OPTIMAL STRATEGIES FOR CSIDH

JESÚS-JAVIER CHI-DOMÍNGUEZ∗
Faculty of Information Technology and Communication Sciences
Tampere University, Hervanta Campus
Korkeakoulunkatu 1, 33720 Tampere, Finland

FRANCISCO RODRÍGUEZ-HENRÍQUEZ
Computer Science Department, Cinvestav IPN, Zacatenco Unit
Av. IPN no. 2508, San pedro Zacatenco
Gustavo A. Madero, 07300 Mexico city, Mexico

(Communicated by Lih-Chung Wang)

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. We also report an estimated number of field operations for larger instantiations of this protocol, namely, CSIDH-1024 and CSIDH-1792.

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 the Couveignes-Rostovtsev-Stolbunov scheme [11, 24, 23], by exploiting 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 more than 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 nine hundred million clock cycles to compute a shared secret (cf. Table 5). For comparison, the SIKE protocol instantiated with a 434-bit prime, requires some twenty million clock cycles [16].

2020 Mathematics Subject Classification: Primary: 94A60, 11T71; Secondary: 14L30.
Key words and phrases: Isogeny-based cryptography, post-quantum cryptography, CSIDH protocol, key-exchange protocol, group action evaluation.

The first author is supported by ERC grant agreement No. 804476.
∗ Corresponding author: Jesús-Javier Chi-Domínguez.
The first constant-time implementation of CSIDH was reported by Bernstein, Lange, Martindale, and Panny in [3]. However, the authors of [3] focused their analysis on assessing the quantum security level provided by CSIDH. 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 secret exponents using different interval bounds depending on the the CSIDH small prime factors \( \ell_i \).

Later, the CSIDH implementation of [17] was further improved by Onuki, Aikawa, Yamazaki, and Takagi in [22], by keeping track of two points to evaluate the action of an ideal: one in \( E(\mathbb{F}_p) \), and one in \( E(\mathbb{F}_{p^2}) \) with its \( x \)-coordinate in \( \mathbb{F}_p \). Moreover, Moriya, Onuki and Takagi [19], and Cervantes-Vázquez et al. in [7], performed more efficient 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.

The group action algorithm proposed in the original CSIDH protocol of [6], relied on a multiplication-based approach for constructing and evaluating isogenies. It was first stated in [3, §8] that this multiplication-based procedure could possibly be improved by adapting the SIDH optimal strategy approach introduced by deFeo, Jao and Plût in [12].

Shortly after [3], 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 presented in [12]; a more efficient re-ordering of the CSIDH small prime factors \( \ell_i \); and a procedure to find optimal bounds for the CSIDH exponents. However, when optimizing the CSIDH-512 variant proposed in [22], the best strategy reported by [14] corresponds to the multiplication-based SIMBA approach of [17]. In spite of its outstanding performance, this result is unsatisfactory in the sense that the main contribution of [14] was to introduce optimal strategies other than the multiplication-based one already proposed and utilized in [17]. In practice it appears as if in this setting, the approach of [14] strictly reduces to finding optimal CSIDH public parameters, namely, optimal bound vectors and SIMBA partitions. Further, the authors of [14] also claim a 13.04% speedup over the publicly available software library of [17], this time using optimal strategies different than multiplication-based ones. Unfortunately, we were unable to compile the software provided by [14] for this CSIDH variant.

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. Let \( L := [\ell_1, \ell_2, \ldots, \ell_{74}] \) be the list of small odd prime numbers such that \( p = 4 \cdot \prod_{i=1}^{74} \ell_i - 1 \) is the prime number used in CSIDH. One can find CSIDH optimal strategies for each possible ordering of the set of primes \( L \). However, since dealing with all such orderings is clearly computationally unfeasible, in this work we heuristically assume that an arrangement of the set \( L \) from the smallest to the largest \( \ell_i \), is close to the global optimal. For this fixed ordering, we present a procedure that finds an optimal strategy with cubic complexity with respect to \( n \).
Optimal strategies for CSIDH

