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
[2] The author is with School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou 510006, P.R. China
[3] The authors are with Guangdong Key Laboratory of Information Security, Guangzhou 510006, P.R. China

Abstract. Supersingular isogeny Diffie-Hellman (SIDH) is attractive for 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. We present several tricks to accelerate key generation of SIDH by precomputing few elements in the base field. We also point out that the main ideas of Costello et al. and Faz-Hernández et al. to improve the ladder performance of SIDH, of which the starting curve is $E_0 : y^2 = x^3 + x$, could be still utilized for the current SIDH protocol. Our experimental results show that our work is more efficient than the current state-of-the-art implementation, which is favorable when the memory resources are limited.

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

1 Introduction

Supersingular isogeny Diffie-Hellman (SIDH) [10] has been regarded as one of the most attractive post-quantum protocols during the last decade because of its small public key size and high security. Up to now, supersingular isogeny key encapsulation (SIKE), which is based on SIDH, still remains active in the NIST post-quantum standardization process. 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 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 and isogeny computation (including isogeny construction and isogeny evaluation at points) are dominant. Although the latter one spends longer time, the optimization of the three-point ladder is still meaningful to improve the performance of SIDH.

Jao and De Feo [10] 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 and P, Q are points defined on the Montgomery curve $E_0 : y^2 = x^3 + x$. The three-point ladder was first improved by Costello et al. [5] by choosing torsion bases to execute base field operations of computing $P + [s]Q$ as possible. Faz-Hernández et al. [8] proposed a new three-point ladder, offering a saving of one differential addition per iteration. In addition, they pointed out that the three-point ladder could be further improved, but it requires large memory requirement, making it hard to be applied when the storage is limited. There is no doubt that setting E_0 as the starting curve brings perfect instantiation of SIDH. Unfortunately, Costello et al. observed that the distortion map of E_0 reduces the entropy of the private and public keys [6]. Hence, 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 relatively heavy computational cost of the three-point ladder.

In this paper, we mainly consider torsion bases used for public-key compression of SIDH [14]. Our contributions are as follows:

- We present Method 1 to speed up the three-point ladder for the case of Alice, requiring an element in the base field to be stored. We also point out that the acceleration techniques mentioned in [8] can be adapted into the current SIDH, as we present in Method 2.
- We show that the method of computing kernel generators proposed in [5], could be still employed to improve the current key generation of SIDH for the case of Bob, as we present in Method 3. Besides, similar to Method 2, we present Method 4 to further improve the performance, with a previous knowledge of a look-up table.

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 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 $y^2 = x^3 + Ax^2 + x$ is denoted by E_A, and $p = 2^{e_2}3^{e_3} - 1$ is a prime satisfying $2^{e_2} \approx 3^{e_3} \approx p^{\frac{1}{2}}$. Denote $r_A = 2^{e_2}$ and $r_B = 3^{e_3}$ for simplicity. In addition, let (x_P, y_P) and $(X_P : Y_P : Z_P)$ denote the affine and projective coordinates of the point P, 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 [18].

Let $\deg(\phi)$ be the degree of ϕ as a rational function, and $ker(\phi)$ is the kernel of ϕ. The isogeny ϕ is called separable if it satisfies $\deg(\phi) = \#ker(\phi)$. A separable isogeny of degree ℓ is abbreviated as ℓ-isogeny.

Two elliptic curves E and E' are said to be ℓ-isogenous over $\mathbb{F}_q$ if there exists an isogeny $\phi : E \rightarrow E'$ defined over $\mathbb{F}_q$. Deciding whether two curves are ℓ-isogenous [10, Problem 5.1] is considered to be a difficult problem [1, 3, 12], which mainly guarantees the security of the SIDH/SIKE protocol. See [16] for details of the structure of the isogeny graphs.

2.2 Montgomery ladder

The Montgomery ladder was first proposed by Montgomery [13] in 1987, aiming to compute multiples of points for a given point. Compared to the double-and-add algorithm [7], the Montgomery ladder can avoid side-channel analysis [11]. Furthermore, the Montgomery ladder is able to compute the x-coordinates of multiples of points efficiently thanks to the following equations:

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

where P and Q are two points over the Montgomery curve. It is obvious 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, projective coordinates are preferred for efficiency.

In each iteration, the Montgomery ladder executes one point doubling and one differential addition, denoted by **dadd**. It costs **five** 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.

