Efficient Implementation of √élu's Formulas

Jianming Lin[1][0009−0007−9144−1432], Weize Wang[2][0009−0000−0516−1776], Changan Zhao[1,3,4][2222−−3333−4444−5555], and Yuhao Zheng[1]

[1] School of Mathematics, Sun Yat-sen University,
Guangzhou 510275, P.R.China
linjm28@mail2.sysu.edu.cn
zhaochan3@mail.sysu.edu.cn
zhengyh57@mail2.sysu.edu.cn
[2] School of Computer Science, Fudan University
Shanghai 200438, P.R.China
wzwang23@m.fudan.edu.cn
[3] Guangdong Key Laboratory of Information Security,
Guangzhou 510006, P.R. China
[4] State Key Laboratory of Information Security (Institute of Information Engineering), Chinese Academy of Science,
Beijing 100093, P.R.China

Abstract. In the implementation of isogeny-based schemes, Vélu's formulas are essential for constructing and evaluating odd degree isogenies. Bernstein et al. proposed an approach known as √élu, which computes an ℓ-isogeny at a cost of $\tilde{\mathcal{O}}(\sqrt{\ell})$ finite field operations. This paper presents two key improvements to enhance the efficiency of the implementation of √élu from two aspects: optimizing the partition involved in √élu and speeding up the computations of the sums of products used in polynomial multiplications over finite field $\mathbb{F}_p$ with large prime characteristic p. To optimize the partition, we adjust it to enhance the utilization of x-coordinates and eliminate the computational redundancy, which can ultimately reduce the number of $\mathbb{F}_p$-multiplications. The speedup of the sums of products is to employ two techniques: lazy reduction (abbreviated as LZYR) and generalized interleaved Montgomery multiplication (abbreviated as INTL). These techniques aim to minimize the underlying operations such as $\mathbb{F}_p$-reductions and assembly memory instructions. We present an optimized C and assembly code implementation of √élu for the CTIDH512 instantiation. In terms of ℓ-isogeny computations in CTIDH512, the performance of clock cycles applying new partition + INTL (resp. new partition + LZYR) offers an improvement up to 16.05% (resp. 10.96%) compared to the previous work.

Keywords: isogeny-based cryptography √élu formulae partition

1 Introduction

Isogeny-based post-quantum scheme was first proposed in 2006 by Couveignes, Rostovtsev and Stolbunov in [19,45] (CRS scheme). The security of this system

is based on the difficulty of computing isogenies between ordinary curves defined over a finite field. Since then, isogeny-based cryptography has gained increasing attention from cryptographers.

The CRS scheme faces bottlenecks in terms of efficiency and security. Childs et al. showed that the private key of CRS could be recovered in subexponential time on a quantum computer [15]. In 2011, the Supersingular Isogeny-based Diffie-Hellman key exchange protocol (SIDH) was presented by Jao and De Feo in [31]. This protocol is more efficient than CRS. In 2014, a supersingular isogeny key encapsulation scheme (SIKE) based on SIDH was proposed [9]. SIKE was submitted to the NIST post-quantum cryptography standardization project (https://www.nist.gov/) in 2017, and passed to the round 4 of this contest as an alternative candidate in 2021.

However, more recently Castryck and Decru presented an efficient key recovery attack on SIDH [12]. Maino et al. [36] provided a subexponential algorithm to break SIDH with arbitrary starting curves. Inspired by these works, Robert [44] presented a deterministic polynomial time attack on SIDH in all cases. All of these attacks can also be applied to Séta [24] and B-SIDH [16].

Fortunately the above attacks do not extend to CSIDH, a commutative class group action protocol proposed by Castryck et al. in 2018 [13]. CSIDH is another isogeny-based key exchange protocol based on the CRS scheme, operating with supersingular curves. Compared with CRS and its variant presented by Kieffer et al. in [33], CSIDH is significantly faster. Banegas et al. presented a constant-time implementation of CSIDH named CTIDH [3] in 2021, which is the most efficient constant-time variant of CSIDH. Nevertheless, CSIDH/CTIDH also suffers from a subexponential-time attack on quantum computers since the endomorphism ring over $\mathbb{F}_p$ is commutative. Therefore, several works suggest that CSIDH should work with larger parameters to reach the NIST security level [8,43,14]. This means that CSIDH needs to involve the constructions and evaluations of larger degree isogenies, which are the most expensive computational tasks.

These key exchange protocols can be used to construct digital signatures. Until now the two kinds of signatures: CSIDH-based [21,7,2,25] and quaternion-based [27,22,23,20] are not vulnerable to the Castryck-Decru-Maino-Martindale-Robert attacks [12,36,44] since they do not reveal the torsion point information. Both of them involve the constructions and evaluations of large degree isogenies. Additionally, an isogeny-based public key encryption (PKE) named FESTA [5] was proposed by Basso et al. in 2023, whose encryption process also requires large degree isogeny computations. Based on the above, the computations of large degree isogeny are the fundamental operations in the majority of isogeny-based schemes.

Denote an isogeny of odd degree ℓ by ℓ-isogeny. Vélu's formulas [48] have been widely used to construct and evaluate ℓ-isogenies. In recent years, there are several works focusing on optimizing the constructions and evaluations of ℓ-isogenies via Vélu's formulas [17,18,37,30,4]. They use the other coordinates besides Montgomery models such as Edwards coordinates to speed up the finite field arithmetic, or apply the technique of addition chains to optimize the

computations of scalar multiplications involved in isogeny computations. These optimizations obtain a cost of $\tilde{\mathcal{O}}(\ell)$ field operations.

Bernstein et al. presented a new technique to construct and evaluate ℓ-isogenies at a cost of about $\tilde{\mathcal{O}}(\sqrt{\ell})$ field operations in 2020 [6]. This approach was achieved by observing that the evaluations of polynomial products can be efficiently done via the baby-step giant-step algorithm. Due to its square root complexity, this variant of Vélu's formulas is named as square-root Vélu's formulas, or simply √élu. The experimental tuning results in [6] demonstrate that √élu performs better for $\ell \geq 83$. Adj et al. presented a more concrete computational analysis of √élu and employed novel techniques to reduce the polynomial multiplications [1] when performing the evaluations of ℓ-isogenies.

Consequently, several works have focused on developing more efficient square-root Vélu's formulas of ℓ-isogenies on other supersingular elliptic curve models beyond Montgomery curves. In 2019, Moriya et al. performed CSIDH on Edwards curves [42], utilizing the w-coordinate for ℓ-isogeny constructions and evaluations. Following the proposal of √élu, they applied it to their new approach and implementation and achieved further optimization [41]. More recent works include constructions and evaluations of ℓ-isogenies on Huff curves [34], Twisted Jacobi quartic curves [28] [29] and Twisted Hessian curves [47].

Our Contributions. We speed up the implementation of √élu from two perspectives: optimizing the partition required in √élu to reduce the number of multiplications over finite fields and saving the underlying finite field arithmetic involved in the polynomial multiplications. Let $\mathbb{F}_p$ be a finite field with p elements in the rest of this paper, where p is an odd prime.

For the previous partition $S = (I \pm J) \cup K$ [6] required in the process of √élu, the set I is completely not included in S which may bring some computational redundancy. Inspired by this, we modify the above partition and rewrite S by $\tilde{S}$ to make the new set $\tilde{I}$ included in $\tilde{S}$. This adjustment can enhance the utilization of the set of x-coordinates and share more computations of scalar multiplications, which can reduce the number of $\mathbb{F}_p$-multiplications.

Inspired by Bernstein *et al.*'s suggestion in Appendix A.5 of [6], we consider lazy reduction [46] and generalized Montgomery multiplication [35] to speed up √élu by reducing the $\mathbb{F}_p$-modular reductions and the underlying memory instructions respectively in concrete implementation.

