Faster Key Generation of Supersingular Isogeny Diffie-Hellman

Kaizhan Lin[1], Fangguo Zhang[2,3], and Chang-An Zhao[1,3]

[1] School of Mathematics, Sun Yat-Sen university, Guangzhou 510275, P.R. China
zhaochan3@mail.sysu.edu.cn
[2] School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou 510006, P.R. China
[3] Guangdong Key Laboratory of Information Security, Guangzhou 510006, P.R. China

Abstract. Supersingular isogeny Diffie-Hellman (SIDH) is attractive for its relatively small public key size, but it is still unsatisfactory due to its efficiency, compared to other post-quantum proposals. In this paper, we focus on the performance of SIDH when the starting curve is $E_6 : y^2 = x^3 + 6x^2 + x$, which is fixed in Round-3 SIKE implementation. Inspired by the previous work [7, 10], we present several tricks to accelerate key generation of SIDH and each process of SIKE. Our experimental results show that the performance of this work is at least 6.09% faster than that of the current SIKE implementation, and we can further improve the performance when large storage is available.

Keywords: SIDH · SIKE · isogeny-based cryptography · post-quantum cryptography · Montgomery ladder

1 Introduction

Supersingular isogeny Diffie-Hellman (SIDH) [12] has been regarded as one of the most attractive post-quantum protocols during the last decade because of its small public key size compared to the other proposals at the same security level. Up to now, supersingular isogeny key encapsulation (SIKE), which is based on SIDH, still remains active in the NIST post-quantum standardization process [1]. Nonetheless, compared to other post-quantum cryptosystems, isogeny-based protocols generally seem to be inefficient, and so do SIDH and SIKE, for the reason why the efficient implementation of SIDH/SIKE has become a hot spot in recent years.

SIDH consists of the key generation phase and the key agreement phase. For each of them, the three-point ladder [12] and isogeny computation (including isogeny construction and isogeny evaluation at points) are dominant. In the SIDH implementation, they take 17%∼19% and 81%∼83% of the overall computational cost, respectively [5]. Although the latter one requires more computational resources, the optimization of the three-point ladder is still meaningful to improve the performance of SIDH and SIKE.

Jao and De Feo [12] developed the three-point ladder when SIDH was presented in 2011. One advantage of the three-point ladder is that the x-coordinate of the point $P + [s]Q$ can be computed directly by the x-coordinates of P, Q and $P - Q$, where s is a secret key. In the original SIDH proposal, the points $P, Q \in E_0(\mathbb{F}_{p^2})$, where $E_0 : y^2 = x^3 + x$. The three-point ladder was first improved by Costello et al. [7] by choosing proper torsion bases to execute base field operations of computing $P+[s]Q$ as possible. Faz-Hernández et al. [10] proposed a new three-point ladder, offering a saving of one differential addition per iteration. Furthermore, they claimed that the three-point ladder could be further improved, but it requires large memory. There is no doubt that setting E_0 as the starting curve brings perfect instantiation of SIDH. Unfortunately, it was observed in [8] that the distortion map of E_0 reduces the entropy of the private and public keys. As a consequence, the original starting curve was replaced by $E_6 : y^2 = x^3+6x^2+x$, while this modification restricts the techniques mentioned above, resulting in a relatively heavy overhead of the three-point ladder.

In this paper, we mainly consider torsion bases used in public-key compression of SIDH [16]. Our contributions are as follows:

- We present Method 1 for Alice to accelerate the kernel generation in the key generation phase, requiring few elements to be stored. When the storage is permitted, the techniques mentioned in [10] can be adapted into the current SIDH with a modification of the kernel generator, as we present in Method 2. In each iteration of the ladder, Methods 1 and 2 save about 17.8 and 21.4 multiplications in $\mathbb{F}_p$, respectively.
- We show that the method of computing kernel generators proposed in [7] could be still employed to improve the kernel generation in the key generation phase of Bob by utilizing several tricks, as we present in Method 3. Besides, we present Method 4 to further improve the performance, with a modification of the kernel generator and a previous knowledge of a look-up table. The performances of Methods 3 and 4 are about 2.2 and 4.7 times faster than that of the previous work, respectively.
- We adapt our methods into the current SIKE. The experimental results show that the performance of our methods is $6.09\% \sim 7.13\%$ faster than that of the previous work. If large storage is permitted, Method 2 and Method 4 improve the performance by $8.72\% \sim 10.30\%$.

The remaining of this paper is organized as follows. In Section 2 we review basic knowledge of isogenies, the Montgomery ladder, the three-point ladder and the SIDH protocol. In Section 3 we show how to speed up the kernel generation of the isogeny during the first phase of SIDH. Our implementation results are presented in Section 4. Section 5 concludes our work.

2 Preliminaries

Throughout the paper, an elliptic curve in Montgomery form $y^2 = x^3 + Ax^2 + x$ defined over $\mathbb{F}_{p^2} = \mathbb{F}_p[i]/\langle i^2+1\rangle$ is denoted by E_A, where $p = 2^{e_2}3^{e_3} - 1$ is a prime

satisfying $2^{e_2} \approx 3^{e_3} \approx \sqrt{p}$. We denote $r_A = 2^{e_2}$ and $r_B = 3^{e_3}$ for simplicity. In addition, let (x_P, y_P), $(X_P : Z_P)$ and $(X_P : Y_P : Z_P)$ denote the point P in affine, Kummer and projective coordinates, respectively.

2.1 Isogeny

Given two elliptic curves E and E' defined over a finite field $\mathbb{F}_q$, an isogeny $\phi : E \to E'$ is a non-constant morphism from $E(\mathbb{F}_q)$ to $E'(\mathbb{F}_q)$ such that

$$\phi(\mathcal{O}_E) \to \mathcal{O}_{E'},$$

where $\mathcal{O}_E$ is the unique point at infinity of E, and $\mathcal{O}_{E'}$ is defined similarly [20]. Let $\deg(\phi)$ be the degree of ϕ as a rational function and $\ker(\phi)$ the kernel of ϕ. The isogeny ϕ is called separable if $\deg(\phi) = \#\ker(\phi)$ [20, Theorem 4.10]. A separable isogeny of degree ℓ is abbreviated as ℓ-isogeny.

