(Short Paper) A Faster Constant-Time Algorithm of CSIDH Keeping Two Points

Hiroshi Onuki[1(✉)], Yusuke Aikawa[1,2], Tsutomu Yamazaki[3], and Tsuyoshi Takagi[1]

[1] Department of Mathematical Informatics, University of Tokyo, Tokyo, Japan
{onuki,takagi}@mist.i.u-tokyo.ac.jp
[2] Department of Mathematics, Hokkaido University, Sapporo, Japan
yusuke@math.sci.hokudai.ac.jp
[3] Graduate School of Mathematics, Kyushu University, Fukuoka, Japan
yamazaki.tsutomu.890@s.kyushu-u.ac.jp

Abstract. At ASIACRYPT 2018, Castryck, Lange, Martindale, Panny and Renes proposed CSIDH, which is a key-exchange protocol based on isogenies between elliptic curves, and a candidate for post-quantum cryptography. However, the implementation by Castryck et al. is not constant-time. Specifically, a part of the secret key could be recovered by the side-channel attacks. Recently, Meyer, Campos, and Reith proposed a constant-time implementation of CSIDH by introducing dummy isogenies and taking secret exponents only from intervals of non-negative integers. Their non-negative intervals make the calculation cost of their implementation of CSIDH twice that of the worst case of the standard (variable-time) implementation of CSIDH. In this paper, we propose a more efficient constant-time algorithm that takes secret exponents from intervals symmetric with respect to the zero. For using these intervals, we need to keep two torsion points on an elliptic curve and calculation for these points. We implemented our algorithm by extending the implementation in C of Meyer et al. (originally from Castryck et al.). Then our implementation achieved 152.8 million clock cycles, which is about 29.03% faster than that of Meyer et al.

Keywords: CSIDH · Post-quantum cryptography · Isogeny-based cryptography · Constant-time implementation · Supersingular elliptic curve isogenies

1 Introduction

RSA and elliptic curve cryptosystems will no longer be secure once a large-scale quantum computer is built. Due to this, the importance of post-quantum cryptography (PQC) has increased. In 2017, the National Institute of Standards and Technology (NIST) started the process of PQC standardization [18]. Candidates for the NIST PQC standardization include supersingular isogeny key encapsulation (SIKE) [14], which is a scheme based on isogenies between elliptic curves.

N. Attrapadung and T. Yagi (Eds.): IWSEC 2019, LNCS 11689, pp. 23–33, 2019.
https://doi.org/10.1007/978-3-030-26834-3_2

SIKE is a variant of supersingular isogeny Diffie-Hellman (SIDH), which was proposed by Jao and De Feo [12] in 2011. SIDH uses isogenies between supersingular elliptic curves over a finite field. SIDH achieves an efficient key-exchange but needs to send torsion points of an elliptic curve as supplementary information. Attacks using this information are discussed in by Galbraith, Petit, Shani, and Ti [11] and Petit [20].

Isogeny-based cryptography was first proposed by Couveignes [8] in 1997 and independently rediscovered by Rostovtsev and Stolbunov [21,23]. Their proposed scheme is a Diffie-Hellman-style key-exchange based on isogenies between ordinary elliptic curves over a finite field and typically called CRS. CRS does not need to send any point of elliptic curves, therefore the attacks to SIDH, which is based on information of points of elliptic curves, cannot be applied to CRS. However, even after optimizations by De Feo, Kieffer, and Smith [9], CRS is much slower than SIDH. In 2018, Castryck, Lange, Martindale, Panny, and Renes [3] proposed commutative SIDH (CSIDH), which adopts supersingular elliptic curves to the CRS scheme. They used supersingular elliptic curves over a finite prime field $\mathbb{F}_p$ and their endomorphism rings over $\mathbb{F}_p$. Since the number of $\mathbb{F}_p$-rational points on a supersingular elliptic curve E over $\mathbb{F}_p$ is $p+1$, one can choose p such that $\#E(\mathbb{F}_p)$ has many small prime factors. This allows CSIDH to compute isogenies faster than CRS.

