Optimal strategies for CSIDH

Jesús-Javier Chi-Domínguez * [1] and Francisco Rodríguez-Henríquez † [2]

[1] Tampere University, Tampere, Finland
[2] Computer Science Department, Cinvestav IPN, Mexico City, Mexico

April 14, 2020

Abstract

Since its proposal in Asiacrypt 2018, the commutative isogeny-based key exchange protocol (CSIDH) has spurred considerable attention to improving its performance and re-evaluating its classical and quantum security guarantees. In this paper we discuss how the optimal strategies employed by the Supersingular Isogeny Diffie-Hellman (SIDH) key agreement protocol can be naturally extended to CSIDH. Furthermore, we report a software library that achieves moderate but noticeable performance speedups when compared against state-of-the-art implementations of CSIDH-512, which is the most popular CSIDH instantiation.

1 Introduction

In late 2018, Castryck, Lange, Martindale, Panny, and Renes presented the isogeny-based key exchange protocol CSIDH [6]. CSIDH can be seen as a fast variant of Couveignes-Rostovtsev-Stolbunov scheme [11, 23, 22], using the ideas presented in [13] but this time operating on supersingular curves defined over prime fields.

One especially attractive feature of CSIDH is that it supports efficient public-key validation, which implies that this scheme can be used as a non-interactive (static-static) key exchange protocol. This is a unique feature that none of the post-quantum cryptographic schemes in the NIST contest enjoys [20]. On the negative side, CSIDH is one order of magnitude slower than its cousin, the SIKE protocol [1]. Indeed, running on a high-end x64 Intel processor, a constant-time implementation of CSIDH requires about $225M$ clock cycles to compute a shared secret (*cf.* Table 4). For comparison, the SIKE protocol instantiated with a 434-bit prime, requires some $20M$ clock cycles [16].

The first constant-time implementation of CSIDH was reported by Bernstein, Lange, Martindale, and Panny in [3]. The authors of [3] focused their analysis on assessing the quantum security level provided by CSIDH, and for this purpose, they strove for

*jesus.chidominguez@tuni.fi
†francisco@cs.cinvestav.mx

producing not only a constant-time CSIDH instantiation but also a randomness-free implementation of it. Shortly after, Jalali, Azarderakhsh, Kermani, and Jao in [15], and Meyer, Campos, and Reith in [17] independently presented constant-time instantiations of CSIDH. The authors of [17] introduced several ingenious algorithmic tricks, including the adoption of the Elligator 2 map of [2], splitting isogeny computations into multiple batches (SIMBA), and sampling the exponents e_i using different interval bounds depending on the prime factors ℓ_i.

More recently, the CSIDH implementation of [17] was improved by Onuki, Aikawa, Yamazaki, and Takagi in [21]. Additionally, Moriya, Onuki and Takagi [19], and Cervantes-Vázquez *et al.* in [7], performed more efficiently the CSIDH isogeny computations using the twisted Edwards model of elliptic curves. The authors of [7] proposed a more computationally demanding dummy-free variant of CSIDH, which in exchange, is arguably better suited to resist attacks from stronger adversaries. Moreover, Hutchinson, LeGrow, Koziel and Azarderakhsh presented in [14] several improvements for achieving faster constant-time implementations of CSIDH. The algorithmic improvements proposed in [14] included a formal framework that permits to adapt to CSIDH, the SIDH optimal strategies discussed in [12] by means of a stochastic procedure; a more efficient re-ordering of the CSIDH small prime factors ℓ_i; and a framework to define the optimal bounds for the CSIDH exponents. Unfortunately, we have so far not being able to reproduce the computational timings of the software library reported in [14].

Our contributions: This is a follow-up paper of previous work presented in [7]. Here, we present a detailed discussion of how to adapt SIDH strategies for the efficient group action evaluation of CSIDH. The main difference of our procedure with the framework presented in [14], is the deterministic nature of our approach and the fact that it works with non-disjoint subsets of prime factors ℓ_i. In particular, the strategies proposed in this paper do not rely on the SIMBA approach of [17], but rather, they are a intuitive generalization of how the SIDH strategies can be applied to CSIDH. Moreover, the CSIDH optimal strategies proposed here comply with the same codification utilized by SIDH. Additionally, we report constant-time C-code implementations of three instantiations of the CSIDH protocol, namely, MCR [17], OAYT [21], and dummy-free [7] variants. Our experimental results achieve performance speedups of 12.09%, 5.46% and 10.58% compared with the MCR, OAYT and dummy-free styles as presented in [7]. Our software library is freely available at,

`https://github.com/JJChiDguez/csidh_withstrategies.`

Note: Let E and E' be two supersingular elliptic curves defined over $\mathbb{F}_p$ for which there exists a separable degree-ℓ isogeny $\phi : E \to E'$ defined over $\mathbb{F}_p$. Quite recently was presented in [4] a new approach for finding at a cost of only $\tilde{O}(\sqrt{\ell})$ operations, the co-domain elliptic curve E' and $\phi(Q)$ and the image of a point $Q \in E(\mathbb{F}_p)$ with $P \notin \mathrm{Ker}(\phi)$. We note that the main contribution presented in [4] is largely orthogonal to the contributions in this paper. Therefore, we leave as a future work to adopt the findings of [4] to further reduce the computational costs of CSIDH reported here.

Organization. In §2 several background algorithmic concepts related to the CSIDH group action computation are given. In §3 an introduction to the efficient computation of the CSIDH class group action is given. Additionally, the usage of optimal strategies for CSIDH is also presented in this section. In §4 additional algorithmic tricks for the computation of three CSIDH variants are given. In §5 we report our experimental results and comparison with related works. Finally, in §6 concluding remarks are drawn.

Notation. **M**, **S**, and **A** denote the cost of computing a single multiplication, squaring, and addition (or subtraction) in $\mathbb{F}_p$, respectively. We assume that a constant-time equality test $\texttt{isequal}(X, Y)$ is defined, returning 1 if $X = Y$ and 0 otherwise. We also assume that a constant-time conditional swap $\texttt{cswap}(X, Y, b)$ is defined, exchanging (X, Y) if $b = 1$ (and not if $b = 0$).

2 Preliminaries

2.1 Differential addition chains for Montgomery ladders

In the CSIDH protocol, any given scalar k is the product of a subset of the collection of the 74 small primes ℓ_i dividing $\frac{p+1}{4}$. Hence, one can simply compute the scalar multiplication operation $[k]P$ as the composition of the shortest differential addition chains for each prime ℓ dividing k. Note that all those shortest additions chains can be pre-computed off-line. Montgomery ladders using differential addition chains can compute the scalar multiplication operation $[k]P$ with an average length of about $1.5\lceil \log_2(k) \rceil$ steps [7]. Each Montgomery ladder step involves the computation of one differential point addition and differential point doubling at a cost of $4\mathbf{M} + 2\mathbf{S} + 6\mathbf{A}$ and $4\mathbf{M} + 2\mathbf{S} + 4\mathbf{A}$, respectively.

Table 1 reports the field arithmetic expenses associated with the computation of $[\ell]P$, where $\ell = 2d + 1$.

2.2 Isogeny constructions and evaluations

Let p be an odd prime number and let ℓ be an odd number $\ell = 2d + 1$, with $d \geq 1$. Let E and E' be two supersingular elliptic curves defined over $\mathbb{F}_p$ for which there exists a separable degree-ℓ isogeny $\phi : E \to E'$ defined over $\mathbb{F}_p$. This implies that there must exist an ℓ-order point $P \in E(\mathbb{F}_p)$ such that $\text{Ker}(\phi) = \{\infty, \pm P, \pm[2]P, \ldots, \pm[d]P\}$. Given the domain elliptic curve E and an ℓ-order point $P \in E(\mathbb{F}_p)$, we are interested in the problem of computing the co-domain elliptic curve E'. Furthermore, given a point $Q \in E(\mathbb{F}_p)$ such that $Q \notin \text{Ker}(\phi)$, a closely related problem is that of finding $\phi(Q)$, *i.e.*, the image of the point Q over E'. In the remainder of this paper, these two tasks will be called isogeny construction and isogeny evaluation computations, respectively.

It has become customary to perform these two tasks by using three main building blocks, namely, `KPS`, `CODOM` and `PEVAL`. Let us define `KPS` as the task of computing the first d multiples of the point P, namely, the set $R = \{P, [2]P, \ldots, [d]P\}$. Using `KPS` as a

Primitive	$\mathbf{M}$	$\mathbf{S}$	$\mathbf{A}$	
			Montgomery[10]	Edwards[7]
$[\ell]P$ [7]	12λ	6λ	15λ	–
KPS	$4(d-1)$	$2(d-1)$	$6d-2$	$6d-2$
PEVAL	$4d$	2	$6d$	$2d+4$
CODOM [9]	$\ell+2\bar{\lambda}+1$	$2(\lambda+2)$	–	0

Table 1: Costs for computing prime degree-ℓ isogenies with $\ell = 2d+1$ using the KPS, PEVAL and CODOM building blocks. Field multiplication ($\mathbf{M}$) and squaring ($\mathbf{S}$) costs are taken from [10, 7, 9]. The cost of performing one scalar multiplication $[\ell]P$ using differential addition chains as in [7], is also presented. The computational costs associated to the point addition and point doubling operations is of $4\mathbf{M} + 2\mathbf{S} + 6\mathbf{A}$ and $4\mathbf{M} + 2\mathbf{S} + 4\mathbf{A}$, respectively. We define $\lambda = \lceil \log_2 \ell \rceil$ and $\bar{\lambda} \approx \frac{\log_2(\lceil \frac{\ell}{8} \rceil)}{3}$.

building block, the module CODOM computes the per-field constants that define the codomain curve E' over $\mathbb{F}_p$. Also, using KPS as a building block, PEVAL computes the image point $\phi(Q)$. Note that KPS becomes more expensive than PEVAL starting from $\ell \geq 11$. When $\ell \leq 7$, the block KPS is considerably cheaper or even free of cost for the case $\ell = 3$.

Observe also that since CODOM and PEVAL show no dependencies between them, once that the kernel points have been computed, it is possible to compute CODOM and PEVAL in parallel. Furthermore, when evaluating an arbitrary number of points in E that do not belong to the $\mathrm{Ker}(\phi)$ subgroup, KPS must be computed only once. This implies that the computational cost associated to KPS gets amortized when computing the image of two or more points.

Table 1 summarizes the field arithmetic costs associated to the KPS and PEVAL operations. Note that KPS is a straightforward computation that can be performed at the cost of one point doubling and $k-2$ point additions. Efficient formulas for computing PEVAL can be found in [10] and [7] for Montgomery and twisted Edwards curves, respectively.

As a numerical example consider the cost of computing isogeny evaluations and constructions for the prime $\ell = 2 \cdot 64 + 1 = 127$.

Example 1 *Let us consider the case for the prime $\ell = 2 \cdot 64 + 1 = 127$. Then, according to Table 1 the computational expenses associated with the computation of the KPS, PEVAL and CODOM primitives and the scalar multiplication $[127]T$, for some point $T \in E(\mathbb{F}_p)$, is shown in Table 2. It can be seen that constructing and evaluating a degree-127 isogeny is 4.34 and 5.03 times more expensive than computing the scalar multiplication $[127]T$, respectively. Note that any extra isogeny evaluation can reuse the KPS computation and therefore it is only two times more expensive than finding the multiple $[127]T$.*

In the remainder of this paper we assume that given a curve E specified in Montgomery form, a point G in $E(\mathbb{F}_p)$ and an odd integer $\ell = 2d+1$, the procedure QuotientIsogeny invoking both the KPS and CODOM primitives, computes the degree-ℓ quotient isogeny $\phi : E \to E' \cong E/\langle G \rangle$, returning (E', R), where $R = \{G, [2]G, \ldots, [d]G\}$.