Algorithm 1 Montgomery ladder

```
Input: $P = (X_P : Z_P) \in E_A$, $s = (s_{\ell-1} \cdots s_1 s_0)_2$ and $A24 = (A + 2)/4$
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:         $(X_2, Z_2, X_1, Z_1) =$ dadd$(X_2, Z_2, X_1, Z_1, X_3, Z_3, A24)$
5:     else
6:         $(X_1, Z_1, X_2, Z_2) =$ dadd$(X_1, Z_1, X_2, Z_2, X_3, Z_3, A24)$
7:     end if
8: end for
9: return $X_2$, $Z_2$
```

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 computes the latter). Afterwards, one can recover the y-coordinate of $[s]Q$ by the Okeya-Sakurai formula [15]:

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

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

We present Algorithm 4 in Appendix B, which is used to recover $[s]Q$, while Algorithm 6 in Appendix C is pseudocode of differential addition.

2.3 Three-point ladder algorithm

Instead of the Montgomery ladder, Jao, De Feo and Plût [10] proposed a three-point ladder to compute $P+[s]Q$, where P and Q are points on the Montgomery curve. The superiority of three-point ladder is that $P + [s]Q$ can be computed directly. That is to say, there is no need recovering the y-coordinate of any point. The three-point ladder was later improved in [8, Algorithm 2].

Algorithm 2 Three-point ladder [8]

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 $A24 = (A + 2)/4$
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:         $(X_1, Z_1, X_3, Z_3) = \mathbf{dadd}(X_1, Z_1, X_3, Z_3, X_2, Z_2, A24)$
5:     else
6:         $(X_1, Z_1, X_2, Z_2) = \mathbf{dadd}(X_1, Z_1, X_2, Z_2, X_3, Z_3, A24)$
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 computation of Q does not depend on the secret key s. Faz-Hernández et al. [8] pointed out that a look-up table can be precomputed to reduce the computational cost:

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

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, while the size of the table is relatively large.

Remark 1. Similar to the Montgomery ladder, one can recover the y-coordinate of $P + [k]Q$ after the last iteration if $[2^\ell]Q$ is precomputed. See 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 $\mathbb{F}_{p^2} = \mathbb{F}_p[i]/\langle i^2 + 1\rangle$. 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 $\langle P_A, Q_A\rangle = E[r_A]$ and $\langle P_B, Q_B\rangle = E[r_B]$. 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 [19] 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 calculates $\phi_B(P_A)$, $\phi_B(Q_A)$. Finally, he sends $\phi_B(P_A), \phi_B(Q_A)$ as well as the image curve parameter B to Alice.

Once Alice receives the public key from Bob, Alice 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 S'_A 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 the corresponding isogeny ϕ'_B. 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 [2, 4, 9, 14, 17, 20].

3 Optimization of Kernel Generator Computation

In this section, we show how to improve the kernel generation of isogeny in key generation of SIDH. We consider the torsion bases selected in [14], which can be utilized to speed up key generation of the compressed version of SIDH as well.

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

3.1 Case of Alice

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

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

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

When implementing the three-point ladder (Algorithm 2) to compute the point $P_A + [s_A]Q_A$, 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, after executing the t-th iteration, $[2^t]Q_A = (X_1 : Z_1)$ always satisfies the following relation:

$$\frac{X_1}{Z_1} \in \mathbb{F}_p.$$

This is because the x-coordinate of $[2^t]Q_A$ is in $\mathbb{F}_p$, where $t = 1, \cdots, e_2 - 1$.

Therefore, Alice can precompute $x_{[2]Q_A}$ (which is an element in $\mathbb{F}_p$) to speed up the computation of kernel generators. After the first iteration of the three-point ladder, she could substitute $(x_{[2]Q} : 1)$ for the values of $(X_1 : Z_1)$ such that the subsequent operations related to X_1 and Z_1 could be executed in the base field $\mathbb{F}_p$ as possible.

In general, Alice can use her secret key s_A to compute $P_A + [s_A]Q_A$ as follows:

Method 1:

- Execute **dadd** once to attain the coordinates $(X_1, Z_1, X_2, Z_2, X_3, Z_3)$ of the tuple $\{[2]Q_A, P_A + Q_A, Q_A - P_A\}$ or $\{[2]Q_A, P_A, [2]Q_A - P_A\}$, with respect to the last bit of the binary representation;
- Replace $(X_1 : Z_1)$ with $(x_{[2]Q} : 1)$;
- Continue executing the three-point ladder as usual.