However, the computational time in the proof-of-concept implementation by Castryck et al. depends on the associated secret key, so their implementation of CSIDH is not side-channel resistant. Recently, Meyer, Campos, and Reith [15] proposed a constant-time implementation of CSIDH and several speedup techniques for their implementation. They achieved the constant-time implementation by using dummy isogenies and by changing intervals of key elements from $[-m, m]$ to $[0, 2m]$, where $m \in \mathbb{N}$. Consequently, their constant-time implementation needs to calculate each degree isogeny $2m$ times, while the worst case of the variable-time CSIDH needs only m times. Therefore, the computational cost of their constant-time implementation is twice as that of the worst case of the variable-time CSIDH. The constant-time implementation in [15] allows variance of the computational time of their implementation with randomness that does not relate to secret information.

On the other hand, implementations which do not allow such variance are proposed by Bernstein, Lange, Martindale, and Panny [2] and Jalali, Azarderakhsh, Kermani, and Jao [13]. The implementation in [2] is for evaluating the performance of quantum attacks for CSIDH. It must not have branches in order to compute in superposition on quantum computers. The implementation in [13] is for classical computers, but it has no branches. As a result, it is slower than the implementation in [15]. We discuss the differences in these implementations in Sect. 3.2.

In this paper, we propose a new constant-time implementation, which is faster than the constant-time implementation by Meyer et al. [15]. Our implementation is "constant-time" in the same sense as that of [15]. In other words, the computational time and the order of scalar multiplications and isogenies in

our implementation do not depend on a secret key. We use the dummy isogenies proposed by [15]. but do not change the key intervals of CSIDH, i.e., we use the interval $[-m, m]$. To achieve a constant-time implementation without changing the key intervals, we need to keep two torsion points of both $E[\pi-1]$ and $E[\pi+1]$ and calculation associated with these points, where π is the Frobenius endomorphism of an elliptic curve E. As a result, our implementation needs almost twice as many scalar multiplications on elliptic curves and twice as many calculations of images of points under isogenies as the worst case of the variable-time CSIDH. However, the number of calculations of the images of curves is the same as in the worst case of the variable-time CSIDH, and scalars in a part of additional scalar multiplications on elliptic curves are smaller. Therefore, our implementation is faster than the implementation in [15]. We implemented our algorithm in C and compared its cycle count and running time with those of the implementation in [15]. Our experiment shows that the cycle count of our implementation is 29.03% less than that of the implementation in [15].

Organization. The rest of this paper is organized as follows. The following section describes CSIDH. Section 3 explains a constant-time implementation in [15]. and briefly introduces constant-time implementations based on another definition. We give the details of our new constant-time implementation of CSIDH in Sect. 4. In Sect. 5, we present experimental results. We conclude our work in Sect. 6.

2 CSIDH

In this section, we overview the protocol of CSIDH and its mathematical backgrounds. For more details, see Castryck et al. [3].

2.1 Protocol of CSIDH

For describing the protocol of CSIDH, we define the following notations. Let p be a prime number, $\mathcal{CL}(\mathbb{Z}[\sqrt{-p}])$ the ideal class group of $\mathbb{Z}[\sqrt{-p}]$ and $\mathcal{ELL}_{\mathbb{F}_p}(\mathbb{Z}[\sqrt{-p}])$ a set of $\mathbb{F}_p$-isomorphism classes of supersingular elliptic curves whose endomorphism ring is isomorphic to $\mathbb{Z}[\sqrt{-p}]$. Then we can define an action

$$\mathcal{CL}(\mathbb{Z}[\sqrt{-p}]) \times \mathcal{ELL}_{\mathbb{F}_p}(\mathbb{Z}[\sqrt{-p}]) \to \mathcal{ELL}_{\mathbb{F}_p}(\mathbb{Z}[\sqrt{-p}]),\ (\mathfrak{a}, E) \mapsto \mathfrak{a} * E.$$

We call this action the class group action. The details of these notations and the action are described in the next subsection. CSIDH is a Diffie-Hellman style key exchange as follows:

Alice and Bob share an elliptic curve $E_0 \in \mathcal{ELL}_{\mathbb{F}_p}(\mathbb{Z}[\sqrt{-p}])$ as a public parameter. Alice chooses an ideal $\mathfrak{a} \in \mathcal{CL}(\mathbb{Z}[\sqrt{-p}])$ as her secret key and sends the curve $\mathfrak{a} * E$ to Bob as her public key. Bob proceeds in the same way by choosing a secret key $\mathfrak{b} \in \mathcal{CL}(\mathbb{Z}[\sqrt{-p}])$. Then, both parties can compute the shared secret $\mathfrak{ab} * E = \mathfrak{ba} * E$. Note that $\mathcal{CL}(\mathbb{Z}[\sqrt{-p}])$ is commutative.