Primitive	M	S	Total Cost	
			S = M	S = 0.8M
$[\ell]P$	84	42	126	118
KPS	252	126	378	352
PEVAL	256	2	256	256
CODOM	151	18	169	166

Table 2: Approximately arithmetic costs for computing prime degree-ℓ isogenies with $\ell = 2d + 12 \cdot 64 + 1 = 127$, using the KPS, PEVAL and CODOM primitives. The cost of computing the scalar multiplication $[127]T$ is also reported.

3 Computing the CSIDH class group action

In this section, an introduction to the efficient computation of the CSIDH class group action is given. We start giving a simplified view of the CSIDH algorithm, which is followed by several algorithmic refinements.

3.1 Setting

Let $\ell_1, \ldots, \ell_n \in \mathbb{Z}$ be small odd prime numbers such that $p = 4\prod_{i=1}^{n} \ell_i - 1$ is also a prime number. We work with the 511-bit prime proposed in [6], using the following labeling: $\ell_{74} = 3, \ell_{73} = 5, \ldots, \ell_2 = 373$, given by the first 73 odd primes, and $\ell_1 = 587$. Let $E/\mathbb{F}_p$ be a supersingular elliptic curve given in Montgomery form as,

$$E/\mathbb{F}_p \colon y^2 = x^3 + Ax^2 + x; \tag{1}$$

It follows that $\#E(\mathbb{F}_p) = (p+1) = 4\prod_{i=1}^{n} \ell_i$. Additionally, let $\pi \colon (x, y) \mapsto (x^p, y^p)$ be the Frobenius map and $N \in \mathbb{Z}$ be a positive integer. Then, $E[N] \coloneqq \{P \in E(\mathbb{F}_p) \colon [N]P = \mathcal{O}\}$ denotes the N-torsion subgroup of $E/\mathbb{F}_p$. Similarly, $E[\pi - 1] \coloneqq \{P \in E(\mathbb{F}_p) \colon (\pi - 1)P = \mathcal{O}\}$ and $E[\pi + 1] \coloneqq \{P \in E(\mathbb{F}_{p^2}) \colon (\pi + 1)P = \mathcal{O}\}$ denote the subgroups of $\mathbb{F}_p$-rational and zero-trace points, respectively. In particular, any point $P \in E[\pi + 1]$ is of the form (x, iy) where $x, y \in \mathbb{F}_p$ and $i = \sqrt{-1}$ so that $i^p = -1$.

3.2 A simplified constant-time CSIDH group action evaluation

The most demanding computational task of CSIDH is the evaluation of the class group action, which is dominated by the cost of performing a number of degree-ℓ_i isogeny constructions. This action takes as input a secret integer vector $e = (e_1, \ldots, e_n)$ such that $e_i \in [\![0, m]\!]$, and then constructs isogenies with kernel generated by $P \in E_A[\ell_i] \cap E[\pi - 1]$ for exactly e_i iterations.

For constant-time implementation of CSIDH, the group action evaluation starts by constructing isogenies with kernel generated by $P \in E_A[\ell_i] \cap E[\pi - 1]$ for e_i iterations, and then it performs dummy isogeny constructions for $(m - e_i)$ iterations.

Algorithm 1 shows a simplified and idealized computation of the CSIDH group action as explained next. The procedure consists of two main loops. At the beginning of the procedure in Step 1, the constants of the input parameter E_A are assigned to E_0. At

Algorithm 1: Simplified constant-time CSIDH class group action for supersingular curves over $\mathbb{F}_p$, where $p = 4\prod_{i=1}^{n} \ell_i - 1$. The ideals $\mathfrak{l}_i = (\ell_i, \pi - 1)$, where π maps to the p-th power Frobenius endomorphism on each curve. This algorithm computes exactly m isogenies for each ideal $\mathfrak{l}_i$.

Input: A supersingular curve E_A over $\mathbb{F}_p$, and an exponent vector $(e_1, \dots, e_n)$ with each $e_i \in [0, m])$, m a positive number.

Output: $E_B = \mathfrak{l}_1^{e_1} * \cdots * \mathfrak{l}_n^{e_n} * E_A$.

1 $E_0 \leftarrow E_A$;
2 `// Outer loop: Each` ℓ_i `prime f. is processed` m `times`
3 **for** $i \in \{1, \dots, m\}$ **do**
4     $T \leftarrow$ `ObtainFullTorsionPoint`(E_0) ; `//` $T \in E_n[\pi - 1]$
5     $T \leftarrow [4]T$; `// Now` $T \in E_n\left[\prod_i \ell_i\right]$
6   `// Inner loop: processing each prime factor` $\ell_i | (p+1)$;
7   **for** $j \in \{0, 1, \dots, n-1\}$ **do**
8         $G_j \leftarrow T$;
9   **for** $k \in \{1, \dots, n-1-j\}$ **do**
10             $G_j \leftarrow [\ell_k]G_j$
11   **if** $e_j \neq 0$ **then**
12             $(E_{(j+1) \bmod n}, R) \leftarrow$ `QuotientIsogeny`(E_j, G_j, ℓ_{n-j}) ;
13             $T \leftarrow$ `PEVAL`(T, R) ;
14             $e_j \leftarrow e_j - 1$;
15   **else**
16   `QuotientIsogeny`(E_j, G_j, ℓ_{n-j}); $\phi(T)$; `// Dummy operations`
17             $T \leftarrow [\ell_{n-j}]T$;
18             $E_{j+1 \bmod n} \leftarrow E_j$;
19 **return** E_0

Step 4 of the outer loop of Steps 3-18, a full order point $T \in E_0$ (*i.e.*, a point having order $\frac{p+1}{4}$), is computed. For the sake of simplicity it has been assumed that the function in Step 4 must always output a full torsion point belonging to $E_n[\pi - 1]$.[1]

Thereafter, the inner loop of Steps 7-18 constructs and evaluates a degree-ℓ_i isogeny for each one of the n prime factors ℓ_j dividing $p + 1$, using G_j as a subgroup kernel generator. At each iteration, an isogenous elliptic curve E_j is computed. When the inner loop completes its computation, the constants defining the elliptic curve E_0 are used in Step 4 to find a new full order point $T \in E_0$. The outer loop of Steps 3-18 simply repeat the execution of the inner loop in order to complete *exactly* m evaluations. At the end of the procedure, the constants defining the curve E_0 (corresponding to the m-th evaluation of the inner loop) is returned. As long as the computations in Steps 11-14 and Steps 15-18 are carefully balanced, and the conditional statements are substitute by conditional swaps (see Algorithm 2), this procedure computes the group action in constant time. Hence, the running time of Algorithm 2 does not depend on the secret key vector e.

[1] Note that in practice the time required for finding a full-torsion point is relatively expensive. Hence, one normally relax this condition and works with points whose order does not necessarily include all the prime factors of $p + 1$.

Algorithm 2: Simplified constant-time CSIDH class group action for supersingular curves over $\mathbb{F}_p$, where $p = 4\prod_{i=1}^{n} \ell_i - 1$. The ideals $\mathfrak{l}_i = (\ell_i, \pi - 1)$, where π maps to the p-th power Frobenius endomorphism on each curve. This algorithm computes exactly m isogenies for each ideal $\mathfrak{l}_i$. (**Low level version**)

Input: A supersingular curve E_A over $\mathbb{F}_p$, and an exponent vector $(e_1, \ldots, e_n)$ with each $e_i \in [0, m])$, m a positive number.

Output: $E_B = \mathfrak{l}_1^{e_1} * \cdots * \mathfrak{l}_n^{e_n} * E_A$.

1 $E_0 \leftarrow E_A$;
2 // Outer loop: Each ℓ_i prime f. is processed m times
3 **for** $i \in \{1, \ldots, m\}$ **do**
4     $T \leftarrow$ `ObtainFullTorsionPoint`(E_0) ; // $T \in E_n[\pi - 1]$
5     $T \leftarrow [4]T$; // Now $T \in E_n\left[\prod_i \ell_i\right]$
6   // Inner loop: processing each prime factor $\ell_i | (p+1)$;
7   **for** $j \in \{0, 1, \ldots, n-1\}$ **do**
8         $G_j \leftarrow T$;
9   **for** $k \in \{1, \ldots, n-1-j\}$ **do**
10             $G_j \leftarrow [\ell_k]G_j$
11         $b \leftarrow$ `isequal`$(e_{n-j}, 0)$;
12         $(E_{(j+1) \bmod n}, R) \leftarrow$ `QuotientIsogeny`(E_j, G_j, ℓ_{n-j}) ; // degree-ℓ_{n-j} isogeny
13         $T' \leftarrow [\ell_{n-j}]T$;
14         $T \leftarrow$ `PEVAL`(T, R); // Evaluate T on degree-ℓ_{n-j} isogeny
15   `cswap`$(E_j, E_{(j+1) \bmod n}, b)$; // undo if $e_{n-j} = 0$
16   `cswap`(T', T, b) ; // undo if $e_{n-j} = 0$
17         $e_{n-j} \leftarrow e_{n-j} - \big((b+1) \mod 2\big)$;
18 **return** E_0

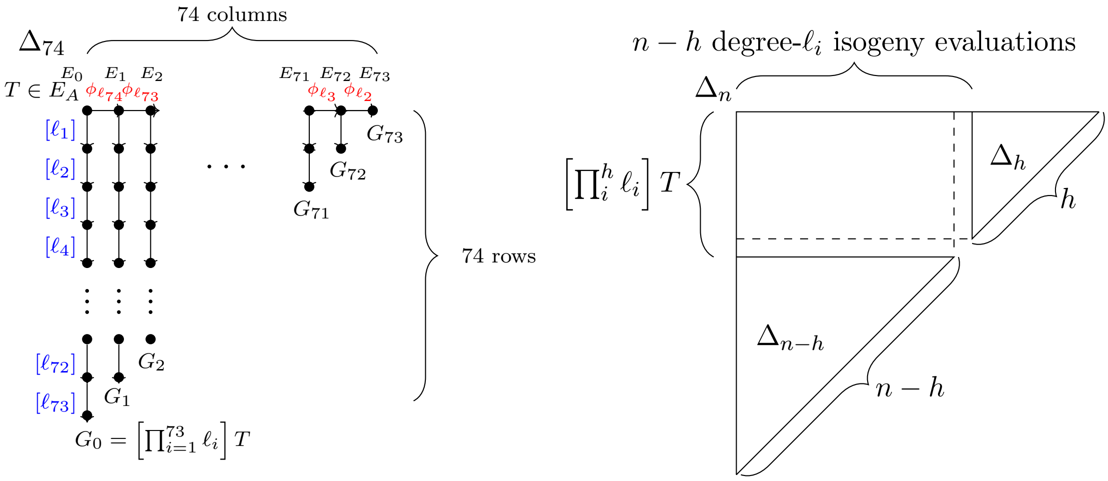
The computational cost of Algorithm 1 is dominated by the computation of n degree-ℓ_i isogeny evaluations and constructions plus a total of $\frac{n(n+1)}{2}$ scalar multiplications by the prime factors ℓ_i, for $i = 1, \ldots, n$.

Remark 1 *A natural instantiation of Algorithm 1 uses the 511-bit CSIDH prime with 74 prime factors dividing $p+1$. In order to guarantee a 128-bit classical security level, it is required to choose $m = 10$, so that the private key space has a size of about $11^{74} \approx 2^{256}$ different keys.*

Algorithm 2 presents a low-level constant-time version of Algorithm 1, where all the conditional statements have been implemented as conditional swaps statements.