The main difference of our procedure with the framework presented in [14], is that the strategies proposed in this paper do not rely on the SIMBA approach of [17], but rather, they are an intuitive generalization of how the SIDH strategies can be applied to CSIDH. We note that our approach never declares a multiplication-based strategy as an optimal strategy. Moreover, the CSIDH optimal strategies proposed here comply with the same codification utilized by the SIDH protocol. Additionally, we report constant-time C-code implementations of three instantiations of the CSIDH protocol, namely, MCR [17], OAYT [22], and the dummy-free [7] variants. Our experimental results achieve performance speedups of 12.09%, 3.36% and 10.58% compared with the MCR, OAYT and dummy-free styles as presented in [7].

For a fair comparison with the implementation results reported in [14], we executed in our server the Octave script provided by the authors of [14]. This script finds OAYT CSIDH-512 optimal public parameters, by producing a header and C files that should be executed using the C software library of [7]. This way, it was experimentally found that the field arithmetic operation and clock cycle associated to the OAYT CSIDH-512 computation using the parameters found by the script of [14], is 0.7% cheaper and 0.43% faster than the ones corresponding to our work for this CSIDH variant, respectively.

We stress that this comparison was made between our optimal strategies à la SIDH against a SIMBA multiplication-based approach as proposed in [17] boosted with optimized public parameters found by [14]. The latter setting does not use a non multiplicative-based strategy.

Finally, we also report estimated field arithmetic operation costs of the CSIDH instantiations CSIDH-1024 and CSIDH-1792. Our software library is freely available at,

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

Remark 1. Let $E$ and $E'$ be two supersingular elliptic curves defined over $\mathbb{F}_p$ for which there exists a cyclic degree-$\ell$ 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 $O(\sqrt{\ell})$ operations, the co-domain elliptic curve $E'$ and $\phi(Q)$ and the image of a point $Q \in E(\mathbb{F}_p)$ with $P \notin \text{Ker}(\phi)$. 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 the CSIDH variants 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 experimental results and comparison with related works are reported. Finally, in §6 we draw some concluding remarks.

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 $\text{isequal}(X,Y)$ is defined, returning 1 if $X = Y$ and 0 otherwise. We also assume that a constant-time conditional swap $\text{cswap}(X,Y,b)$ 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 \( \ell \), dividing \( \frac{p+1}{3} \). Hence, one can simply compute the scalar multiplication operation \([k]P\) as the composition of the shortest differential addition chains for each prime \( \ell \) dividing \( k \). Note that all these shortest additions chains can be pre-computed off-line. Montgomery ladders using differential addition chains can perform the scalar multiplication operation \([k]P\) with an average length of about 1.5\( \log_2(k) \) steps [7]. Each Montgomery ladder step involves the computation of one differential point addition and differential point doubling at a cost of \( 4M + 6A \) and \( 4M + 2S + 4A \), 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 \( \ell \) 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-\( \ell \) isogeny \( \phi : E \rightarrow E' \) defined over \( \mathbb{F}_p \). This implies that there must exist an \( \ell \)-order point \( P \in E(\mathbb{F}_p) \) such that \( \text{Ker}(\phi) = \{ \infty, \pm P, \pm 2P, \ldots, \pm [d]P \} \). Given the domain elliptic curve \( E \) and an \( \ell \)-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, \( \text{KPS}, \text{xISOG} \) and \( \text{xEVAL} \). Let us define \( \text{KPS} \) as the task of computing the first \( d \) multiples of the point \( P \), namely, the set \( \langle P \rangle = \{ P, [2]P, \ldots, [d]P \} \). Using \( \text{KPS} \) as a building block, the module \( \text{xISOG} \) computes the per-field constants that define the co-domain curve \( E' \) over \( \mathbb{F}_p \). Also, using \( \text{KPS} \) as a building block, \( \text{xEVAL} \) computes the image point \( \phi(Q) \). Note that \( \text{KPS} \) becomes more expensive than \( \text{xEVAL} \) starting from \( \ell \geq 11 \). When \( \ell \leq 7 \), the block \( \text{KPS} \) is considerably cheaper or even free of cost for the case \( \ell = 3 \).

