://www.sciencedirect.com/science/article/pii/S0020019005003364, doi:10.1016/j.ipl.2005.11.015.

[KH98] C. K. Koç and Ching-Yu Hung. Fast algorithm for modular reduction. In *IEE Proceedings: Computers and Digital Techniques*, 1998. URL: https://cetinkayakoc.net/docs/j48.pdf.

[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. doi:10.1155/2018/5747642.

[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. doi:10.1109/TIT.2019.2938984.

[LW17] Arjen K. Lenstra and Benjamin Wesolowski. Trustworthy public randomness with sloth, unicorn, and trx. *Int. J. Appl. Cryptogr.*, 3(4):330–343, 2017. doi:10.1504/IJACT.2017.10010315.

[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. doi:10.1109/ACSSC.2003.1291939.

[MMP+23] Luciano Maino, Chloe Martindale, Lorenz Panny, Giacomo Pope, and Benjamin Wesolowski. A direct key recovery attack on SIDH. In Carmit Hazay and Martijn Stam, editors, *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*, volume 14008 of *Lecture Notes in Computer Science*, pages 448–471. Springer, 2023. doi:10.1007/978-3-031-30589-4_16.

[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. doi:10.1109/TC.2020.3043400.

[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. doi:10.1090/mcom/3036.

[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. URL: http://drops.dagstuhl.de/opus/volltexte/2018/10153, doi:10.4230/LIPIcs.ITCS.2019.60.

[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. URL: https://www.sciencedirect.com/science/article/pii/0898122183901335, doi:10.1016/0898-1221(83)90133-5.

[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. doi:10.1109/TVLSI.2019.2905899.

[Rob23] Damien Robert. Breaking SIDH in polynomial time. In Carmit Hazay and Martijn Stam, editors, *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*, volume 14008 of *Lecture Notes in Computer Science*, pages 472–503. Springer, 2023. doi:10.1007/978-3-031-30589-4_17.

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

[SB23] Guantong Su and Guoqiang Bai. Towards high-performance supersingular isogeny cryptographic hardware accelerator design. *Electronics*, 12(5), 2023. URL: https://www.mdpi.com/2079-9292/12/5/1235, doi:10.3390/electronics12051235.

[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. doi:10.46586/tches.v2022.i4.163-187.

[Sil09] J.H. Silverman. *The Arithmetic of Elliptic Curves*. Graduate Texts in Mathematics. Springer New York, 2009. URL: https://books.google.at/books?id=Z90CA_EUCCkC.

[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. doi:10.1109/EUC.2008.30.

[Tat66] J. Tate. Endomorphisms of Abelian Varieties over Finite Fields. *Inventiones Mathematicae*, 2:134, January 1966. doi:10.1007/BF01404549.

[Vél71] J. Vélu. Isogénies entre courbes elliptiques. *Comptes-Rendus de l'Académie des Sciences, Série I*, 273:238–241, juillet 1971.

[Wal64] Christopher S. Wallace. A suggestion for a fast multiplier. *IEEE Trans. Electron. Comput.*, 13(1):14–17, 1964. doi:10.1109/PGEC.1964.263830.

[Was08] L.C. Washington. *Elliptic Curves: Number Theory and Cryptography, Second Edition (2nd ed.)*. Chapman and Hall/CRC, 2008. doi:10.1201/9781420071474.

[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. URL: https://doi.org/10.1007/978-3-030-17659-4_13.

[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. doi:10.46586/tches.v2023.i1.438-462.

A Background about 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'.

A.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 optimization techniques with respect to point doubling and algorithms for different elliptic curves later in Sec. 3.3. 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, $\phi_4 : (x_P, y_P) \to \left(x_{\phi_4(P)}, y_{\phi_4(P)}\right)$ for a point $P = (x_P, y_P) \notin G$ can be described by the following two equations:

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

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

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 optimized 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:

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

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 $(X, Y, Z) \to (X/Z, Y/Z)$.

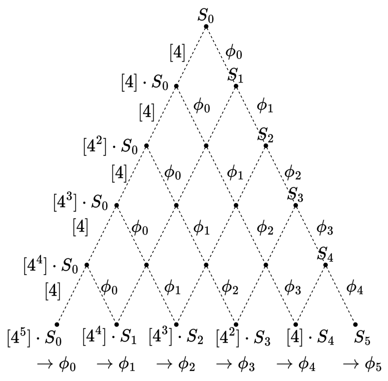
S_0
$[4]$
ϕ_0
$[4]\cdot S_0$
S_1
ϕ_1
$[4^2]\cdot S_0$
S_2
ϕ_2
$[4^3]\cdot S_0$
S_3
ϕ_3
$[4^4]\cdot S_0$
S_4
ϕ_4
$[4^5]\cdot S_0$ $[4^4]\cdot S_1$ $[4^3]\cdot S_2$ $[4^2]\cdot S_3$ $[4]\cdot S_4$ S_5
$\to\phi_0$ $\to\phi_1$ $\to\phi_2$ $\to\phi_3$ $\to\phi_4$ $\to\phi_5$

Figure 7: Computation structure for $\phi = \phi_5 \circ \phi_4 \circ \phi_3 \circ \phi_2 \circ \phi_1 \circ \phi_0$

B Strategies for computing isogenies

Ever since the SIDH protocol was first proposed, a lot of work has been done to optimize the computation of smooth large-degree isogenies [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. 7 for $k = 6$ and $\ell = 4$, 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. 7. There are two main operations in isogeny "arithmetic": point quadrupling (or two point doubling) and 4-isogenies with $\ell = 4$. 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. fig. 8(a) 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 parallelized. All of the isogeny evaluations through the same isogeny ϕ_i can be computed in parallel (fig. 8(b)).

The Optimized strategy: first introduced by [FJP14, JAC+22], this strategy (fig. 8(c)) is done by finding an optimum computation strategy. Those strategies, as shown in the right figure of fig 8, 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. 8, 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.

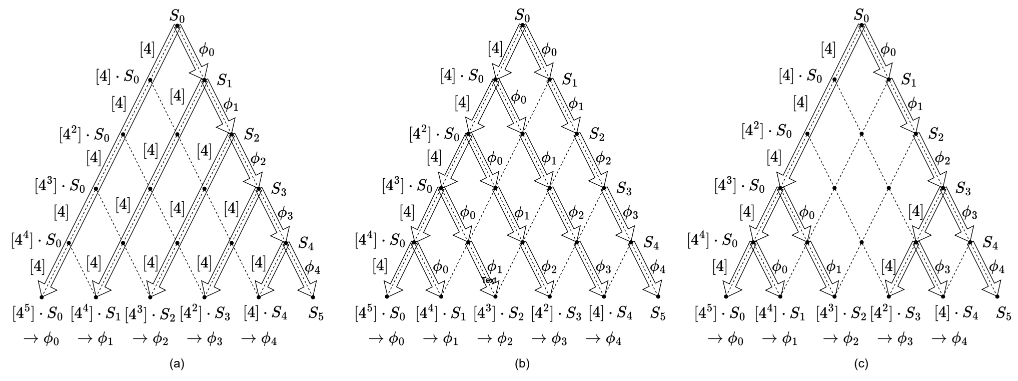

Figure 8: (a) "Basic", (b) Full evaluation and, (c) Optimized isogeny strategies