Based on the code of [3] (`https://ctidh.isogeny.org/`) and [35], we implement √élu on CTIDH512 instantiation using our improvements. More precisely, our main contributions are summarized as follows:

1. Our new partition saves the $\mathbb{F}_p$-multiplications by up to 5.64% compared with the previous implementation of CTIDH512 when computing large ℓ-isogenies. Moreover, it can effectively expand the utilization of √élu in isogeny-based cryptosystems.
2. We combine the new partition with lazy reduction to speed up the computation of large ℓ-isogenies over $\mathbb{F}_p$. The experimental results of clock cycles (resp. multiplication instructions) for computing large ℓ-isogenies indi-

cate a reduction of up to 10.96% (resp. 15.75%) compared to the previous work [3,6].
3. We compute the sums of products used in polynomial multiplications over $\mathbb{F}_p$ by applying generalized interleaved multiplication and combine it with our new partition. This approach leads to a speedup in ℓ-isogeny computations, reducing the number of memory instructions by up to about 24.25% ($\ell = 347$). The performing results for computing large degree ℓ-isogenies show a clock cycle saving of up to 16.05% ($\ell = 137$) compared to the previous work [3,6].

Outline. The remainder of this paper is organized as follows. In Section 2, we give a brief overview of traditional Vélu's formulas and $\sqrt{\text{élu}}$. We also provide an introduction of Montgomery multiplication and lazy reduction. Section 3 presents the approaches to speed up the computation of large degree ℓ-isogenies. In Section 4, we compare the experimental results of the cost of computing ℓ-isogenies on CTIDH512 utilizing our methods with the previous work. Finally, our conclusion and future work are drawn in Section 5.

Notations. Denote the $\mathbb{F}_p$-multiplications and $\mathbb{F}_p$-squares by $\boldsymbol{M}$ and $\boldsymbol{S}$, respectively.

2 Preliminaries

In this section, we introduce the corresponding mathematical preliminaries used in our methods, including isogeny formulas, Montgomery multiplications, and lazy reduction.

2.1 Vélu's Formulas

Let E and E' be two elliptic curves defined over $\mathbb{F}_p$. An isogeny $\varphi : E \to E'$ defined over $\mathbb{F}_p$ can be regarded as a non-constant group homomorphism from $E(\mathbb{F}_p)$ to $E'(\mathbb{F}_p)$ such that $\varphi(\mathcal{O}_E) = \mathcal{O}_{E'}$, where $\mathcal{O}_E$ (resp. $\mathcal{O}_{E'}$) denotes the point at infinity on E (resp. E'). The degree of φ is its degree as a group homomorphism, denoted by $\deg(\varphi)$. The kernel of φ is a finite subgroup $G \subset E(\mathbb{F}_p)$ such that $\varphi(G) = \{\mathcal{O}_E\}$, and we name it $\ker(\varphi)$. For *separable isogenies*, the equality $\deg(\varphi) = \#\ker(\varphi)$ always holds [31]. Concrete formulas of φ were first given by Vélu [48,26].

A Montgomery curve [39] is an elliptic curve defined through the following equation:

$$E_A/\mathbb{F}_p : y^2 = x^3 + Ax^2 + x,$$

where $A \in \mathbb{F}_p \setminus \{\pm 2\}$. The Montgomery model is preferred in isogeny-based cryptosystems due to its efficient arithmetic.

Vélu's Formulas on Montgomery curves. Costello and Hisil proposed the odd-degree isogenies (Vélu's) formulas on Montgomery curves using x-coordinates [17]. The computations include the evaluations of the image point $Q' = \varphi(Q)$ and the codomain curve coefficient A'.

Theorem 1. *For a finite field $\mathbb{F}_p$, let ℓ and s be two integers satisfying $\ell = 2s+1$, where ℓ is odd. Let $P \in E_A(\mathbb{F}_p)$ be a point of order ℓ on the Montgomery curve $E_A/\mathbb{F}_p : y^2 = x^3 + Ax^2 + x$, and write $\sigma = \sum_{i=1}^{s} x_{[i]P}$, $\tilde{\sigma} = \sum_{i=1}^{s} \frac{1}{x_{[i]P}}$ and $\pi = \prod_{i=1}^{s} x_{[i]P}$. The curve*

$$E_{A'}/\mathbb{F}_p : y^2 = x^3 + A'x^2 + x$$

with

$$A' = \pi^2 \cdot (6\tilde{\sigma} - 6\sigma + A)$$

is the codomain of ℓ-isogeny $\varphi : E_A \to E_{A'}$ with $Ker(\varphi) = \langle P \rangle$, which is defined by the rational map:

$$\varphi : x \mapsto f(x),$$

where

$$f(x) = x \cdot \prod_{i=1}^{s} \left(\frac{xx_{[i]P} - 1}{x - x_{[i]P}} \right)^2. \tag{1}$$

The Costello-Hisil algorithm [17] is the state-of-the-art for evaluating relatively small odd degree isogenies on Montgomery models. However, for larger degree ℓ-isogenies, there exists a more effective algorithm. We will describe it in the next subsection.

2.2 √élu's Formulas

Bernstein et al. first proposed a baby-step giant-step like method which is named √élu for computing large degree isogenies [6]. In the following, we state the computation process of √élu in detail.

From Theorem 1, the x-coordinate of $\varphi(Q)$ can be represented as

$$x_{\varphi(Q)} = x^{\ell} \cdot \prod_{i=1}^{s} \left(\frac{x_{[i]P} - 1/x}{x - x_{[i]P}} \right)^2.$$

Let $h_S(x) = \prod_{i \in S}(x - x_{[i]P})$. Then $x_{\varphi(Q)}$ can be expressed as a function of $h_S(x)$:

$$x_{\varphi(Q)} = x^{\ell} \cdot \prod_{i \in S} \left(\frac{h_S(1/x)}{h_S(x)} \right)^2$$

where $S = \{1, 3, \cdots, \ell - 2\}$.

The formulas from Theorem 1 give an approach to computing the codomain coefficient A'. Another approach is pointed out in [37]. One can transform it to twisted Edwards form and use the formulas in [40] to obtain $A' = 2 \cdot \frac{1+d}{1-d}$ where

$$d = \left(\frac{A-2}{A+2}\right)^{\ell} \left(\frac{h_S(1)}{h_S(-1)}\right)^{8}.$$

Note that both evaluating isogenies and computing codomain curves require the evaluation of $h_S(x)$. A straightforward approach is to perform $\tilde{\mathcal{O}}(\#S)$ $\mathbb{F}_p$-operations by enumerating the points in S. Alternatively, Bernstein et al. proposed a strategy to efficiently evaluate $h_S(x)$ and obtained a square root complexity speedup over the Costello-Hisil algorithm [17]. For our purpose, we present the following lemma in [11] to illustrate the relationship between x_P, x_Q, x_{P+Q} and x_{P-Q}.

Lemma 1. *Let $E/\mathbb{F}_p$ be an elliptic curve. There exist biquadratic polynomials F_0, F_1 and F_2 in $\mathbb{F}_p[x_1, x_2]$ such that*

$$(x - x_{P+Q})(x - x_{P-Q}) = x^2 + \frac{F_1(x_P, x_Q)}{F_0(x_P, x_Q)}x + \frac{F_2(x_P, x_Q)}{F_0(x_P, x_Q)}$$

for all P and Q in E such that $\mathcal{O}_E \notin \{P, Q, P+Q, P-Q\}$.