Observe also that since \( \text{xISOG} \) and \( \text{xEVAL} \) show no dependencies between them, once that the kernel points have been computed, it is possible to compute \( \text{xISOG} \) and \( \text{xEVAL} \) in parallel. Furthermore, when evaluating an arbitrary number of points in \( E \) that do not belong to the \( \text{Ker}(\phi) \) subgroup, \( \text{KPS} \) must be computed only once. This implies that the computational cost associated to \( \text{KPS} \) gets amortized when computing the image of two or more points. Table 1 summarizes the field arithmetic costs associated to the \( \text{KPS}, \text{xISOG}, \) and \( \text{xEVAL} \) procedures. Note that \( \text{KPS} \) is a straightforward computation that can be performed at the cost of one point doubling and \( k - 2 \) point additions. Efficient formulas for computing \( \text{xEVAL} \) can be found in [10] and [7] for Montgomery and twisted Edwards curves, respectively.

Example. 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 \( \text{KPS}, \text{xEVAL} \) and \( \text{xISOG} \) 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 \( \text{KPS} \) computation and therefore it is only two times more expensive than finding the multiple [127]T.
Optimal strategies for CSIDH

Montgomery [10] Edwards [7]

\[ \ell P \]

\[ KPS \]

\[ xEVAL \]

\[ xISOG \]

\[ \ell + 2\lambda + 1 \]

\[ 2(\lambda + 2) \]

\[ \lambda \]

\[ \bar{\lambda} \approx \log_2(\lambda) \]

Table 1. Costs for computing prime degree-\( \ell \) isogenies with \( \ell = 2d + 1 \) using the KPS, xEVAL and xISOG building blocks. Field multiplication (M) and squaring (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 \( 4M + 2S + 6A \) and \( 4M + 2S + 4A \), respectively. We define \( \lambda = \lceil \log_2(\ell) \rceil \) and \( \bar{\lambda} \approx \log_2(\lambda) \).

Table 2. Approximate arithmetic costs for computing prime degree-\( \ell \) isogenies with \( \ell = 2d + 12 \cdot 64 + 1 = 127 \), using the KPS, xEVAL, and xISOG primitives. The cost of computing the scalar multiplication \([127]T \) is also reported.

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

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.

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: \( \ell_7 = 3, \ell_7 = 5, \ldots, \ell_2 = 373, \) given by the first 73 odd primes, and \( \ell_1 = 587. \) Let \( E/F_p \) be a supersingular elliptic curve given in Montgomery form as,

\[ E/F_p: y^2 = x^3 + Ax^2 + x; \]

It follows that \( \#E(F_p) = (p + 1) = 4 \prod_{i=1}^{n} \ell_i. \) Additionally, let \( \pi: (x, y) \mapsto (x^p, y^p) \) be the Frobenius map and \( N \in \mathbb{Z} \) be a positive integer. Then, \( E[N] := \{ P \in E(F_p) : [N]P = \mathcal{O} \} \) denotes the \( N \)-torsion subgroup of \( E/F_p \). Similarly, \( E[\pi - 1] := \{ P \in E(F_p) : (\pi - 1)P = \mathcal{O} \} \) and \( E[\pi + 1] := \{ P \in E(F_p) : (\pi + 1)P = \mathcal{O} \} \) denote the subgroups of \( F_p \)-rational and zero-trace points, respectively. In particular, any
Algorithm 1 Simplified constant-time CSIDH class group action for supersingular curves over \( \mathbb{F}_p \), where \( p = 4 \prod_{i=1}^{\ell_i} \ell_i - 1 \). The ideals \( \ell_i = (\ell_i, \pi - 1) \), where \( \pi \) maps to the \( p \)-th power Frobenius endomorphism on each curve. This algorithm computes exactly \( m \) isogenies for each ideal \( \ell_i \).