2.2 Ideal Class Group

Let p be a large prime of the form $4\ell_1 \cdots \ell_n - 1$, where $\ell_1, \ldots, \ell_n$ are small distinct odd primes. Let $E \in \mathcal{ELL}_{\mathbb{F}_p}(\mathbb{Z}[\sqrt{-p}])$ and π be its p-th power Frobenius endomorphism. Since E is supersingular, the primes ℓ_i split in $\mathbb{Z}[\sqrt{-p}]$ as $(\ell_i) = \mathfrak{l}_i\bar{\mathfrak{l}}_i$, where $\mathfrak{l}_i = (\ell_i, \pi - 1)$ and $\bar{\mathfrak{l}}_i = (\ell_i, \pi + 1)$. It can be seen that the actions of $\mathfrak{l}_i$ and $\bar{\mathfrak{l}}_i$ can be computed efficiently. In the ideal class group, $\bar{\mathfrak{l}}_i$ is the inverse of $\mathfrak{l}_i$, so we can compute the action of an ideal of the form $\mathfrak{l}_1^{e_1} \cdots \mathfrak{l}_n^{e_n}$, $e_1, \ldots, e_n \in \mathbb{Z}$ by the composition of the actions of $\mathfrak{l}_i$ and $\bar{\mathfrak{l}}_i$. Castryck et al. [3] showed that under some heuristics, $\mathfrak{l}_1^{e_1} \cdots \mathfrak{l}_n^{e_n}$, $-m \leq e_i \leq m$ represent uniformly "almost" all the ideal classes in $\mathcal{CL}(\mathbb{Z}[\sqrt{-p}])$, where $m \in \mathbb{N}$ such that $(2m+1)^n \geq \#\mathcal{CL}(\mathbb{Z}[\sqrt{-p}])$. We denote the exponents (e_i) by secret exponents.

3 Previous Works for Constant-Time Implementation of CSIDH

In this section, we explain a constant-time implementation proposed by Meyer et al. [15] and briefly describe related works.

3.1 Constant-Time Implementation

As already mentioned by Castryck et al. [3], their proof-of-concept implementation is not side-channel resistant because the computational time for a public key and a shared secret depends on the associated secret key. To solve this problem, Meyer et al. [15] proposed a constant-time implementation of CSIDH. According to [15], "a constant-time implementation" means an implementation whose computational time and order of scalar multiplications of each size and isogenies of each degree do not depend on a secret key. Their constant-time implementation is described in Algorithm 1.

To achieve a constant-time implementation, they used dummy isogenies and changed the intervals of the integer key elements from $[-m, m]$ to $[0, 2m]$. We explain these techniques below. In this algorithm, one samples a point on an elliptic curve by using Elligator [1] for CSIDH, which was proposed by Bernstein, Lange, Martindale, and Panny [2]. Elligator enables us to generate x-coordinates of points with suitable y-coordinate by computing only one Legendre symbol. For the details, see Bernstein et al. [2].

Dummy Isogenies. It seems that one should compute a constant number of isogenies of each degree ℓ_i and only use the ones required by the secret key. The function for dummy isogenies is designed to use the same operations on $\mathbb{F}_p$ as the function for isogenies. For more details, see [15,16].

Changing the Key Intervals. By using dummy isogenies, the number of isogeny computations is fixed. However, this is not sufficient to achieve a constant-time implementation, since the sizes of the scalar multiplications vary in accordance with the signs of secret exponents. To remove this effect, Meyer et al. [15] proposed changing the intervals from $[-m, m]$ to $[0, 2m]$.

Algorithm 1. Constant-time evaluation of the class group action in CSIDH [15]

Input: $A \in \mathbb{F}_p$, $m \in \mathbb{N}$, a list of integers $(e_1, \ldots, e_n)$ s.t. $0 \leq e_i \leq 2m$ for $i = 1, \ldots, n$, and distinct odd primes $\ell_1, \ldots, \ell_n$ s.t. $p = 4\prod_i \ell_i - 1$.