Consider the set $S = \{1, 3, \cdots, \ell - 2\}$. Let $I = \{2b + 4bi \mid 0 \leq i < b'\}$ and $J = \{2j + 1 \mid 0 \leq j < b\}$ where $b = \lfloor \frac{\sqrt{\ell-1}}{2} \rfloor$, $b' = \lfloor \frac{\ell-1}{4b} \rfloor$ if $b > 0$, and $b' = 0$ if $b = 0$. Then we have $S = (I \pm J) \cup K$ such that $h_S(x) = h_{(I \pm J) \cup K}(x)$ where the set $K = \{4bb' + 1, \cdots, \ell - 4, \ell - 2\}$. We can represent most elements of S through $(I + J) \cup (I - J) = I \pm J$ and make sure $I + J$, $I - J$ are disjoint. Consequently, we need to determine the sets of x-coordinates: $xI = \{x([i]P) \mid i \in I\}$, $xJ = \{x([j]P) \mid j \in J\}$ and $xK = \{x([k]P) \mid k \in K\}$ using the partition mentioned above while evaluating $h_S(x)$. Finally, we apply Lemma 1 to express $h_{I \pm J}(x)$ as a resultant of two polynomials of smaller size:

$$\begin{aligned} h_{I \pm J}(x) &= \prod_{(i,j) \in I \times J} (x - x_{[i+j]P})(x - x_{[i-j]P}) \\ &= \frac{Res_z(h_I(z), E_J(x, z))}{Res_z(h_I(z), D_J(z))} \end{aligned}$$

where

$$E_J(x, z) = \prod_{j \in J} (F_0(z, x_{[j]P})x^2 + F_1(z, x_{[j]P})x + F_2(z, x_{[j]P}))$$

and $D_J(z) = \prod_{j \in J} F_0(z, x_{[j]P})$.

By using the above techniques, the computational cost of performing an ℓ-isogeny has been reduced from $\tilde{\mathcal{O}}(\ell)$ to $\tilde{\mathcal{O}}(\sqrt{\ell})$. Specifically, Adj et al. [1] showed that the best algorithm for the polynomial multiplications inherent in $\sqrt{\text{élu}}$ is to apply Karatsuba's method, then the complexity becomes approximately $\tilde{\mathcal{O}}(k(\sqrt{\ell})^{\log_2 3})$ where k is a constant. Due to the constant k, $\sqrt{\text{élu}}$'s formulas perform better only when the prime degree ℓ is large.

2.3 Montgomery multiplication and lazy reduction

A widely-used method known as Montgomery multiplication was proposed by Montgomery in [38]. This method speeds up the modular multiplication by replacing long divisions by simple divisions with powers of 2.

To apply the Montgomery multiplication, all of $\mathbb{F}_p$-elements are represented in the Montgomery domain [38]. Let $R = 2^N$, and $p' = -p^{-1} \mod R$, where $N = nw, n = \lceil \frac{l}{w} \rceil, l = \lceil \log_2 p \rceil$, the value w is the computer wordsize. Since we implement the modular multiplication on the x64 platform, we set $w = 64$. For two $\mathbb{F}_p$-elements a and b, their Montgomery representations are given by $\tilde{a} = aR \mod p$ and $\tilde{b} = bR \mod p$, respectively. If it holds that $\tilde{a}\tilde{b} < pR$, the p-residue $c = \tilde{a}\tilde{b}R^{-1} \mod p$ can be computed as

$$c = (\tilde{a}\tilde{b} + ((\tilde{a}\tilde{b} \mod R)p' \mod R)p)/R. \tag{2}$$

Since it satisfies: $0 \leq c < \frac{pR+pR}{R} = 2p$, we need to execute a modular correction to ensure that c lies within the range $[0, p-1]$. Finally the result $ab \mod p$ can be easily computed by dividing c by the value R.

Let $\{a_0, a_1, \cdots, a_{t-1}\}$ and $\{b_0, b_1, \cdots, b_{t-1}\}$ be two sets of elements in $\mathbb{F}_p$. The computations of the sums of products: [35] $c = \sum_{i=0}^{t-1} \pm a_i b_i \mod p$ can be found at the core of many cryptologic computations, such as pairing and isogeny computations.

In practice, the technique of **lazy reduction** is widely used to optimize the implementation of computing the sums of products c. Instead of performing each modular multiplication $a_i b_i \mod p$ separately, we first compute the sum of unreduced integer products $\sum_{i=1}^{n} a_i b_i$, and finally perform the modular reduction $c = \sum_{i=1}^{n} a_i b_i \mod p$. This approach can significantly reduce the number of $\mathbb{F}_p$-modular reductions.

3 Main Results

In this section, we present our main optimizations to the implementation of √élu. Using the same notation as above, let P be a point of order ℓ. We modify the partition of sets I, J, K which can reduce the number of $\mathbb{F}_p$-multiplications in the computations of xI, xJ and xK. Moreover, since it needs to perform polynomial multiplications when dealing with $h_I(z)$, $E_J(x, z)$ and their resultant $Res_z(h_I(z), E_J(x, z))$ [6], we accelerate the computations of the sums of products over $\mathbb{F}_p$ which are widely used in polynomial multiplications by applying lazy reduction and generalized interleaved Montgomery multiplication [35].

3.1 Modifying the partition

From Section 2.2, the computation of $xI = \{x([i]P) \mid i \in I\}$, $xJ = \{x([j]P) \mid j \in J\}$ and $xK = \{x([k]P) \mid k \in K\}$ is a crucial part of the evaluation of $h_S(x)$ on √élu. In the previous partition [6], the set I is not included in S which may lead to some additional computations of scalar multiplications. Inspired by this,

we redefine the set S as $\tilde{S} = \{2, 4, \cdots, \ell - 1\}$ to make all the elements in $\tilde{S}$ be even. By the properties of x-coordinates on Montgomery curves we have $h_S(x) = h_{\tilde{S}}(x)$. We modify the partition of $I,\ J,\ K$ as follows:

$$\begin{aligned}
\tilde{I} &= \{2\tilde{b} + 2 + i(4\tilde{b} + 2) |\ i = 0, 1, \cdots, \tilde{b}' - 1\} \\
\tilde{J} &= \{2j + 2 |\ j = 0, 1, \cdots, \tilde{b} - 1\} \\
\tilde{K} &= \{4\tilde{b}\tilde{b}' + 2\tilde{b}' + 2, 4\tilde{b}\tilde{b}' + 2\tilde{b}' + 4, \cdots, \ell - 1\}
\end{aligned}$$

where $\tilde{b} = \lfloor \frac{\sqrt{\ell-1}}{2} \rfloor$ and $\tilde{b}' = \lfloor \frac{\ell-1}{4\tilde{b}+2} \rfloor$ are the sizes of the sets $\tilde{J}$ and $\tilde{I}$, respectively. It can be verified that the set $\tilde{I} \pm \tilde{J} = (\tilde{I} + \tilde{J}) \cup (\tilde{I} - \tilde{J})$ is a disjoint union as before. Let i_0 be an arbitrary element in $\tilde{I}$, we have:

$$2 \leq 2\tilde{b} + 2 \leq i_0 \leq 4\tilde{b}\tilde{b}' + 2\tilde{b}' - 2\tilde{b} < \ell - 1$$

Due to the arbitrariness of i_0, $\tilde{I}$ is completely contained in $\tilde{S}$. Consequently, $\tilde{S}$ can be expressed as: $\tilde{S} = (\tilde{I} \pm \tilde{J}) \cup \tilde{I} \cup \tilde{K}$. The evaluation formula of $h_{\tilde{S}}(x)$ can be represented as:

$$h_{\tilde{S}}(x) = h_{\tilde{I}}(x) h_{\tilde{K}}(x) \cdot Res_z(h_{\tilde{I}}(z), E_{\tilde{J}}(z, x_j)).$$

After the above adjustment, the set $\tilde{I}$ plays the same role as $\tilde{K}$ which can improve the utilization of the x-coordinates during the evaluation of $h_{\tilde{S}}(x)$. In the following, we state how this modification impacts the computational cost.

In practice, we do not fix $b = \lfloor \frac{\sqrt{\ell-1}}{2} \rfloor$ (resp. $\tilde{b} = \lfloor \frac{\sqrt{\ell-1}}{2} \rfloor$) and $b' = \lfloor \frac{\ell-1}{4b} \rfloor$ (resp. $\tilde{b}' = \lfloor \frac{\ell-1}{4\tilde{b}+2} \rfloor$) since the corresponding partition may not be optimal. We use a tuning program mentioned in [6] to identify the unique (b, b') (resp. $(\tilde{b}, \tilde{b}')$) which achieves the highest efficiency by looping through the values b and b' (resp. $\tilde{b}$ and $\tilde{b}'$), calculating and comparing the computational cost ($\mathbb{F}_p$-multiplication) of performing an ℓ-isogeny using the corresponding partition. The following theorem highlights that if the tuning results of the previous partition verify a specific condition, we can always save several $\mathbb{F}_p$-multiplications.