Remark 2 *Notice that the scalar multiplication required in Step 13 of algorithm 2, can be performed by invoking the `QuotientIsogeny()` procedure using as input parameter the point T, instead of the point G_j. Let R be the array of $n-j$ points $\{T, [2]T, \ldots, [d_{n-j}]T\}$, and*

$$
\begin{aligned}
[l_{n-j}]T &\coloneqq [2d_{n-j}+1]T = [d_{n-j}]T + [d_{n-j}+1]T \\
&= R[d_{n-j}] + [d_{n-j}+1]T = R[d_{n-j}] + \big(R[d_{n-j}] + R[1]\big)
\end{aligned}
$$



(a) The multiplicative strategy for computing the CSIDH group action as given in Algorithm 1

(b) Optimal strategies *à la* SIDH for CSIDH

Figure 1: Subfigure 1a shows a discrete triangle used to compute the inner loop of the CSIDH group action Algorithm 1. The main goal of this task is to find the field constants that define the elliptic curve E_B. As stated in Algorithm 1, the discrete triangle of Subfigure 1a must be computed exactly m times. Using an optimal strategy as in [12], a discrete triangle Δ_n is processed by splitting it into two sub-triangles as shown in Subfigure 1b.

can be computed with two additions. Thus, the points T and G_j must be swapped before the `QuotientIsogeny()` *procedure is invoked.*

3.3 A multiplicative-based Strategy for CSIDH

In order to efficiently compute the group action of Algorithm 1, one can adapt the canonical strategies for traversing a weighted directed graph presented in[12], which is represented as a discrete right triangle Δ_n of side n having $\frac{n(n+1)}{2}$ vertices distributed in n columns and rows (See Figure 1a).

The vertices of Δ_n represent elliptic curve points and its vertical and horizontal edges have as associated weight p_{ℓ_i} and q_{ℓ_i}, defined as the cost of performing one scalar multiplication by ℓ_i and evaluating a degree-ℓ_i isogeny, respectively. The j-th column of the triangle contains exactly $n - j$ vertices representing elliptic curve points belonging to the isogenous elliptic curve E_j, for $j = 0, \ldots, n-1$. A leaf is defined as the most bottom point in a given column of the triangle. The set of n leaves define the hypotenuse of Δ_n. A ramification (or split) vertex is defined as a vertex having both horizontal and vertical edges leaving from it. The weight of a split vertex is the number of vertices between it and either the next split vertex in the column, or the leave in the column. Each one of the n columns of Δ_n corresponds to an isogenous supersingular elliptic curve E_j, for $j = n, 1, 2 \ldots, n-1$.

Remark 3 *As a mechanism to obtain a constant-time implementation of the group action the procedure shown in Algorithm 1, as well as most constant-time implementations of CSIDH, use dummy computations. Hence, it may occur that* $E_k = E_l$ *with* $0 \leq k < l \leq n-1$.

At the beginning of the group action evaluation, only the base elliptic curve $E_A = E_0$ is known. Then, a point $T \in E_A$ (ideally) with order $\frac{p+1}{4} = \prod_i \ell_i$ must be found. This torsion point can be *descended* by performing a scalar multiplication with each one of the n prime factors of $p+1$ (see the first column of Figure 1a).

The leaf of the first column represents the point $G_0 = \left[\prod_i \ell_{i=1}^{n-1}\right] T$. If G_0 is finite, then it has to have order ℓ_n and can be used to generate the subgroup corresponding to the kernel of the isogeny ϕ_{ℓ_n}. The leaf G_1 is defined as,

$$G_1 = \begin{cases} \left[\prod_i \ell_{i=1}^{n-2}\right] \phi_{\ell_n}(T) & \text{if } e_n \neq 0; \\ \left[\prod_i \ell_{i=1}^{n-2}\right] ([\ell_n]T) & \text{if } e_n = 0. \end{cases} \tag{2}$$

The point G_1 is guaranteed to be finite and of order ℓ_{n-1}, provided that T is a full order point. In general, if the exponents $e_j \neq 0$ for $j = n, n-1, \ldots, 3, 2$. Then

$$G_{n-(j-1)} = \left[\prod_i \ell_{i=1}^{j-2}\right] \phi_{\ell_j}(\ldots(\phi_{\ell_n}(T))\ldots). \tag{3}$$

If some $e_k = 0$, then the corresponding isogeny evaluation ϕ_{ℓ_k} of Eq. (3) must be substituted by the scalar multiplication $[\ell_k]T$.

The goal of the group action computation is thus seen as the task to obtain one by one, all the leaves $G_j \in \Delta_n$ for $j = 1, 2, \ldots, n$, until the farthest right one, G_{n-1}, has been calculated. Then, the elliptic curve E_B determined by $\phi_{\ell_n} : E_{n-1} \to E_n$ can be obtained by simply constructing a degree-ℓ_n isogeny with kernel G_{n-1}, which coincides with the domain or image of ϕ_{ℓ_n} depending if $e_1 = 0$ or not, respectively.

The naive strategy followed by Algorithm 1 is depicted in Figure 1a, instantiated for the CSIDH prime p_{512} with 74 prime factors ℓ_i such that $\ell_i | (p+1)$. The computation of the triangle Δ_n shown in Figure 1a represents one full execution of the inner loop of Steps 7-18. This computation should be repeated $m = 10$ times in order to complete the CSIDH group action (*cf.* Remark 1). From Figure 1a, it can be seen that Algorithm 1 follows a pure multiplicative strategy, where $\frac{n(n+1)}{2} = 2775$ scalar multiplications by the scalars ℓ_i for $i = 1, \ldots, 74$, are performed; plus the construction and evaluation of only 74 degree-ℓ_i isogenies.

Assuming that in average, one scalar multiplication computation $[\ell]T$ is at least five times less expensive than a degree-ℓ isogeny construction or evaluation, one can see that there is room for optimizing the multiplicative strategy followed by Algorithm 1.[2] In the following we briefly review optimal strategies as they were presented in [12].

[2] Another computational reason for considering other approaches is that a multiplicative strategy is eminently sequential. Alternative strategies exploiting the inherent parallelism of the isogeny evaluation computations can be much more attractive for multi-core platforms.

3.4 Optimal strategies for CSIDH

Let $L := [\ell_1, \ell_2, \dots, \ell_n]$ be the list of small odd prime numbers such that $p = 4 \cdot \prod_{i=1}^{n} \ell_i - 1$ is a prime number. A CSIDH strategy is a weighted subgraph $S_n(L)$ contained into a discrete rectangular triangle Δ_n of side n. Any strategy $S_n(L)$ has an associated cost which is defined as,

$$C(S_n) = \sum_{x \in \mathbf{edges}(S_n(L))} \omega(x) + \sum_{j=0}^{n} \nu((n-1-j, j)), \tag{4}$$

where $\omega(x)$ and $\nu((n-1-j,j))$ denote the weights of the edge x and leaf $(n-1-j,j)$, respectively.

In addition $S_n(L)$ is called optimal, if for any different strategy $S'_n(L)$ the inequality $C_n\big(S_n(L)\big) < C_n\big(S'_n(L)\big)$ holds. Optimal strategies were defined in [12] within the context of the SIDH protocol. In [12] the fact that a triangle Δ_n can be optimally and recursively decomposed into two sub-triangles Δ_h and Δ_{n-h} was exploited as shown in Figure 1b. Let us denote as Δ^h the design decision of splitting a triangle Δ_n at row h. The sequential cost of walking across the strategy $S_n(L)$, which is a subgraph of Δ_n^h, is given as

$$C(S_n^h(L)) = C(S_h(L_h)) + C(S_{n-h}(L_{n-h})) + \sum_{i=1}^{n-h} q_{\tilde{\ell}_i} + \sum_{i=0}^{h-1} p_{\tilde{\ell}_{n-i}},$$

where $L_h = [\tilde{\ell}_{n-h+1}, \dots, \tilde{\ell}_n]$ and $L_{n-h} = [\tilde{\ell}_1, \dots, \tilde{\ell}_{n-h}]$ are two disjoint sublists of L and size h and $n-h$, respectively. We say that $S_n^{\hat{h}}(L)$ is optimal if $C(S_n^{\hat{h}}(L))$ is minimal among all $S_n^h(L)$ for $h \in [1, n-1]$. Applying this strategy recursively leads to a procedure that computes the CSIDH group action at an optimal cost. The associated number of scalar multiplications is reduced at the price of increasing the total number of isogeny evaluations and constructions.

In the context of SIDH, optimal strategies tend to balance the number of isogeny evaluations and scalar multiplications to $O(n \log(n))$. However, CSIDH optimal strategies are expected to be largely multiplicative, *i.e.*, optimal strategies will tend to favor computing more scalar multiplications. This is due to the fact that these operations are several times cheaper than isogeny evaluations for sufficiently large prime degree ℓ (*cf.* Example 1).

As proposed in [12], optimal strategies can be obtained using dynamic programming (see [1, 8] for concrete algorithms). A brief description of the process of finding optimal strategies for CSIDH is given next.

3.4.1 Finding Optimal strategies for CSIDH

Notice that the computation of SIDH strategies are a very special case of CSIDH strategies, where q_{ℓ_i} and p_{ℓ_j} are fixed, and the required number of different weighted sub-triangles is given as $\sum_{i=1}^{n-1} i = \frac{(n-1)n}{2}$.

This is not the case for CSIDH, where each pair of sub-triangles Δ_h and Δ_{n-h} requires different (and disjoint) sub-lists L_h and L_{n-h} chosen from $L := [\ell_1, \ell_2, \ldots, \ell_n]$. Additionally, since the ordering of each sub-list impacts on the cost of any strategy in Δ_h and Δ_{n-h}, the search space of different weighted sub-triangles to be considered is exceedingly large, given as, $\sum_{i=1}^{n-1} i! \cdot \binom{n}{i} \gg 2^n$. Therefore, searching for an optimal ordering of the small prime factors in L and determining if a given strategy is optimal, become infeasible.

Heuristically, one can expect that the optimal ordering of prime factors $\ell_i \in L$, has a computational cost quite close to the one associated to processing the isogenies from the smallest to the the largest. Under this assumption, it is enough to compute optimal strategies for each sub-list of ordered small odd primes (starting from the smallest). This implies that the search space of different weighted sub-triangles gets reduced to a space of cubic complexity since

$$\begin{aligned} n + \sum_{j=2}^{n-1}(n+1-j)(j-1) &= n + n\sum_{j=2}^{n-1}(j-1) - \sum_{j=2}^{n-1}(j-1)^2 = n + n\sum_{j'=1}^{n-2} j' - \sum_{j'=1}^{n-2}\left(j'\right)^2 \\ &= n + n\left(\frac{(n-2)(n-1)}{2}\right) - \left(\frac{(n-2)(n-1)(2n-3)}{6}\right) \\ &= n + \frac{(n-2)(n-1)}{6}\big(3n-(2n-3)\big) \\ &= n + \frac{(n-2)(n-1)(n+3)}{6}. \end{aligned}$$

Notice also that any CSIDH strategy can be coded following the linearized representation used in [1]. In [1], a strategy is described as a list of exactly $(n-1)$ positive integers smaller than n, such that each entry determines the number of vertical edges before a ramification or a leaf is reached. For example, the multiplicative-based strategy of Algorithm 1, can be coded as $S_n(L) := [n-1, n-2, \ldots, 2, 1]$.

Based on the approach described in [1], the following cubic complexity procedure outlines how to obtain a CSIDH optimal strategy. This procedure outputs a vector of $(n-1)$ positive integers smaller than n. For k, j positive integers, let us define a sub-list of prime factors $\mathbf{N}_{k,j} := [\ell_{j+1}, \ell_{j+2}, \ldots, \ell_{j+k}] \in L$. Then,

1. For each $j := 0, 1 \ldots, n-1$, the optimal stragegy for each $\mathbf{N}_{1,n-1-j}$ is $S_1(\mathbf{N}_{1,n-1-j}) = []$ and has a cost equals $C_1\big(S_1(\mathbf{N}_{1,n-1-j})\big) = \nu\big((n-1-j, j)\big)$.