The curves E and E' are said to be ℓ-isogenous over $\mathbb{F}_q$ if there exists an ℓ-isogeny $\phi : E \to E'$ defined over $\mathbb{F}_q$. Deciding whether two curves are ℓ-isogenous [12, Problem 5.1] is considered to be a difficult problem [2,14], which mainly guarantees the security of the SIDH/SIKE protocol. See [18] for details of the structure of isogeny graphs.

2.2 Montgomery ladder

The Montgomery ladder was first proposed by Montgomery [15] in 1987, aiming to compute multiples of points for a given point. Compared to the double-and-add algorithm [9], the Montgomery ladder can avoid side-channel attacks [13]. Furthermore, the Montgomery ladder can efficiently compute the x-coordinates of multiples of points thanks to the following equations [15]:

$$\begin{cases} x_{[2]P} = \dfrac{(x_P^2 - 1)^2}{4x_P(x_P^2 + Ax_P + 1)}, \\ x_{P-Q}x_{P+Q} = \dfrac{(x_P x_Q - 1)^2}{(x_P - x_Q)^2}, \end{cases} \tag{1}$$

where P and Q are two points of E_A. It is easy to se that we can also use the Montgomery point $P = (X_P : Z_P)$ to compute the Montgomery point $[k]P = (X_{[k]P} : Z_{[k]P})$. Typically, Kummer coordinates are preferred for efficiency.

In each iteration, the Montgomery ladder executes one point doubling plus one differential addition, denoted by **dadd** [3, Algorithm 5]. It costs **six** (or **five** when $Z_{P-Q} = 1$) field multiplications and **four** field squarings. Pseudocode of **dadd** is referred to Appendix A, and pseudocode of the Montgomery ladder is available in Algorithm 1.

In the SIDH protocol, a Montgomery point of the form $S = P + [s]Q$ on E_A is required to be computed. The Montgomery ladder can be used to compute the x-coordinates of $[s]Q$ and $[s+1]Q$ (Note that the Montgomery ladder also

Algorithm 1 Montgomery ladder [15]

Input: $P = (X_P : Z_P) \in E_A$, $s = (s_{\ell-1} \cdots s_1 s_0)_2$ and $A_{24} = (A+2)/4$
Output: $[s]P$.

1: $(X_1 : Z_1) = [2]P$, $(X_2 : Z_2) = P$, $(X_3 : Z_3) = P$
2: **for** $j = \ell - 2$ down to 0 **do**
3: **if** $s_i = 0$ **then**
4: $(X_2, Z_2, X_1, Z_1) = \mathbf{dadd}(X_2, Z_2, X_1, Z_1, X_3, Z_3, A_{24})$
5: **else**
6: $(X_1, Z_1, X_2, Z_2) = \mathbf{dadd}(X_1, Z_1, X_2, Z_2, X_3, Z_3, A_{24})$
7: **end if**
8: **end for**
9: **return** X_2, Z_2

computes the latter). Afterwards, one can recover the y-coordinate of $[s]Q$ by the Okeya-Sakurai formula [17]:

$$y_{[s]Q} = \frac{\left(x_{[s]Q}x_Q + 1\right)\left(x_{[s]Q} + x_Q + 2A\right) - 2A - \left(x_{[s]Q} - x_Q\right)^2 x_{[s+1]Q}}{2y_Q}. \quad (2)$$

Thus, we can get $P + [s]Q$ by one differential addition.

We present Algorithm 4 in Appendix B to recover $[s]Q$, while Algorithms 6 and 7 in Appendix C aim to perform differential addition.

2.3 Three-point ladder algorithm

Instead of the Montgomery ladder, Jao, De Feo and Plût [12] proposed a three-point ladder to compute $P + [s]Q$. The superiority of the three-point ladder is that one can compute $P+[s]Q$ directly. That is to say, there is no need recovering the y-coordinate of any point. The three-point ladder was later improved in [10, Algorithm 2].

Algorithm 2 Three-point ladder [10]

Input: $P = (X_P : Z_P)$, $Q = (X_Q : Z_Q)$, $Q-P = (X_{Q-P} : Z_{Q-P})$, $s = (s_{\ell-1} \cdots s_1 s_0)_2$ and $A_{24} = (A+2)/4$
Output: $P + [k]Q$

1: $(X_1 : Z_1) = Q$, $(X_2 : Z_2) = P$, $(X_3 : Z_3) = Q - P$
2: **for** $j = 0$ to $\ell - 1$ **do**
3: **if** $s_i = 0$ **then**
4: $(X_1, Z_1, X_3, Z_3) = \mathbf{dadd}(X_1, Z_1, X_3, Z_3, X_2, Z_2, A_{24})$
5: **else**
6: $(X_1, Z_1, X_2, Z_2) = \mathbf{dadd}(X_1, Z_1, X_2, Z_2, X_3, Z_3, A_{24})$
7: **end if**
8: **end for**
9: **return** X_2, Z_2

As we can see in Algorithm 2, in each iteration the point doubling does not depend on the secret key s. The authors of [10] pointed out that a look-up table can be precomputed to reduce the computational cost:

$$T(Q) = \left(\frac{x_{[2]Q}+1}{x_{[2]Q}-1}, \frac{x_{[4]Q}+1}{x_{[4]Q}-1}, \ldots, \frac{x_{[2^\ell]Q}+1}{x_{[2^\ell]Q}-1} \right), \tag{3}$$

where $i = 1, \cdots, \ell = e_2 - 3$ or $\lceil \log r_B \rceil$. In this case it costs only **three** field multiplications and **two** squarings per iteration, but requires relative large memory.

Remark 1. Similar to the Montgomery ladder, one can recover the y-coordinate of $P + [k]Q$. See Algorithm 5 in Appendix B for more details.

2.4 SIDH protocol

In this subsection, we introduce the classical SIDH protocol briefly. Let $E_6 : y^2 = x^3 + 6x^2 + x$ be a supersingular elliptic curve over $\mathbb{F}_{p^2}$. For two subgroups $E[r_A]$ and $E[r_B]$, there are two pairs of torsion points $\{P_A, Q_A\}$ and $\{P_B, Q_B\}$ such that $\langle P_A, Q_A \rangle = E_6[r_A]$ and $\langle P_B, Q_B \rangle = E_6[r_B]$. All mentioned above are considered as public domain parameters.

