Pairing Optimizations for Isogeny-based Cryptosystems

Shiping Cai[1], Kaizhan Lin[1], and Chang-An Zhao[⋆,1,2]

[1] School of Mathematics, Sun Yat-sen University, Guangzhou, Guangdong, China
caishp6@mail2.sysu.edu.cn
linkzh5@mail2.sysu.edu.cn
zhaochan3@mail.sysu.edu.cn
[2] Guangdong Key Laboratory of Information Security, Guangzhou, Guangdong, China

Abstract. In isogeny-based cryptography, bilinear pairings are regarded as a powerful tool in various applications, including key compression, public-key validation and torsion basis generation. However, in most isogeny-based protocols, the performance of pairing computations is unsatisfactory due to the high computational cost of the Miller function. Reducing the computational expense of the Miller function is crucial for enhancing the overall performance of pairing computations in isogeny-based cryptography.
This paper addresses this efficiency bottleneck. To achieve this, we propose several techniques for a better implementation of pairings in isogeny-based cryptosystems. We use (modified) Jacobian coordinates and present new algorithms for Miller function computations to compute pairings of order $2^\bullet$ and $3^\bullet$. For pairings of arbitrary order, which are crucial for key compression in some SIDH-based schemes (such as M-SIDH and bin-SIDH), we combine Miller doublings with Miller additions/subtractions, leading to a considerable speedup. Moreover, the optimizations for pairing applications in CSIDH-based protocols are also considered in this paper. In particular, our approach for supersingularity verification in CSIDH is 15.3% faster than Doliskani's test, which is the state-of-the-art.

Keywords: Pairing computations · Isogeny-based cryptography · Supersingularity verification · Torsion basis generation

1 Introduction

As one of the essential tools in elliptic curve cryptography, pairings are now widely applied in isogeny-based protocols. To compress the public key, pairings are considered to improve the performance of SIDH/SIKE [1]. Although SIDH and SIKE are broken by the attacks proposed in [9,28,37], one can still use

Authors are listed in alphabetical order.
⋆ Corresponding auther

pairings to improve the public-key compression in other SIDH-like schemes [25], such as the SIDH-PoK-based identification protocol [16], M-SIDH [20], binSIDH and terSIDH [4].

Recently, the implementations of isogeny-based signatures also involve pairing computations. In SQIsign [17,18], the ideal-to-isogeny translation is the most costly procedure. To enhance the performance, pairings are used to simplify the discrete logarithm computations on elliptic curves [27]. As the most compact signature, SQIsignHD [15] also applies pairings to reduce the signature size. Compared with SQIsign, SQIsignHD has a simpler signing phase, and pairing computations become one of the efficiency bottlenecks. A faster approach to compute pairings would make SQIsignHD more attractive in isogeny-based cryptography. Furthermore, pairings are also applied to isogeny-based public-key encryption protocols, such as FESTA [5] and QFESTA [33]. Besides, pairings have been considered as a powerful tool for supersingularity verification and full-torsion basis generation/verification in CSIDH [35].

However, pairing computations in most of isogeny-based schemes are not efficient now. In compressed SIDH/SIKE, one can use the precomputation technique proposed in [32] to accelerate the pairing computation significantly. But in most of isogeny-based schemes we mentioned above, the precomputation technique does not benefit the performance because of the costly dual isogeny computation. Therefore, it is important to explore how to compute pairings efficiently on a generic supersingular elliptic curve. For the case considered in CSIDH, the pairing-based supersingularity verification [35] is still slightly less efficient than the state-of-the-art [19,2], even with multiple techniques in the literature to speed up the implementation.

In this paper, we mainly consider how to efficiently compute the pairings in different isogeny-based protocols, including SIDH-based schemes (e.g., binSIDH), SQISignHD, FESTA, QFESTA, CSIDH and dCSIDH [8]. To be precise,

- We optimize pairing computations when the embedding degree $k = 1$. Firstly, a new formula is proposed to compute Miller tripling, which benefits the computation of $3^\bullet$-pairings. Besides, we reduce the cost of Miller line function evaluation for quadrupling in [27] to speed up $2^\bullet$-pairing computations. Compared with the pairing computation in projective coordinates [14], our new approach performs better in (modified) Jacobian coordinates. Moreover, we explore how to compute pairings of arbitrary order by proposing new formulas for Miller doubling-and-addition and Miller doubling-and-subtraction. These formulas improve the pairing performance in SIDH-like schemes, such as M-SIDH, binSIDH and terSIDH. Since our new formulas do not rely on specific underlying fields, they possess the potential to be applied to other protocols.
- We enhance the performance of pairing computations for the case of $k = 2$, which is adapted in CSIDH, dCSIDH, etc. Particularly, we revisit the pairing computation for torsion points verification, supersingularity verification and torsion basis generation in CSIDH [35]. Some technical details to improve the performance are also presented. The experimental results show that the

improvements bring a considerable speedup. It is worth noting that our algorithm yields a factor 1.2 acceleration for the pairing-based supersingularity verification, which beats Doliskani's test [2,19].

Related work. Very recently, independent work by Robert [38] proposed an elegant approach to compute pairings efficiently, which can also be used to accelerate the pairing computations in isogeny-based protocols. We note that our work optimizes the pairing implementation from a distinct perspective. It is possible that combining the techniques in both would lead to a better performance in applications.

The organization of this paper is as follows. We give the preliminaries of this paper in Section 2. In Sections 3 and 4, we present our main optimizations for pairing computation in isogeny-based cryptography when the embedding degree $k = 1$ and $k = 2$, respectively. Finally, we conclude in Section 5.

2 Preliminaries

This section provides the necessary background required for the remainder of the paper. We begin with a brief overview of elliptic curves and then recall the basic definition of pairings, including the Weil pairing and the reduced Tate pairing. Furthermore, we introduce Miller's algorithm.

Elliptic curves. Let $q = p^k$ where p is a prime. An elliptic curve E defined over a finite field $\mathbb{F}_q$ has the following form:

$$E/\mathbb{F}_q : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6,$$

where $a_1, a_2, a_3, a_4, a_6 \in \mathbb{F}_q$ such that E is non-singular. When $a_1 = a_3 = a_6 = 0$ and $a_4 = 1$, the curve is of the form $E_A : y^2 = x^3 + Ax^2 + x$. We call it a Montgomery curve. Setting $a_1 = a_2 = a_4 = 0$ gives a curve in the form $E_{a,b} : y^2 = x^3 + ax + b$, known as short Weierstrass curve. We denote the point at infinity of an elliptic curve E as ∞_E, or ∞ for simplicity when there is no ambiguity in the context. For a curve E defined over $\mathbb{F}_q$, all the rational points on E, including ∞_E, form an abelian group under point addition. We let $E(\mathbb{F}_q)$ denote the group. For a positive integer n, define

$$E[n] = \{P \in E(\overline{\mathbb{F}_p}) | [n]P = \infty\},$$
$$E(\mathbb{F}_q)[n] = \{P \in E(\mathbb{F}_q) | [n]P = \infty\}.$$

An curve E in characterstic p is called supersingular if $E[p] = \{\infty\}$. It is called ordinary otherwise.

Weil pairing. In 1940, André Weil first introduced the Weil pairing [40]. The efficient computation of Weil pairing was proposed by Miller [29]. Let E be an elliptic curve over the finite field $\mathbb{F}_p$ and r be an integer with $p \nmid r$. Choose points

$P, Q \in E[r]$, we can define a map

$$\omega_r : E[r] \times E[r] \to \mu_r$$
$$(P, Q) \mapsto (-1)^r \frac{f_{r,P}(Q)}{f_{r,Q}(P)},$$

where μ_r is the r-th root of unity. We call ω_r the Weil pairing. Here, $f_{r,P}$ and $f_{r,Q}$ are the Miller functions with $\mathrm{div}(f_{r,P}) = r(P) - r(\infty)$ and $\mathrm{div}(f_{r,Q}) = r(Q) - r(\infty)$.

Reduced Tate pairing. Based on the Weil pairing, the Tate pairing is proposed in the case that the second argument $Q \in E(\mathbb{F}_{p^k})/rE(\mathbb{F}_{p^k})$, and we only require the first argument P is in the r-torsion group. Let k be the embedding degree of E with respect to r, i.e., k is the smallest integer such that $r \mid p^k - 1$. Then, we can define the reduced Tate pairing τ_r as follows:

$$\tau_r : E(\mathbb{F}_{p^k})[r] \times E(\mathbb{F}_{p^k})/rE(\mathbb{F}_{p^k}) \to \mu_r$$
$$(P, Q) \mapsto f_{r,P}(Q)^{\frac{p^k-1}{r}}.$$

The computation of raising the power of $f_{r,P}(Q)$ to $\frac{p^k-1}{r}$ is called the final exponentiation, which maps the result of Miller function evaluation into a group formed by the r-th roots of unity in $\mathbb{F}_{p^k}^*$.

As stated above, the Weil pairing computation involves two Miller function computations, whereas the reduced Tate pairing computation contains one Miller function computation and the final exponentiation. In most isogeny-based cryptosystems, the final exponentiation typically requires less computational resources if the embedding degree is less than or equal to 2. Conversely, Miller function computation dominates the computational cost of the pairings. Compared with the Weil pairing computation, the reduced Tate pairing computation saves one Miller function computation. This is why the reduced Tate pairing is preferred in isogeny-based protocols.

Miller's algorithm. The computation of $f_{r,P}(Q)$ contains point operations and Miller function evaluation, which can be computed by Miller's algorithm [29] and the elliptic net algorithm [41]. Although the elliptic net algorithm has been improved significantly in recent years [12,7], it is still less efficient than Miller's algorithm. This paper mainly considers how to efficiently compute pairings by Miller's algorithm in isogeny-based schemes.