2. For each $k := 2, 3, \ldots, n$ and $j := 0, 1 \ldots, n-k$, the optimal strategy is

$$S_k(\mathbf{N}_{k,j}) = [s] \texttt{ cat } S_{k-s}(\mathbf{N}_{k-s,j+s}) \texttt{ cat } S_s(\mathbf{N}_{s,j})$$

and has a cost equals $C_k(S_k\big(\mathbf{N}_{k,j})\big) = \min_h \alpha$, where $s = \arg\min_h \alpha$, and

$$\alpha = \{ \;\; C_{k-h}\big(S_{k-h}(\mathbf{N}_{k-h,h+j})\big) + C_h\big(S_h(\mathbf{N}_{h,j})\big) + \omega\big([(0,0),(h,0)]\big) + \omega\big([(0,0),(0,k-h)]\big) : h = 1, 2, \ldots, k-1 \;\; \}.$$

Here, $\omega\big([(0,0),(0,h)]\big)$ and $\omega\big([(0,0),(k-h,0)]\big)$ represent a vertical segment and a horizontal segment of length h and $k-h$, respectively. It has been assumed that the root vertex $(0,0)$ corresponds with the root of the sub-triangle Δ_k, associated with the sub-list of prime factors $\mathbf{N}_{k,j}$. See Figure 1b for an illustration of the first level of this recursive process with $k=n$.

The remaining task is how to evaluate a CSIDH optimal strategy $S_n(L)$ as obtained in the above procedure. We discuss this problem in the next section.

4 Additional algorithmic refinements for constant-time group action evaluation

In this section, we focus our attention to the algorithmic tricks presented by three recent CSIDH variants, namely, the Meyer–Campos–Reith constant-time algorithm of [17], the Onuki–Aikawa–Yamazaki–Takagi constant-time algorithm of [21], and the dummy-free algorithm of [7].

4.1 One torsion point with dummy isogeny constructions (MCR-style)

Meyer, Campos and Reith proposed in [17] several ingenious optimizations that compared to Algorithm 1, lead to a much faster constant-time CSIDH group action computation.

One of the optimizations introduced in [17], was to sample a point using the Elligator 2 map of [2] and [3]. Typically, the Elligator 2 mapping does not return a full order point. Let $T \in E(\mathbb{F}_p)$, with $p = 4\prod_{i=1}^{n}\ell_i - 1$. As pointed out in [7], under reasonable heuristics assumptions experimentally verified in [3], it is observed that

$$\Pr\left[\left[\frac{p+1}{\ell_i}\right]T = \mathcal{O}\right] = \frac{1}{\ell_i}, \text{ for } i = 1,\ldots,n.$$

In the event that the Elligator procedure outputs a point T that is not of full order, then extra points must be sampled in order *to repair* the missing prime factors.

A second optimization in [17], dubbed SIMBA-σ-κ, consisted of splitting the processing of the prime factors ℓ_i as defined above, into σ disjoint sets (batches) of size $\frac{n}{\sigma}$. Afterwards, a multiplicative strategy is applied to each batch. Each multiplicative strategy is evaluated κ times.

Finally as in [18], instead of using a fixed interval $[0,10]$ for all the isogeny computations, the authors of [17] proposed to define a customized interval per each entry in the secret vector e. Thus, a vector m is defined such that $0 \le e_i \le m_i$, for $i = 1,\ldots,n$. The missing prime factors are repaired using a multiplicative strategy, until all the m_i degree-ℓ_i isogeny constructions have been performed.

In this work, we adopted the Elligator 2 procedure for point sampling, plus the definition of a vector m with a customized interval per each entry in the secret vector e. However, we dismiss the usage of the SIMBA approach.

In the remaining of this paper we will refer to this approach, which uses one torsion point and dummy isogeny constructions, as the *MCR-style* CSIDH group action evaluation. The details of how to execute an optimal strategy using this approach are given in Appendix B.1.

4.2 Two torsion point with dummy isogeny constructions (OAYT-style)

Onuki, Aikawa, Yamazaki and Takagi proposed a faster constant-time version of CSIDH in [21]. Their key idea is to use two points to evaluate the action of an ideal, one in $\ker(\pi - 1)$ (i.e., in $E(\mathbb{F}_p)$) and one in $\ker(\pi + 1)$ (*i.e.*, in $E(\mathbb{F}_{p^2})$ with the x-coordinate in $\mathbb{F}_p$). This allows them to avoid timing attacks, while keeping the same primes and exponent range $[-5, 5]$ as in the original CSIDH algorithm of [6]. Their algorithm also employs dummy isogenies to mitigate some power analysis attacks, as in [17]. With these improvements, the authors achieve a considerable speed-up compared to [17]. The saving comes from the fact that the procedure proposed by [21] performs approximately five isogeny constructions (as opposed to the ten constructions in [17]) and ten isogeny evaluations per ℓ_i. Algorithm 3 of Appendix A summarizes the main idea proposed by Onuki *et al.* [21].

In the remaining of this paper we will refer to this approach, which uses two torsion points and dummy isogeny constructions, as the *OAYT-style* CSIDH group action evaluation. We stress that OAYT-style considers both, the Elligator 2 procedure for sampling points and a customized bound vector m, but does not make use of the SIMBA strategy (*cf.* §4.1). The details of how to execute an optimal strategy using OAYT-style can be found in Appendix B.2.

4.3 Two torsion point without dummy isogeny constructions (Dummy-free style)

A constant-time CSIDH group action computation that does not use dummy computations, thus making every computation essential for a correct final result was proposed in [7]. This yields some natural resistance to fault attacks, at the cost of approximately a twofold slowdown. For the approach in [7], the exponents e_i are uniformly sampled from sets

$$\mathcal{S}(m_i) = \{e \mid e = m_i \bmod 2 \text{ and } |e| \leq m_i\},$$

i.e., centered intervals containing only even or only odd integers. The action of vectors drawn from $\mathcal{S}(m)^n$ can be computed by interpreting the coefficients e_i as,

$$|e_i| = \underbrace{1 + 1 + \cdots + 1}_{e_i \text{ times}} + \underbrace{(1-1) - (1-1) + (1-1) - \cdots}_{m_i - e_i \text{ times}},$$

i.e., the algorithm starts by acting by $\mathfrak{l}_i^{\mathtt{sign}(e_i)}$ for e_i iterations, then alternates between $\mathfrak{l}_i$ and $\mathfrak{l}_i^{-1}$ for $m_i - e_i$ iterations. Algorithm 4 of of Appendix A describes the approach presented in [7].

In the remaining of this paper we will refer to this approach, which uses two torsion points without dummy isogeny constructions, as the *Dummy-free-style* CSIDH group action evaluation. We stress that Dummy-free-style considers both, the Elligator 2 procedure for sampling points and a customized bound vector m, but does not make use of the SIMBA strategy (*cf.* §4.1). The details of how to execute an optimal strategy using Dummy-free-style can be found in Appendix B.3.

4.4 Finding an optimal bound vector for the CSIDH group action

All three of the MCR-, OAYT- and Dummy-free styles previously described in this section, use a bound vector $m = (m_1, m_2, \ldots, m_n)$. The bound vector m specifies the intervals where each secret exponent e_i associated to each degree-ℓ_i isogeny with $i = 1, \ldots, n$, must be sampled. Given a bound vector m, the computational cost of the CSIDH group action is a complex function that must take into consideration not only the expenses associated to the number of isogeny constructions/evaluations and scalar multiplications, but also the costs of repairing missing prime factors due to the probabilistic nature of the Elligator 2 procedure (*cf.* 4.1). A heuristic solution to the optimization problem of finding a vector m such that the computational cost of the group action evaluation is minimized while its classical security level is preserved (cf. Remark 1), can be found by means of a greedy algorithm.

Let us assume that an initial vector $m = (m_1, m_2, \ldots, m_n)$ that achieves λ-bits of classical security is given, where all m_i for $i = 1, \ldots, n$ are positive integers. Then, one first proceeds by reducing one of the entries of the vector m by one, while increasing one or more other entries, until the perturbed vector m provides a classical security of λ-bits, but hopefully a lesser computational cost for the group action. If the modified vector has an smaller cost than the initial one, then the vector m is updated accordingly. Let us use $\delta = 2$ if the group action evaluation is performed using OAYT-style, and $\delta = 1$ if MCR- or Dummy-free styles are chosen. Then, a greedy algorithm that finds an optimal vector m achieving λ-bits of classical security can be summarized as follows:

0. Initial bound $(m_1, m_2, \ldots, m_n)$ that yields λ-bits of classical security for the group action. In other words, $\lfloor \sum_{i=1}^{n} \log_2(\delta \cdot m_i + 1) \rfloor = 2\lambda$;

1. For each $i \coloneqq 1, 2, \ldots, n$:

   (a) Set $\overrightarrow{m} = (m_1, m_2, \ldots, m_n)$;

   (b) Decrease the i-th coordinate of $\overrightarrow{m}$ by one unit;

   (c) Compute

   $$\mu_i = \left\{ \tilde{m} = \overrightarrow{m} + \Delta \colon \Delta \in (\mathbb{Z}_+ \cup \{0\})^n,\ \Delta_i = 0,\ \left\lfloor \sum_{j=1}^{n} \log_2(\delta \cdot \tilde{m}_j + 1) \right\rfloor = 2\lambda \right\}$$

   (d) Select the local optimal element $\hat{m}$ of μ_i that minimizes the cost;

(e) If $\hat{m}$ has a smaller cost than the initial bound $(m_1, m_2, \ldots, m_n)$, then replace each m_i by $\hat{m}_i$.

2. Output $(m_1, m_2, \ldots, m_n)$.

For our Python script experiments, we set the initial bound vector as $(m_1, m_2, \ldots, m_n)$ with $m_i = \frac{10}{\delta}$ for each $i := 1, 2, \ldots, n$. Additionally, in order to ensure that at least one degree-ℓ_i isogeny construction will be performed for each small odd prime ℓ_i (*i.e.* that all the entries in the bound vector are strictly greater than 0), the above greedy method was applied iteratively $\left(\frac{10}{\delta} - 1\right)$ times. We heuristically found out that setting $m_n = \frac{3}{2} \cdot \frac{10}{\delta}$, tends to obtain better bound vectors. A Python-script implementation of the above greedy procedure found the following bounds,

$$\overrightarrow{m}_{MCR} = \begin{array}{rrrrrrrrrrrrr} (3, & 4, & 4, & 4, & 4, & 4, & 4, & 4, & 4, & 4, & 5, & 5, & 5, \\ 5, & 5, & 5, & 5, & 5, & 5, & 6, & 6, & 6, & 6, & 6, & 6, & 6, \\ 6, & 6, & 7, & 7, & 7, & 8, & 8, & 8, & 8, & 9, & 9, & 9, & 10, \\ 10, & 11, & 11, & 12, & 13, & 12, & 14, & 15, & 16, & 16, & 16, & 20, & 23, \\ 21, & 23, & 23, & 23, & 23, & 23, & 23, & 23, & 22, & 20, & 19, & 22, & 22, \\ 22, & 22, & 22, & 22, & 21, & 21, & 20, & 18, & 15); & & & & \end{array}$$

$$\overrightarrow{m}_{OAYT} = \begin{array}{rrrrrrrrrrrrr} (1, & 2, & 2, & 2, & 2, & 2, & 2, & 2, & 2, & 2, & 2, & 3, & 3, \\ 3, & 3, & 3, & 3, & 3, & 3, & 3, & 3, & 3, & 3, & 3, & 3, & 3, \\ 3, & 3, & 3, & 4, & 4, & 4, & 4, & 4, & 4, & 4, & 4, & 5, & 5, \\ 5, & 5, & 6, & 6, & 7, & 7, & 7, & 7, & 8, & 8, & 8, & 9, & 11, \\ 9, & 11, & 11, & 11, & 11, & 11, & 11, & 11, & 11, & 11, & 11, & 11, & 11, \\ 11, & 10, & 10, & 10, & 10, & 10, & 9, & 9, & 7); & & & & \end{array}$$

$$\overrightarrow{m}_{Dummy-free} = \begin{array}{rrrrrrrrrrrrr} (3, & 4, & 4, & 4, & 4, & 4, & 4, & 4, & 4, & 4, & 5, & 5, & 5, \\ 5, & 5, & 5, & 5, & 5, & 5, & 5, & 6, & 6, & 6, & 6, & 7, & 6, \\ 6, & 6, & 6, & 8, & 7, & 8, & 8, & 8, & 8, & 9, & 9, & 9, & 9, \\ 11, & 11, & 11, & 12, & 13, & 12, & 14, & 15, & 16, & 16, & 16, & 19, & 23, \\ 23, & 23, & 23, & 23, & 23, & 23, & 23, & 23, & 22, & 20, & 19, & 22, & 22, \\ 22, & 22, & 22, & 22, & 21, & 21, & 20, & 18, & 15); & & & & \end{array}$$

for MCR, OAYT, and dummy-free styles, respectively. Let us recall that each entry of these bound vectors corresponds with the number of degree-(ℓ_i) isogeny constructions to be performed, with $\ell_1 = 587 > \ell_2 > \cdots > \ell_n = 3$.