**Require:** 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.

**Ensure:** \( E_B = \ell_1^{e_1} \cdots \ell_n^{e_n} E_A \).

1: \( E_0 \leftarrow E_A \)
2: // Outer loop: Each \( \ell_i \) prime is processed \( m \) times
3: for \( i \leftarrow 1 \) to \( m \) do
4: \( T \leftarrow \text{ObtainFullTorsionPoint}(E_0) \quad \text{// } T \in E_0[\pi - 1] \)
5: \( T \leftarrow [4]T \quad \text{// } \text{Now } T \in E_0 \cap \prod \ell_i \)
6: // Inner loop: processing each prime factor \( \ell_j(p + 1) \)
7: for \( j \leftarrow 0 \) to \( (n - 1) \) do
8: \( G_j \leftarrow T \)
9: for \( k \leftarrow 1 \) to \( (n - 1 - j) \) do
10: \( G_j \leftarrow (\ell_k)G_j \)
11: end for
12: if \( e_{n-j} \neq 0 \) then
13: \( (E_{j+1} \mod n, (G_{j+1})) \leftarrow \text{QuotientIsogeny}(E_j, G_j, \ell_{n-j}) \)
14: \( T \leftarrow \text{xEVAL}(T, (G_j)) \)
15: \( e_{n-j} \leftarrow e_{n-j} - 1 \)
16: else
17: // Dummy operations
18: \( (E_{j+1} \mod n, (G_{j+1})) \leftarrow \text{QuotientIsogeny}(E_j, G_j, \ell_{n-j}) \)
19: \( \text{xEVAL}(T, (G_j)) \)
20: \( T \leftarrow [\ell_{n-j}]T \)
21: \( E_{j+1} \mod n \leftarrow E_j \)
22: end if
23: end for
24: end for
25: return \( E_0 \)

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 \( \pi^2 = -1 \).

### 3.1. 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-\( \ell_i \) 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 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 Step 4 of the outer loop of Steps 3-24, a full order point \( T \in E_0 \) (i.e., a point having order \( p+1 \)), 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 \( E_n[\pi - 1] \).

1Note 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
Algorithm 2 Simplified constant-time CSIDH class group action for supersingular curves over $\mathbb{F}_p$, where $p = 4 \prod_{i=1}^{74} \ell_i - 1$. The ideals $\mathfrak{l}_i = (\ell_i, \pi - 1)$, where $\pi$ 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)

Require: 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.
Ensure: $E_{0'} = \ell_1^{e_1} \cdots \ell_n^{e_n} + E_A$.

1: $E_0 \leftarrow E_A$  
2: // Outer loop: Each $\mathfrak{l}_i$ prime f. is processed $m$ times  
3: for $i \leftarrow 1$ to $m$ do  
4: $T \leftarrow \text{ObtainFullTorsionPoint}(E_0)$ // $T \in E_0[0, \pi - 1]$  
5: $T' \leftarrow |4|T$ // Now $T \in E_0[\prod_1 \mathfrak{l}_i]$  
6: // Inner loop: processing each prime factor $\mathfrak{l}_i | (p + 1)$  
7: for $j \leftarrow 0$ to $(n - 1)$ do  
8: $G_j \leftarrow T$  
9: for $k \leftarrow 1$ to $(n - 1 - j)$ do  
10: $G_j \leftarrow [e_k]G_j$  
11: end for  
12: $b \leftarrow \text{isequal}(e_{n - j}, 0)$  
13: $(E_{j+1}) \leftarrow \text{QuotientIsogeny}(E_j, G_j, \mathfrak{l}_{n - j})$  
14: $T' \leftarrow [\mathfrak{l}_{n-j}]T'$  
15: $T \leftarrow \text{xEVAL}(T; \langle G_j \rangle)$ // Evaluate $T$ on degree-$\ell_{n-j}$ isogeny  
16: $\text{cswap}(E_j, E_{j+1})$ mod $n$, $b$ // undo if $e_{n - j} = 0$  
17: $\text{cswap}(T', T, b)$ // undo if $e_{n - j} = 0$  
18: $e_{n - j} \leftarrow e_{n - j} - ((b + 1) \mod 2)$  
19: end for  
20: end for  
21: return $E_0$

Thereafter, the inner loop of Steps 7-23 constructs and evaluates a degree-$\ell_{n-j}$ isogeny for each one of the prime factors 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 3-24 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 12-15 and Steps 16-22 are carefully balanced, and the conditional statements are substituted 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 $e$.