Non-Adjacent Form (NAF). In elliptic curve cryptography, the non-adjacent form [23](NAF) representation that is a binary signed-digit representation for an integer is applied to improve the scalar multiplication [31,22]. Compared with the binary digit representation, the NAF representation ensures that the integer has fewer non-zero digits. Ideally, the number of non-zero digits is about one-third of the length of the NAF representation. This enables a more efficient implementation of Miller's algorithm. By using the NAF representation, one can save a significant amount of Miller additions. To summarize, we show Miller's algorithm with the NAF form for the reduced Tate pairing in Algorithm 1.

Notation. In this paper, let $\boldsymbol{M}, \boldsymbol{S}, \boldsymbol{m}, \boldsymbol{s}, \boldsymbol{i}$ denote the costs of one multiplication, squaring in $\mathbb{F}_{p^2}$, one multiplication, one squaring and one inversion in

Algorithm 1 Miller's algorithm in NAF form

Require: $P \in E(\mathbb{F}_p)[r], Q \in E(\mathbb{F}_{p^k})/rE(\mathbb{F}_{p^k})$ and an integer $r = (r_{h-1}, r_{h-2}, \cdots, r_0)_2$ in NAF form.
Ensure: $f_{r,P}(Q)^{\frac{p^k-1}{r}}$.
1: $T \leftarrow P$, $f \leftarrow 1$;
2: **for** i **from** $h-2$ **to** 0 **do**
3: $f \leftarrow f^2 \cdot \frac{\ell_{T,T}}{v_{[2]T}}(Q)$;
4: $T \leftarrow [2]T$;
5: **if** $r_i = 1$ **then**
6: $f \leftarrow f \cdot \frac{\ell_{T,P}}{v_{T+P}}(Q)$;
7: $T \leftarrow T + P$;
8: **else if** $r_i = -1$ **then**
9: $f \leftarrow f \cdot \frac{\ell_{T,-P}}{v_{T-P}}(Q)$;
10: $T \leftarrow T - P$;
11: **end if**
12: **end for**
13: $f \leftarrow f^{\frac{p^k-1}{r}}$;
14: **return** f.

$\mathbb{F}_p$, respectively. We estimate that $1\boldsymbol{M} \approx 3\boldsymbol{m}$, $1\boldsymbol{S} \approx 2\boldsymbol{m}$, $1\boldsymbol{s} \approx 0.8\boldsymbol{m}$ and $1\boldsymbol{i} \approx 30\boldsymbol{m}$. For simplicity, the operations of field additions/subtractions are not counted which are much cheaper compared with field multiplications and squarings. We denote the line passing through the points P and Q by $L_{P,Q}$, and the notation $\ell_{P,Q}$ is the line function that defines $L_{P,Q}$. Similarly, denote the vertical line passing through P by v_P, and define v_P as its line function. When there is no ambiguity in the context, we use L_{k_1,k_2}, ℓ_{k_1,k_2}, V_{k_1}, v_{k_1} to represent $L_{[k_1]P,[k_2]P}$, $\ell_{[k_1]P,[k_2]P}$, $V_{[k_1]P}$, $v_{[k_1]P}$ respectively. Besides, let $f_{k,P}$ be a rational function with $\mathrm{div}(f_{k,P}) = k(P) - ([k]P) - (k-1)(\infty)$. Finally, we denote $P = (x_P, y_P)$ in affine coordinates, $P = (X_P : Y_P : Z_P)$ in Jacobian coordinates and $P = (X_P : Y_P : Z_P : T_P)$ in modified Jacobian coordinates, where $x_P = X_P/Z_P^2$, $y_P = Y_P/Z_P^3$ and $T_P = aZ_P^4$.

3 Efficient Pairing Computation for Embedding Degree 1

This section considers pairing computations when the embedding degree is equal to one, which is widely used in isogeny-based schemes. For instance, one can utilize pairings to transfer elliptic curve discrete logarithm computation to discrete logarithm computation over the finite field in compressed SIDH-like schemes. In addition, pairings are utilized to check the relative sign for two points in FESTA. These require the pairing e satisfies that $e(P, P) = 1$ for every point $P \in E(\mathbb{F}_{p^2})$, where E is a supersingular elliptic curve defined over $\mathbb{F}_{p^2} = \mathbb{F}_p[i]/\langle i^2 + 1\rangle$. with $p \equiv 3 \bmod 4$. The Weil pairing always satisfies this property, while the reduced Tate pairing satisfies it under specific situations, as stated in Theorem 1.

Theorem 1 ([25,27]). *Let E be a supersingular elliptic curve defined over $\mathbb{F}_{p^2}$ with $p \equiv 3 \bmod 4$. Suppose that $P \in E(\mathbb{F}_{p^2})[N]$ and $N \mid p+1$. Then $\tau_N(P,P) = 1$ for every $P \in E(\mathbb{F}_{p^2})[N]$ if*

- *N is odd, or*
- *N is a power of 2.*

As we analyzed in Section 2, computing the reduced Tate pairing is more efficient than computing the Weil pairing in general. Indeed, we can adapt the reduced Tate pairing in most isogeny-based protocols since we often encounter the scenarios described in Theorem 1 in practice.

In this section, we will first describe how to efficiently compute the reduced Tate pairings of degree 2^{e_2} and 3^{e_3}, where $e_2, e_3 \in \mathbb{N}$. After that, we will consider pairings of arbitrary order, which can be used to improve compressed SIDH-like schemes.

3.1 Pairing computation of degree 2^{e_2} and 3^{e_3}

In isogeny-based cryptography, elliptic curves in Montgomery form are widely used. Adapting the isomorphism

$$\begin{aligned} \phi : E_A &\to E_{a,b}, \\ (x, y) &\mapsto (x + \frac{A}{3}, y), \end{aligned} \tag{1}$$

we can efficiently translate a Montgomery curve $E_A : y^2 = x^3 + Ax^2 + x$ to a Weierstrass curve $E_{a,b} : y^2 = x^3 + ax + b$, where

$$a = 1 - \frac{A^2}{3} \text{ and } b = -\frac{A}{3} + \frac{2A^3}{27}.$$

For a better performance, we mainly consider computing the reduced Tate pairing $\tau_r(P,Q)$ on the supersingular elliptic curve $E_{a,b} : y^2 = x^3 + ax + b$, where $P, Q \in E_{a,b}(\mathbb{F}_{p^2})$ are represented in affine coordinates.

Miller function computation consists of point operations and Miller function evaluation. For example, Miller doubling involves one point doubling and Miller evaluation for doubling. Generally, Miller's algorithm only considers Miller doubling and Miller addition. While in [14,26,27], the authors proposed Miller tripling and Miller quadrupling, improving the pairing performance in the specific setting of isogeny-based protocols. Based on these works, we explore how to further optimize Miller tripling and Miller quadrupling.

We first consider the case when the reduced Tate pairing of degree 3^{e_3}, i.e., given points $P, Q \in E_{a,b}(\mathbb{F}_{p^2})$, construct the Miller function $f_{3^{e_3},P}$ with $\mathrm{div}(f_{3^{e_3},P}) = 3^{e_3}(P) - 3^{e_3}(\infty)$ and evaluate it at Q. In this case, one can decompose the Miller function computation into multiple Miller triplings. Suppose that $R = [m]P$, where m is an integer. Let λ_1 and λ_2 be the slopes of the lines $L_{m,m}$ and $L_{-m,-2m}$. The values of λ_1 and λ_2 can be easily obtained during the

computations of $[2m]P \leftarrow [2]([m]P)$ and $[3m]P \leftarrow [2m]P + [m]P$. In [14], the authors evaluate the Miller function for tripling with the following formula:

$$\mathrm{div}(f_{3m,P}) = \mathrm{div}\left(f_{m,P}^3 \frac{\ell_{m,m} \cdot \ell_{m,2m}}{v_{2m} \cdot v_{3m}}\right). \tag{2}$$

Applying Equation (2), one needs to evaluate four Miller line functions. Here we propose Equation (3), a more efficient formula for Miller function evaluation in Miller tripling:

$$\begin{aligned} \mathrm{div}(f_{3m,P}) &= \mathrm{div}\left(f_{m,P}^3 \frac{\ell_{m,m} \cdot v_m}{\ell_{-m,-2m}}\right) \\ &= \mathrm{div}\left(f_{m,P}^3 \frac{[\lambda_1(x - x_{2m}) - (y + y_{2m})] \cdot (x - x_m)}{\lambda_2(x - x_{2m}) - (y + y_{2m})}\right). \end{aligned} \tag{3}$$

Proposition 1. *Equation* (3) *is correct.*

Proof. Since $\mathrm{div}(f_{m,P}) = m(P) - ([m]P) - (m-1)(\infty)$ and $\mathrm{div}(f_{3m,P}) = 3m(P) - ([3m]P) - (3m-1)(\infty)$, we have $\mathrm{div}(f_{3m,P}/f_{m,P}^3) = 3([m]P) - ([3m]P) - 2(\infty)$. On the other hand, from

$$\begin{aligned} & 3([m]P) - ([3m]P) - 2(\infty) \\ =& (2([m]P) + ([-2m]P) - 3(\infty)) + (([m]P) + ([-m]P) - 2(\infty)) \\ & - (([-m]P) + ([-2m]P) + ([3m]P) - 3(\infty)) \\ =& \mathrm{div}(\ell_{m,m}) + \mathrm{div}(v_m) - \mathrm{div}(\ell_{-m,-2m}) \\ =& \mathrm{div}\left(\frac{\ell_{m,m} \cdot v_m}{\ell_{-m,-2m}}\right), \end{aligned}$$

we can imply that the first equality in Equation (3) holds.