Alice chooses a random integer $s_A \in [0, r_A - 1]$ as her secret to begin the key generation phase. To prevent simple side-channel attacks, she adapts the three-point ladder to compute $S_A = P_A + [s_A]Q_A$ of order r_A. Thereafter, Alice constructs the r_A-isogeny with kernel $\langle S_A \rangle$ by Vélu's formula [21] with the help of the smoothness of r_A. Finally, Alice transmits $\phi_A(P_B)$, $\phi_A(Q_B)$ and the image curve E_A to Bob. Similarly, Bob selects his secret key $s_B \in [0, r_B - 1]$ to compute $S_B = P_B + [s_B]Q_B$, and then finds the r_B-isogeny ϕ_B with kernel $\langle S_B \rangle$ to calculate $\phi_B(P_A)$, $\phi_B(Q_A)$. Finally, he sends $\phi_B(P_A), \phi_B(Q_A)$ as well as the image curve E_B to Alice.

Once Alice receives the public key from Bob, she begins her key agreement phase. In the first place she computes $S'_A = \phi_B(P_A) + [s_A]\phi_B(Q_A)$. Next, she constructs the isogeny ϕ'_A with kernel $\langle S'_A \rangle$ and finds out the image curve E_{BA} of ϕ'_A. Similar to Alice, Bob evaluates $S'_B = \phi_A(P_B) + [s_B]\phi_A(Q_B)$ and constructs the corresponding isogeny ϕ'_B with kernel $\langle S'_B \rangle$. Note that only the image curve parameter is needed. To end the key agreement phase, each of them evaluates the j-invariant of their respective image curve as their shared secret.

As mentioned in Section 2.3, Alice can make full use of the x-coordinates of $P_A, Q_A, R_A = P_A - Q_A$ to compute the x-coordinate of S_A using the three-point ladder. Instead of $\{\phi_A(P_B), \phi_A(Q_B), A\}$, she could utilize x_{S_A} to evaluate $\{x_{\phi_A(P_B)}, x_{\phi_A(Q_B)}, x_{\phi_A(R_B)}\}$ and transmits it to Bob. The same case is also available for Bob. Similarly, the key agreement phase can be optimized in the same way as well. Besides, the public keys of Alice and Bob can be further compressed. For detailed techniques used in public-key compression of SIDH/SIKE, we refer to [3, 6, 11, 16, 19, 22].

3 Optimization of Kernel Generator Computation

In this section, we show how to improve the kernel generation of the isogeny in key generation of SIDH. We consider the torsion bases selected in [16], which can be utilized to speed up key generation of the compressed version of SIDH as well.

When counting field operations, we use **M** and **S** to denote the respective costs of a multiplication and a squaring in the field $\mathbb{F}_{p^2}$. Besides, the notations **m** and **s** are used to represent the costs of a multiplication and a squaring in $\mathbb{F}_p$, respectively. To measure the performance of the algorithms, we estimate $\mathbf{M} \approx 3\mathbf{m}$, $\mathbf{S} \approx 2\mathbf{m}$ and $\mathbf{s} \approx 0.8\mathbf{m}$ [10, 22].

3.1 Case of Alice

Naehrig and Renes [16] chose a r_A-torsion basis $\{P_A, Q_A\}$ such that

$$[2]P_A = (x, iy), \quad [2]Q_A \in E_6(\mathbb{F}_p), \tag{4}$$

where $x, y \in \mathbb{F}_p$, to speed up public-key compression. In fact, the features of this basis could be used to accelerate the three-point ladder as well.

Remark 2. Since $E_6(\mathbb{F}_p)[r_A]$ is isomorphic to $\mathbb{Z}/2^{e_2-1}\mathbb{Z} \times \mathbb{Z}/2\mathbb{Z}$ [7], it is impossible to find a r_A-torsion basis such that one of the torsion points belongs to $E_6(\mathbb{F}_p)$.

When implementing the three-point ladder (Algorithm 2) to compute the point $S_A = P_A + [s_A]Q_A$, all the operations are in $\mathbb{F}_{p^2}$, for the reason that both P_A and Q_A are defined on $E_6(\mathbb{F}_{p^2})\backslash E_6(\mathbb{F}_p)$. However, note that

$$[2^i]Q_A \in E_6(\mathbb{F}_p), i = 1, 2, \cdots, e_2 - 1.$$

Therefore, instead of the three-point ladder used to compute $P_A + [s_A]Q_A$ directly in $\mathbb{F}_{p^2}$, Alice could execute the Montgomery ladder to compute $[s_A - 1]Q_A$ or $[s_A]Q_A$ in the base field if she precomputes $[2]Q_A$ in affine coordinate, and then obtain $P_A + [s_A]Q_A$ with several operations in $\mathbb{F}_{p^2}$. Our idea to accelerate the kernel generation for Alice comes mostly from Method 1.

Method 1:

- Use the Montgomery ladder to compute the points $R_0 = [\lfloor \frac{s_A}{2} \rfloor]([2]Q_A)$ and $R_1 = [\lfloor \frac{s_A}{2} \rfloor + 1]([2]Q_A)$ in Kummer coordinates;
- Utilize the Okeya-Sakurai formula (2) to recover R_0 in projective coordinate;
- If $(s_A \bmod 2) \equiv 1$, compute $(P_A + Q_A) + R_0$ in Kummer coordinate, otherwise compute $P_A + R_0$ in Kummer coordinate.

Lemma 1. *One can compute the point $S_A = P_A + [s_A]Q_A$ in Kummer coordinate by applying Method 1.*

Proof. Note that

$$R_0 = [\lfloor \frac{s_A}{2} \rfloor]([2]Q_A) = \begin{cases} [s_A - 1]Q_A, \text{when } s_A \text{ is odd,} \\ [s_A]Q_A, \text{else.} \end{cases}$$

Therefore, by performing the Montogomery ladder, we have $[s_A - 1]Q_A$ and $[s_A + 1]Q_A$ in Kummer coordinates when s_A is odd, or $[s_A]Q_A$ and $[s_A + 2]Q_A$ in Kummer coordinates when s_A is even.

Now consider $i = (s_A \bmod 2)$. If $i = 1$, then the point that we recover is $[s_A - 1]Q_A$. In this case, one should compute the point $P_A + [s_A]Q_A = (P_A + Q_A) + [s_A - 1]Q_A = (P_A + Q_A) + R_0$ in Kummer coordinate. When i is equal to 0, i.e., s_A is even, we directly compute $P_A + [s_A]Q_A = P_A + [\lfloor \frac{s_A}{2} \rfloor]([2]Q_A) = P_A + R_0$ in Kummer coordinate.