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

Remark 2. 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 $1174 \approx 2^{256}$ 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 3. Notice that the scalar multiplication required in Step 14 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 \(d_{n-j}\) points \([T, 2T, \ldots, [d_{n-j}]T]\), and

\[
[l_{n-j}]T := [2d_{n-j} + 1]T = [d_{n-j}]T + [d_{n-j} + 1]T = R[d_{n-j}] + [d_{n-j} + 1]T = R[d_{n-j}] + (R[d_{n-j}] + R[1])
\]

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

3.2. 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 \(\Delta_n\) of side \(n\) having \(\frac{n(n+1)}{2}\) vertices distributed in \(n\) columns and rows (See Figure 1a).

The vertices of \(\Delta_n\) represent elliptic curve points and its vertical and horizontal edges have as associated weight \(p_{\ell_i}\) and \(q_{\ell_i}\), defined as the cost of performing one scalar multiplication by \(\ell_i\) and evaluating a degree-\(\ell_i\) 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 \(\Delta_n\). 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 leaf in the column. Each one of the \(n\) columns of \(\Delta_n\) corresponds to an isogenous supersingular elliptic curve \(E_j\), for \(j = n, 1, 2, \ldots, n - 1\).
Remark 4. 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, make an abundant use of 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 $p + 1 = \prod_i \ell_i$ 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 $G_0 = \prod_{i=1}^{n-1} \ell_i T$. If $G_0$ is finite, then it has to have order $\ell_n$ and can be used to generate the subgroup corresponding to the kernel of the isogeny $\phi_{\ell_n}$. The leaf $G_1$ is defined as,

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

Provided that $T$ is a full order point, the point $G_1$ is guaranteed to be finite and of order $\ell_n - 1$. In general, for $j = 1, \ldots, n - 1$, one has

$$G_j = \prod_{i=1}^{n-1-j} \ell_i \phi_{\ell_{n+1-j}}(\ldots(\phi_{\ell_n}(T))\ldots).$$

If some secret exponent $e_k = 0$, for $k = 1, \ldots, n$, then the corresponding isogeny evaluation $\phi_{\ell_k}$ 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 of obtaining, 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 $\phi_{\ell_{n-1}} : E_{n-1} \rightarrow E_n$, can be obtained by simply constructing a degree-$\ell_{n-1}$ isogeny with kernel $G_{n-1}$, which coincides with the domain or image of $\phi_{\ell_{n-1}}$ depending if $e_1 = 0$ or not, respectively.

The naïve strategy followed by Algorithm 1 is depicted in Figure 1a, instantiated for the CSIDH prime $p_{512}$ with 74 prime factors $\ell_i$ such that $\ell_i | (p + 1)$. The computation of the triangle $\Delta_n$ shown in Figure 1a represents one full execution of the inner loop of Steps 7-23. This computation should be repeated $m = 10$ times in order to complete the CSIDH group action (cf. Remark 2). From Figure 1a, it can be seen that Algorithm 1 follows a pure multiplicative strategy, where $n(n+1)/2 = 2775$ scalar multiplications by the scalars $\ell_i$ for $i = 1, \ldots, 74$, are performed; plus the construction and evaluation of only 74 degree-$\ell_i$ isogenies.

Assuming that in average, one scalar multiplication computation $[\ell]T$ is at least five times less expensive than a degree-$\ell$ 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].