Note that $L_{m,m}$ not only passes through $[m]P$ but also $[-2m]P$. It follows that the second equality in Equation (3) holds, which completes the proof. ■

Equation (3) only requires three Miller line function evaluations, thereby saving one line evaluation compared to the previous work (Equation (2)). In addition, we explore the computational efficiency of various coordinates and find out that using Jacobian coordinates is the best choice for efficiency. We present Algorithm 2, which executes point operation and line evaluation for tripling at a cost of $19\boldsymbol{M} + 11\boldsymbol{S}$.

Algorithm 2 TPL: Miller tripling

Input: Miller function $f = f_{m,P}(Q)$, points $R = [m]P = (X_R : Y_R : Z_R)$, $Q = (x_Q, y_Q)$ and the coefficient a of $E_{a,b} : y^2 = x^3 + ax + b$.
Output: $[3]R = (X_3 : Y_3 : Z_3)$ and $f_{3m,R}(Q)$.

1: $XX \leftarrow X_R^2$
2: $YY \leftarrow Y_R^2$
3: $ZZ \leftarrow Z_R^2$
4: $Y_4 \leftarrow YY^2$
5: $T_0 \leftarrow ZZ^2$
6: $T_1 \leftarrow a \cdot T_0$
7: $M \leftarrow T_1 + 3 \cdot XX$
8: $MM \leftarrow M^2$
9: $T_2 \leftarrow 6 \cdot ((X_R + YY)^2 - XX - Y_4)$
10: $E \leftarrow T_2 - MM$
11: $EE \leftarrow E^2$
12: $T \leftarrow 16 \cdot Y_4$
13: $T_3 \leftarrow (M + E)^2 - EE - MM$
14: $U \leftarrow T_3 - T$
15: $T_4 \leftarrow 4 \cdot YY \cdot U$
16: $X_3 \leftarrow 4 \cdot (X_R \cdot EE - T_4)$
17: $T_5 \leftarrow T - U$
18: $T_6 \leftarrow E \cdot EE$
19: $T_7 \leftarrow U \cdot T_5$
20: $Y_3 \leftarrow 8 \cdot Y_R \cdot (T_7 - T_6)$
21: $Z_3 \leftarrow (Z_R + E)^2 - ZZ - EE$
22: $U \leftarrow U - T_3/2$
23: $T_1 \leftarrow ZZ \cdot x_Q - X_R$
24: $T_2 \leftarrow E \cdot T_1$
25: $T_3 \leftarrow M \cdot T_1$
26: $T_4 \leftarrow Y_R \cdot ZZ$
27: $T_5 \leftarrow T_4 \cdot Z_R$
28: $T_6 \leftarrow y_Q \cdot T_5$
29: $T_7 \leftarrow T_2 \cdot (T_3 - 2 \cdot T_6 + 2 \cdot YY)$
30: $T_8 \leftarrow E \cdot (T_6 + YY)$
31: $T_1 \leftarrow U \cdot T_1 - 2 \cdot T_8$
32: $T_2 \leftarrow ZZ \cdot T_1$
33: $f \leftarrow f^3 \cdot T_7 \cdot \overline{T_2}$
34: **return** $(X_3 : Y_3 : Z_3)$, f

$Cost : 19\boldsymbol{M} + 11\boldsymbol{S}$

Compared with the previous work [14], our approach saves $10\boldsymbol{M} - 3\boldsymbol{S} \approx 24\boldsymbol{m}$. Overall, it offers an approximate 23% speedup.

Remark 1. In Miller function evaluation, all inversion operations can be replaced by conjugate operations which is easy to compute. Furthermore, the final exponentiation step involves raising $f_{r,P}(Q) \in \mathbb{F}_{p^2}$ to a power divided by $p-1$. Hence, for every element $\alpha + \beta i \in \mu_{p+1} = \{x \in \mathbb{F}_{p^2} | x^{p+1} = 1\}$ with $\alpha, \beta \in \mathbb{F}_p$, we have

$$(\alpha + \beta i)^{-1} = (\alpha + \beta i)^p = \alpha - \beta i.$$

Moreover, we can omit the inversion of an element defined on $\mathbb{F}_p$ as raising it to the power $p - 1$ is equal to 1. This technique, so-called denominator elimination [3], is widely applied for efficient pairing computations.

Next, we explore the pairing computation of degree 2^{e_2}. In [26,27], the authors merged two Miller doublings into one Miller quadrupling, and used Equation (4) to evaluate Miller functions for quadrupling:

$$\begin{aligned} \mathrm{div}(f_{4m,P}) &= \mathrm{div}\left(\frac{\left[f_{m,P}^2 \cdot \ell_{m,m}\right]^2}{\ell_{-2m,-2m}}\right) \\ &= \mathrm{div}\left(\frac{\left[f_{m,P}^2[\lambda_1(x - x_{2m}) - (y + y_{2m})]\right]^2}{\lambda_2(x - x_{2m}) - (y + y_{2m})}\right), \end{aligned} \tag{4}$$

where λ_1 and λ_2 are the slopes of the lines $L_{m,m}$ and $L_{-2m,-2m}$, respectively. The values of λ_1 and λ_2 can be easily obtained by $[2m]P \leftarrow [2]([m]P)$ and $[4m]P \leftarrow [2]([2m]P)$. We also use Equation (4) to compute Miller quadrupling. However, compared with the previous work [27] we save $1\boldsymbol{M}$ in each loop of Miller quadrupling, according to Algorithm 3.

Algorithm 3 QDL: Miller quadrupling

Input: Miller function $f = f_{m,P}(Q)$, points $R = [m]P = (X_R : Y_R : Z_R : T_R)$, $Q = (x_Q, y_Q)$ and the coefficient a of $E_{a,b} : y^2 = x^3 + ax + b$.
Output: $[4]R = (X_4 : Y_4 : Z_{R_4} : T_{R_4})$ and $f_{4m,R}(Q)$.

1: $XX \leftarrow X_R^2$
2: $T_1 \leftarrow 2 \cdot Y_R^2$
3: $T_2 \leftarrow T_1^2$
4: $T_3 \leftarrow (X_R + T_1)^2 - XX - T_2$
5: $T_4 \leftarrow 2 \cdot T_2$
6: $\lambda_1 \leftarrow 3 \cdot XX + T_R$
7: $X_2 \leftarrow \lambda_1^2 - 2 \cdot T_3$
8: $Y_2 \leftarrow \lambda_1 \cdot (T_3 - X_2) - T_4$
9: $Z_2 \leftarrow 2 \cdot Y_R \cdot Z_R$
10: $T_{R_2} \leftarrow 2 \cdot T_4 \cdot T_R$
11: $XX \leftarrow X_2^2$
12: $T_1 \leftarrow 2 \cdot Y_2^2$
13: $T_2 \leftarrow T_1^2$
14: $T_3 \leftarrow (X_2 + T_1)^2 - XX - T_2$
15: $T_4 \leftarrow 2 \cdot T_2$
16: $\lambda_2 \leftarrow 3 \cdot XX + T_{R_2}$
17: $X_4 \leftarrow \lambda_2^2 - 2 \cdot T_3$
18: $Y_4 \leftarrow \lambda_2 \cdot (T_3 - X_4) - T_4$
19: $Z_4 \leftarrow 2 \cdot Y_2 \cdot Z_2$
20: $T_{R_4} \leftarrow 2 \cdot T_4 \cdot T_{R_2}$
21: $ZZ \leftarrow Z_2^2$
22: $T_0 \leftarrow Z_2 \cdot ZZ$
23: $f \leftarrow f^2$
24: $T_1 \leftarrow ZZ \cdot x_Q - X_2$
25: $T_2 \leftarrow y_Q \cdot T_0 + Y_2$
26: $T_3 \leftarrow T_1 \cdot \lambda_1 - T_2$
27: $f \leftarrow f \cdot T_3$
28: $f \leftarrow f^2$
29: $f \leftarrow f \cdot Y_2$
30: $T_1 \leftarrow T_1 \cdot \lambda_2$
31: $T_3 \leftarrow 2 \cdot T_2 \cdot Y_2$
32: $T_1 \leftarrow (T_1 + T_3) \cdot T_0$
33: $f \leftarrow f \cdot \overline{T_1}$
34: **return** $(X_4 : Y_4 : Z_4 : T_{R_4})$, f
$Cost : 16\boldsymbol{M} + 13\boldsymbol{S}$

3.2 Pairing computation in a generic case

In this subsection, we intend to speed up pairing computations for generic cases, i.e., computing the reduced Tate pairing of arbitrary order efficiently. The techniques in this subsection can be adapted to improve pairing computations in M-SIDH, binSIDH, etc.