Hence, in both cases Method 1 correctly computes the point $P_A + [s_A]Q_A$ in Kummer coordinate. ■

One can precompute a lookup table to speed up the kernel generation of the isogeny when the storage is available. This technique is also adapted in [10]. Find a point $P'_A \in E(\mathbb{F}_p)$ of order 3, and then precompute $Q'_A = [2]Q_A - P'_A$ and the lookup table

$$T([2]Q_A) = \left(\frac{x_{[2]Q_A} + 1}{x_{[2]Q_A} - 1}, \frac{x_{[4]Q_A} + 1}{x_{[4]Q_A} - 1}, \cdots, \frac{x_{[2^{\ell+1}]Q_A} + 1}{x_{[2^{\ell+1}]Q_A} - 1} \right),$$

where $\ell = e_2 - 4$. After that, Alice can improve the performance of the three-point ladder and compute the kernel generator of the isogeny by the following:

Method 2:

- Use the three-point ladder to compute $R_0 = P'_A + [\lfloor \frac{s_A}{2} \rfloor \bmod 2^{e_2-3}]([2]Q_A)$ and $R_1 = [2^{e_2-3} - \lfloor \frac{s_A}{2} \rfloor \bmod 2^{e_2-3})]([2]Q_A) - P'_A$ in Kummer coordinates with the help of the lookup table $T([2]Q_A)$ and Q'_A;
- Set $[2^{e_2-1}]Q_A = (0 : 0)$, then perform one differential addition;
- Use Algorithm 5 to recover R_0 in projective coordinate;
- Set

$$R_2 = \begin{cases} R_0, \text{if } \lfloor \frac{s_A}{2^{e_2-2}} \rfloor = 0, \\ R_0 + [2^{e_2-2}]Q_A, \text{if } \lfloor \frac{s_A}{2^{e_2-2}} \rfloor = 1, \\ R_0 + [2^{e_2-1}]Q_A, \text{if } \lfloor \frac{s_A}{2^{e_2-2}} \rfloor = 2, \\ R_0 - [2^{e_2-2}]Q_A, \text{if } \lfloor \frac{s_A}{2^{e_2-2}} \rfloor = 3; \end{cases} \tag{5}$$

- If $(s_A \bmod 2) = 1$, compute $R_3 = (P_A + Q_A) + R_2$ in Kummer coordinate, otherwise $R_3 = P_A + R_2$ in Kummer coordinate;
- Compute $[3]R_3$ in Kummer coordinate.

Lemma 2. *One can compute the point $[3]P_A + [3s_A]Q_A$ in Kummer coordinate and regard it as the kernel generator of a r_A-isogeny by applying Method 2. This modification of the kernel generator does not change the key space size.*

Proof. Note that the t-th iteration, the three-point ladder computes the points $P'_A + [\lfloor \frac{s_A}{2} \rfloor \bmod 2^t]([2]Q_A)$ and $[2^t - (\lfloor \frac{s_A}{2} \rfloor \bmod 2^t)]([2]Q_A) - P'_A$. Hence, we can utilize $T([2]Q_A)$ to efficiently compute R_0 and R_1.

Step 4 computes $R_2 = R_0 + [\lfloor \frac{s_A}{2^{e_2-2}} \rfloor]([2]Q_A)$, i.e.,

$$R_2 = \begin{cases} P'_A + [s_A - 1]Q_A, \text{when } s_A \text{ is odd}, \\ P'_A + [s_A]Q_A, \text{else}. \end{cases}$$

When s_A is odd, we get $R_2 = P'_A + [s_A - 1]Q_A$. In this case, the point $R_3 = P_A + Q_A + R_2 = P_A + P'_A + [s_A]Q_A$. When s_A is even, compute $R_3 = P_A + R_2 = P_A + P'_A + [s_A]Q_A$.

The last step is to eliminate P'_A by tripling R_3:

$$[3]R_3 = [3]P_A + [3]P'_A + [3s_A]Q_A = [3]P_A + [3s_A]Q_A.$$

Since $\gcd(3, r_A) = 1$, the order of $[3]R_3$ is r_A. Hence, the point $[3]R_3$ could be regarded as the kernel generator of a r_A-isogeny. Besides, the group endomorphism

$$\begin{array}{rccc} \eta_3 : & (\mathbb{Z}_{r_A}, +) & \to & (\mathbb{Z}_{r_A}, +), \\ & x & \mapsto & 3x, \end{array}$$

is an isomorphism. Therefore, the key space size is not changed. This completes the proof. ■

One may ask how to run in constant time when computing R_2 in Equation (5). It is natural to compute three point additions directly and output the right point with respect to $\lfloor \frac{s_A}{2^{e_2-2}} \rfloor$. Here we give another approach to compute R_2 efficiently by utilizing the property $[2^{e_2-1}]Q_A = (0, 0)$.

We first compute $R'_2 = R_0 + [2^{e_2-2}]Q_A$. Thereafter, according to the addition law on elliptic curves, we have

$$R_0 + (0,0) = \left(\frac{1}{x_{R_0}}, -\frac{y_{R_0}}{x_{R_0}^2} \right) = \left(X_{R_0} Z_{R_0} : -Y_{R_0} Z_{R_0} : X_{R_0}^2 \right),$$

$$R_0 - [2^{e_2-2}]Q_A = \left(\frac{1}{x_{R'_2}}, -\frac{y_{R'_2}}{x_{R'_2}^2} \right) = \left(X_{R'_2} Z_{R'_2} : -Y_{R'_2} Z_{R'_2} : X_{R'_2}^2 \right).$$

Therefore, there is no need to compute three differential additions. Instead, we compute two field multiplications and one field squaring with respect to $\lfloor \frac{s_A}{2^{e_2-2}} \rfloor \bmod 2$. Finally, output the right point with respect to $\lfloor \frac{s_A}{2^{e_2-1}} \rfloor$. Further, to defend the attacker who performs one fault injection, we can simply compute the above points twice and check whether the two results are the same. This countermeasure is also adapted in the CSIDH with dummy-operations against fault injection attacks [4].