Output: $B \in \mathbb{F}_p$ s.t. $E_B = (\mathfrak{l}_1^{e_1} \cdots \mathfrak{l}_n^{e_n}) * E_A$, where $\mathfrak{l}_i = (\ell_i, \pi - 1)$ for $i = 1, \ldots, n$, and π is the p-th power Frobenius endomorphism of E_A.

1: Set $e'_i = 2m - e_i$ for $i = 1, \ldots, n$.
2: **while** some $e_i \neq 0$ or $e'_i \neq 0$:
3: Set $S = \{i \mid e_i \neq 0 \text{ or } e'_i \neq 0\}$.
4: Set $k = \prod_{i \in S} \ell_i$.
5: Generate a point $P \in E_A[\pi - 1]$ by Elligator.
6: Let $P \leftarrow [(p+1)/k]P$.
7: **for** $i \in S$:
8: Set $Q = [k/\ell_i]P$.
9: **if** $Q \neq \infty$: /* **branch not involving secret information** */
10: **if** $e_i \neq 0$: /* **branch involving secret information** */
11: Compute an isogeny $\varphi : E_A \to E_B$ with $\ker\varphi = \langle Q \rangle$.
12: Let $A \leftarrow B$, $P \leftarrow \varphi(P)$, and $e_i \leftarrow e_i - 1$.
13: **else**
14: Dummy computation.
15: Let $A \leftarrow A$, $P \leftarrow [\ell_i]P$, and $e'_i \leftarrow e'_i - 1$.
16: **end if**
17: **end if**
18: Let $k \leftarrow k/\ell_i$.
19: **end for**
20: **end while**
21: **return** A.

3.2 Constant-Time Implementations Based on Another Definition

As we stated above, Meyer et al. [15] allow variance of the computational time of their implementation with randomness that does not relate to secret information (caused by the branch **if** $Q \neq \infty$ in line 9 in Algorithm 1). On the other hand, constant-time implementations that do not allow this variance are known. Bernstein et al. [2] constructed a constant-time implementation of CSIDH for evaluating the performance of quantum attacks. For calculating the class group actions in superposition on a quantum computer, a completely constant-time implementation is required. Therefore, their constant-time implementation has no branches (such as **if** branch). Jalali, Azarderakhsh, Kermani, and Jao [13] proposed a constant-time implementation for classical computers, which also

has no branches. As a result of removing all branches, these implementations are slower than that of [15]. We propose a constant-time implementation based on the definition in [15], i.e., our implementation allows branches which do not depend on secret information.

4 Our Constant-Time Implementation

In this section, we propose a new constant-time implementation that is faster than that of [15].

The constant-time implementation in [15]. requires the cost to be the same as that of calculating the action of the ideal class corresponding to secret exponents $(2m, \ldots, 2m)$. This cost is twice the cost corresponding to secret exponents $(m, \ldots, m)$, which is the worst case in the variable-time CSIDH. We mitigate the cost for achieving constant-time by using positive and negative secret exponents.

4.1 Basic Idea

To achieve a constant-time implementation without fixing the signs of secret exponents, we compute isogenies corresponding to positive and negative secret exponents in the same round in the while loop in Algorithm 1. This requires keeping two points of both $E[\pi - 1]$ and $E[\pi + 1]$ and computing scalar multiplications and images under isogenies for both points. This means that our new method needs almost twice as many scalar multiplications and twice as many computations of images of points per isogeny calculation (the reason we need "almost" twice as many scalar multiplications is explained later). However, it needs only one computation for an isogenous curve coefficient. Therefore, the cost of our method is less than twice of the worst case of the variable-time CSIDH. Combining this method and dummy isogenies of [15,16], we achieve a more efficient constant-time implementation.

4.2 Proposed Algorithm

Our constant-time implementation for computing the class group action is described in Algorithm 2.