4.5 Number of optimal strategies required for a group action computation

Let γ and Γ be equal to the minimum and maximum entries in the integer bound vector m, respectively. Once again, let $L = [\ell_1, \ell_2, \ldots, \ell_n]$ be the list of small odd prime numbers such that $p = 4 \cdot \prod_{i=1}^{n} \ell_i - 1$ is a prime number. Then as discussed in 3.4, one can find a strategy $S_n(L)$ that performs an optimal number of isogeny constructions/evaluations with degrees equal to each one of the n prime factors in L. The strategy $S_n(L)$ must be executed γ times. At this point with high probability all the degree-ℓ_i isogenies having entries $m_i = \lambda$ for $i = 1, \ldots, n$, do not need to be considered any further.[3] Additionally one still needs to process L' isogenies, where L' is a subset of L such that its corresponding entries in the bound vector m are strictly greater than λ. To proceed forward, all the entries of m must be subtracted by λ, disregarding the zero entries. Then, a new minimum entry λ' is computed and a new strategy $S_{n'}(L')$ must be found and executed λ' times with $n' = \#L'$. This procedure is repeated until there are no more isogenies to be processed. In fact, after Γ rounds, the estimated number of missing degree-ℓ_i isogeny constructions is $\approx \left(\frac{m_i}{\ell_i}\right)$. A simple multiplicative strategy can be executed to repair those missing isogeny constructions/evaluations. We formalize the preceding discussion as follows.

We require to find and execute t strategies, where $t \leq n$ is the number of different integer entries in the bound vector m. Let $m^{(k)}$ be a multiset of bound vector with length n_k for $k = 1, \ldots, t$. Let $\gamma_k = \min\ m^{(k)}$. By definition, $m^{(1)} = m$, $n_1 = n$ and $\gamma_1 = \gamma$. Then, the k-th strategy must be executed γ_k times, where

$$
\begin{aligned}
m^{(1)} &= \{m_1, m_2, \ldots, m_n\}, \\
m^{(2)} &= \{m_1^{(1)} - \gamma_1, \ldots, m_{n_1}^{(1)} - \gamma_1\} \setminus \{0\}, \\
m^{(3)} &= \{m_1^{(2)} - \gamma_2, \ldots, m_{n_2}^{(2)} - \gamma_2\} \setminus \{0\}, \\
&\vdots \\
m^{(t)} &= \{m_1^{(t-1)} - \gamma_{t-1}, \ldots, m_{n_{t-1}}^{(t-1)} - \gamma_{t-1}\} \setminus \{0\}.
\end{aligned}
$$

The k-th strategy must be optimal with respect to the list L_k, defined as follows:

$$
\begin{aligned}
L_1 &= [\ell_1, \ell_2, \ldots, \ell_n], \\
L_2 &= [\ell_i \in L_1 \colon L_i^{(1)} > \gamma_1], \\
L_3 &= [\ell_i \in L_2 \colon L_i^{(2)} > \gamma_2], \\
&\vdots \\
L_t &= [\ell_i \in L_{t-1} \colon L_i^{(t-1)} > \gamma_{t-1}].
\end{aligned}
$$

The cost of the final multiplicative strategy to account for the missing isogenies can be skipped or at least minimized, if the group action is evaluated by considering the following adjusted bounds,

$$
m_i' := \left\lfloor m_i \cdot \left(\frac{\ell_i}{\ell_i - 1}\right)\right\rceil \text{ for } i = 1, \ldots, n.
$$

[3]In fact the probability of having completed all the degree-ℓ_i isogenies whose entries $m_i = \lambda$ for $i = 1, \ldots, n$, depend on the order of the points output by the Elligator 2 procedure as discussed in §4.1.

In particular, using m'_i instead of m_i, the expected number of degree-ℓ_i isogeny constructions to be performed is m_i. To be more precise, we propose "to use" m'_i times each ℓ_i in order to reach all the m_i degree-ℓ_i isogeny constructions.

Remark 4 *Our analysis only depends on the cost of isogeny evaluations and scalar multiplications, and thus it can be easily applied to the work of Castryck and Decru [5] (CSURF).*

5 Experiments and comparisons

In this section we report the CSIDH-512 group action evaluation considering the three strategies discussed in §4, namely, i) MCR-style, ii) OAYT-style, and ii) Dummy-free-style, by adopting the bound vectors presented in §4.4. We present a comparison of our results versus the SIMBA-based methods that use the exponent bounds m as reported in [7, §5.2].[4]

All of our experiments were ran on a Intel(R) Core(TM) i7-6700K CPU 4.00GHz machine with 16GB of RAM, with Turbo boost disabled and using `gcc` version 5.5. Our software library is freely available from,

`https://github.com/JJChiDguez/csidh_withstrategies.`

Implementation	**Group action evaluation**	**M**	**S**	**a**	**Speedup (%)**
Cervantes-Vázquez *et al.* [7]	MCR with SIMBA	0.900	0.310	0.964	—
	OAYT with SIMBA	0.658	0.210	0.691	—
	Dummy-free with SIMBA	1.319	0.423	1.389	—
Hutchinson *et al.* [14]	MCR with SIMBA	0.905	0.312	0.860	-0.58
	OAYT with SIMBA	0.632	0.209	0.704	3.11
This work	MCR-style	0.856	0.241	0.816	9.34
	OAYT-style	0.662	0.182	0.642	2.76
	Dummy-free-style	1.266	0.333	1.195	8.21

Table 3: Field operation counts for constant-time CSIDH-512 group action evaluation. Counts are given in millions of operations, averaged over 1024 random experiments. The three speedups given in the last column are calculated with respect to the MCR, OAYT and dummy-free using the SIMBA approach as they were reported in [7]. We considered only multiplication and squaring operations and assumed $\mathbf{M} = \mathbf{S}$.

Tables 3 4 report the field arithmetic counting and clock cycles timings obtained for the CSIDH-512 constant-time group action evaluation, averaged over 1024 random experiments. The three speedup figures given in the last column are calculated with respect to the MCR, OAYT and Dummy-free using the SIMBA approach as they were reported in [7]. It can be seen that our approach produces noticeable savings compared

[4] The subsets of small odd primes and optimal strategies implmented can be easily obtained from our library.

Implementation	Group action evaluation	Mcycles	Speedup (%)
Cervantes-Vázquez *et al.* [7]	MCR-style	339	—
	OAYT-style	238	—
	Dummy-free	482	—
This work	MCR-style	298	12.09
	OAYT-style	225	5.46
	Dummy-free	431	10.58

Table 4: Clock cycle timings for constant-time CSIDH-512 group action evaluation, averaged over 1024 runs. The three speedups given in the last column are calculated with respect to the MCR, OAYT and dummy-free using the SIMBA approach as they were reported in [7].

against the MCR and Dummy-free SIMBA-based implementation of [7]. In the case of our OAYT-style implementation, the savings are more modest. Concretely, optimal strategies as applied to the MCR- OAYT- and Dummy-free- styles implementations yield a 12.09%, 5.46% and 10.58% speedup over [7], respectively (See Table 4).

As shown in Table 3, the OAYT SIMBA-based field operation count reported in [14] for the CSIDH group action stands as the smallest reported till date. Unfortunately, the source code corresponding to the implementation in [14] was not freely available, making a direct comparison with our implementation impossible. Moreover, the experiments reported in [14] correspond to the average of 200 random samples, which appear to be insufficient to eliminate experimental noise.[5]

For completeness, we give in Table 5, the expected field arithmetic counts for computing the CSIDH-512 group action using several combinations of the SIMBA-based method along with strategies. These estimates correspond to the output of a Python-script that interprets the algorithms and code presented by Cervantes *et al.* in [7] as they apply to the following settings:[6]

1. SIMBA-1-$\left(\frac{10}{\delta}\right)$ method with bounds $\overrightarrow{m} = (\frac{10}{\delta}, \ldots, \frac{10}{\delta})$ where $\delta = 1, 2$. This setting corresponds to the constant-time multiplicative-based strategy presented in Algorithm 2.

2. SIMBA-σ-κ method with σ and κ as proposed in [17] and [21], and using as bounds $\overrightarrow{m} = (\frac{10}{\delta}, \ldots, \frac{10}{\delta})$ with $\delta = 1, 2$, respectively.

3. *SIMBA-σ-κ method with strategies.* This is a SIMBA-σ-κ method but using optimal strategies on each batch. At each batch, an optimal strategy process isogenies

[5]The experimental noise is correlated to the number of rational elliptic curve points of torsion $\frac{(p+1)}{4\ell_i}$ (after k random samples) which is $\approx \frac{k}{\ell_i}$. Hence, the experiments of Hutchinson *et al.* in [14] do not appear to account for the case when large values for $\ell_i > 200$, are missing.

[6]Let us recall that the *SIMBA-σ-κ method* splits the set of n small odd primes ℓ_i into σ disjoint sets (batches) of size $\frac{n}{\sigma}$. Then it applies a multiplicative strategy on each batch. Each multiplicative strategy is evaluated κ times. Finally, it performs a multiplicative strategy on the set of unprocessed small odd primes until all the m_i degree-ℓ_i isogeny construction have been performed (See §4.1 for more details).

starting from the largest to the smallest.

4. A Python-code version of the C-code implementation presented in [7].

5. The improvements presented in this work with the following bound vectors:

   (a) $\overrightarrow{m} = (\frac{10}{\delta}, \ldots, \frac{10}{\delta})$ where $\delta = 1, 2$,

   (b) The ones proposed in Meyer-Campos-Reith [17] and Onuki *et al.* [21], and

   (c) The ones presented in section 4.4.

The last column in Table 5 gives the expected speedups for MCR- OAYT- and Dummy-free- styles using as a baseline the field arithmetic counts for multiplicative-based SIMBA-1-10 MCR style and multiplicative-based SIMBA-1-5 OAYT- and Dummy-free- styles, respectively.[7] The last three rows in Table 5 report the highest speedups. Notice that hese three rows correspond with the last three rows in Table 3. Interestingly, the usage of optimal strategies for the SIMBA-based approach cost approximately the same as a multiplicative-based SIMBA method. A graphical view of several of these CSIDH strategies can be found in Figures 2 and 3 of Appendix C.

6 Conclusions

The computational cost of the CSIDH group action evaluation directly depends on the number and degree of isogenies to be processed, which are determined by the n prime factors of $p + 1$. Another influential factor in the cost of this operation is given by the bound vector, which specifies the number of times that each one of those isogenies must be processed. In this work, we have given further evidence that the application of optimal strategies to the CSIDH computation can provide a noticeable performance speedup.

In the context of CSIDH, optimal strategies can be used to speedup the SIMBA method proposed in [17], which roughly speaking, corresponds to the framework reported by Hutchinson *et al.* in [14]. In this work, we dismiss the usage of the SIMBA method by employing optimal strategies as an intuitive generalization of the way that this technique is applied to SIDH. When optimal strategies *à la* SIDH are applied to CSIDH, they tend to exploit the cheap cost of isogeny evaluations with smaller degrees.