As mentioned in Section 2.3, Alice could further optimize the implementation by storing a look-up table $T(Q)$.

Method 2:

- For each iteration (except the last two), execute a differential addition with the help of the look-up table $T(Q)$;
- Set $(X_1 : Z_1) = (1 : 1)$, and then execute a differential addition;
- Set $(X_1 : Z_1) = (0 : 1)$, and then execute a differential addition.

Since $[2]Q_A \in E(\mathbb{F}_p)$, this optimization requires additional storage, but it is unbearable for storage constrained environments. Furthermore, this method is not as efficient as the method mentioned in [8, Section 3.2.2] because there is no obvious way to perform the three-point ladder in the base field.

Remark 2. There does not exist a point of order r_A on $E_6(\mathbb{F}_p)$ because $E_6(\mathbb{F}_p)[r_A]$ is isomorphic to $\mathbb{Z}/2^{e_2-1}\mathbb{Z} \times \mathbb{Z}/2\mathbb{Z}$. Hence, it is impossible to find a r_A-torsion basis such that one of the torsion points belongs to $E_6(\mathbb{F}_p)$.

To sum up, the estimates given in Table 1 show the computational cost for each iteration. The table shows that Methods 1 and 2 are able to improve the three-point ladder, and the implementation of the latter method performs better, but it requires large storage.

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

Method	Cost estimates
Current SIDH [2]	$6\mathbf{M}+4\mathbf{S} \approx (6\times3 + 4\times2)\ \mathbf{m} = 26\mathbf{m}$
Method 1	$3\mathbf{M}+2\mathbf{S}+5\mathbf{m}+2\mathbf{s} \approx (3\times3+2\times2+5+2\times0.8)\ \mathbf{m} = 19.6\mathbf{m}$
Method 2	$2\mathbf{M}+2\mathbf{S}+2\mathbf{m} \approx (2\times3+2\times2+2)\ \mathbf{m} = 12\mathbf{m}$

3.2 Case of Bob

To compress public keys faster, Naehrig and Renes 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 of the isogeny. Similar to the ideas proposed in [5], 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 the X-coordinates and Z-coordinates of $[s_B]P_3$ and $[s_B + 1]P_3$, respectively;
- Utilize the Okeya-Sakurai formula (1) to recover the projective coordinates of $[s_B]P_3$;
- Compute $[s_B]P_3 + Q_3$;
- Complete the evaluation of the isogeny ϕ_2 at $[s_B]P_3 + Q_3$.

It is obvious 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_B is r_B and $\mathbf{gcd}(2, r_B) = 1$, the order of the point S_B is exactly r_B. In this case, only the point P_3 in affine coordinates should be stored (Q_3 could be recovered by $\psi(P_3)$).

The reason why Bob executes the Montgomery ladder is that $Q_3 \in E_0(\mathbb{F}_p)$, i.e., all the operations are implemented in the base field. Therefore, Bob could compute the point S_B much more efficient than before. It is worth noting that the modification of the kernel form of the isogeny does not reduce the size of key space because P_B is also a point of order r_B.

Set $Q_3' = (1, \sqrt{2}) \in E(\mathbb{F}_p)$ (Note that 2 is a square in $\mathbb{F}_p$ because $p \equiv 7(mod\ 8)$). Analogous to Alice, 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. In addition, Bob needs to precompute $\sqrt{2}$ and P_3' to execute the three-point ladder in the base field. Moreover, $[2^{\lceil \log r_B \rceil + 2}]P_3$ is also needed to recover the Y-coordinate of $[4s_B]P_3$. In this light, large memory is required. The main procedure is as follows:

Method 4:

- Use the three-point ladder to compute the X-coordinates and Z-coordinates of $[s_B]P_3 + Q_3'$ and $[2^{\lceil \log r_B \rceil} - s_B]P_3 + Q_3'$, respectively;
- Compute $[4]([s_B]P_3 + Q_3') = [4s_B]P_3$ and $[4]([2^{\lceil \log r_B \rceil} - s_B]P_3 + Q_3') = [2^{\lceil \log r_B \rceil + 2} - 4s_B]P_3$;
- Utilize Algorithm (5) to recover $[4s_B]P_3$ in projective coordinates;
- Compute $[4s_B]P_3 + Q_3$;
- Complete the evaluation of the isogeny ϕ_2 at $[4s_B]P_3 + Q_3$.