In each Miller loop, Miller doublings are always executed in the original Miller's algorithm. When a Miller addition is required, the original Miller's algorithm handles the Miller doubling step and the Miller addition step individually. Our main idea of improving the performance is to combine doubling with addition (subtraction) to evaluate the Miller function for doubling and addition (subtraction) at the same time. More precisely, given a point $[m]P$ and the Miller line evaluation $f_{m,P}(Q)$ from the previous step, we can directly evaluate

$f_{2m+1,P}$ ($f_{2m-1,P}$) at Q with Equation (5) (Equation (6)). That is,

$$\begin{aligned}\mathrm{div}(f_{2m+1,P}) &= \mathrm{div}\left(f_{m,P}^2 \frac{\ell_{m,m}\cdot v_1}{\ell_{-2m,-1}}\right)\\ &= \mathrm{div}\left(f_{m,P}^2 \frac{[\lambda_1(x-x_{2m})-(y+y_{2m})]\cdot(x-x_1)}{\lambda_2(x-x_{2m})-(y+y_{2m})}\right),\end{aligned} \tag{5}$$

where λ_1 and λ_2 are the slopes of the lines $L_{m,m}$ and $L_{-2m,-1}$, respectively, and

$$\begin{aligned}\mathrm{div}(f_{2m-1,P}) &= \mathrm{div}\left(f_{m,P}^2 \frac{\ell_{m,m}}{\ell_{-2m,1}}\right)\\ &= \mathrm{div}\left(f_{m,P}^2 \frac{\lambda_1(x-x_{2m})-(y+y_{2m})}{\lambda_2(x-x_{2m})-(y+y_{2m})}\right).\end{aligned} \tag{6}$$

where λ_1 and λ_2 are the slopes of the lines $L_{m,m}$ and $L_{-2m,1}$, respectively. Analogous to the cases in Section 3.1, during the computation of $[2m]P \leftarrow [2]([m]P)$ and $[2m+1]P \leftarrow [2m]P + P$ ($[2m-1]P \leftarrow [2m]P - P$), it is easy to obtain the slopes of the lines $L_{m,m}$ and $L_{-1,-2m}$ ($L_{-2m,1}$).

Proposition 2. *Equations* (5) *and* (6) *are correct.*

Proof. Since $\mathrm{div}(f_{m,P}) = m(P) - ([m]P) - (m-1)(\infty)$, and $\mathrm{div}(f_{2m+1,P}) = (2m+1)(P) - ([2m+1]P) - 2m(\infty)$, we have

$$\mathrm{div}(f_{2m+1,P}/f_{m,P}^2) = (P) + 2([m]P) - ([2m+1]P) - 2(\infty).$$

On the other hand, from

$$\begin{aligned}&(P) + 2([m]P) - ([2m+1]P) - 2(\infty)\\ =&(2([m]P) + ([-2m]P) - 3(\infty)) + ((P) + (-P) - 2(\infty))\\ &- (([-2m]P) + (-P) + ([2m+1]P) - 3(\infty))\\ =&\mathrm{div}(\ell_{m,m}) + \mathrm{div}(v_1) - \mathrm{div}(\ell_{-2m,-1})\\ =&\mathrm{div}\left(\frac{\ell_{m,m}\cdot v_1}{\ell_{-2m,-1}}\right),\end{aligned}$$

we can imply the first equality in Equation (5) holds. Similarly, from the divisors of $f_{m,P}$ and $f_{2m-1,P}$, one can deduce that $\mathrm{div}(f_{2m-1,P}/f_{m,P}^2) = 2([m]P) - (P) - ([2m-1]P)$. It follows that

$$\begin{aligned}&2([m]P) - (P) - ([2m-1]P)\\ =&(2([m]P) + ([-2m]P) - 3(\infty)) - (([-2m]P) + (P) + ([2m-1]P) - 3(\infty))\\ =&\mathrm{div}(\ell_{m,m}) - \mathrm{div}(\ell_{-2m,1})\\ =&\mathrm{div}\left(\frac{\ell_{m,m}}{\ell_{-2m,-1}}\right),\end{aligned}$$

and thus the first equality of Equation (6) holds. Since $L_{m,m}$ passes through $[-2m]P$, it is easy to see that the second equality in Equation (5) and that in Equation (6) hold as well. This completes the proof. ■

Remark 2. If the x-coordinates of P and Q are in $\mathbb{F}_p$, then Equation (5) can be simplified as

$$\mathrm{div}(f_{2m+1,P}) = \mathrm{div}\left(f_{m,P}^2 \frac{\lambda_1(x - x_{2m}) - (y + y_{2m})}{\lambda_2(x - x_{2m}) - (y + y_{2m})}\right).$$

due to the final exponentiation $\frac{p^2-1}{r} = (p-1) \cdot \frac{p+1}{r}$.

Moreover, to obtain an optimized implementation, we explore pairing computations in Jacobian coordinates and modified Jacobian coordinates, respectively. We present Table 1 to give an efficiency comparison. It shows that point doubling using modified Jacobian coordinates is more efficient, but point addition, evaluation for Miller doubling and evaluation for Miller quadrupling are more expensive. In general, the original Miller's algorithm is superior in performance when using Jacobian coordinates. However, we use the NAF representation of r to compute pairings, which typically leads to a sparse representation of r, i.e., the NAF form of r has few non-zero bits. Hence, Miller doublings dominate the computations. In this case, we prefer using modified Jacobian coordinates to compute the reduced Tate pairings.

In summary, we present Algorithm 4 to compute pairings for a generic case. Note that we also perform one Miller quadrupling (Algorithm 3) instead of two consecutive Miller doublings to further improve the performance. The algorithms for Miller doubling (**DBL**), Miller doubling-and-addtion (**DBLADD**) and Miller doubling-and-subtration (**DBLSUB**) can be seen in Appendix A.

Procedures	Jacobian	Modified Jacobian
Point doubling	$2\boldsymbol{M} + 8\boldsymbol{S}$	$3\boldsymbol{M} + 5\boldsymbol{S}$
Point addition	$7\boldsymbol{M} + 4\boldsymbol{S}$	$8\boldsymbol{M} + 6\boldsymbol{S}$
Evaluation for Miller doubling	$7\boldsymbol{M} + 1\boldsymbol{S}$	$7\boldsymbol{M} + 2\boldsymbol{S}$
Evaluation for Miller doubling-and-addition	$9\boldsymbol{M} + 1\boldsymbol{S}$	$9\boldsymbol{M} + 1\boldsymbol{S}$
Evaluation for Miller doubling-and-subtraction	$8\boldsymbol{M} + 1\boldsymbol{S}$	$8\boldsymbol{M} + 1\boldsymbol{S}$
Evaluation for Miller quadrupling	$10\boldsymbol{M} + 2\boldsymbol{S}$	$10\boldsymbol{M} + 3\boldsymbol{S}$

Table 1: Comparison of computational costs in (Modified) Jacobian coordinates for procedures of pairing computations

Remark 3. According to [11, Corollary 1], we can compute $\tau_{c \cdot r}(P, Q)$ instead of $\tau_r(P, Q)$, where c is an integer coprime to r. Therefore, we can choose a small integer c such that $c \cdot r$ has lower Hamming weight in NAF form compared with that of r in NAF form. Although it slightly increases the number of iterations, the practical performance would be better as it may save considerable Miller additions.

Algorithm 4 Pairing computation for a generic case

Require: Points $P = (x_P, y_P)$, $Q = (x_Q, y_Q)$ of order $r = (r_{h-1}r_{h-2}\cdots r_0)$ in NAF form, the curve coefficient a of $E_{a,b} : y^2 = x^3 + ax + b$.

Ensure: $\tau_r(P,Q) = f_{r,P}(Q)^{\frac{p^2-1}{r}}$.

$f \leftarrow 1$, $i \leftarrow h-1$, $T \leftarrow P$. // Transform P into modified Jacobian coordinates.
while $i \neq 0$ **do**
  if $i \neq 1$ and $r_i = 1$ **then**
    $(T, f) \leftarrow \mathbf{DBLADD}(Q, T, P, f)$. // $T \leftarrow [2]T + P$
    $i \leftarrow i - 1$.
  else if $i \neq 1$ and $r_i = -1$ **then**
    $(T, f) \leftarrow \mathbf{DBLSUB}(Q, T, P, f)$. // $T \leftarrow [2]T - P$
    $i \leftarrow i - 1$.
  else if $i \neq 2$ and $r_i = 0$ and $r_{i-1} = 0$ **then**
    $(T, f) \leftarrow \mathbf{QDL}(Q, T, f)$. // $T \leftarrow [4]T$
    $i \leftarrow i - 2$.
  else
    $(T, f) \leftarrow \mathbf{DBL}(Q, T, f)$. $T \leftarrow [2]T$
    $i \leftarrow i - 1$.
  end if
end while
if $r_0 = 1$ **then**
  $f \leftarrow f \cdot (x_Q - x_P)$.
else if $r_0 = 0$ **then**
  $f \leftarrow f^2 \cdot (x_Q - x_T)$.
end if
$f \leftarrow f^{\frac{p^2-1}{r}}$.
return f.

4 Efficient Pairing Computation for Embedding Degree 2

The pairing computation for $k = 2$ is mainly considered in CSIDH and its variants. Reijnders [35] presented a general idea to adapt pairings to improve the performance of full torsion basis verification, supersingularity verification and torsion basis generation. Some techniques are also utilized to speed up the implementation. In this section, we will propose several techniques to further improve the performance, and give an efficiency comparison between our methods and the previous work.

Assume that $p = 4\ell_1\ell_2\cdots\ell_n - 1$, where ℓ_j $(1 \leq j \leq n)$ are primes with $\ell_1 < \ell_2 < \cdots < \ell_n$. Let $\mathbb{F}_{p^2} = \mathbb{F}_p[i]/\langle i^2 + 1\rangle$. Same as what we handled in Section 3, we use the isomorphism in Equation (1) to transfer the points defined on the Montgomery curve to the Weierstrass curve and consider the computation of $\tau_r(P,Q)$, where

$$P \in \mathbb{G}_{1,r} = \{(x,y) \in E_{a,b}[r] | x, y \in \mathbb{F}_p\},$$
$$Q \in \mathbb{G}_{2,r} = \{(x,yi) \in E_{a,b}[r] | x, y \in \mathbb{F}_p\}.$$

All the pairings in this section have the property that the order r divides $p+1$, i.e., r is a product of small odd primes.