Theorem 2. *Let $\tilde{S} = (\tilde{I} \pm \tilde{J}) \cup \tilde{I} \cup \tilde{K}$ be our partition mentioned above. Let $S = (I \pm J) \cup K$ be the previous partition in [6]. If the tuning results of the previous partition satisfies $\#K \geq \#I$, we can save a certain amount of $\mathbb{F}_p$-multiplications using our partition after tuning when computing an ℓ-isogeny.*

Proof. We fix $\tilde{b} = b$ and $\tilde{b}' = b'$ for our partition after obtaining the optimal (b, b') for the previous partition in [6]. The size of K is: $\#K = \frac{\ell-1}{2} - 2bb'$. If it satisfies $\#K \geq \#I$, we have:

$$\#\tilde{K} = \frac{\ell - 1}{2} - \tilde{b}'(2\tilde{b} + 1) = \#K - \#I \geq 0.$$

Thus we can derive a relationship between $\#K$ and $\#\tilde{K}$:

$$\#K - \#\tilde{K} = \#I = \#\tilde{I}.$$

Therefore, we can save $\#\tilde{I}$ scalar multiplication computations compared with the previous partition when computing the sets of x-coordinates $x\tilde{I},\ x\tilde{J}$ and $x\tilde{K}$.

Furthermore, the tuple $(\tilde{b}, \tilde{b}') = (b, b')$ may not correspond to the optimal partition for our situation. We can reduce more $\mathbb{F}_p$-multiplications by taking the tuning result $(\tilde{b}, \tilde{b}')$ with respect to our optimal partition.

Remark 1. Note that the tuning program counts the whole $\mathbb{F}_p$-multiplications for an ℓ-isogeny computation, including the evaluation of the resultant . Therefore, the cost we save does not depend only on the computations of $xI,\ xJ$ and xK. We may also save some $\mathbb{F}_p$-multiplications although the condition $\#K \geq \#I$ is not verified.

We can share some computations of scalar multiplications when computing $xI,\ xJ$ and xK (resp. $x\tilde{I},\ x\tilde{J}$ and $x\tilde{K}$). Now we propose an example below for illustration. We take the odd prime $\ell = 191$, and the tuning result of the 191-isogeny using the previous partition corresponds to the tuple $(b, b') = (7, 6)$. Thus the size of K is $\frac{\ell-1}{2} - 2bb' = 11$, which satisfies the condition $\#K \geq \#I$ as stated in Theorem 2.

Example 1. Consider the odd prime $\ell = 191$. Using the previous partition, we have:

$$I = \{14, 42, \cdots, 154\},\ J = \{1, 3, \cdots, 13\},$$
$$K = \{169, 171, \cdots, 189\}.$$

The set xK can be represented as:

$$xK = \{x([2]P),\ x([4]P),\ \cdots,\ x([22]P)\}.$$

And it takes a point doubling and six differential additions to obtain xJ and $xK[0]$. After obtaining xJ, the elements: $xK[2i],\ i = 1, \cdots, 5$ can be directly doubled from $xJ[i],\ i = 1, \cdots, 5$, respectively. In addition, we can execute five point doublings to compute the remaining elements of xK except for $xK[0]$. Hence it takes ten point doublings to obtain xK. Finally we deal with xI. The first element $xI[0]$ has been computed in xK, and the addition step can be obtained by doubling it. Therefore, the cost of computing xI includes five differential additions and a point doubling.

For Montgomery model, a differential addition and a doubling need $4\boldsymbol{M} + 2\boldsymbol{S}$ and $4\boldsymbol{M} + 2\boldsymbol{C}$, respectively [9]. The notation $\boldsymbol{C}$ represents multiplication of an element of $\mathbb{F}_p$ by a constant of $\mathbb{F}_p$. Since we need to execute a chain of ℓ-isogenies on CTIDH, the two constants in $\mathbb{F}_p$ that need to be multiplied are not fixed. Thus $\boldsymbol{C}$ can be roughly regarded as $\boldsymbol{M}$. For simplicity we take $\boldsymbol{M} = \boldsymbol{S}$, then the total computational cost of $xI,\ xJ$ and xK becomes $138\boldsymbol{M}$.

Take $(\tilde{b}, \tilde{b}') = (b, b') = (7, 6)$ for our partition, we have:

$$\tilde{I} = \{16, 46, \cdots, 166\},\ \tilde{J} = \{2, 4, \cdots, 14\},$$
$$\tilde{K} = \{182, 184, \cdots, 190\}.$$

In this case $x\tilde{K} = \{x(P), x([3]P), \cdots, x([9]P)\}$. Similarly, it takes four differential additions and a point doubling to obtain $x\tilde{K}$ and $x\tilde{J}[0]$. Then the elements: $x\tilde{J}[2]$, $x\tilde{J}[4]$ and $x\tilde{J}[6]$ can be doubled from $x\tilde{K}[1]$, $x\tilde{K}[2]$ and $x\tilde{K}[3]$, respectively. The remaining elements except for $x\tilde{J}[0]$ in $x\tilde{J}$ can be obtained by performing three point doublings. Finally the first element $x\tilde{I}[0]$ can be doubled from $x\tilde{J}[3]$. The addition step of $x\tilde{I}$ can be obtained by adding the first element of $x\tilde{I}$ and the last element of $x\tilde{J}$. Thus it takes six differential additions and a point doubling to obtain $x\tilde{I}$. The total computational cost is $108\boldsymbol{M}$. We can save $30\boldsymbol{M}$ compared with the partition in [6] when computing $x\tilde{I}$, $x\tilde{J}$ and $x\tilde{K}$. From Theorem 2, the tuple $(\tilde{b}, \tilde{b}') = (7, 6)$ may not correspond to the optimal partition. Therefore we can reduce at least 30 $\mathbb{F}_p$-multiplications using our partition when computing the 191-isogeny.

On the basis of the above analysis, sharing the computations of scalar multiplications as far as possible can also reduce the cost of $\mathbb{F}_p$-multiplications. In our current partition, the intersection of $x\tilde{J}$ and $x\tilde{K}$ is empty. In other words, the shared computations between $x\tilde{K}$ and $x\tilde{J}$ are limited which may also bring some computational redundancy if $\#\tilde{K}$ is large. To avoid this, we still need to make an adjustment.

By exploiting the symmetry of the points in $\langle P \rangle$, we can rewrite the set S as $\tilde{S} = \{1, 2, \cdots, \frac{\ell-1}{2}\}$ and modify the partition of I, J, K as follows:

$$\tilde{I} = \{\frac{\ell-1}{2} - \tilde{b}'(2\tilde{b}+1) + \tilde{b} + 1 + i(2\tilde{b}+1) |\ i = 0, 1, \cdots, \tilde{b}' - 1\},$$

and

$$\begin{aligned}\tilde{J} =& \{j + 1 |\ j = 0, 1, \cdots, \tilde{b} - 1\},\\ \tilde{K} =& \{1, 2, \cdots, \frac{\ell-1}{2} - \tilde{b}'(2\tilde{b}+1)\}\end{aligned}$$

where $\tilde{b}$ and $\tilde{b}'$ are the sizes of the sets $\tilde{J}$ and $\tilde{I}$, respectively. Similarly to our first partition, we also have $h_S(x) = h_{\tilde{S}}(x)$ and the set $\tilde{I} \pm \tilde{J} = (\tilde{I} + \tilde{J}) \cup (\tilde{I} - \tilde{J})$ is a disjoint union. Let i_0 be an element of $\tilde{I}$, according to the above adjustment we have

$$1 \leq \frac{\ell-1}{2} - 2\tilde{b}\tilde{b}' - \tilde{b}' + \tilde{b} + 1 \leq i_0 \leq \frac{\ell-1}{2}$$

which implies that the set $\tilde{I}$ is completely included in $\tilde{S}$. Hence, the evaluation formula of $h_{\tilde{S}}(x)$ can also be rewritten as:

$$h_{\tilde{S}}(x) = h_{\tilde{I}}(x) h_{\tilde{K}}(x) \cdot Res_z(h_{\tilde{I}}(z), E_{\tilde{J}}(z, x_j)).$$