In Algorithm 2, the points P_0 and P_1 are k-torsion of $E[\pi - 1]$ and $E[\pi + 1]$, respectively. The indicator s is the sign bit of a secret exponent e_i (line 8), i.e., $s = 0$ if $e_i \geq 0$ and $s = 1$ if $e_i < 0$. This can be computed by bit operations. For example, $s = e_i \gg 7$ if e_i is stored as a signed 8-bit integer. The point Q is ℓ_i-torsion of $E[\pi - 1]$ if $e_i \geq 0$ or of $E[\pi + 1]$ is $e_i < 0$ (line 9). Therefore, the algorithm computes the isogeny corresponding to the sign of e_i in line 13–17. Note that we need a scalar multiplication on P_{1-s} by ℓ_i in line 10 because the ℓ_i-torsion parts of P_0 and P_1 should drop in order to update k to k/ℓ_i. The ℓ_i-torsion part of P_s is Q and drops by the isogeny φ, since Q is in the kernel of φ. In contrast, the ℓ_i-torsion part of P_{1-s} does not drop by φ. We also note that we need to calculate this scalar multiplication even when $Q = \infty$, i.e., one fails

Algorithm 2. Our constant-time evaluation of the class group action in CSIDH

Input: $A \in \mathbb{F}_p$, $m \in \mathbb{N}$, a list of integers $(e_1, \ldots, e_n)$ s.t. $-m \leq e_i \leq m$ for $i = 1, \ldots, n$, and distinct odd primes $\ell_1, \ldots, \ell_n$ s.t. $p = 4\prod_i \ell_i - 1$.

Output: $B \in \mathbb{F}_p$ s.t. $E_B = (\mathfrak{l}_1^{e_1} \cdots \mathfrak{l}_n^{e_n}) * E_A$, where $\mathfrak{l}_i = (\ell_i, \pi - 1)$ for $i = 1, \ldots, n$, and π is the p-th power Frobenius endomorphism of E_A.

1: Set $e'_i = m - |e_i|$ for $i = 1, \ldots, n$.
2: **while** some $e_i \neq 0$ or $e'_i \neq 0$:
3: Set $S = \{i \mid e_i \neq 0 \text{ or } e'_i \neq 0\}$.
4: Set $k = \prod_{i \in S} \ell_i$.
5: Generate points $P_0 \in E_A[\pi - 1]$ and $P_1 \in E_A[\pi + 1]$ by Elligator.
6: Let $P_0 \leftarrow [(p+1)/k]P_0$ and $P_1 \leftarrow [(p+1)/k]P_1$.
7: **for** $i \in S$:
8: Set s the sign bit of e_i.
9: Set $Q = [k/\ell_i]P_s$.
10: Let $P_{1-s} \leftarrow [\ell_i]P_{1-s}$.
11: **if** $Q \neq \infty$: /* **branch not involving secret information** */
12: **if** $e_i \neq 0$: /* **branch involving secret information** */
13: Compute an isogeny $\varphi : E_A \to E_B$ with $\ker\varphi = \langle Q \rangle$.
14: Let $A \leftarrow B$, $P_0 \leftarrow \varphi(P_0)$, $P_1 \leftarrow \varphi(P_1)$, and $e_i \leftarrow e_i - 1 + 2s$.
15: **else**
16: Dummy computation.
17: Let $A \leftarrow A$, $P_s \leftarrow [\ell_i]P_s$, and $e'_i \leftarrow e'_i - 1$.
18: **end if**
19: **end if**
20: Let $k \leftarrow k/\ell_i$.
21: **end for**
22: **end while**
23: **return** A.

to obtain a generator of the kernel of an isogeny. The equation $Q = \infty$ means the ℓ_i-torsion part of P_s has already vanished but does not mean the ℓ_i-torsion part of P_{1-s} has vanished. Therefore, for updating k to k/ℓ_i, we need the scalar multiplication on P_{1-s} by ℓ_i. In contract, in the variable-time CSIDH algorithm, one calculates nothing when $Q = \infty$. This is why we said "we need "almost" twice as many scalar multiplications" in the previous subsection. However, the number of these additional scalar multiplications is much smaller than the total number of scalar multiplications. For example, it is about 2% of the total number of scalar multiplications in CSIDH-512, which is the parameter set for CSIDH proposed by Castryck et al. [3].

Remark 1. The same as in the implementation in [15], we use Elligator for CSIDH. It enables us to generate x-coordinates of P_0 and P_1 in line 5 in Algorithm 2 by computing only one Legendre symbol. For the details, see Bernstein et al. [2].

Remark 2. Our dummy isogeny includes a dummy calculation corresponding to evaluations of P_1 under φ not only of P_0 so that the calculation costs of lines 13–14 and lines 16–17 in Algorithm 2 are the same.

4.3 Security Comparison with the Implementation by Meyer et al.

We claim that the security of our implementation against side-channel attacks is equivalent to that of the implementation in [15]. Although Algorithm 2 contains a conditional branch on secret information, one can replace the branch by conditional swaps and implement it without conditional branches and memory accesses which depend on secret information.