4.1 Torsion basis verification

In dCSIDH [8], the full torsion basis is a part of the public key. Hence, it is vulnerable to side-channel attacks without the verification of a full torsion basis. In this subsection, we focus on the following problem:

Problem 1. Given a supersingular elliptic curve $E_A/\mathbb{F}_p$ and two points $P \in \mathbb{G}_{1,r}$ and $Q \in \mathbb{G}_{2,r}$ where $r = (p+1)/4$, check that P and Q have full order.

It is easy to see that P and Q have order $r = (p+1)/4$ if and only if the value $\tau_r(P,Q)$ is a generator of μ_r. Therefore, one can compute $\tau_r(P,Q)$ and then verify that it is indeed a generator of μ_r using a divide-and-conquer approach [42, Algorithm 7.3]. Fixing a (public) primitive root ζ_0 of μ_{p+1}, there exists $\lambda \in \mathbb{Z}^*_{p+1}$ such that $\tau_r(P,Q) = \zeta_0^\lambda$. When $[\lambda^{-1}]Q$ is known, one can directly verify $\tau_r(P,[\lambda^{-1}]Q) = \zeta_0$ to avoid the order verification. In this case, we can regard $[\lambda^{-1}]Q$ as the original Q [35, Section 4.1]. However, the efficiency bottleneck of torsion basis verification is the pairing computation, especially Miller function evaluation.

Since the embedding degree is 2, one can use denominator elimination (Remark 1) to reduce the computational cost. In addition, some techniques we proposed in Section 3 can be applied to improve the performance. For example, the previous work [35] used projective coordinates to compute the Miller function [14], but adapting other coordinates to compute pairings will be more efficient. Table 2 provides cost estimates for executing Miller doubling (**DBL**) and Miller addition (**ADD**) using different coordinates. Similar to the case in Section 3, the doubling operations in modified Jacobian coordinates is the most efficient one. As for Miller addition, using Jacobian coordinates is the best choice. Similar to what we did in Section 3.2, merging the Miller doubling and Miller

Coordinates	Operation	$\boldsymbol{m}$	$\boldsymbol{s}$	Total ($\boldsymbol{m}$)
Projective [14,35]	**DBL**	15	5	19
	ADD	20	4	23.2
Jacobian	**DBL**	11	8	17.4
	ADD	13	4	16.2
Modified Jacobian	**DBL**	12	6	16.8
	ADD	14	6	18.8

Table 2: Comparison of **DBL** and **ADD** computational costs in different coordinates. We estimate that $1\boldsymbol{s} \approx 0.8\boldsymbol{m}$.

addition/subtraction can further reduce the computational cost. Note that the trick in Remark 2 can be applied in this case. Besides, one can also execute one Miller quadrupling instead of two consecutive Miller doublings. For comparison,

Table 3 compares the cost of Miller operations involved in Algorithm 4 in Jacobian coordinates and modified Jacobian coordinates, respectively. According to our experimental results, computing $\tau_r(P, Q)$ in modified Jacobian coordinates is more efficient.

It should be noted that the technique in Remark 3 does not work well in this case. This is because $r = (p+1)/4$ is divisible by a number of small primes. Same as [25,35], checking the order of the pairing value in μ_r can be done efficiently with the help of Lucas sequences [36].

	Jacobian	Modified Jacobian
DBLADD	$2\boldsymbol{M} + 1\boldsymbol{S} + 15\boldsymbol{m} + 12\boldsymbol{s}$	$2\boldsymbol{M} + 1\boldsymbol{S} + 16\boldsymbol{m} + 11\boldsymbol{s}$
DBLSUB	$2\boldsymbol{M} + 1\boldsymbol{S} + 15\boldsymbol{m} + 12\boldsymbol{s}$	$2\boldsymbol{M} + 1\boldsymbol{S} + 16\boldsymbol{m} + 11\boldsymbol{s}$
QDL	$2\boldsymbol{M} + 2\boldsymbol{S} + 11\boldsymbol{m} + 16\boldsymbol{s}$	$2\boldsymbol{M} + 2\boldsymbol{S} + 13\boldsymbol{m} + 11\boldsymbol{s}$
DBL	$1\boldsymbol{M} + 1\boldsymbol{S} + 6\boldsymbol{m} + 9\boldsymbol{s}$	$1\boldsymbol{M} + 1\boldsymbol{S} + 7\boldsymbol{m} + 6\boldsymbol{s}$

Table 3: Computational costs of Miller operations in Algorithm 4 for computing $\tau_r(P, Q)$.

4.2 Supersingularity verification

For long-term security, it is crucial to validate the public key in the implementations of CSIDH and its variants. In this subsection, we address the following problem:

Problem 2. Given an elliptic curve E_A defined over $\mathbb{F}_p$, verify that E_A is supersingular.

To solve this problem, a direct approach is to randomly select a point P and check its order $\mathrm{ord}(P)$ satisfies that $\mathrm{ord}(P) > 4\sqrt{p}$ and $\mathrm{ord}(P) \mid p + 1$[10]. Reijnders [35, Section 4.2] suggested to randomly select $P = (x_P, y_P)$ and $Q = (x_Q, iy_Q)$ with $x_P, y_P, x_Q, y_Q \in \mathbb{F}_p$, and then check whether the pairing value $\tau_r([4]P, Q)$ has order larger than $4\sqrt{p}$ in the group μ_{p+1}, where $r = (p + 1)/4$. If the order of P divides $p + 1$, then this verification confirms that the order of P is larger than $4\sqrt{p}$, and thus the elliptic curve is supersingular. One can also compute the order of the pairing value $\tau_N(P', Q)$, to confirm $\mathrm{ord}(P) > 4\sqrt{p}$, where N is slightly larger than $4\sqrt{p}$ and $P' = [(p + 1)/N]P$. According to the estimate in [35], the latter approach for verification is more efficient as the Miller loop is shorter. Since $E_A(\mathbb{F}_p)[\ell_j] \cong \mathbb{Z}/\ell_j\mathbb{Z}$, $j = 1, 2, \cdots, n$, the probability that $\ell_j \mid \mathrm{ord}(P)$ for a random point $P \in E_A[p + 1]$ is $1/\ell_j$. Hence, it is best to set $N = \ell_s\ell_{s+1} \cdots \ell_n$ with $N > 4\sqrt{p}$.

However, the pairing-based verification proposed in [35, Algorithm 5] only verifies that the order of $\tau_N(P', Q)$ is larger than $4\sqrt{p}$. As we stated above, one must also check $\mathrm{ord}(P)$ divides $p+1$. The previous work suggested to verify that the order of the pairing value divides $p+1$, to imply the curve is supersingular [35, Remark 5]. However, this method does not work when adapting the reduced Tate pairing because of the final exponentiation, i.e., the value is always in μ_{p+1}.

Therefore, the verification is incomplete: given an ordinary elliptic curve E, the orders of the points P' and Q may not divide $p+1$. Consequently, using Miller's algorithm to compute $\tau_N(P', Q)$ does not make sense.

Therefore, the algorithm should be modified with an additional verification that $[p+1]P = \infty$. At first glance, we need to perform an extra scalar multiplication. Fortunately, at the last iteration of Miller's algorithm, we obtain the coordinates of $[N-1]P'$ if the least significant bit of the NAF form of N is 1, or $[N+1]P'$ if the least significant bit of the NAF form of N is -1. Therefore, the condition that $[p+1]P = \infty$ can be efficiently checked by verifying

$$\begin{cases} [N-1]P' = P', \text{if the least significant bit of the NAF form of } N \text{ is 1,} \\ [N+1]P' = -P', \text{otherwise.} \end{cases} \tag{7}$$

All the optimizations mentioned in Section 4.1 can be still applied to improve the performance. As N does not have a low Hamming weight in general and it is not divisible by the small odd primes $\ell_j \mid p+1$ with $j = 1, 2, \cdots, s-1$, we can adapt the technique in Remark 3 to optimize Miller function evaluation.

Besides, before computing $\tau_N(P', Q)$, we need to compute $P' = [(p+1)/N]P$ first. A naive way is to use the Montgomery ladder [30] to compute the x-coordinate of P' and then recover the y-coordinate by computing the square roots of $x_{P'}^3 + Ax_{P'}^2 + x_{P'}$. Note that the Montgomery ladder not only outputs the x-coordinate of P', but also that of $[(p+1)/N+1]P = P' + P$. Thanks to Okeya-Sakurai formula [34], we have

$$y_{P'} = \frac{(x_P x_{P'} + 1)(x_{P'} + x_P + 2A) - 2A - (x_{P'} - x_P)^2 x_{P+P'}}{2y_P}.$$

Therefore, one can compute the y-coordinate of P' efficiently, instead of computing the square roots of $x_{P'}^3 + Ax_{P'}^2 + x_{P'}$.

4.3 Torsion basis generation

To speed up the performance of key agreement in dCSIDH, the public key involves the full torsion basis, but torsion basis generation overheads the key generation phase. Besides, for the constant-time implementation of group actions in CSIDH-based protocols (such as SQALE [13]), computing full torsion points $P \in \mathbb{G}_{1,r}$ and $Q \in \mathbb{G}_{2,r}$ with $r = (p+1)/4$ in advance is necessary. In this subsection, we target the following problem:

Problem 3. Given an elliptic curve E_A defined over $\mathbb{F}_p$, generate $P \in \mathbb{G}_{1,r}$ and $Q \in \mathbb{G}_{2,r}$ such that $\langle P, Q \rangle = E_A[r]$, where $r = (p+1)/4$.

In [35], the author proposed an efficient algorithm to generate a full torsion basis. However, it may fail with low but nonnegligible probability. Inspired by [25], we propose Algorithm 5 to generate the torsion basis efficiently. It should be noted that our algorithm always proceeds successfully. In the following, we describe how the algorithm works in detail.