Remark 3. In the key agreement phase, Alice could compute the kernel generator $S_{A'} = \phi_B(P_A) + [s_A]\phi_B(Q_A)$ as usual, since S'_A and $[3]S'_A$ generate the same kernel.

To sum up, the estimates given in Table 1 show the computational cost for each iteration of the ladder. The table shows that Methods 1 and 2 can improve the three-point ladder, and the implementation of the latter method performs better when large storage is available.

Table 1. Cost estimates for each iteration of the three-point ladder during the key generation phase of Alice.

Method	Cost estimates
Current SIDH [3]	$6\mathbf{M}+4\mathbf{S} \approx (6\times 3 + 4\times 2)\ \mathbf{m} = 26\mathbf{m}$
Method 1	$5\mathbf{m}+4\mathbf{s} \approx (5 + 4\times 0.8)\ \mathbf{m} = 8.2\mathbf{m}$
Method 2	$3\mathbf{m}+2\mathbf{s} \approx (3 + 2\times 0.8)\ \mathbf{m} = 4.6\mathbf{m}$

3.2 Case of Bob

To compress public keys faster, Naehrig and Renes [16] selected the r_B-torsion basis $\{P_3, Q_3\}$ on E_0 such that

$$P_3 = (x, y), Q_3 = \psi(P_3) = (-x, iy),$$

where $x, y \in \mathbb{F}_p$. Then they set $\{\phi_2(P_3), \phi_2(Q_3)\}$ as the r_B-torsion basis of E_6, where ϕ_2 is the 2-isogeny with kernel $\langle (i, 0) \rangle$:

$$\begin{aligned} \phi_2 : E_0 &\to E_6, \\ (x, y) &\mapsto \left(\frac{ix^2 - x}{x - i}, y\frac{ix^2 + 2x + i}{(x - i)^2} \right). \end{aligned}$$

Instead of $P_B + [s_B]Q_B$, we consider $[s_B]P_B + Q_B$ as the kernel generator of the isogeny. Similar to the ideas proposed in [7], Bob can use his secret key s_B to compute $[s_B]P_B + Q_B$ as follows:

Method 3:

- Use the Montgomery ladder to compute $[s_B]P_3$ and $[s_B + 1]P_3$ in Kummer coordinates;
- Utilize the Okeya-Sakurai formula (2) to recover $[s_B]P_3$ in projective coordinate;
- Compute $[s_B]P_3 + Q_3$ in Kummer coordinate;
- Complete the evaluation of the isogeny ϕ_2 at $[s_B]P_3 + Q_3$.

Lemma 3. *One can compute the point $[s_B]P_B + Q_B$ in Kummer coordinate and regard it as the kernel generator of a 3^{e_3}-isogeny by applying Method 3. This modification of the kernel generator does not change the key space size.*

Proof. It is easy to check that one can correctly compute $[s_B]P_3 + Q_3$. The rest is to prove $S_B = \phi_2([s_B]P_3 + Q_3)$ is a point of order r_B. Note that

$$S_B = \phi_2([s_B]P_3 + Q_3) = [s_B]\phi_2(P_3) + \phi_2(Q_3) = [s_B]P_B + Q_B.$$

Since the order of Q_3 is r_B and $\gcd(2, r_B) = 1$, the order of the point Q_B is r_B. This implies that S_B is a point of order r_B, and it could be regarded as the kernel generator of a r_B-isogeny. Obviously, this modification of the kernel generator does not change the key space size because there are exactly r_B choices of s_B to compute the kernel generators and any two of them do not generate the same group of order r_B. ■

Since $Q_3 \in E_0(\mathbb{F}_p)$, all the operations of the Montgomery ladder are implemented in the base field. Therefore, Bob could compute the point S_B much more efficient than before. In this case, only the point P_3 in affine coordinate should be stored (Q_3 could be recovered by $\psi(P_3)$).

Set $Q_3' = (1, \sqrt{2}) \in E_0(\mathbb{F}_p)$ (Note that 2 is a square in $\mathbb{F}_p$ because $p \equiv 7(mod\ 8)$). It is easy to check Q_3' is a point of order 4. Analogous to Method 2, Bob could store the table

$$T(P_3) = \left(\frac{x_{[2]P_3} + 1}{x_{[2]P_3} - 1}, \frac{x_{[4]P_3} + 1}{x_{[4]P_3} - 1}, \cdots, \frac{x_{[2^\ell]P_3} + 1}{x_{[2^\ell]P_3} - 1}\right),$$

where $\ell = \lceil \log r_B \rceil$ and $P_3' = P_3 - Q_3' \in E_0(\mathbb{F}_p)$, to speed up the implementation of the three-point ladder. The main procedure is as follows:

Method 4:

- Use the three-point ladder to compute $R_0 = [s_B]P_3 + Q_3'$ and $R_1 = [2^{\lceil \log r_B \rceil} - s_B]P_3 - Q_3'$ in Kummer coordinates;
- Compute the points $R_2 = [4]R_0$ and $R_3 = [4]R_1$ in Kummer coordinates;
- Utilize Algorithm 5 to recover R_2 in projective coordinate;
- Compute $R_4 = R_2 + Q_3$;
- Complete the evaluation of the isogeny ϕ_2 at R_4.

Lemma 4. *One can compute the point $[4s_B]P_B + Q_B$ in Kummer coordinate and regard it as the kernel generator of a r_B-isogeny by applying Method 4. This modification of the kernel generator does not change the key space size.*

Proof. In the t-th iteration, the three-point ladder computes $[s_B \bmod 2^t]P_3 + Q_3'$ and $[2^t - (s_B \bmod 2^t)]P_3 - Q_3'$. Hence, we can utilize $T(P_3)$ to efficiently compute R_0 and R_1.