Meyer et al. [15] claimed that their implementation is constant-time in the sense that it can prevent the two leakage scenarios they consider [15, §3]: timing leakage and power analysis. Timing leakage is leaking information on a secret key by the computational time. Power analysis measures the power consumption of the algorithm and determines blocks that represent the two main primitives in CSIDH, scalar multiplications, and isogeny computation. Their implementation prevents these leakage scenarios because the computational time and the order of scalar multiplications of each size and isogenies of each degree in their implementation do not depend on a secret key.

Our implementation also prevents the above two leakage scenarios. Its computational time does not depend on information on a secret key because of dummy isogenies. By calculating isogenies whose exponents have different signs in the same loop, the order of scalar multiplications of each size and isogenies of each degree do not depend on information on a secret key. Furthermore, our implementation has two branches, the same as the implementation in [15]. The first is **if** $Q \neq \infty$ in line 11 in Algorithm 2, which does not involve secret information and affects the computational time (the corresponding branch in the implementation in [15] is in line 9 in Algorithm 1). The second is **if** $e_i \neq 0$, line 12 in Algorithm 2, which involves secret information and does not affect the computational time (the corresponding branch in the implementation in [15] is in line 10 in Algorithm 1). This branch can be removed by using conditional swaps and implemented securely. See the code of [15], that is available at https://zenon.cs.hs-rm.de/pqcrypto/constant-csidh-c-implementation. We note that our implementation switches calculation for isogenies associated to positive and negative secret exponents by the indicator s in line 8 in Algorithm 2, which can be computed by bit operations. There are memory accesses which depend on the secret bit s in line 9–10 in Algorithm 2. But one can implement it securely by using a conditional swap to swap the values of P_0 and P_1. As a result, we conclude that our implementation is constant-time as that of [15].

5 Experimental Results

We implemented our algorithm with the speedup techniques proposed by Meyer et al. [15] in C. For the parameters used for the speedup techniques, see our full paper [19]. Our code is based on the code of [15][1]. (originally from

[1] The code by Meyer et al. is available for download at https://zenon.cs.hs-rm.de/pqcrypto/constant-csidh-c-implementation. The commit ID of the version we used is 7fc2abdd, the latest version on 15 Feb, 2019.

Castryck et al. [3]). Table 1 shows the cycle counts and running times for our implementation and that in [15] For the implementation in [15], we used the code on which our code is based (the code in the footnote 1). We ran both codes on an Intel Xeon Gold 6130 Skylake processor running Ubuntu 16.04.5 LTS. Our implementation has 29.03% fewer clock cycles than the implementation in [15], which is almost the same as the reduction ratio expected by the evaluation of our cost model.

Table 1. Performance comparison, averaged over 10,000 runs.

	Clock cycles $\times 10^6$	Wall clock time
Implementation in [15]	215.3	102.742 ms
Our implementation	**152.8**	**72.913 ms**

6 Conclusion

We improved a constant-time implementation of commutative supersingular isogeny Diffie-Hellman (CSIDH), which is isogeny-based Diffie-Hellman-style key exchange and a candidate for post-quantum cryptography. Our implementation is based on the constant-time implementation in Meyer et al. [15]. Whereas they used only non-negative key intervals, we used key intervals symmetric with respect to zero. To achieve a constant-time implementation using these intervals, we constructed a new algorithm that keeps two torsion points on an elliptic curve. The additional cost for calculation associated with this point is less than the additional cost in [15] to achieve constant-time. Consequently, our implementation is faster than the implementation in [15]. We implemented our algorithm in C and measuring its clock cycles. The reduction ratio measured by clock cycles is 29.03%.

Acknowledgment. This work was supported by JST CREST Grant Number JPMJCR14D6, Japan.