Line 1: Generate two points $P \in \mathbb{G}_{1,r}$ and $Q \in \mathbb{G}_{2,r}$ using Elligator [6].
Lines 2-4: Compute $v = \tau_r(P, Q)$ and its order in μ_r, then we have $I_v = \{j | v^{(p+1)/\ell_j} = 1\}$. For each $j \in I_v$, one can deduce that $[(p+1)/\ell_j]P = \infty$ or $[(p+1)/\ell_j]Q = \infty$. Conversely, it follows that $\ell_j \mid \mathrm{ord}(P)$ and $\ell_j \mid \mathrm{ord}(Q)$ for $j \notin I_v$. Therefore, we have $I_P \subset I_v$ and $I_Q \subset I_v$, where $I_P = \{j | \ell_j \nmid \mathrm{ord}(P)\}$ and $I_Q = \{j | \ell_j \nmid \mathrm{ord}(Q)\}$.
Lines 5-7: This step aims to determine I_P and I_Q. Line 5 computes $P_v = [\prod_{j \notin I_v}]P$ and $Q_v = [\prod_{j \notin I_v}]Q$. It is obvious that $\ell_j \nmid \mathrm{ord}(P_v)$ (resp. $\ell_j \nmid \mathrm{ord}(Q_v)$) if and only if $\ell_j \nmid \mathrm{ord}(P)$ (resp. $\ell_j \nmid \mathrm{ord}(Q)$) for $j \in I_v$. Therefore, we compute $\mathrm{ord}(P_v)$ and $\mathrm{ord}(Q_v)$ using a divide-and-conquer approach in Line 6, and then we obtain I_P and I_Q, as shown in Line 7.
Lines 8-15: Line 8 checks if I_P is empty or not. If I_P is empty, then we can deduce that $P \in \mathbb{G}_{1,r}$ has full order r. Given $I_P \neq \emptyset$, Lines 9-10 generate a new point P' whose order divides $\prod_{j \in I_P} \ell_j$. If P' is not the point at infinity, then we set $P \leftarrow P + P'$ and modify the corresponding set I_P. Meanwhile, we execute $P_v \leftarrow P_v + P'$, which will be useful for generating the full order point in $\mathbb{G}_{2,r}$. We repeat this procedure until P has full order r, i.e., $I_P = \emptyset$. Simultaneously, we also obtain P_v, a point of order $\prod_{j \in I_v} \ell_j$.
Lines 16-26: Line 16 checks if I_Q is empty or not. If $I_Q = \emptyset$, then Q is a full-order point of $\mathbb{G}_{2,r}$, and thus we have generated a full torsion basis successfully. Otherwise, compute $P_Q = [u]P_v$ with $u = \prod_{j \in I_v \setminus I_Q} \ell_j$. It is clear that P_Q has order $\prod_{j \in I_Q} \ell_j$. After that, we generate another point $Q' \in \mathbb{G}_{2,p+1}$ using Elligator and compute the pairing $v' = \tau_{u'}(P_Q, Q')$ where $u' = \prod_{j \in I_Q} \ell_j$. We repeat generating Q' until v' has order u', i.e., the point $[(p+1)/u']Q'$ has order u'. This implies that $Q + [(p+1)/u']Q'$ has full order in $\mathbb{G}_{2,r}$.
Line 27: Output P and Q.

Remark 4. When generating a full order point in $\mathbb{G}_{1,r}$, we may perform several point additions in Line 13 to update P until $I_P = \emptyset$. Conversely, when $I_Q \neq \emptyset$ we just find one point Q' such that $v' = \tau_{u'}(P_Q, Q')$ is a generator of $\mu_{u'}$, and then perform one point addition (Line 23) to generate the full order point in $\mathbb{G}_{2,r}$. This is because updating Q is a costly step: before updating Q, we have to perform an expensive scalar multiplication $Q' \leftarrow [(p+1)/u']Q'$ first. According to our experiments, our strategy to generate the full order point Q is faster than that to update Q and I_Q frequently.

It is easy to see that the algorithm is correct. Except for the technique mentioned in Remark 3, all the other techniques we mentioned Sections 4.1 and 4.2 benefit Algorithm 5. For a faster implementation, we also utilize the following tricks to save the computational cost:

- Note that the Elligator technique is able to generate points $P \in \mathbb{G}_{1,p+1}$ and $Q \in \mathbb{G}_{2,p+1}$ simultaneously. Hence, when generating P' (in Line 9), we also store the points which are defined on $G_{2,p+1}$. In Line 19, we can just use these points instead of generating Q' by Elligator.

Algorithm 5 New torsion basis generation

Require: Integer $r = \frac{p+1}{4}$ and the coefficient A of the curve $E_A : y^2 = x^3 + Ax^2 + x$.
Ensure: Two points $P \in \mathbb{G}_{1,r}$ and $Q \in \mathbb{G}_{2,r}$ such that $\langle P, Q\rangle = E_A[r]$.
1: Generate $P \in \mathbb{G}_{1,r}$ and $Q \in \mathbb{G}_{2,r}$.
2: $v \leftarrow \tau_r(P, Q)$.
3: Compute the order of v in μ_{p+1}.
4: $I_v \leftarrow \{j | v^{(p+1)/\ell_j} = 1\}$.
5: $u \leftarrow \prod_{j \notin I_v} \ell_j$, $P_v \leftarrow [u]P$, $Q_v \leftarrow [u]Q$.
6: Compute $\mathrm{ord}(P_v)$ and $\mathrm{ord}(Q_v)$.
7: $I_P \leftarrow \{j | \ell_j \nmid \mathrm{ord}(P_v)\}$, $I_Q \leftarrow \{j | \ell_j \nmid \mathrm{ord}(Q_v)\}$.
8: **while** $I_P \neq \emptyset$ **do**
9: Generate $P' \in \mathbb{G}_{1,r}$.
10: $u \leftarrow \prod_{j \notin I_P} \ell_j$, $P' \leftarrow [u]P'$.
11: **if** $P' \neq \infty$ **then**
12: Compute $\mathrm{ord}(P')$.
13: $P \leftarrow P + P'$, $P_v \leftarrow P_v + P'$, $I_P \leftarrow I_P \backslash \{j | \ell_j$ divides $\mathrm{ord}(P')\}$.
14: **end if**
15: **end while**
16: **if** $I_Q \neq \emptyset$ **then**
17: $u \leftarrow \prod_{j \in I_v \backslash I_Q} \ell_j$, $P_Q \leftarrow [u]P_v$, $label \leftarrow 0$.
18: **while** $label = 0$ **do**
19: Generate $Q' \in \mathbb{G}_{2,p+1}$.
20: $v' \leftarrow \tau_{u'}(P_Q, Q')$ where $u' = \prod_{j \in I_Q} \ell_j$.
21: Compute the order of v' in μ_{p+1} (denoted by $\mathrm{ord}(v')$).
22: **if** $\mathrm{ord}(v') = u'$ **then**
23: $Q' \leftarrow [(p+1)/u']Q'$, $Q \leftarrow Q + Q'$, $label \leftarrow 1$.
24: **end if**
25: **end while**
26: **end if**
27: **return** P, Q.

– Since the first argument of $\tau_{u'}(P_Q, Q')$ (Line 20) is fixed inside the while loop (Lines 18-25), it is feasible to store information that solely depends on P_Q when computing v' for the first time. It benefits the performance when we have to evaluate the pairings (Line 20) multiple times. As u' is generally small, using this technique requires relatively low memory.

There are still some techniques in the literature that have the potential to optimize Algorithm 5. For example, when u' (Line 20) is a prime, checking the order of the value $v' = f_{u',P_Q}(Q')^{(p^2-1)/u'}$ (Line 21) in μ_{p+1} is equivalent to checking whether $f_{u',P_Q}(Q')^{p+1}$ is a u'-th power residue modulo p. When $u' \leq 11$, computing the u'-th power residue symbol is cheaper than executing the hard part of the final exponentiation [24]. In [21], Galbraith and Lin proposed algorithms to compute compressed pairing using x-only coordinates. Compared with the traditional approach to compute pairings, the x-only pairing computation is more expensive, and the gap will widen as the order of the pairing increases. However,

there is no need to recover the y-coordinate of Q' when just computing $\mathrm{ord}(v')$ (Line 21). In the case that $\mathrm{ord}(v')$ is not equal to u', one may save the square root computation when executing Elligator. Therefore, it is possible that computing v' (Line 21) with x-only coordinates would be more efficient. We leave these explorations as future works.

4.4 Experimental Results

To give an efficiency comparison between the previous work and ours, we implement our algorithms in SageMath. The code is available at:

`https://github.com/LinKaizhan/Pairingoptimizations`

As mentioned in Section 2, we neglect field additions/substractions and only count field multiplications and squarings. Furthermore, we also count field inversions. Since it is unnecessary to execute the algorithms in constant time, the field inversion is relatively efficient.

As shown in Table 4, computing pairings in modified Jacobian coordinates is more competitive compared to that in Jacobian coordinates. The experimental results indicate that we not only optimize torsion basis verification but also supersingularity verification in CSIDH. It should be noted that pairing-based supersingularity verification in [35] is slightly less efficient than Doliskani's test [19,2]. With our optimizations, pairing-based supersingularity verification is 15.3% faster than Doliskani's test.