Step 2 aims to eliminate Q_3', which is a point of order 4. We simply quadruple R_0 and R_1, or, alternatively, double them twice. Then,

$$\begin{aligned} R_2 &= [4]([s_B]P_3 + Q_3') = [4s_B]P_3, \\ R_3 &= [4]([2^{\lceil \log r_B \rceil} - s_B]P_3 - Q_3') = [2^{\lceil \log r_B \rceil + 2} - 4s_B]P_3. \end{aligned}$$

After recovering the projective coordinates of the point R_2, we compute

$$R_4 = R_2 + Q_3 = [4s_B]P_3 + Q_3.$$

The rest is to evaluate ϕ_2 to R_4. Note that P_3, Q_3 are points of order r_B and $\mathbf{gcd}(2, r_B) = 1$. The point

$$\phi_2(R_4) = [4s_B]\phi_2(P_3) + \phi_2(Q_3) = [4s_B]P_B + Q_B$$

has order r_B on E_6 as well. Consequently, Bob could use $[4s_B]P_B + Q_B$ to determine a r_B-isogeny. This modification of the kernel generator does not reduce the key space size as $\mathbf{gcd}(r_B, 4) = 1$. ■

We estimate the cost of each iteration of the ladder by utilizing the methods mentioned above, and draw a comparison between the cost of the methods and that of the previous, as shown in Table 2. We can predict that Method 3 improves the performance obviously, and so does Method 4.

Table 2. Cost estimates for each iteration of the ladder during the key generation phase of Bob.

Method	Cost estimates
Current SIDH [3]	$6\mathbf{M}+4\mathbf{S} \approx (6\times3 + 4\times2)\ \mathbf{m} = 26\mathbf{m}$
Method 3	$5\mathbf{m}+4\mathbf{s} \approx (5 + 4\times0.8)\ \mathbf{m} = 8.2\mathbf{m}$
Method 4	$3\mathbf{m}+2\mathbf{s} \approx (3 + 2\times0.8)\ \mathbf{m} = 4.6\mathbf{m}$

4 Implementation

In this section we present the implementation of key generation of SIDH and SIKE by utilizing our techniques, and then give a comparison in efficiency.

In Tables 1 and 2 we give cost estimates for each iteration of the ladder. Indeed, the cost of the ladder dominates the cost of the kernel generation of isogenies, so its performance mainly depends on the implementation of the ladder.

Our implementation makes use of the SIDH C library [3]. The following experimental results have been obtained by using an 11th Gen Intel(R) Core(TM) i7-1185G7 @ 3.00GHz on 64-bit Linux. We benchmarked our code and observed the performance of key generation of SIDH and SIKE by using different methods in comparison with the current SIDH/SIKE. The results are reported in Table 3.

Table 3. Performance comparison of key generation of SIDH (including the ladder and isogeny computation) by using different methods. All timings are presented in millions of clock cycles. We use M1, M2, M3 and M4 to denote the situation when using Method 1, Method 2, Method 3 and Method 4, respectively.

Setting	Alice's key generation					Bob's key generation				
	Current	Ours		Speedup		Current	Ours		Speedup	
	SIDH [3]	M1	M2	M1	M2	SIDH [3]	M3	M4	M3	M4
SIKEp434	3.20	2.82	2.72	11.88%	15.00%	3.55	3.15	3.07	11.27%	13.52%
SIKEp503	4.50	3.92	3.76	12.89%	16.44%	4.98	4.34	4.22	12.85%	15.26%
SIKEp610	9.00	8.05	7.85	10.56%	12.78%	8.97	8.03	7.82	10.48%	12.82%
SIKEp751	13.75	12.25	11.60	10.91%	15.64%	15.50	13.84	13.30	10.71%	14.19%

As can be seen in Table 3, when the storage is constrained, the performance of ours is 10.56%∼ 12.89% faster than key generation of the current SIDH for the case of Alice, and 10.48%∼ 12.85% faster for the case of Bob. When the storage is permitted, it performs better with a previous knowledge of a look-up table.

Note that in SIKE, each process (KeyGen, Encaps and Decaps) calls the ladder in key generation of SIDH once. Hence, we improve each process with the help of our methods. Table 4 reports the experimental results by using our techniques and the comparison with that of the current SIKE.

Table 4. Performance comparison of SIKE by using different methods. All timings are presented in millions of clock cycles. We use M1, M2, M3 and M4 to denote the situation when using Method 1, Method 2, Method 3 and Method 4, respectively.

Setting		M1&M3	M2&M4	Current SIKE [3]	Speedup	
					M1&M3	M2&M4
SIKEp434	Keygen	3.21	3.11	3.56	9.83%	12.64%
	Encaps	5.52	5,36	5.82	5.15%	7.90%
	Decaps	5.91	5.76	6.21	4.83%	7.25%
	Total	**14.64**	**14.23**	**15.59**	**6.09%**	**8.72%**
SIKEp503	Keygen	4.55	4.37	5.09	10.61%	14.15%
	Encaps	7.81	7.62	8.34	6.35%	8.63%
	Decaps	8.35	8.16	8.87	5.86%	8.12%
	Total	**20.71**	**20.15**	**22.30**	**7.13%**	**9.64%**
SIKEp610	Keygen	8.05	7.79	9.06	11.15%	14.02%
	Encaps	15.60	15.36	16.71	6.64%	8.08%
	Decaps	15.81	15.40	16.72	5.44%	7.89%
	Total	**39.46**	**38.55**	**42.49**	**7.13%**	**9.27%**
SIKEp751	Keygen	13.75	13.19	15.34	10.37%	14.02%
	Encaps	23.39	22.42	24.87	5.95%	9.85%
	Decaps	25.22	24.38	26.67	5.44%	8.59%
	Total	**62.36**	**59.99**	**66.88**	**6.76%**	**10.30%**

In Table 5 we report the additive memory requirements for Methods 2 and 4 that require to store a precomputed table in the SIDH settings. It shows that large memory is necessary for applying the methods. Hence, Methods 1 and 3 would be preferred for memory constrained environments.

Table 5. Additive memory requirements (in KiB) for Method 2 and Method 4

Setting	SIKEp434	SIKEp503	SIKEp610	SIKEp751
Alice	12.2	16.1	24.5	35.6
Bob	12.2	16.1	24.2	36.0
Total	24.4	32.2	48.7	71.6

5 Conclusion

In this paper, we proposed several tricks to utilize these techniques to the key generation phase of the current SIDH. Some of our methods change the generator form of the isogeny, but the key space size is not reduced. When large storage is permitted, we could improve the ladder performance further. Our new idea may make SIDH/SIKE more attractive in post-quantum cryptography.

Acknowledgements

We thank the anonymous reviewers for helpful comments and suggestions. This work is supported by Guangdong Major Project of Basic and Applied Basic Research (No. 2019B030302008), the National Natural Science Foundation of China (No.s 61972429 and 61972428).