By following this approach, we proposed an efficient deterministic algorithm for computing optimal strategies for CSIDH. We report constant-time C-code implementations of three CSIDH variants: MCR-, OAYT-, and Dummy-free styles. As shown in Table 4, our experimental results achieve performance speedups of 12.09%, 5.46% and 10.58% compared with the MCR, OAYT and dummy-free SIMBA-based implementations reported in [7]. As a future work, we would like to apply our framework to larger primes that provide a larger quantum security than CSIDH-512.

[7]Notice that the cost of validating the public key was omitted from these estimates. However, as shown in the last row of Table reftab:estimates, the computational cost of this task is negligible.

Algorithm	Strategy	Bounds: $\vec{m}$	Group action evaluation	M	S	a	Speedup (%)
SIMBA-1-10	multiplicative	$(10,\ldots,10)$	MCR-style	1.635	0.638	1.943	—
	optimal			1.122	0.271	0.993	38.72
	multiplicative		dummy-free	1.933	0.660	2.132	—
	optimal			1.744	0.336	1.425	19.78
SIMBA-1-5	multiplicative	$(5,\ldots,5)$	OAYT-style	0.971	0.332	1.073	—
	optimal			0.888	0.178	0.739	18.19
This work	optimal	$(10,\ldots,10)$	MCR-style	1.037	0.257	0.926	43.07
			dummy-free	1.523	0.345	1.334	27.96
		$(5,\ldots,5)$	OAYT-style	0.776	0.183	0.695	26.40
SIMBA-5-11	multiplicative	$(10,\ldots,10)$	MCR-style	0.981	0.311	1.000	43.16
	optimal			0.981	0.311	1.000	43.16
	multiplicative		dummy-free	1.411	0.399	1.382	30.20
	optimal			1.412	0.399	1.382	30.16
SIMBA-3-8	multiplicative	$(5,\ldots,5)$	OAYT-style	0.719	0.206	0.710	29.00
	optimal			0.720	0.206	0.710	28.93
SIMBA-5-11	multiplicative	as given in [17]	MCR-style	0.900	0.297	0.939	47.34
	optimal			0.900	0.296	0.939	47.38
	multiplicative		dummy-free	1.309	0.392	1.324	34.40
	optimal			1.308	0.392	1.322	34.44
SIMBA-3-8	multiplicative	as given in [21]	OAYT-style	0.642	0.198	0.661	35.53
	optimal			0.643	0.198	0.661	35.46
This work	optimal	as given in [17]	MCR-style	0.930	0.242	0.851	48.44
			dummy-free	1.378	0.335	1.249	33.94
		as given in [21]	OAYT-style	0.670	0.173	0.626	35.30
This work	optimal	as given in section 4.4	MCR-style	0.835	0.231	0.784	**53.10**
			dummy-free	1.244	0.322	1.158	**39.61**
			OAYT-style	0.642	0.172	0.610	**37.53**
Public key validation	—			0.021	0.010	0.030	—

Table 5: Expected number of field operation for the constant-time CSIDH-512 group action evaluation. Counts are given in millions of operations, averaged over 1024 random experiments. The Speedup is computed using the multiplicative version of SIMBA-1-$\left(\frac{10}{\delta}\right)$ (with $\delta = 1, 2$) as a baseline, by only considering multiplication and squaring operations, and by assuming $\mathbf{M} = \mathbf{S}$. The last three rows in this table report the highest speedups. These three rows correspond with the last three rows in Table 3. Public key validation was separately measured, and presented in the last row of the table.

Acknowledgements. This work was partially done while the second author was visiting the University of Waterloo. The authors would like to thank Daniel Cervantes-Vázquez for his valuable comments that helped to improve the technical material of this paper. This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 804476).

References

[1] R. Azarderakhsh, M. Campagna, C. Costello, L. D. Feo, B. Hess, A. Jalali, D. Jao, B. Koziel, B. LaMacchia, P. Longa, M. Naehrig, G. Pereira, J. Renes, V. Soukharev, and D. Urbanik. "Supersingular isogeny key encapsulation". second round candidate of the NIST's post-quantum cryptography standardization process, 2017. Available at: `https://sike.org/`.

[2] D. J. Bernstein, M. Hamburg, A. Krasnova, and T. Lange, "Elligator: elliptic-curve points indistinguishable from uniform random strings". In 2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013, pages 967–980, 2013.

[3] D. J. Bernstein, T. Lange, C. Martindale, and L. Panny, "Quantum Circuits for the CSIDH: Optimizing Quantum Evaluation of Isogenies", *Advances in Cryptology — EUROCRYPT 2019*, LNCS 11477 (2019), 409–441.

[4] D. J. Bernstein, L. De Feo, A. Leroux, and B. Smith. Faster computation of isogenies of large prime degree. Cryptology ePrint Archive, Report 2020/341, 2020. Available at: `https://eprint.iacr.org/2020/341`,

[5] W. Castryck and T. Decru, "CSIDH on the surface", Cryptology ePrint Archive, Report 2019/1404, Available at `https://eprint.iacr.org/2019/1404`.

[6] W. Castryck, T. Lange, C. Martindale, L. Panny, and J. Renes, "CSIDH: An Efficient Post-Quantum Commutative Group Action", *Advances in Cryptology — ASIACRYPT 2018*, LNCS 11274 (2018), 395–427.

[7] D. Cervantes-Vázquez, M. Chenu, J.-J. Chi-Domínguez, L. De Feo, F. Rodríguez-Henríquez, and Benjamin Smith, Stronger and Faster Side-Channel Protections for CSIDH, *Progress in Cryptology - LATINCRYPT 2019.* LNCS 11774 (2019), 173-193

[8] D. Cervantes-Vázquez and E. Ochoa-Jiménez and F. Rodríguez-Henríquez Parallel strategies for SIDH: Towards computing SIDH twice as fast, Cryptology ePrint Archive, Report 2020/383, 2020. Available at: `https://eprint.iacr.org/2020/383`.

[9] D. Cervantes-Vázquez and F. Rodríguez-Henríquez. A note on the cost of computing odd degree isogenies. Cryptology ePrint Archive: Report 2019/1373, 2019. Available at: `https://eprint.iacr.org/2019/1373`.

[10] C. Costello and H. Hisil, "A simple and compact algorithm for SIDH with arbitrary degree isogenies", In T. Takagi and T. Peyrin, editors, Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security Part II, volume 10625 of Lecture Notes in Computer Science, pages 303–329. Springer, 2017.

[11] J.-M. Couveignes. Hard homogeneous spaces. Cryptology ePrint Archive, Report 2006/291, 2006. Available at: `http://eprint.iacr.org/2006/291`.

[12] L. De Feo, D. Jao and J. Plût, "Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies", *Journal of Mathematical Cryptology*, 8 (2014), 209–247.

[13] L. De Feo, J. Kieffer, and B. Smith, "Towards Practical Key Exchange from Ordinary Isogeny Graphs", *Advances in Cryptology — ASIACRYPT 2018*, LNCS 11274 (2018), 365–394.

[14] A. Hutchinson, J. LeGrow, B. Koziel, and R. Azarderakhsh. Further Optimizations of CSIDH: A Systematic Approach to Efficient Strategies, Permutations, and Bound Vectors. Cryptology ePrint Archive: Report 2019/1121, 2019. Available at `http://eprint.iacr.org/2019/1121`.

[15] A. Jalali, R. Azarderakhsh, M. Kermani, and D. Jao, "Towards Optimized and Constant-Time CSIDH on Embedded Devices", *Constructive Side-Channel Analysis and Secure Design — COSADE 2019*, LNCS 11421 (2019), 215–231.

[16] P. Longa. Practical quantum-resistant key exchange from supersingular isogenies and its efficient implementation. *Latincrypt 2019* Invited Talk. Available at: `https://latincrypt2019.cryptojedi.org/slides/latincrypt2019-patrick-longa.pdf`

[17] M. Meyer, F. Campos, and S. Reith, "On Lions and Elligators: An Efficient Constant-Time Implementation of CSIDH", *Post-Quantum Cryptography — PQCrypto 2019*, LNCS 11505 (2019), 307–325.

[18] M. Meyer and S. Reith, "A Faster Way to the CSIDH", *Progress in Cryptology — INDOCRYPT 2018*, LNCS 11356 (2018), 137–152.

[19] T. Moriya, H. Onuki, and T. Takagi. How to Construct CSIDH on Edwards Curves. Cryptology ePrint Archive: Report 2019/843, 2019. Available at `http://eprint.iacr.org/2019/843`.

[20] National Institute of Standards and Technology, "Submission requirements and evaluation criteria for the post-quantum cryptography standardization process", December 2016. Available from `https://csrc.nist.gov/csrc/media/projects/post-quantum-cryptography/documents/call-for-proposals-final-dec-2016.pdf`.

[21] H. Onuki, Y. Aikawa, T. Yamazaki, and T. Takagi. A Faster Constant-time Algorithm of CSIDH keeping Two Torsion Points. Cryptology ePrint Archive: Report 2019/353, 2019. Available at `http://eprint.iacr.org/2019/353`.

[22] A. Rostovtsev and A. Stolbunov. Public-key cryptosystem based on isogenies. Cryptology ePrint Archive, Report 2006/145, 2006.

[23] A. Stolbunov. Constructing public-key cryptographic schemes based on class group action on a set of isogenous elliptic curves. *Advances in Mathematics of Communication*, 4(2), 2010.

A Constant-time Algorithms for computing the CSIDH group action

Algorithm 3: OAYT style from [21]. Simplified constant-time CSIDH class group action for supersingular curves over $\mathbb{F}_p$, where $p = 4\prod_{i=1}^{n} \ell_i - 1$. The ideals $\mathfrak{l}_i = (\ell_i, \pi - 1)$ and $\mathfrak{l}_i^{-1} = (\ell_i, \pi + 1)$, where π maps to the p-th power Frobenius endomorphism on each curve. This algorithm computes exactly m isogenies for each ideal $\mathfrak{l}_i$ (or $\mathfrak{l}_i^{-1}$).

Input: A supersingular curve E_A over $\mathbb{F}_p$, and an exponent vector $(e_1, \ldots, e_n)$ with each $e_i \in [-m, m])$, m a positive number.

Output: $E_B = \mathfrak{l}_1^{e_1} * \cdots * \mathfrak{l}_n^{e_n} * E_A$.

$E_0 \leftarrow E_A$;
// Outer loop: Each ℓ_i prime f. is processed m times
for $i \in \{1, \ldots, m\}$ **do**
    $T_+, T_- \leftarrow$ `ObtainFullTorsionPoint`(E_0) ; // $T_\pm \in E_n[\pi \mp 1]$
    $T_+, T_- \leftarrow [4]T_+, [4]T_-$; // Now $T_+, T_- \in E_n\left[\prod_i \ell_i\right]$
    // Inner loop: processing each prime factor $\ell_i|(p+1)$;
    for $j \in \{0, 1, \ldots, n-1\}$ **do**
        $s \leftarrow$ `isequal`(`sign`$(e_j), -1)$;
        `cswap`(T_+, T_-, s) ; // swap ideals $\mathfrak{l}_{n-j}$ and $\mathfrak{l}_{n-j}^{-1}$
        $G_j \leftarrow T_+$;
        for $k \in \{1, \ldots, n-1-j\}$ **do**
            $G_j \leftarrow [\ell_k]G_j$
        $b \leftarrow$ `isequal`$(e_{n-j}, 0)$;
        $(E_{(j+1) \bmod n}, R) \leftarrow$ `QuotientIsogeny`(E_j, G_j, ℓ_{n-j}) ; // degree-ℓ_{n-j} isogeny
        $T'_+ \leftarrow [\ell_{n-j}]T_+$;
        $T_+ \leftarrow$ `PEVAL`(T_+, R); // Evaluate T_+ on degree-ℓ_{n-j} isogeny
        $T_- \leftarrow$ `PEVAL`(T_-, R); // Evaluate T_- on degree-ℓ_{n-j} isogeny
        `cswap`$(E_j, E_{(j+1) \bmod n}, b)$; // undo if $e_{n-j} = 0$
        `cswap`(T'_+, T_+, b) ; // undo if $e_{n-j} = 0$
        `cswap`(T'_-, T_-, b) ; // undo if $e_{n-j} = 0$
        $T_- \leftarrow [\ell_{n-j}]T_-$;
        `cswap`(T_+, T_-, s) ; // swap ideals $\mathfrak{l}_{n-j}$ and $\mathfrak{l}_{n-j}^{-1}$
        $e_{n-j} \leftarrow e_{n-j} - \big((b+1) \mod 2\big)$;
return E_0

B Executing optimal strategies for CSIDH

In this appendix, we give explicit details of how an optimal strategy can be executed in constant-time using the MCR, OAYT and Dummy-free approaches as described in §§4.1 4.3.