Procedures	Coordinates	Field operations					Total($\boldsymbol{m}$)	[35]	Speedup
		$\boldsymbol{M}$	$\boldsymbol{S}$	$\boldsymbol{m}$	$\boldsymbol{s}$	$\boldsymbol{i}$			
Torsion basis verification	Jacobian	683	518	21235	17842	1	17395	19109	9.0%
	Modified Jacobian	683	509	8299	7093	1	17140		10.3%
Torsion basis verification (given $[\lambda^{-1}]Q$)	Jacobian	683	509	4516	4773	1	11501	13248	13.2%
	Modified Jacobian	683	509	5025	3819	1	11247		15.1%
Supersingularity verification	Jacobian	352	273	5652	5622	3	11842	14230	16.8%
	Modified Jacobian	352	273	5925	5088	3	11687		17.9%
Torsion basis generation	Jacobian	694	517	21222	17928	8	38920	-	-
	Modified Jacobian	694	517	21912	16983	8	38614		-

Table 4: Efficiency comparison of different pairing applications in CSIDH between [35] and this work. The performance of torsion basis generation is not in constant time, hence we execute 10^4 times and record the average cost. We estimate that $1\boldsymbol{M} \approx 3\boldsymbol{m}$, $1\boldsymbol{S} \approx 2\boldsymbol{m}$, $1\boldsymbol{s} \approx 0.8\boldsymbol{m}$ and $1\boldsymbol{i} \approx 30\boldsymbol{m}$.

5 Conclusion

In this paper, we utilized various techniques for pairing computations in isogeny-based cryptography. We presented novel algorithms for computing pairings in

(modified) Jacobian coordinates. These algorithms improve the implementations of SIDH-like schemes and SQIsignHD. Furthermore, we optimized pairing computations used in CSIDH and targeted three applications: full torsion basis verification, supersingularity verification and torsion basis generation. To enhance the performance, we developed new algorithms that achieved speedups of 10.3%, 15.1% and 17.9%, respectively. As a future work, we aim to implement these optimizations in Rust or C, leveraging technical skills such as lazy reduction [39] to achieve even greater performance improvements.

Acknowledgments

This work is supported by Guangdong Major Project of Basic and Applied Basic Research (No. 2019B030302008).

References

1. Azarderakhsh, R., Jao, D., Kalach, K., Koziel, B., Leonardi, C.: Key Compression for Isogeny-Based Cryptosystems. In: Proceedings of the 3rd ACM International Workshop on ASIA Public-Key Cryptography. pp. 1–10 (2016)
2. Banegas, G., Gilchrist, V., Smith, B.: Efficient supersingularity testing over GF(p) and CSIDH key validation. Mathematical Cryptology **2**(1), 21–35 (2022)
3. Barreto, P.S.L.M., Kim, H.Y., Lynn, B., Scott, M.: Efficient Algorithms for Pairing-Based Cryptosystems. In: Yung, M. (ed.) Advances in Cryptology — CRYPTO 2002. pp. 354–369. Springer Berlin Heidelberg, Berlin, Heidelberg (2002)
4. Basso, A., Fouotsa, T.B.: New SIDH Countermeasures for a More Efficient Key Exchange. In: Guo, J., Steinfeld, R. (eds.) Advances in Cryptology – ASIACRYPT 2023. pp. 208–233. Springer Nature Singapore, Singapore (2023)
5. Basso, A., Maino, L., Pope, G.: FESTA: Fast Encryption from Supersingular 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., Hamburg, M., Krasnova, A., Lange, T.: Elligator: Elliptic-curve points indistinguishable from uniform random strings. In: Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security. p. 967–980 (2013)
7. Cai, S., Hu, Z., Yao, Z.A., Zhao, C.A.: The elliptic net algorithm revisited. Journal of Cryptographic Engineering (Nov 2022)
8. 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), `https://eprint.iacr.org/2023/793`
9. Castryck, W., Decru, T.: An Efficient Key Recovery Attack on SIDH. In: Hazay, C., Stam, M. (eds.) Advances in Cryptology – EUROCRYPT 2023. pp. 423–447. Springer Nature Switzerland, Cham (2023)
10. Castryck, W., Lange, T., Martindale, C., Panny, L., Renes, J.: CSIDH: an efficient post-quantum commutative group action. In: Advances in Cryptology–ASIACRYPT 2018: 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2–6, 2018, Proceedings, Part III 24. pp. 395–427. Springer (2018)

11. Chang-An, Z., Fangguo, Z., Jiwu, H.: All Pairings Are in a Group. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences **E91.A**(10), 3084–3087 (2008)
12. Chen, B., Zhao, C.A.: An Improvement of the Elliptic Net Algorithm. IEEE Transactions on Computers **65**(9), 2903–2909 (2016)
13. Chávez-Saab, J., Chi-Domínguez, J.J., Jaques, S., Rodríguez-Henríquez, F.: The SQALE of CSIDH: sublinear Vélu quantum-resistant isogeny action with low exponents. Journal of Cryptographic Engineering **12**(3), 349–368 (Sep 2022)
14. Costello, C., Jao, D., Longa, P., Naehrig, M., Renes, J., Urbanik, D.: Efficient Compression of SIDH Public Keys. In: Coron, J.S., Nielsen, J.B. (eds.) Advances in Cryptology – EUROCRYPT 2017. pp. 679–706. Springer International Publishing, Cham (2017)
15. Dartois, P., Leroux, A., Robert, D., Wesolowski, B.: SQISignHD: New Dimensions in Cryptography. Cryptology ePrint Archive, Paper 2023/436 (2023), `https://eprint.iacr.org/2023/436`
16. De Feo, L., Dobson, S., Galbraith, S.D., Zobernig, L.: SIDH proof of knowledge. In: Advances in Cryptology–ASIACRYPT 2022: 28th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, December 5–9, 2022, Proceedings, Part II. pp. 310–339. Springer (2023)
17. 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)
18. De Feo, L., Leroux, A., Longa, P., Wesolowski, B.: New Algorithms for the Deuring Correspondence: Towards Practical and Secure SQISign Signatures. In: Hazay, C., Stam, M. (eds.) Advances in Cryptology – EUROCRYPT 2023. pp. 659–690. Springer Nature Switzerland, Cham (2023)
19. Doliskani, J.: On division polynomial PIT and supersingularity. Applicable Algebra in Engineering, Communication and Computing **29**(5), 393–407 (Nov 2018)
20. Fouotsa, T.B., Moriya, T., Petit, C.: M-SIDH and MD-SIDH: Countering SIDH Attacks by Masking Information. In: Hazay, C., Stam, M. (eds.) Advances in Cryptology – EUROCRYPT 2023. pp. 282–309. Springer Nature Switzerland, Cham (2023)
21. Galbraith, S.D., Lin, X.: Computing pairings using x-coordinates only. Designs, Codes and Cryptography **50**(3), 305–324 (Mar 2009)
22. Hitchcock, Y., Montague, P.: A New Elliptic Curve Scalar Multiplication Algorithm to Resist Simple Power Analysis. In: Batten, L., Seberry, J. (eds.) Information Security and Privacy. pp. 214–225. Springer Berlin Heidelberg, Berlin, Heidelberg (2002)
23. I. Blake, G. Seroussi, N.S.: Elliptic Curves in Cryptography, London Mathematical Society Lecture Note Series, vol. 265. Cambridge University Press (1999)
24. Joye, M., Lapiha, O., Nguyen, K., Naccache, D.: The Eleventh Power Residue Symbol. Journal of Mathematical Cryptology **15**(1), 111–122 (2021)
25. Lin, K., Lin, J., Cai, S., Wang, W., Zhao, C.A.: Compressed M-SIDH: an instance of compressed SIDH-like schemes with isogenies of highly composite degrees. Designs, Codes and Cryptography (Mar 2024), `https://doi.org/10.1007/s10623-024-01368-z`
26. Lin, K., Lin, J., Wang, W., Zhao, C.A.: Faster Public-key Compression of SIDH with Less Memory. IEEE Transactions on Computers pp. 1–9 (2023)

27. Lin, K., Wang, W., Xu, Z., Zhao, C.A.: A Faster Software Implementation of SQISign. Cryptology ePrint Archive, Paper 2023/753 (2023), https://eprint.iacr.org/2023/753
28. Maino, L., Martindale, C., Panny, L., Pope, G., Wesolowski, B.: A Direct Key Recovery Attack on SIDH. In: Hazay, C., Stam, M. (eds.) Advances in Cryptology – EUROCRYPT 2023. pp. 448–471. Springer Nature Switzerland, Cham (2023)
29. Miller, V.S.: The Weil Pairing, and Its Efficient Calculation. Journal of Cryptology **17**(4), 235–261 (2004)
30. Montgomery, P.L.: Speeding the Pollard and elliptic curve methods of factorization. Mathematics of Computation **48**, 243–264 (1987)
31. Morain, F., Olivos, J.: Speeding up the computations on an elliptic curve using addition-subtraction chains. ITA **24**, 531–544 (1990)
32. Naehrig, M., Renes, J.: Dual Isogenies and Their Application to Public-Key Compression for Isogeny-Based Cryptography. In: Galbraith, S.D., Moriai, S. (eds.) Advances in Cryptology – ASIACRYPT 2019. pp. 243–272. Springer International Publishing, Cham (2019)
33. Nakagawa, K., Onuki, H.: QFESTA: Efficient Algorithms and Parameters for FESTA using Quaternion Algebras. Cryptology ePrint Archive, Paper 2023/1468 (2023), https://eprint.iacr.org/2023/1468
34. Okeya, K., Sakurai, K.: Efficient Elliptic Curve Cryptosystems from a Scalar Multiplication Algorithm with Recovery of the y-coordinate on a Montgomery-Form Elliptic Curve. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) Cryptographic Hardware and Embedded Systems — CHES 2001. pp. 126–141. Springer Berlin Heidelberg, Berlin, Heidelberg (2001)
35. Reijnders, K.: Effective Pairings in Isogeny-Based Cryptography. In: Aly, A., Tibouchi, M. (eds.) Progress in Cryptology – LATINCRYPT 2023. pp. 109–128. Springer Nature Switzerland, Cham (2023)
36. Richard Crandall, C.B.P.: Prime numbers: a computational perspective. Springer, 2nd ed edn. (2005)
37. Robert, D.: Breaking SIDH in Polynomial Time. In: Hazay, C., Stam, M. (eds.) Advances in Cryptology – EUROCRYPT 2023. pp. 472–503. Springer Nature Switzerland, Cham (2023)
38. Robert, D.: Fast pairings via biextensions and cubical arithmetic. Cryptology ePrint Archive, Paper 2024/517 (2024), https://eprint.iacr.org/2024/517
39. Scott, M.: Implementing Cryptographic Pairings. In: Takagi, T., Okamoto, T., Okamoto, E., Okamoto, T. (eds.) Pairing-Based Cryptography – Pairing 2007. pp. 177–196. Springer Berlin Heidelberg, Berlin, Heidelberg (2007)
40. Silverman, J.H.: The arithmetic of elliptic curves, vol. 106. Graduate Texts in Mathematics. Springer-Verlag, New York (2009)
41. Stange, K.E.: The Tate Pairing Via Elliptic Nets. In: Takagi, T., Okamoto, T., Okamoto, E., Okamoto, T. (eds.) Pairing-Based Cryptography – Pairing 2007. pp. 329–348. Springer Berlin Heidelberg, Berlin, Heidelberg (2007)
42. Sutherland, A.V.: Order computations in generic groups (2007), https://math.mit.edu/~drew/sutherland-phd.pdf