References

1. The National Institute of Standards and Technology (NIST). Post-quantum cryptography standardization (2017–2018), `https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Post-Quantum-Cryptography-Standardization`
2. Adj, G., Cervantes-Vázquez, D., Chi-Domínguez, J.J., Menezes, A., Rodríguez-Henríquez, F.: On the Cost of Computing Isogenies Between Supersingular Elliptic Curves. In: Cid, C., Jacobson Jr., M.J. (eds.) Selected Areas in Cryptography – SAC 2018. pp. 322–343. Springer International Publishing, Cham (2019)
3. Azarderakhsh, R., Campagna, M., Costello, C., De Feo, L., Hess, B., Hutchinson, A., Jalali, A., Jao, D., Karabina, K., Koziel, B., LaMacchia, B., Longa, P., Naehrig, M., Pereira, G., Renes, J., Soukharev, V., Urbanik, D.: Supersingular Isogeny Key Encapsulation (2020), `http://sike.org`
4. Campos, F., Kannwischer, M.J., Meyer, M., Onuki, H., Stöttinger, M.: Trouble at the CSIDH: Protecting CSIDH with Dummy-Operations Against Fault Injection Attacks. In: 2020 Workshop on Fault Detection and Tolerance in Cryptography (FDTC). pp. 57–65 (2020). https://doi.org/10.1109/FDTC51366.2020.00015

5. Cervantes-Vázquez, D., Ochoa-Jiménez, E., Rodríguez-Henríquez, F.: Extended supersingular isogeny diffie–hellman key exchange protocol: Revenge of the sidh. IET Information Security **n/a**(n/a). https://doi.org/https://doi.org/10.1049/ise2.12027, https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/ise2.12027
6. 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)
7. 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)
8. Costello, C., Longa, P., Naehrig, M., Renes, J., Virdia, F.: Improved Classical Cryptanalysis of SIKE in Practice. In: Kiayias, A., Kohlweiss, M., Wallden, P., Zikas, V. (eds.) Public-Key Cryptography – PKC 2020. pp. 505–534. Springer International Publishing, Cham (2020)
9. Donald E. Knuth: The Art of Computer Programming, v.2. Seminumerical algorithms. Addison-Welsley, 2nd edition (1981)
10. Faz-Hernández, A., López, J., Ochoa-Jiménez, E., Rodríguez-Henríquez, F.: A Faster Software Implementation of the Supersingular Isogeny Diffie-Hellman Key Exchange Protocol. IEEE Transactions on Computers **67**(11), 1622–1636 (2018)
11. Hutchinson, A., Karabina, K., Pereira, G.: Memory Optimization Techniques for Computing Discrete Logarithms in Compressed SIKE. In: Cheon, J.H., Tillich, J.P. (eds.) Post-Quantum Cryptography. pp. 296–315. Springer International Publishing, Cham (2021)
12. 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)
13. Kocher, P., Jaffe, J., Jun, B.: Differential Power Analysis. In: Wiener, M. (ed.) Advances in Cryptology — CRYPTO' 99. pp. 388–397. Springer Berlin Heidelberg, Berlin, Heidelberg (1999)
14. Longa, P., Wang, W., Szefer, J.: The Cost to Break SIKE: A Comparative Hardware-Based Analysis with AES and SHA-3. In: Malkin, T., Peikert, C. (eds.) Advances in Cryptology – CRYPTO 2021. pp. 402–431. Springer International Publishing, Cham (2021)
15. Montgomery, P.L.: Speeding the Pollard and elliptic curve methods of factorization. Mathematics of Computation **48**, 243–264 (1987)
16. Naehrig, M., Renes, J.: Dual Isogenies and Their Application to Public-key Compression for Isogeny-based Cryptography. In: Advances in Cryptology - ASIACRYPT 2019 (December 2019)
17. 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)
18. Onuki, H., Aikawa, Y., Takagi, T.: The Existence of Cycles in the Supersingular Isogeny Graphs Used in SIKE. In: 2020 International Symposium on Information Theory and Its Applications (ISITA). pp. 358–362 (2020)
19. Pereira, G., Doliskani, J., Jao, D.: x-only point addition formula and faster compressed SIKE. Journal of Cryptographic Engineering pp. 1–13 (2020)

20. Silverman, J.H.: The Arithmetic of Elliptic Curves, 2nd Edition. Graduate Texts in Mathematics. Springer (2009)
21. Vélu, J.: Isogénies entre courbes elliptiques. C. R. Acad. Sci., Paris, Sér. A **273**, 238–241 (1971)
22. Zanon, G.H.M., Simplicio, M.A., Pereira, G.C.C.F., Doliskani, J., Barreto, P.S.L.M.: Faster Key Compression for Isogeny-Based Cryptosystems. IEEE Transactions on Computers **68**(5), 688–701 (2019)

A Point Doubling and Differential Addition

Algorithm 3 dadd: doubling and differential addition [3, Algorithm 5]

Input: $(X_P : Z_P)$, $(X_Q : Z_Q)$, $(X_{P-Q} : Z_{P-Q})$ and $A_{24} = (A+2)/4$
Output: $(X_{[2]P} : Z_{[2]P})$ and (X_{P+Q}, Z_{P+Q})

1: $t_0 \leftarrow X_P + Z_P$
2: $t_1 \leftarrow X_P - Z_P$
3: $X_P \leftarrow t_0^2$
4: $t_2 \leftarrow X_Q - Z_Q$
5: $X_Q \leftarrow X_Q + Z_Q$
6: $t_0 \leftarrow t_0 \cdot t_2$
7: $Z_P \leftarrow t_1^2$
8: $t_1 \leftarrow t_1 \cdot X_Q$
9: $t_2 \leftarrow X_P - Z_P$
10: $X_P \leftarrow X_P \cdot Z_P$
11: $X_Q \leftarrow A_{24} \cdot t_2$
12: $Z_Q \leftarrow t_0 - t_1$
13: $Z_P \leftarrow Z_P + X_Q$
14: $X_Q \leftarrow t_0 + t_1$
15: $Z_P \leftarrow t_2 \cdot Z_P$
16: $Z_Q \leftarrow Z_Q^2$
17: $X_Q \leftarrow X_Q^2$
18: $Z_Q \leftarrow X_{P-Q} \cdot Z_Q$
19: $X_Q \leftarrow Z_{P-Q} \cdot X_Q$

Remark 4. When $Z_{P-Q} = 1$, **one** field multiplication can be saved in Line 19.