By applying the new partition above, the sets $x\tilde{I}$, $x\tilde{J}$ and $x\tilde{K}$ fulfill the condition: $x\tilde{J} \cap x\tilde{K} \neq \emptyset$ and $\tilde{I} \subset \tilde{S}$. Therefore, we can share more computations of scalar multiplications compared with our initial partition when computing $x\tilde{I}$, $x\tilde{J}$ and $x\tilde{K}$. Besides, we can determine the first element of $x\tilde{I}$ by adding the last element of $x\tilde{K}$ with the point $[\tilde{b}+1]P$ using x-only differential addition

after computing $x\tilde{J}$ and $x\tilde{K}$ and the addition step of $x\tilde{I}$ can be obtained by executing: $[2\tilde{b}+1]P \leftarrow [\tilde{b}+1]P + [\tilde{b}]P$. Algorithm 1 shows the computational procedure of $x\tilde{I}$, $x\tilde{J}$ and $x\tilde{K}$. We only consider the case $\#\tilde{J} > \#\tilde{K}$, to which the procedures of the other cases are similar.

Algorithm 1 Computing the sets of x-coordinates according to the new partition.

Input: The kernel of the ℓ-isogeny $\varphi : \langle P \rangle$. Positive integers $(\tilde{b}, \tilde{b}')$ which are the sizes of $\tilde{I}$ and $\tilde{J}$, respectively. The partition $\tilde{J}, \tilde{I}$ and $\tilde{K}$ which satisfies $\#\tilde{J} > \#\tilde{K}$.
Output: The sets of x-coordinates $x\tilde{I}$, $x\tilde{J}$ and $x\tilde{K}$.

$x2P \leftarrow \text{xDBL}(xP)$, $x\tilde{J}[0] \leftarrow xP$, $x\tilde{J}[1] \leftarrow x2P$, $i \leftarrow 2$
while $i < \tilde{b}$ **do**
  $x\tilde{J}[i] \leftarrow \text{xADD}(x\tilde{J}[i-1],\ xP,\ x\tilde{J}[i-2])$
  $x\tilde{J}[i+1] \leftarrow \text{xDBL}(x\tilde{J}[\lfloor i/2 \rfloor])$
  $i \leftarrow i+2$
end while
if $\tilde{b}\ \%\ 2 == 0$ **then**
  $x\tilde{J}[\tilde{b}] \leftarrow \text{xADD}(x\tilde{J}[\tilde{b}-1],\ xP,\ x\tilde{J}[\tilde{b}-2])$
end if
for i from 0 to $\frac{\ell-1}{2} - \tilde{b}'(2\tilde{b}+1) - 1$ **do**
  $x\tilde{K}[i] \leftarrow x\tilde{J}[i]$
end for
$x\tilde{I}[0] \leftarrow \text{xADD}(x\tilde{J}[\tilde{b}],\ x\tilde{J}[\frac{\ell-1}{2} - \tilde{b}'(2\tilde{b}+1) - 1],\ x\tilde{J}[\tilde{b}+1-\frac{\ell-1}{2} + \tilde{b}'(2\tilde{b}+1)])$
$Istep \leftarrow \text{xADD}(x\tilde{J}[\tilde{b}],\ x\tilde{J}[\tilde{b}-1],\ xP)$
$x\tilde{I}[1] \leftarrow \text{xADD}(Istep,\ x\tilde{I}[0],\ x\tilde{J}[\tilde{b} - \frac{\ell-1}{2} + \tilde{b}'(2b\tilde{b}+1)])$
for i from 2 to $\tilde{b}' - 1$ **do**
  $x\tilde{I}[i] \leftarrow \text{xADD}(x\tilde{I}[i-1],\ Istep,\ x\tilde{I}[i-2])$
end for
return $(x\tilde{I}, x\tilde{J}, x\tilde{K})$

In Algorithm 1, the function $\text{xADD}(x_P, x_Q, x_{P-Q})$ represents the x-only differential addition on Montgomery curves. The output of xADD is the x-coordinate of $P+Q$. The function $\text{xDBL}(x_P)$ returns the x-coordinate of $[2]P$.

We continue to take $\ell = 191$ and use the partition with respect to the tuple $(\tilde{b}, \tilde{b}') = (7, 6)$ for our new partition. We obtain that:

$$\tilde{I} = \{13, 28, \cdots, 88\},\ \tilde{J} = \{1, 2, \cdots, 7\},\ \tilde{K} = \{1, 2, \cdots, 5\}.$$

According to Algorithm 1, it takes three differential additions and three point doublings to obtain $x\tilde{J}$. After computing $x\tilde{J}$, we need to perform one more differential addition to obtain $x([8]P)$ which is used to compute the addition step of $x\tilde{I}$. Observing that $\tilde{K}$ is completely included in $\tilde{J}$, we do not need to redundantly compute $x\tilde{K}$. Finally, the first element of $x\tilde{I}$ can be obtained by $x\tilde{I}[0] = x\tilde{J}[5] + x\tilde{J}[6]$ and the addition step of $x\tilde{I}$ can be obtained by adding $x\tilde{J}[6]$ and $x([8]P)$. Thus, it needs seven differential additions to obtain $x\tilde{I}$. The

total computational cost is $84\boldsymbol{M}$. We can save $54\boldsymbol{M}$ and $24\boldsymbol{M}$ compared with the partition in [6] and our first partition respectively, when computing $x\tilde{I}$, $x\tilde{J}$ and $x\tilde{K}$. In conclusion, we consider the latter partition for implementation.

3.2 Optimizing the implementation of polynomial multiplications

In this subsection, we use the techniques of lazy reduction and generalized interleaved Montgomery multiplication [35] to speed up the computations of polynomial multiplications involved in evaluating ℓ-isogenies by improving the implementations of the sums of products.

Let $A(x) = \sum_{i=1}^{m} a_i x^i$ and $B(x) = \sum_{j=1}^{n} b_j x^j \in \mathbb{F}_p[x]$ be two polynomials of degree m and n. Let $C(x) = A(x)B(x) = \sum_{k=1}^{m+n} c_k x^k$ be the product of $A(x)$ and $B(x)$. The Karatsuba multiplication [32] is done by dividing the polynomials $A(x)$ (resp. $B(x)$) into two parts: $A_0 + A_1 x^{\lceil \frac{m}{2} \rceil}$ (resp. $B_0 + B_1 x^{\lceil \frac{n}{2} \rceil}$) and then using the equation: $A_0B_1 + A_1B_0 = (A_0 + A_1)(B_0 + B_1) - A_0B_0 - A_1B_1$ to obtain $A_0B_1 + A_1B_0$. Recurse the procedures above and recombine the coefficients, we finally obtain the product $C(x)$.

After repeatedly recursing the polynomials $A(x)$ and $B(x)$, we only need to deal with the product of two small polynomials in the final step. Let $m \times n$ represents the product of two polynomials of degrees m and n. Let $(a_0, a_1, \cdots, a_m)$ represents a degree-m polynomial A with coefficients $a_i, i = 0, \cdots, m$. Now we focus on the three products: $\mathbf{2 \times 1}$, $\mathbf{3 \times 1}$ and $\mathbf{2 \times 2}$ which are recursed down from the starting product C. Note that the sums of products $\sum_{i=0}^{1} a_i b_i$ and $\sum_{i=0}^{2} a_i b_i$ are required when constructing the above small products. In the current implementation of CTIDH, the technique of lazy reduction or generalized interleaved Montgomery multiplication has not yet been utilized to optimize the computation of the sums of products.

The "low", "middle" and "high" products of two polynomials can also be obtained by combining the Karatsuba multiplication with lazy reduction or generalized interleaved Montgomery multiplication like the general products.

Using lazy reduction. In the following, we explore how to combine lazy reduction and the Karatsuba multiplication [32] when performing the small products mentioned above in the last step of polynomial multiplications over $\mathbb{F}_p$.