Appendix A Sub-algorithms in Algorithm 4

Algorithm 6 DBL: Miller doubling

Input: Miller function $f = f_{m,P}(Q)$, points $R = [m]P = (X_R : Y_R : Z_R : T_R)$, $Q = (x_Q, y_Q)$ and the coefficient a of $E_{a,b} : y^2 = x^3 + ax + b$.
Output: $[2]R = (X_2 : Y_2 : Z_2 : T_{R_2})$ and $f_{2m,P}(Q)$.

1: $XX \leftarrow X_R^2$
2: $T_1 \leftarrow 2 \cdot Y_R^2$
3: $T_2 \leftarrow T_1^2$
4: $T_3 \leftarrow (X_R + T_1)^2 - XX - T_2$
5: $T_4 \leftarrow 2 \cdot T_2$
6: $\lambda \leftarrow 3 \cdot XX + T_R$
7: $X_2 \leftarrow \lambda^2 - 2 \cdot T_3$
8: $Y_2 \leftarrow \lambda \cdot (T_3 - X_2) - T_4$
9: $Z_2 \leftarrow 2 \cdot Y_R \cdot Z_R$
10: $T_{R_2} \leftarrow 2 \cdot T_4 \cdot T_R$
11: $ZZ \leftarrow Z_2^2$
12: $T_1 \leftarrow ZZ \cdot Z_2$
13: $T_2 \leftarrow ZZ \cdot x_Q - X_2$
14: $T_3 \leftarrow T_2 \cdot Z_2$
15: $T_2 \leftarrow T_2 \cdot \lambda$
16: $T_2 \leftarrow T_2 - Y_2 - T_1 \cdot y_Q$
17: $T_2 \leftarrow T_2 \cdot \overline{T_3}$
18: $f \leftarrow f^2$
19: $f \leftarrow f \cdot T_2$
20: **return** $(X_2 : Y_2 : Z_2 : T_{R_2})$, f

$Cost : 10\boldsymbol{M} + 7\boldsymbol{S}$

Algorithm 7 DBLADD: Miller doubling-and-addition

Input: Miller function $f = f_{m,P}(Q)$, points $R = [m]P = (X_R : Y_R : Z_R : T_R)$, $P = (x_P, y_P)$, $Q = (x_Q, y_Q)$ and the coefficient a of $E_{a,b} : y^2 = x^3 + ax + b$.
Output: $[2m+1]P = (X_3 : Y_3 : Z_3 : T_{R_3})$ and $f_{2m+1,P}(Q)$.

1: $XX \leftarrow X_R^2$
2: $T_1 \leftarrow 2 \cdot Y_R^2$
3: $T_2 \leftarrow T_1^2$
4: $T_3 \leftarrow (X_R + T_1)^2 - XX - T_2$
5: $T_4 \leftarrow 2 \cdot T_2$
6: $\lambda_1 \leftarrow 3 \cdot XX + T_R$
7: $X_2 \leftarrow \lambda_1^2 - 2 \cdot T_3$
8: $Y_2 \leftarrow \lambda_1 \cdot (T_3 - X_2) - T_4$
9: $Z_2 \leftarrow 2 \cdot Y_R \cdot Z_R$
10: $T_{R_2} \leftarrow 2 \cdot T_4 \cdot T_R$
11: $ZZ \leftarrow Z_2^2$
12: $T_1 \leftarrow ZZ \cdot x_P - X_2$
13: $T_2 \leftarrow T_1^2$
14: $T_3 \leftarrow 4 \cdot T_2$
15: $T_4 \leftarrow T_1 \cdot T_3$
16: $T_0 \leftarrow ZZ \cdot Z_2$
17: $T_5 \leftarrow T_0 \cdot y_P - Y_2$
18: $T_6 \leftarrow 2 \cdot T_5$
19: $T_7 \leftarrow T_3 \cdot X_2$
20: $X_3 \leftarrow T_6^2 - T_4 - 2 \cdot T_7$
21: $Y_3 \leftarrow T_6 \cdot (T_7 - X_3) - 2 \cdot Y_2 \cdot T_4$
22: $Z_3 \leftarrow (Z_2 + T_1)^2 - ZZ - T_2$
23: $T_{R_3} \leftarrow a \cdot Z_3^4$
24: $T_2 \leftarrow ZZ \cdot x_Q - X_2$
25: $T_3 \leftarrow y_Q \cdot T_0 + Y_2$
26: $T_4 \leftarrow \lambda_1 \cdot T_2 - T_3$
27: $T_3 \leftarrow T_1 \cdot T_3$
28: $T_5 \leftarrow T_5 \cdot T_2$
29: $T_5 \leftarrow T_3 + T_5$
30: $T_1 \leftarrow T_1 \cdot T_4$
31: $T_1 \leftarrow T_1 \cdot (x_Q - x_P)$
32: $T_1 \leftarrow T_1 \cdot \overline{T_5}$
33: $f \leftarrow f^2$
34: $f \leftarrow f \cdot T_1$
35: **return** $(X_3 : Y_3 : Z_3 : T_{R_3})$, f

$Cost : 20\boldsymbol{M} + 11\boldsymbol{S}$

Algorithm 8 DBLSUB: Miller doubling-and-subtraction

Input: Miller function $f = f_{m,P}(Q)$, points $R = [m]P = (X_R : Y_R : Z_R : T_R)$, $P = (x_P, y_P)$, $Q = (x_Q, y_Q)$ and the coefficient a of $E_{a,b} : y^2 = x^3 + ax + b$.
Output: $[2m-1]P = (X_3 : Y_3 : Z_3 : T_{R_3})$ and $f_{2m-1,P}(Q)$.

1: $XX \leftarrow X_R^2$
2: $T_1 \leftarrow 2 \cdot Y_R^2$
3: $T_2 \leftarrow T_1^2$
4: $T_3 \leftarrow (X_R + T_1)^2 - XX - T_2$
5: $T_4 \leftarrow 2 \cdot T_2$
6: $\lambda_1 \leftarrow 3 \cdot XX + T_R$
7: $X_2 \leftarrow \lambda_1^2 - 2 \cdot T_3$
8: $Y_2 \leftarrow \lambda_1 \cdot (T_3 - X_2) - T_4$
9: $Z_2 \leftarrow 2 \cdot Y_R \cdot Z_R$
10: $T_{R_2} \leftarrow 2 \cdot T_4 \cdot T_R$
11: $ZZ \leftarrow Z_2^2$
12: $T_1 \leftarrow ZZ \cdot x_P - X_2$
13: $T_2 \leftarrow T_1^2$
14: $T_3 \leftarrow 4 \cdot T_2$
15: $T_4 \leftarrow T_1 \cdot T_3$
16: $T_0 \leftarrow ZZ \cdot Z_2$
17: $T_5 \leftarrow -T_0 \cdot y_P - Y_2$
18: $T_6 \leftarrow 2 \cdot T_5$
19: $T_7 \leftarrow T_3 \cdot X_2$
20: $X_3 \leftarrow T_6^2 - T_4 - 2 \cdot T_7$
21: $Y_3 \leftarrow T_6 \cdot (T_7 - X_3) - 2 \cdot Y_2 \cdot T_4$
22: $Z_3 \leftarrow (Z_2 + T_1)^2 - ZZ - T_2$
23: $T_{R_3} \leftarrow a \cdot Z_3^4$
24: $T_2 \leftarrow ZZ \cdot x_Q - X_2$
25: $T_3 \leftarrow y_Q \cdot T_0 + Y_2$
26: $T_4 \leftarrow \lambda_1 \cdot T_2 - T_3$
27: $T_3 \leftarrow T_1 \cdot T_3$
28: $T_5 \leftarrow T_5 \cdot T_2$
29: $T_5 \leftarrow T_3 + T_5$
30: $T_1 \leftarrow T_1 \cdot T_4$
31: $T_1 \leftarrow T_1 \cdot \overline{T_5}$
32: $f \leftarrow f^2$
33: $f \leftarrow f \cdot T_1$
34: **return** $(X_3 : Y_3 : Z_3 : T_{R_3})$, f

$Cost : 19\boldsymbol{M} + 11\boldsymbol{S}$