Remark 3. This modification of the kernel of the isogeny does not change the size of key space since $\mathbf{gcd}(r_B, 4) = 1$.

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. We can predict that Method 3 improves the performance obviously, and so Method 4 does. However, based on Method 3, the acceleration effect of Method 4 may be not so significant.

4 Implementation

In this section we present the implementation of key generation of SIDH 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 depends on the implementation of the ladder.

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

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

Our implementation makes use of the SIDH C library[4]. The following experimental results have been obtained by utilizing 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 by using different methods in comparison with the current SIDH. The results are reported in Table 3.

Table 3. Performance comparison of key generation of SIDH by using different methods. All timings are present in millions of clock cycles.

Setting	Alice's key generation			Bob's key generation		
	Current SIDH[2]	Method 1	Method 2	Current SIDH[2]	Method 3	Method 4
SIKEp434	3.25	3.05	2.92	3.61	3.25	3.17
SIKEp503	4.63	4.32	4.13	5.12	4.50	4.45
SIKEp610	9.52	8.87	8.51	9.55	8.27	8.21
SIKEp751	13.87	12.93	12.46	15.79	14.02	13.85

As can be seen in Table 3, when the storage is constrained, the performance of ours is 6.56%∼ 7.35% faster than the current SIDH for the case of Alice, and 11.08%∼ 15.48% faster for the case of Bob. When the storage permits, it performs better with a previous knowledge of a look-up table, especially for the case of Alice.

In Table 4 we report the storage 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. Therefore, we suppose that Methods 1 and 3 would be more suitable for memory constrained environments.

5 Conclusion

In this paper, we review the improvement for the ladder in SIDH when the starting curve is E_0, and show that these techniques can still be used to speed up key generation of the current SIDH. Compared to the methods which require large storage, we utilize several tricks to make SIDH faster by storing few elements

[4] `https://github.com/Microsoft/PQCrypto-SIDH`

Table 4. Storage requirements (in KiB) for Method 2 and Method 4

Setting	SIKEp434	SIKEp503	SIKEp610	SIKEp751
Alice	29.3	39.4	58.6	87.5
Bob	30.6	40.9	59.9	90.8
Total	59.9	80.3	118.5	178.2

in the base field. Our new idea may make SIDH/SIKE more attractive in the condition of shortage of memory.

Acknowledgements

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. 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)
2. 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`
3. Costello, C.: The Case for SIKE: A Decade of the Supersingular Isogeny Problem. Cryptology ePrint Archive, Report 2021/543 (2021), `https://eprint.iacr.org/2021/543`
4. 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)
5. 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)
6. 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)
7. Donald E. Knuth: The Art of Computer Programming, v.2. Seminumerical algorithms. Addison-Welsley, 2nd edition (1981)
8. 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)

9. 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)
10. 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)
11. 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)
12. 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)
13. Montgomery, P.L.: Speeding the Pollard and elliptic curve methods of factorization. Mathematics of Computation **48**, 243–264 (1987)
14. 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)
15. 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)
16. 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)
17. Pereira, G., Doliskani, J., Jao, D.: x-only point addition formula and faster compressed SIKE. Journal of Cryptographic Engineering pp. 1–13 (2020)
18. Silverman, J.H.: The Arithmetic of Elliptic Curves, 2nd Edition. Graduate Texts in Mathematics. Springer (2009)
19. Vélu, J.: Isogénies entre courbes elliptiques. C. R. Acad. Sci., Paris, Sér. A **273**, 238–241 (1971)
20. 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

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: $X_Q \leftarrow Z_{P-Q} \cdot X_Q$
```

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

After the t-th iteration of the three-point ladder, Bob can utilize $[2^t]Q = (X_0 : Z_0)$, $P + [s]Q = (X_1 : Z_1)$, $[2^t - s]Q - P = (X_2 : Z_2)$ to compute $P + [2^t + s]Q = (X_3 : Z_3)$, and then recover $[4s_B]P_3 + Q_3$ in projective coordinates by the following algorithm:

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 affine coordinates to a point Q represented in projective coordinates, and output the result $P+Q = (X_{P+Q} : Z_{P+Q})$.

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: $X_{P+Q} \leftarrow t_0 - t_1$
```