Algorithm 2 illustrates the computational procedure of product $\mathbf{2 \times 1}$ using the Karatsuba multiplication and lazy reduction. The functions **fp-mul**(a, b) and **fp-rdcn**(a) are the integer multiplication and $\mathbb{F}_p$-reduction, respectively.

From Algorithm 2 we can see that it takes five integer multiplications and four $\mathbb{F}_p$-reductions to obtain the product using lazy reduction. If not, we have to perform five $\mathbb{F}_p$- multiplications. This is equal to a saving of one modular reduction each time we compute $\mathbf{2 \times 1}$. The situations of the products $\mathbf{3 \times 1}$ and $\mathbf{2 \times 2}$ are similar to $\mathbf{2 \times 1}$ thus we omit them for simplicity.

Using generalized interleaved Montgomery multiplication. Now we apply the generalized interleaved Montgomery multiplication [35] to speed up the computations of the sums of products $\sum_{i=0}^{t} a_i b_i$.

Algorithm 2 Product $\mathbf{2 \times 1}$: utilizing the Karatsuba multiplication and lazy reduction.

Input: A degree-2 polynomial $A = (a_0, a_1, a_2)$ and a degree-1 polynomial $B = (b_0, b_1)$.
Output: The product $C = A * B = (c_0, c_1, c_2, c_3)$.
1: $c_0 \leftarrow$ fp-muln$(a_0, b_0) = a_0 b_0$
2: $tmp_1 \leftarrow$ fp-muln$(a_0 + a_1, b_0 + b_1) = (a_0 + a_1)(b_0 + b_1)$
3: $tmp_2 \leftarrow$ fp-muln$(a_1, b_1) = a_1 b_1$
4: $c_1 \leftarrow tmp_1 - tmp_2 - c_0 = a_0 b_1 + a_1 b_0$
5: $c_2 \leftarrow$ fp-muln$(a_2, b_0) + tmp_2 = a_2 b_0 + a_1 b_1$
6: $c_3 \leftarrow$ fp-muln$(a_2, b_1) = a_2 b_1$
7: **for** i from 0 to 3 **do**
8: $\quad c_i \leftarrow$ fp-rdcn$(c_i, c_i) = c_i R^{-1} \mod p$
9: **end for**
10: **return** $C \leftarrow (c_0, c_1, c_2, c_3)$

We first recall the Montgomery multiplication and use the same notations as in Sec. 2.3. A straight implementation of Equation 2 requires the use of a huge number of GPRs (general purpose registers). Another approach which processes the recovery of one digit at a time by reducing with $r = 2^w$ at each iteration can balance the use of GPRs. It is called radix-r interleaved Montgomery multiplication [35].

The method above can be easily generalized to the sums of products [35], which is called generalized Montgomery multiplication. Algorithm 3 illustrates the computational procedure of $\mathbf{2 \times 2}$. Let **fp-mul**(a, b), **fp-sum2**(a, b) and **fp-sum3**(a, b) be functions that compute $abR^{-1} \mod p$, $\sum_{i=0}^{1} a_i b_i \cdot R^{-1}$ and $\sum_{i=0}^{2} a_i b_i \cdot R^{-1}$ utilizing (generalized) interleaved Montgomery multiplication, respectively.

Algorithm 3 Case $\mathbf{2 \times 2}$: computing the sums of products using interleaved Montgomery multiplications

Input: A degree-2 polynomial $A = (a_0, a_1, a_2)$ and a degree-1 polynomial $B = (b_0, b_1)$.
Output: The product $C = A * B = (c_0, c_1, c_2, c_3, c_4)$.
1: $c_0 \leftarrow$ fp-mul$(a_0, b_0) = a_0 b_0 R^{-1} \mod p$
2: $c_1 \leftarrow$ fp-sum2$(a_0, a_1, b_1, b_0) = (a_0 b_1 + a_1 b_0) R^{-1} \mod p$
3: $c_2 \leftarrow$ fp-sum3$(a_0, a_1, a_2, b_2, b_1, b_0) = (a_0 b_2 + a_1 b_1 + a_2 b_0) R^{-1} \mod p$
4: $c_3 \leftarrow$ fp-sum2$(a_1, a_2, b_2, b_1) = (a_1 b_2 + a_2 b_1) R^{-1} \mod p$
5: $c_4 \leftarrow$ fp-mul$(a_2, b_2) = a_2 b_2 R^{-1} \mod p$
6: **return** $C \leftarrow (c_0, c_1, c_2, c_3, c_4)$

Although using generalized interleaved Montgomery multiplication requires more $\mathbb{F}_p$-modular reductions and multiplications to perform than using lazy reduction, the great balance between the intermediate results and GPRs ensures the crucial savings of the use of memory. It is a trade-off between the underlying

instructions of multiplications and memory reads/writes. Note that Algorithm 3 directly calculates the sums of products modulo p which is incompatible with the procedures of lazy reduction. Hence we cannot combine the above two techniques together.

4 Cost analysis and implementation on large degree isogenies

In this section, we first compare the computational cost ($\mathbb{F}_p$ multiplication) between applying the previous partition in [6] and ours. The numbers of underlying instructions using lazy reduction and generalized Montgomery multiplication are also illustrated. Finally we present the clock cycles required for executing an ℓ-isogeny in our implementation. The technique of lazy reduction (resp. generalized Montgomery multiplication) is abbreviated as LZYR (resp. INTL) throughout the remaining part of this paper.

4.1 Efficiency comparison

From Section 3 we know that in practice the optimal partition is obtained by tuning the values of (b, b'), which are the cardinalities of $\tilde{J}$ and $\tilde{I}$, respectively. We use a tuning program to obtain the optimal partition for each ℓ-isogeny in CTIDH512. Figure 1 illustrates the cost associated with the ℓ-isogenies involved in CTIDH512, comparing the original partition [6] with ours. The horizontal and vertical axes represent the degree of isogenies ($3 \leq \ell \leq 587$) and the $\mathbb{F}_p$-multiplications divided by the square root of degree, respectively. We reduce the number of $\mathbb{F}_p$-multiplications for almost all ℓ-isogenies except $\ell = 353$ in CTIDH512 compared to the original partition. Especially for $\ell = 191$, the optimization is about 5.64%.

The tuning results also indicate that our partition can enhance the application of $\sqrt{\text{élu}}$. According to Figure 1 , $\sqrt{\text{élu}}$ performs better when $\ell \geq 83$ in the previous implementation of CTIDH512 [3]. However, by using our new partition, the condition is relaxed to $\ell \geq 73$.

Figure 2 presents the cost of instructions of multiplications and memory reads/writes when executing different prime degree ℓ-isogenies, respectively. The horizontal and vertical axes represent the degree of isogenies ($3 \leq \ell \leq 587$) and the number of assembly instructions divided by the square root of degree, respectively.

The average ratio of the number of multiplication instructions when computing the following ℓ-isogenies ($\ell > 89$) using LZYR to the previous work is 89.3%. When $\ell = 137$, we can achieve the maximum saving of multiplication instructions at about 15.75%. As for INTL, the average ratio of the number of memory reads/writes instructions to the previous work is 83.04%. When $\ell = 347$, the saving is up to 24.25%.

Fig. 1. Comparison of $\mathbb{F}_p$-multiplications for the ℓ-isogenies required in CTIDH512 between the previous work [6] and our new partition (the result has been tuned). The x-axis represents the degree of each isogeny.