Algorithm 4: Dummy-free Style from [7]. Simplified constant-time CSIDH class group action for supersingular curves over $\mathbb{F}_p$, where $p = 4\prod_{i=1}^{n} \ell_i - 1$. The ideals $\mathfrak{l}_i = (\ell_i, \pi - 1)$ and $\mathfrak{l}_i^{-1} = (\ell_i, \pi + 1)$, where π maps to the p-th power Frobenius endomorphism on each curve. This algorithm computes exactly m isogenies for each ideal $\mathfrak{l}_i$ (or $\mathfrak{l}_i^{-1}$).

Input: A supersingular curve E_A over $\mathbb{F}_p$, and an exponent vector $(e_1, \ldots, e_n)$ with each $e_i \in \mathcal{S}(m))$, m a positive number.

Output: $E_B = \mathfrak{l}_1^{e_1} * \cdots * \mathfrak{l}_n^{e_n} * E_A$.

$E_0 \leftarrow E_A$;
// Outer loop: Each ℓ_i prime f. is processed m times
for $i \in \{1, \ldots, m\}$ **do**
  $T_+, T_- \leftarrow$ ObtainFullTorsionPoint(E_0) ; // $T_\pm \in E_n[\pi \mp 1]$
  $T_+, T_- \leftarrow [4]T_+, [4]T_-$; // Now $T_+, T_- \in E_n\left[\prod_i \ell_i\right]$
  // Inner loop: processing each prime factor $\ell_i | (p+1)$;
  for $j \in \{0, 1, \ldots, n-1\}$ **do**
    $s \leftarrow$ isequal(sign$(e_j), -1)$;
    cswap(T_+, T_-, s) ; // swap ideals $\mathfrak{l}_{n-j}$ and $\mathfrak{l}_{n-j}^{-1}$
    $G_j \leftarrow T_+$;
    for $k \in \{1, \ldots, n-1-j\}$ **do**
      $G_j \leftarrow [\ell_k]G_j$
    $(E_{(j+1) \bmod n}, R) \leftarrow$ QuotientIsogeny(E_j, G_j, ℓ_{n-j}) ; // degree-ℓ_{n-j} isogeny
    $T_+ \leftarrow$ PEVAL(T_+, R); // Evaluate T_+ on degree-ℓ_{n-j} isogeny
    $T_- \leftarrow$ PEVAL(T_-, R); // Evaluate T_- on degree-ℓ_{n-j} isogeny
    $T_- \leftarrow [\ell_{n-j}]T_-$;
    cswap(T_+, T_-, s) ; // swap ideals $\mathfrak{l}_{n-j}$ and $\mathfrak{l}_{n-j}^{-1}$
    $e_{n-j} \leftarrow e_{n-j} - 1$;
return E_0

B.1 Using one torsion point and dummy isogeny constructions (MCR-style)

The vertices of $S_n(L)$ are labeled as the pair of integers (i, j), where $0 \leq j < n$ and $0 \leq i < (n-j)$. The vertex (i, j) determines a single torsion-$\left(\prod_{k=i}^{n-j} \ell_k\right)$ point $T_{i,j} \in E_j(\mathbb{F}_p)$. The root of $S_n(L)$ is $(0, 0)$, its leaves are the vertices of the form $(n-1-j, j)$, and two vertices of the form (i, j) and (k, j) determines rational elliptic curve points on the same curve E_j. Now, for each ℓ_i let us define

$$b_{n-j-1} := \begin{cases} 1 & \text{if a dummy degree-}\ell_{n-j}\text{ isogeny construction is required,} \\ 0 & \text{otherwise.} \end{cases}$$

The navigation rules to walk across Δ_n are described as follows:

1. There are two types of edges: *horizontal* and *vertical* edges. Any *horizontal* edge $\left[(i, j), (i, j+1)\right]$ can be computed if and only if the leaf $(n-1-j, j)$ has been reached. Additionally, *vertical* edges of the form $\left[(i, j), (i+1, j)\right]$ are allowed for $0 \leq i < n-1-j$.

2. A ramification is a vertex having both *horizontal* and *vertical* edges.

3. At leaf $(n-1-j, j)$, the following computations and constant-time swaps take place:

$$\begin{aligned} T_{0,j}, T_{n-1-j,j} &\leftarrow \textbf{cswap}(T_{0,j}, T_{n-1-j,j}, b_{n-1-j}), \\ E_{j+1}, R &\leftarrow \textbf{QuotientIsogeny}(E_j, T_{n-1-j,j}, \ell_{n-j}), \\ E_j, E_{j+1} &\leftarrow \textbf{cswap}(E_j, E_{j+1}, b_{n-1-j}), \text{ and} \\ T_{0,j}, T_{n-1-j,j} &\leftarrow \textbf{cswap}(T_{0,j}, T_{n-1-j,j}, b_{n-1-j}). \end{aligned}$$

Here, $\textbf{QuotientIsogeny}(E_j, T_{n-1-j,j,\ell_{n-j}})$ is performed by assuming that $T_{n-1-j,j}$ has order-(ℓ_{n-j}), and its second output is a list of the multiples $R_k := [k]T_{n-1-j,j}$, for $k = 1, \ldots, d_{n-j}$ with $\ell_{n-j} = 2d_{n-j} + 1$ (*cf.* §2.2).

4. A *horizontal* edge corresponds to a decrement in the order of the current point by a factor ℓ_j. To be more precise, the edge $\big[(i,j),(i,j+1)\big]$ means that the following computations must be performed:

   (a) If $i = 0$: $R_{(d_{n-j}+1)} \leftarrow R_{d_{n-j}} + R_1$ and $T_{i,j} \leftarrow R_{d_{n-j}} + R_{(d_{n-j}+1)} = [\ell_{n-j}]R_1$.

   (b) Otherwise: $T_{i,j} \leftarrow [\ell_{n-j}]T_{i,j}$.

   In both cases, the following evaluation and constant-time swap have also been performed

$$\begin{aligned} T_{i,j+1} &\leftarrow \textbf{EvaluateIsogeny}(T_{i,j}, R), \text{ and} \\ T_{i,j}, T_{i,j+1} &\leftarrow \textbf{cswap}(T_{i,j}, T_{i,j+1}, b_{n-j-1}). \end{aligned}$$

5. A *vertical* edge corresponds to a decrease in the order of the current point by a scalar multiplication. In othe words, the edge $\big[(i,j),(i+1,j)\big]$ means that $T_{i+1,j} \leftarrow [\ell_i]T_{i,j}$ has been performed.

B.2 Using two torsion points and dummy isogeny constructions (OAYT-style)

The vertices of $S_n(L)$ are labeled by a integer pair (i,j) where $0 \leq j < n$ and $0 \leq i < (n-j)$. The vertex (i,j) determines a pair of torsion-$\big(\prod_{k=i}^{n-j} \ell_k\big)$ points $T_{+,i,j} \in E_i[\pi - 1]$ and $T_{-,i,j} \in E_i[\pi + 1]$. The root of $S_n(L)$ is $(0,0)$, its leaves are the vertices of the form $(n-1-j, j)$, and two vertices of the form (i,j) and (k,j) determines rational elliptic curve points on the same curve E_j. Now, for each ℓ_i let us define,

$$b_{n-1-j} := \begin{cases} 1 & \text{if dummy degree-}\ell_{n-j}\text{ isogeny construction is required,} \\ 0 & \text{otherwise;} \end{cases}$$

and

$$s_{n-1-j} := \begin{cases} 1 & \text{if } T_{-,i,j} \text{ is required,} \\ 0 & \text{if } T_{+,i,j} \text{ is required.} \end{cases}$$

Then, the navigation rules to walk across Δ_n are described as follows:

1. There are two types of edges: *horizontal* and *vertical* edges. Any *horizontal* edge $\big[(i,j),(i,j+1)\big]$ can be computed if and only if the leaf $(n-1-j,j)$ has been reached. Additionally, *vertical* edges $\big[(i,j),(k,j)\big]$ are only allowed when $i<k$.

2. A ramification is a vertex having both *horizontal* and *vertical* edges.

3. At leaf $(n-1-j,j)$, the following computations and constant-time swaps are perform:

$$\begin{aligned}
T_{+,0,j}, T_{-,0,j} &\leftarrow \textbf{cswap}(T_{+,0,j}, T_{-,0,j}, s_{n-1-j}),\\
T_{+,n-1-j,j}, T_{-,n-1-j,j} &\leftarrow \textbf{cswap}(T_{+,n-1-j,j}, T_{-,n-1-j,j}, s_{n-1-j}),\\
T_{+,0,j}, T_{n-1-j,j} &\leftarrow \textbf{cswap}(T_{+,0,j}, T_{+,n-1-j,j}, b_{n-1-j}),\\
E_{j+1}, R &\leftarrow \textbf{QuotientIsogeny}(E_j, T_{+,n-1-j,j}),\\
E_j, E_{j+1} &\leftarrow \textbf{cswap}(E_j, E_{j+1}, b_{n-1-j}),\\
T_{+,0,j}, T_{n-1-j,j} &\leftarrow \textbf{cswap}(T_{+,0,j}, T_{+,n-1-j,j}, b_{n-1-j}),\\
T_{+,n-1-j,j}, T_{-,n-1-j,j} &\leftarrow \textbf{cswap}(T_{+,n-1-j,j}, T_{-,n-1-j,j}, s_{n-1-j}), \text{ and}\\
T_{+,0,j}, T_{-,0,j} &\leftarrow \textbf{cswap}(T_{+,0,j}, T_{-,0,j}, s_{n-1-j}).
\end{aligned}$$

Here, $\textbf{QuotientIsogeny}(E_j, T_{+,n-1-j,j})$ is performed by assuming that $T_{+,n-1-j,j}$ has order-(ℓ_{n-j}), and its second output is a list of the multiples $R_k := [k]T_{+,n-1-j,j}$ for $k=1,\ldots,d_{n-j}$ with $\ell_{n-j}=2d_{n-j}+1$ (*cf.* §2.2).

4. A *horizontal* edge corresponds to a decrease in the order of the current point by a factor of ℓ_{n-j}. To be more precise, the edge $\big[(i,j),(i,j+1)\big]$ means that the following computations have been performed:

$$T_{+,i,j}, T_{-,i,j} \leftarrow \textbf{cswap}(T_{+,i,j}, T_{-,i,j}, s_{n-1-j}), \text{ and}$$

   (a) If $i=0$: $R_{(d_{n-j}+1)} \leftarrow R_{d_{n-j}} + R_1$ and $T_{+,i,j} \leftarrow R_{d_{n-j}} + R_{(d_{n-j}+1)} = [2d_{n-j}+1]R_1$.

   (b) Otherwise: $T_{+,i,j} \leftarrow [\ell_{n-j}]T_{+,i,j}$.