References

1. Bernstein, D.J., Hamburg, M., Krasnova, A., Lange, T.: Elligator: elliptic-curve points indistinguishable from uniform random strings. In: Proceedings of the 2013 ACM Conference on Computer and Communications Security, pp. 967–980 (2013)
2. Bernstein, D.J., Lange, T., Martindale, C., Panny, L.: Quantum circuits for the CSIDH: optimizing quantum evaluation of isogenies. IACR Cryuptography ePrint Archive 2018/1059. https://eprint.iacr.org/2018/1059 (to appear at Eurocrypt 2019)
3. Castryck, W., Lange, T., Martindale, C., Panny, L., Renes, J.: CSIDH: an efficient post-quantum commutative group action. In: Peyrin, T., Galbraith, S. (eds.) ASIACRYPT 2018. LNCS, vol. 11274, pp. 395–427. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03332-3_15

4. Cohen, H., Lenstra Jr., H.W.: Heuristics on class groups of number fields. Number Theory Noordwijkerhout **1983**, 33–62 (1984)
5. Costello, C., Hisil, H.: A simple and compact algorithm for SIDH with arbitrary degree isogenies. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10625, pp. 303–329. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70697-9_11
6. Costello, C., Longa, P., Naehrig, M.: Efficient algorithms for supersingular isogeny Diffie-Hellman. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016. LNCS, vol. 9814, pp. 572–601. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53018-4_21
7. Costello, C., Smith, B.: Montgomery curves and their arithmetic. J. Crypt. Eng. **8**(3), 227–240 (2018)
8. Couveigne, J.-M.: Hard homogeneous spaces. IACR Cryptology ePrint Archive 2006/291. https://eprint.iacr.org/2006/291
9. De Feo, L., Kieffer, J., Smith, B.: Towards practical key exchange from ordinary isogeny graphs. In: Peyrin, T., Galbraith, S. (eds.) ASIACRYPT 2018. LNCS, vol. 11274, pp. 365–394. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03332-3_14
10. Delfs, C., Galbraith, S.D.: Computing isogenies between supersingulrar elliptic curves over $\mathbb{F}_p$. Des. Codes Crypt. **78**(2), 425–440 (2016)
11. Galbraith, S.D., Petit, C., Shani, B., Ti, Y.B.: On the security of supersingular isogeny cryptosystems. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. LNCS, vol. 10031, pp. 63–91. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53887-6_3
12. Jao, D., De Feo, L.: Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In: Yang, B.-Y. (ed.) PQCrypto 2011. LNCS, vol. 7071, pp. 19–34. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25405-5_2
13. Jalali, A., Azarderakhsh, R., Kermani, M.M., Jao, D.: Towards optimized and constant-time CSIDH on embedded devices. In: Polian, I., Stöttinger, M. (eds.) COSADE 2019. LNCS, vol. 11421, pp. 215–231. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-16350-1_12. https://eprint.iacr.org/2019/297
14. Jao, D., et al.: Supersingular isogeny key encapsulation. Submission to the NIST Post-Quantum Cryptography Standardization project. https://sike.org
15. Meyer, M., Campos, F., Reith, S.: On Lions and Elligators: an efficient constatn-time implementation of CSIDH. IACR Cryptology ePrint Archive 2018/1198. https://eprint.iacr.org/2018/1198 (to appear at PQCrypto 2019)
16. Meyer, M., Reith, S.: A faster way to the CSIDH. In: Chakraborty, D., Iwata, T. (eds.) INDOCRYPT 2018. LNCS, vol. 11356, pp. 137–152. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-05378-9_8
17. Montgomery, P.L.: Speeding the Pollard and elliptic curve methods of factorization. Math. Comput. **48**(177), 24–264 (1987)
18. National Institute of Standards and Technology (NIST): NIST Post-Quantum Cryptography Standardization (2016). https://csrc.nist.gov/Projects/Post-Quantum-Cryptography
19. Onuki, H., Aikawa, Y., Yamazaki, T., Takagi, T.: A faster constant-time algorithm of CSIDH keeping two points IACR Cryuptography ePrint Archive 2019/353. https://eprint.iacr.org/2019/353
20. Petit, C.: Faster algorithms for isogeny problems using torsion point images. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10625, pp. 330–353. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70697-9_12

21. Rostovtsev, A., Stolbunov, A.: Public-key cryptosystem based on isogenies. IACR Cryptology ePrint Archive 2006/145. https://eprint.iacr.org/2006/145
22. Siegel, C.: Über die Classenzahl quadratischer Zahlkörper. Acta Arith. **1**(1), 83–86 (1935)
23. Stolbunov, A.: Constructing public-key cryptographic schemes based on class group action on a set of isogenous elliptic curves. Adv. Math. Commun. **4**(2), 215–235 (2010)