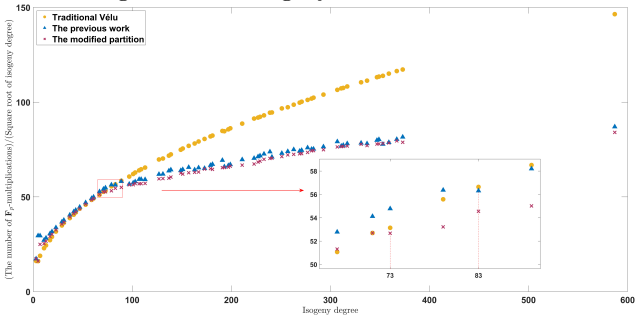
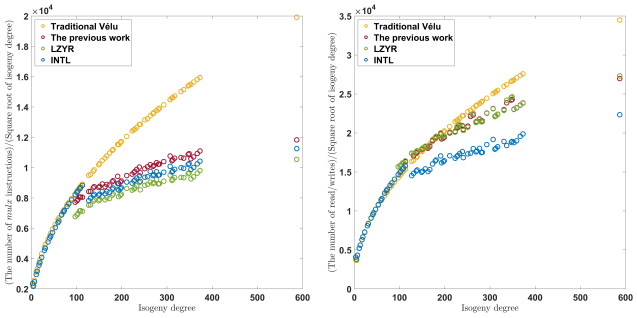


Fig. 2. Comparison of instruction counts between utilizing LZYR/INTL (Algorithm 2 and Algorithm 3) and the previous work [3] for computing each ℓ-isogeny over $\mathbb{F}_p$. The x-axis represents the degree of each isogeny.



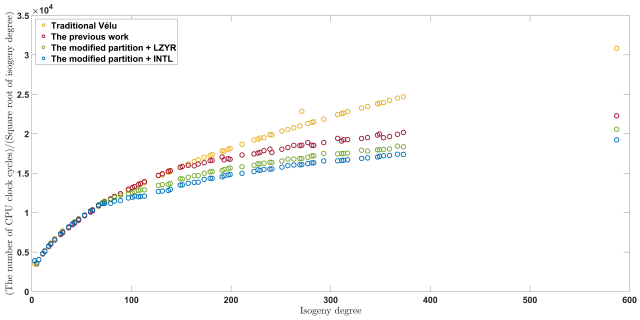
4.2 Improvement of √élu's Formula on CTIDH512

Based on the code provided in [3], we compile and benchmark our code on Intel(R) Core(TM) i9-12900K 3.20 GHz with TurboBoost and hyperthreading features disabled. Compilation was carried out using the command `clang -O3`. The version of clang is 11.3.0.

We combine the new partition with LZYR and INTL respectively for implementation. To make the data more reliable, we executed each ℓ-isogeny which needs to be computed in CTIDH512 for 1.6×10^3 times and took the average cycle counts. The performance results of CPU clock cycles for each ℓ-isogeny are presented in Figure 3. The horizontal and vertical axes represent the degree of isogenies ($3 \leq \ell \leq 587$) and CPU clock cycles divided by the square root of degree, respectively.

Compared with the previous work, when computing the large degree ℓ-isogenies, the performance of applying our new partition + LZYR is 4.44% −

Fig. 3. Performance comparison (in terms of clock cycles) between the proposed methods (Algorithm 1 + Algorithm 2 and Algorithm 1 + Algorithm 3) and the previous work [3] for computing each ℓ-isogeny over $\mathbb{F}_p$.



10.96% faster than that of the previous work. In the case of utilizing our new partition + INTL, we reduced the clock cycles by approximately $8.54\% - 16.05\%$.

5 Conclusion and Future Work

In this paper, we propose a new partition in the computational process of ℓ-isogenies to reduce the number of scalar multiplications. At the underlying level, we also apply two approaches to save the instructions of multiplications and memory reads/writes, which speed up the polynomial multiplications over finite field $\mathbb{F}_p$ in the process of $\sqrt{\text{élu}}$. To a certain extent, they impact the software implementation of cryptographic schemes including CSIDH/CTIDH, the CSIDH-based [21,7,2,25] and the quaternion-based [27,22,23,20] digital signature schemes. Furthermore, our new partition can also be extended to the public key encryption FESTA [5] and the key exchange protocol dCSIDH [10].

The future work can involve: finding a more efficient partition or even reduce the computational complexity of $\sqrt{\text{élu}}$, and studying the performance of the proposed method using school-book forms in combination with the AVX-512 vector instructions available in some Intel processors. Moreover, we aim to adapt our method for implementations on other software platforms, constrained devices and hardware platforms.

Acknowledgement

References

1. Adj, G., Chi-Domínguez, J.J., Rodríguez-Henríquez, F.: Karatsuba-based square-root Vélus formulas applied to two isogeny-based protocols. Journal of Cryptographic Engineering pp. 1–18 (2022)

2. Atapoor, S., Baghery, K., Cozzo, D., Pedersen, R.: CSI-SharK: CSI-Fish withăSharing-friendly Keys. In: Simpson, L., Rezazadeh Baee, M.A. (eds.) Information Security and Privacy. pp. 471–502. Springer Nature Switzerland, Cham (2023)
3. Banegas, G., Bernstein, D.J., Campos, F., Chou, T., Lange, T., Meyer, M., Smith, B., Sotáková, J.: CTIDH: faster constant-time CSIDH. IACR Transactions on Cryptographic Hardware and Embedded Systems **2021**(4), 351387 (Aug 2021)
4. Banegas, G., Gilchrist, V., Dévéhat, A.L., Smith, B.: Fast andăFrobenius: Rational Isogeny Evaluation overăFinite Fields. In: Aly, A., Tibouchi, M. (eds.) Progress in Cryptology – LATINCRYPT 2023. pp. 129–148. Springer Nature Switzerland, Cham (2023)
5. Basso, A., Maino, L., Pope, G.: FESTA: Fast Encryption fromăSupersingular Torsion Attacks. In: Guo, J., Steinfeld, R. (eds.) Advances in Cryptology – ASIACRYPT 2023. pp. 98–126. Springer Nature Singapore, Singapore (2023)
6. Bernstein, D.J., de Feo, L., Leroux, A., Smith, B.: Faster computation of isogenies of large prime degree. In: Galbraith, S. (ed.) ANTS-XIV - 14th Algorithmic Number Theory Symposium. Proceedings of the Fourteenth Algorithmic Number Theory Symposium (ANTS-XIV), vol. 4, pp. 39–55. Mathematical Sciences Publishers, Auckland, New Zealand (Jun 2020)
7. Beullens, W., Kleinjung, T., Vercauteren, F.: CSI-FiSh: Efficient Isogeny Based Signatures Through Class Group Computations. In: Galbraith, S.D., Moriai, S. (eds.) Advances in Cryptology – ASIACRYPT 2019. pp. 227–247. Springer International Publishing, Cham (2019)
8. Bonnetain, X., Schrottenloher, A.: Quantum Security Analysis of CSIDH. In: Canteaut, A., Ishai, Y. (eds.) Advances in Cryptology – EUROCRYPT 2020. pp. 493–522. Springer International Publishing, Cham (2020)
9. Campagna, M., Costello, C., Hess, B., Jalali, A., Koziel, B., LaMacchia, B., Longa, P., Naehrig, M., Renes, J., Urbanik, D., et al.: Supersingular isogeny key encapsulation (2019)
10. Campos, F., Chavez-Saab, J., Chi-Domínguez, J.J., Meyer, M., Reijnders, K., Rodríguez-Henríquez, F., Schwabe, P., Wiggers, T.: Optimizations and practicality of high-security csidh. Cryptology ePrint Archive, Paper 2023/793 (2023)
11. Cassels, J.W.S.: LMSST: 24 Lectures on Elliptic Curves. London Mathematical Society Student Texts, Cambridge University Press (1991)
12. Castryck, W., Decru, T.: An Efficient Key Recovery Attack onăSIDH. In: Hazay, C., Stam, M. (eds.) Advances in Cryptology – EUROCRYPT 2023. pp. 423–447. Springer Nature Switzerland, Cham (2023)
13. Castryck, W., Lange, T., Martindale, C., Panny, L., Renes, J.: CSIDH: An Efficient Post-Quantum Commutative Group Action. In: Peyrin, T., Galbraith, S. (eds.) Advances in Cryptology – ASIACRYPT 2018. pp. 395–427. Springer International Publishing, Cham (2018)
14. Chávez-Saab, J., Chi-Domínguez, J., Jaques, S., Rodríguez-Henríquez, F.: The SQALE of CSIDH: sublinear Vélu quantum-resistant isogeny action with low exponents. J. Cryptogr. Eng. **12**(3), 349–368 (2022)
15. Childs, A., Jao, D., Soukharev, V.: Constructing elliptic curve isogenies in quantum subexponential time. Journal of Mathematical Cryptology **8**(1), 1–29 (2014)
16. Costello, C.: B-SIDH: Supersingular Isogeny Diffie-Hellman Using Twisted Torsion. In: Moriai, S., Wang, H. (eds.) Advances in Cryptology – ASIACRYPT 2020. pp. 440–463. Springer International Publishing, Cham (2020)