In both cases, the following evaluation and constant-time swap have also been performed

$$\begin{aligned}
T_{-,i,j} &\leftarrow [\ell_{n-j}]T_{-,i,j},\\
T_{+,i,j+1} &\leftarrow \textbf{EvaluateIsogeny}(T_{+,i,j}, R),\\
T_{-,i,j+1} &\leftarrow \textbf{EvaluateIsogeny}(T_{-,i,j}, R),\\
T_{+,i,j}, T_{+,i,j+1} &\leftarrow \textbf{cswap}(T_{+,i,j}, T_{+,i,j+1}, b_{n-1-j}),\\
T_{-,i,j}, T_{-,i,j+1} &\leftarrow \textbf{cswap}(T_{-,i,j}, T_{-,i,j+1}, b_{n-1-j}), \text{ and}\\
T_{+,i,j+1}, T_{-,i,j+1} &\leftarrow \textbf{cswap}(T_{+,i,j+1}, T_{-,i,j+1}, s_{n-1-j})
\end{aligned}$$

5. A *vertical* edge corresponds to a decrease in the order of the current point by a scalar multiplication. In othe words, the edge $\big[(i,j),(i+1,j)\big]$ means that the following operations has been performed:

(a) If there are no ramifications between the vertices (i, j) and $(n-1-j, j)$:

$$\begin{aligned} T_{+,i,j}, T_{-,i,j} &\leftarrow \mathbf{cswap}(T_{+,i,j}, T_{-,i,j}, s_{n-1-j}), \\ T_{+,i+1,j} &\leftarrow [\ell_i]T_{+,i,j}, \\ T_{+,i+1,j}, T_{-,i+1,j} &\leftarrow \mathbf{cswap}(T_{+,i+1,j}, T_{-,i+1,j}, s_{n-1-j}), \text{ and} \\ T_{+,i,j}, T_{-,i,j} &\leftarrow \mathbf{cswap}(T_{+,i,j}, T_{-,i,j}, s_{n-1-j}). \end{aligned}$$

(b) Otherwise:

$$\begin{aligned} T_{+,i+1,j} &\leftarrow [\ell_i]T_{+,i,j}, \text{ and} \\ T_{-,i+1,j} &\leftarrow [\ell_i]T_{-,i,j}. \end{aligned}$$

B.3 Using two torsion point without dummy isogeny constructions (Dummy-free style)

The vertices of $S_n(L)$ are labeled by a integer pair (i, j) where $0 \leq j < n$ and $0 \leq i < (n-j)$. The vertex (i, j) determines a pair of torsion-$\left(\prod_{k=i}^{n-j} \ell_k\right)$ points $T_{+,i,j} \in E_i[\pi - 1]$ and $T_{-,i,j} \in E_i[\pi + 1]$. The root of $S_n(L)$ is $(0, 0)$, its leaves are the vertices of the form $(n-1-j, j)$, and two vertices of the form (i, j) and (k, j) determines rational elliptic curve points on the same curve E_j. Now, for each ℓ_i let's define

$$s_{n-1-j} := \begin{cases} 1 & \text{if } T_{-,i,j} \text{ is required,} \\ 0 & \text{if } T_{+,i,j} \text{ is required;} \end{cases}$$

then, the navigation rules to walk across Δ_n are described as follows:

1. There are two types of edges: *horizontal* and *vertical* edges. Any *horizontal* edge $\big[(i, j), (i, j+1)\big]$ can be computed if and only if the leaf $(n-1-j, j)$ has been reached. Additionally, *vertical* edges $\big[(i, j), (k, j)\big]$ are only allowed when $i < k$.

2. A ramification is a vertex having both *horizontal* and *vertical* edges.

3. At leaf $(n-1-j, j)$, the following computations and constant-time swaps are perform:

$$\begin{aligned} T_{+,n-1-j,j}, T_{-,n-1-j,j} &\leftarrow \mathbf{cswap}(T_{+,n-1-j,j}, T_{-,n-1-j,j}, s_{n-1-j}), \\ E_{j+1}, R &\leftarrow \mathbf{QuotientIsogeny}(E_j, T_{+,n-1-j,j}), \\ E_j, E_{j+1} &\leftarrow \mathbf{cswap}(E_j, E_{j+1}, b_{n-1-j}), \text{ and} \\ T_{+,n-1-j,j}, T_{-,n-1-j,j} &\leftarrow \mathbf{cswap}(T_{+,n-1-j,j}, T_{-,n-1-j,j}, s_{n-1-j}). \end{aligned}$$

   Here, **QuotientIsogeny**$(E_j, T_{+,n-1-j,j})$ is performed by assuming that $T_{+,n-1-j,j}$ has order-(ℓ_{n-j}), and its second output is a list of the multiples $R_k := [k]T_{+,n-1-j,j}$ for $k = 1, \ldots, d_{n-j}$ with $\ell_{n-j} = 2d_{n-j} + 1$ (*cf.* §2.2).

4. A *horizontal* edge corresponds to a decrease in the order of the current point by a factor of ℓ_{n-j}. To be more precise, the edge $\big[(i,j),(i,j+1)\big]$ means that the following computations and constnat-time swaps have been performed:

$$\begin{aligned}
T_{+,i,j}, T_{-,i,j} &\leftarrow \mathbf{cswap}(T_{+,i,j}, T_{-,i,j}, s_{n-1-j}),\\
T_{-,i,j} &\leftarrow [\ell_{n-j}]T_{-,i,j},\\
T_{+,i,j+1} &\leftarrow \mathbf{EvaluateIsogeny}(T_{+,i,j}, R),\\
T_{-,i,j+1} &\leftarrow \mathbf{EvaluateIsogeny}(T_{-,i,j}, R), \text{ and}\\
T_{+,i,j}, T_{-,i,j} &\leftarrow \mathbf{cswap}(T_{+,i,j}, T_{-,i,j}, s_{n-1-j}).
\end{aligned}$$

5. A *vertical* edge corresponds to a decrease in the order of the current point by a scalar multiplication. In othe words, the edge $\big[(i,j),(i+1,j)\big]$ means that the following operations has been performed:

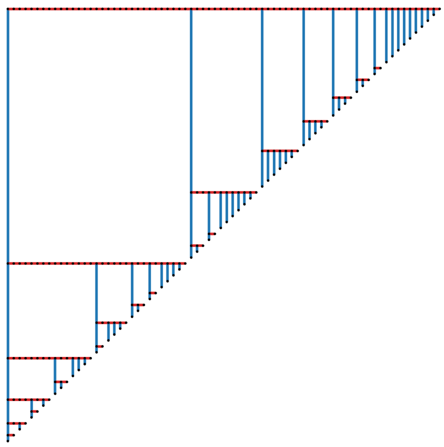
   (a) If there are no ramifications between the vertices (i,j) and $(n-1-j,j)$:

$$\begin{aligned}
T_{+,i,j}, T_{-,i,j} &\leftarrow \mathbf{cswap}(T_{+,i,j}, T_{-,i,j}, s_{n-1-j}),\\
T_{+,i+1,j} &\leftarrow [\ell_i]T_{+,i,j},\\
T_{+,i+1,j}, T_{-,i+1,j} &\leftarrow \mathbf{cswap}(T_{+,i+1,j}, T_{-,i+1,j}, s_{n-1-j}), \text{ and}\\
T_{+,i,j}, T_{-,i,j} &\leftarrow \mathbf{cswap}(T_{+,i,j}, T_{-,i,j}, s_{n-1-j}).
\end{aligned}$$

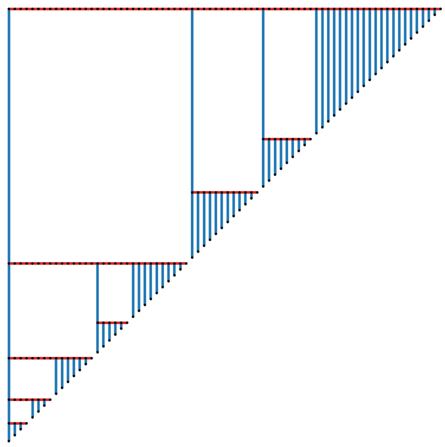
   (b) Otherwise:

$$\begin{aligned}
T_{+,i+1,j} &\leftarrow [\ell_i]T_{+,i,j}, \text{ and}\\
T_{-,i+1,j} &\leftarrow [\ell_i]T_{-,i,j}.
\end{aligned}$$

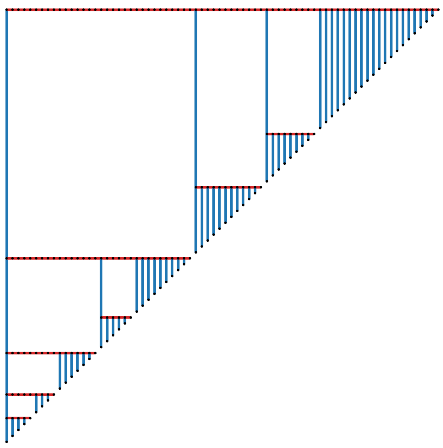
C A graphical view of CSIDH strategies



(a) Simplified MCR-style: requires 10 rounds.

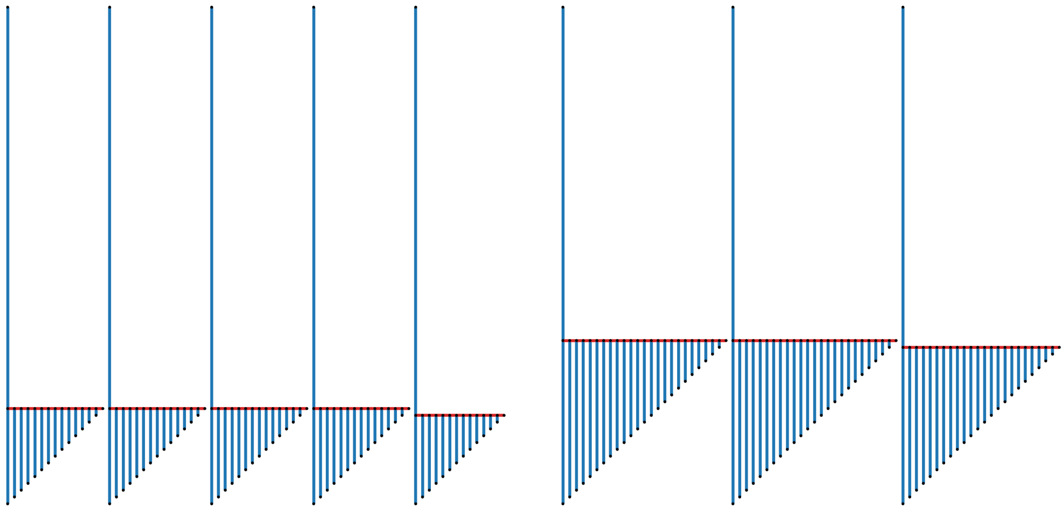


(b) Simplified OAYT-style: requires 5 rounds.



(c) Simplified Dummy-free style: requires 10 rounds.

Figure 2: A graphical view of the strategies followed by three variants of the CSIDH group action evaluation: MCR style as presented in [17], OAYT style as proposed in [21] and dummy-free style as presented in [7]. Horizontal edges (in red) and vertical edges (in blue) represent isogeny evaluations q_{ℓ_i}, and scalar multiplications p_{ℓ_i}, respectively.



SIMBA-5-11: MCR style

SIMBA-3-8: OAYT style

Figure 3: Two variants of the CSIDH group action evaluation: MCR style as proposed in [17] and OAYT style as proposed in [21]. Each one of the two aproaches depicted in this figure, computes a group action using the SIMBA-σ-κ method, constructing isogenies of prime degree grouped in σ batches. Each round must be repeated κ times. A final repair round applies a *multiplicative strategy* to process the prime factors not covered during the κ rounds. Horizontal edges (in red) and vertical edges (in blue) represent isogeny evaluations q_{ℓ_i}, and scalar multiplications p_{ℓ_i}, respectively.