B Recovering the Y-coordinate

B.1 The case of the Montgomery ladder

Algorithm 4 Recovering the Y-coordinate after executing the Montgomery ladder

Input: (x_Q, y_Q), $(X_{[s]Q} : Z_{[s]Q})$, $(X_{[s+1]Q} : Z_{[s+1]Q})$ and A
Output: $(X_{[s]Q} : Y_{[s]Q} : Z_{[s]Q})$

1: $t_0 \leftarrow x_Q \cdot Z_{[s]Q}$
2: $t_1 \leftarrow t_0 + X_{[s]Q}$
3: $t_2 \leftarrow X_{[s]Q} - t_0$
4: $t_2 \leftarrow t_2^2$
5: $t_2 \leftarrow t_2 \cdot X_{[s+1]Q}$
6: $t_0 \leftarrow Z_{[s]Q} + Z_{[s]Q}$
7: $t_0 \leftarrow A \cdot t_0$
8: $t_1 \leftarrow t_0 + t_1$
9: $t_3 \leftarrow x_Q \cdot X_{[s]Q}$
10: $t_3 \leftarrow t_3 + Z_{[s]Q}$
11: $t_1 \leftarrow t_1 \cdot t_3$
12: $t_0 \leftarrow t_0 \cdot Z_{[s]Q}$
13: $t_1 \leftarrow t_1 - t_0$
14: $t_1 \leftarrow t_1 \cdot Z_{[s+1]Q}$
15: $Y_{[s]Q} \leftarrow t_1 - t_2$
16: $t_0 \leftarrow y_Q + y_Q$
17: $t_0 \leftarrow t_0 \cdot Z_{[s]Q}$
18: $t_0 \leftarrow t_0 \cdot Z_{[s+1]Q}$
19: $X_{[s]Q} \leftarrow t_0 \cdot X_{[s]Q}$
20: $Z_{[s]Q} \leftarrow t_0 \cdot Z_{[s]Q}$

B.2 The case of the three-point ladder

Algorithm 5 Recovering the Y-coordinate after the t-th iteration of the three-point ladder

Input:$[2^t]Q = (X_0 : Z_0)$, $P + [s]Q = (X_1 : Z_1)$, $[2^t - s]Q - P = (X_2 : Z_2)$ to compute $P + [2^t + s]Q = (X_3 : Z_3)$
Output: $P + [s]Q = (X_1 : Y_1 : Z_1)$

1: $t_0 \leftarrow X_2 \cdot Z_3$
2: $t_1 \leftarrow X_3 \cdot Z_2$
3: $t_0 \leftarrow t_0 - t_1$
4: $t_1 \leftarrow X_0 \cdot Z_1$
5: $t_1 \leftarrow X_1 - t_1$
6: $t_1 \leftarrow t_1^2$
7: $Y_1 \leftarrow t_0 \cdot t_1$
8: $t_0 \leftarrow Y_0 + Y_0$
9: $t_0 \leftarrow t_0 + t_0$
10: $t_0 \leftarrow t_0 \cdot Z_1$
11: $t_0 \leftarrow t_0 \cdot Z_2$
12: $t_0 \leftarrow t_0 \cdot Z_3$
13: $X_1 \leftarrow t_0 \cdot X_1$
14: $Z_1 \leftarrow t_0 \cdot Z_1$

C Point Addition

Algorithm 6 is used to add a point P represented in projective coordinate to a point Q represented in affine coordinate, and output the result $P+Q = (X_{P+Q} : Z_{P+Q})$ on the elliptic curve E_0.

Algorithm 6 Point differential addition

Input:$(X_P : Y_P : Z_P)$ and (x_Q, y_Q)
Output: $(X_{P+Q} : Z_{P+Q})$

1: $t_0 \leftarrow x_Q \cdot Z_P$
2: $t_1 \leftarrow X_P - t_0$
3: $t_1 \leftarrow t_1^2$
4: $Z_{P+Q} \leftarrow Z_P \cdot t_1$
5: $t_0 \leftarrow X_P + t_0$
6: $t_1 \leftarrow t_0 \cdot t_1$
7: $t_0 \leftarrow y_Q \cdot Z_P$
8: $t_0 \leftarrow Y_P - t_0$
9: $t_0 \leftarrow t_0^2$
10: $t_0 \leftarrow t_0 \cdot Z_P$
11: $X_{P+Q} \leftarrow t_0 - t_1$

Algorithm 7 is used to add a point P represented in projective coordinate to a point Q represented in affine coordinate, and output the result $P + Q = (X_{P+Q} : Y_{P+Q} : Z_{P+Q})$ on the elliptic curve E_6.

Algorithm 7 Point differential addition

Input:$(X_P : Y_P : Z_P)$ and (x_Q, y_Q)
Output: $(X_{P+Q} : Y_{P+Q} : Z_{P+Q})$

1: $t_0 \leftarrow X_P + Z_P$
2: $t_0 \leftarrow t_0 + Z_P$
3: $t_1 \leftarrow x_Q + 2$
4: $t_2 \leftarrow y_Q \cdot Z_P$
5: $t_2 \leftarrow t_2 - y_Q$
6: $t_3 \leftarrow t_2^2$
7: $t_4 \leftarrow t_1 \cdot Z_P$
8: $t_4 \leftarrow t_4 - t_0$
9: $t_5 \leftarrow t_4^2$
10: $t_6 \leftarrow t_5 \cdot t_4$
11: $t_7 \leftarrow t_0 \cdot t_5$
12: $t_8 \leftarrow t_3 \cdot Z_P$
13: $t_8 \leftarrow a - t_6$
14: $t_8 \leftarrow a - t_7$
15: $t_8 \leftarrow a - t_7$
16: $X_{P+Q} \leftarrow t_4 \cdot t_8$
17: $t_9 \leftarrow t_6 \cdot Y_P$
18: $Y_{P+Q} \leftarrow t_7 - t_8$
19: $Y_{P+Q} \leftarrow t_2 \cdot Y_{P+Q}$
20: $Y_{P+Q} \leftarrow Y_{P+Q} - t_9$
21: $Z_{P+Q} \leftarrow t_6 \cdot Z_P$
22: $X_{P+Q} \leftarrow X_{P+Q} - Z_{P+Q}$
23: $X_{P+Q} \leftarrow X_{P+Q} - Z_{P+Q}$