17. Costello, C., Hisil, H.: A Simple and Compact Algorithm for SIDH with Arbitrary Degree Isogenies. In: Takagi, T., Peyrin, T. (eds.) Advances in Cryptology – ASIACRYPT 2017. pp. 303–329. Springer International Publishing, Cham (2017)
18. Costello, C., Longa, P., Naehrig, M.: Efficient Algorithms for Supersingular Isogeny Diffie-Hellman. In: Robshaw, M., Katz, J. (eds.) Advances in Cryptology – CRYPTO 2016. pp. 572–601. Springer Berlin Heidelberg, Berlin, Heidelberg (2016)
19. Couveignes, J.M.: Hard Homogeneous Spaces. Cryptology ePrint Archive, Paper 2006/291 (2006)
20. Dartois, P., Leroux, A., Robert, D., Wesolowski, B.: SQISignHD: New Dimensions in Cryptography. Cryptology ePrint Archive, Paper 2023/436 (2023)
21. De Feo, L., Galbraith, S.D.: SeaSign: Compact Isogeny Signatures from Class Group Actions. In: Ishai, Y., Rijmen, V. (eds.) Advances in Cryptology – EUROCRYPT 2019. pp. 759–789. Springer International Publishing, Cham (2019)
22. De Feo, L., Kohel, D., Leroux, A., Petit, C., Wesolowski, B.: SQISign: Compact Post-quantum Signatures from Quaternions and Isogenies. In: Moriai, S., Wang, H. (eds.) Advances in Cryptology – ASIACRYPT 2020. pp. 64–93. Springer International Publishing, Cham (2020)
23. De Feo, L., Leroux, A., Longa, P., Wesolowski, B.: New Algorithms forătheăDeuring Correspondence. In: Hazay, C., Stam, M. (eds.) Advances in Cryptology – EUROCRYPT 2023. pp. 659–690. Springer Nature Switzerland, Cham (2023)
24. De Feo, L., Delpech de Saint Guilhem, C., Fouotsa, T.B., Kutas, P., Leroux, A., Petit, C., Silva, J., Wesolowski, B.: Séta: Supersingular Encryption fromăTorsion Attacks. In: Tibouchi, M., Wang, H. (eds.) Advances in Cryptology – ASIACRYPT 2021. pp. 249–278. Springer International Publishing, Cham (2021)
25. Feo, L.D., Fouotsa, T.B., Kutas, P., Leroux, A., Merz, S.P., Panny, L., Wesolowski, B.: Scallop: Scaling theăcsi-fish. In: Boldyreva, A., Kolesnikov, V. (eds.) Public-Key Cryptography – PKC 2023. pp. 345–375. Springer Nature Switzerland, Cham (2023)
26. Galbraith, S.D.: Mathematics of Public Key Cryptography. Cambridge University Press, USA, 1st edn. (2012)
27. Galbraith, S.D., Petit, C., Silva, J.: Identification Protocols and Signature Schemes Based on Supersingular Isogeny Problems. In: Takagi, T., Peyrin, T. (eds.) Advances in Cryptology – ASIACRYPT 2017. pp. 3–33. Springer International Publishing, Cham (2017)
28. Hu, Z., Liu, Z., Wang, L., Zhou, Z.: Simplified isogeny formulas on twisted Jacobi quartic curves. Finite Fields and Their Applications **78**, 101981 (2022)
29. Hu, Z., Wang, L., Zhou, Z.: Isogeny Computation onăTwisted Jacobi Intersections. In: Deng, R., Bao, F., Wang, G., Shen, J., Ryan, M., Meng, m., Wang, D. (eds.) Information Security Practice and Experience. pp. 46–56. Springer International Publishing, Cham (2021)
30. Huang, Y., Jin, Y., Hu, Z., Zhang, F.: Optimizing the evaluation of -isogenous curve for isogeny-based cryptography. Information Processing Letters **178**, 106301 (2022)
31. Jao, D., De Feo, L.: Towards Quantum-Resistant Cryptosystems from Supersingular Elliptic Curve Isogenies. In: Yang, B.Y. (ed.) Post-Quantum Cryptography. pp. 19–34. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
32. Karatsuba, A.: Multiplication of multidigit numbers on automata. In: Soviet physics doklady. vol. 7, pp. 595–596 (1963)

33. Kieffer, J.: Étude et accélération du protocole déchange de clés de Couveignes–Rostovtsev–Stolbunov. Ph.D. thesis, Masters thesis, Inria Saclay & Université Paris VI (2017)
34. Kim, S.: Complete Analysis of Implementing Isogeny-Based Cryptography Using Huff Form of Elliptic Curves. IEEE Access **9**, 154500–154512 (2021)
35. Longa, P.: Efficient Algorithms for Large Prime Characteristic Fields and Their Application to Bilinear Pairings. IACR Transactions on Cryptographic Hardware and Embedded Systems **2023**(3), 445472 (Jun 2023)
36. Maino, L., Martindale, C., Panny, L., Pope, G., Wesolowski, B.: A Direct Key Recovery Attack onăSIDH. In: Hazay, C., Stam, M. (eds.) Advances in Cryptology – EUROCRYPT 2023. pp. 448–471. Springer Nature Switzerland, Cham (2023)
37. Meyer, M., Reith, S.: A Faster Way to the CSIDH. In: Chakraborty, D., Iwata, T. (eds.) Progress in Cryptology – INDOCRYPT 2018. pp. 137–152. Springer International Publishing, Cham (2018)
38. Montgomery, P.L.: Modular multiplication without trial division. Mathematics of Computation (Apr 1985)
39. Montgomery, P.L.: Speeding the Pollard and elliptic curve methods of factorization. Mathematics of computation **48**(177), 243–264 (1987)
40. Moody, D., Shumow, D.: Analogues of Vélus formulas for isogenies on alternate models of elliptic curves. Mathematics of Computation **85**(300), 1929–1951 (2016)
41. Moriya, T., Onuki, H., Takagi, T.: How to Construct CSIDH on Edwards Curves. Cryptology ePrint Archive, Paper 2019/843 (2019)
42. Moriya, T., Onuki, H., Takagi, T.: How to Construct CSIDH on Edwards Curves. In: Jarecki, S. (ed.) Topics in Cryptology – CT-RSA 2020. pp. 512–537. Springer International Publishing, Cham (2020)
43. Peikert, C.: He Gives C-Sieves on the CSIDH. In: Canteaut, A., Ishai, Y. (eds.) Advances in Cryptology – EUROCRYPT 2020. pp. 463–492. Springer International Publishing, Cham (2020)
44. Robert, D.: Breaking SIDH inăPolynomial Time. In: Hazay, C., Stam, M. (eds.) Advances in Cryptology – EUROCRYPT 2023. pp. 472–503. Springer Nature Switzerland, Cham (2023)
45. Rostovtsev, A., Stolbunov, A.: PUBLIC-KEY CRYPTOSYSTEM BASED ON ISOGENIES. Cryptology ePrint Archive, Paper 2006/145 (2006)
46. Scott, M.: Implementing cryptographic pairings. In: Proceedings of the First International Conference on Pairing-Based Cryptography. p. 177196. Pairing'07, Springer-Verlag, Berlin, Heidelberg (2007)
47. Tao, Z., Hu, Z., Zhou, Z.: Faster isogeny computation on twisted hessian curves. Applied Mathematics and Computation **444**, 127823 (2023)
48. Vélu, J.: Isogénies entre courbes elliptiques. CR Acad. Sci. Paris, Séries A **273**, 305–347 (1971)