---

2Another 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.
3.3. Optimal strategies for CSIDH. 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. A CSIDH strategy is a weighted subgraph \( S_n(L) \) contained into a discrete rectangular triangle \( \Delta_n \) of side \( n \). Any strategy \( S_n(L) \) has an associated cost defined as,

\[
C(S_n) = \sum_{x \in \text{edges}(S_n(L))} \omega(x) + \sum_{j=0}^n \nu((n-1-j,j)),
\]

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 \( C_n(S_n(L)) < C_n(S'_n(L)) \) holds. Optimal strategies were defined in [12] within the context of the SIDH protocol. In [12] the fact that a triangle \( \Delta_n \) can be optimally and recursively decomposed into two sub-triangles \( \Delta_h \) and \( \Delta_{n-h} \) was exploited as shown in Figure 1b. Let us denote as \( \Delta^h_n \) the design decision of splitting a triangle \( \Delta_n \) at row \( h \). The sequential cost of walking across the strategy \( S_n(L) \), which is a subgraph of \( \Delta^h_n \), is given as

\[
C(S^h_n(L)) = C(S_h(L_h)) + C(S_{n-h}(L_{n-h})) + \sum_{i=1}^{n-h} q_i + \sum_{i=0}^{h-1} p_{\ell_{n-i}},
\]

where \( L_h = [\ell_{n-h+1}, \ldots, \ell_n] \) and \( L_{n-h} = [\ell_1, \ldots, \ell_{n-h}] \) are two disjoint sublists of \( L \) and have size \( h \) and \( n-h \), respectively. We say that \( S^h_n(L) \) is optimal if \( C(S^h_n(L)) \) is minimal among all \( S^h_n(L) \) 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 the computation of more scalar multiplications. This is due to the fact that these operations are several times cheaper than isogeny evaluations for a sufficiently large prime degree \( \ell \) (cf. Example 2.2).

On the other hand, since evaluating/constructing an odd degree-\( \ell \) isogeny with \( \ell \in \{3, 5, 7\} \) has a cheaper cost than a scalar multiplication by \( \ell \) (see for example [9][Table 3]), one can easily show that any multiplicative-based strategy containing any of these three small degree isogenies could never be optimal.

As proposed in [12], optimal strategies can be obtained using dynamic programming (see [1, 8] for concrete algorithms). In [14][§3.1], the cost of adapting the optimal strategies of [12] to the CSIDH setting was presented.

A brief description of our process to finding optimal strategies for CSIDH is given next.

3.3.1. Finding Optimal strategies for CSIDH. Notice that the computation of SIDH strategies are a very special case of CSIDH strategies, where \( q_\ell_i \) (resp. \( p_\ell_i \)) have the same fixed value. Hence, 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 \( \Delta_h \) and \( \Delta_{n-h} \) 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 $\Delta_h$ and $\Delta_{n-h}$, the search space of different weighted sub-triangles to be considered is exceedingly large, given as, $\sum_{i=1}^{n-1} i! \cdot \left(\begin{array}{c} n \\ i \end{array}\right) \gg 2^n$. Therefore, searching for an optimal strategy and ordering of the small prime factors in $L$, quickly 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. Besides, processing the $\ell_i$ primes in the above order favors the usage of non multiplicative-based strategies. This assumption can be informally justified as follows.

Referring to Figure 1a, the isogeny maps required to move from column 0 to column 73 of the right triangle $\Delta_{74}$, will start from the smallest degree-3 isogeny $\phi_{\ell_{74}}$, all the way until the costly degree-373 isogeny $\phi_{\ell_2}$ must be computed to move from column 72 to column 73. We stress that in the case of dealing with the OAYT-style, two isogeny evaluations must be performed when moving from one column to the other.

Swapping the primes $\ell_i$ and $\ell_j$ for $i < j$, produces a small computational saving due to the fact that the scalar multiplication associated to $\ell_j$ has a cheaper cost than the one associated to $\ell_i$. However, this swap also implies evaluating earlier the isogeny $\phi_{\ell_i}$, which is more expensive than $\phi_{\ell_j}$. We note that for almost all the trail of an optimal strategy, there will be more degree-$\ell_j$ isogeny evaluations than degree-$\ell_i$ isogeny evaluations. Moreover, each one of these two isogeny evaluations are more expensive than the cost associated to scalar multiplications by $\ell_i$ and $\ell_j$. Of course, there might be some few pairs $\ell_i$ and $\ell_j$, whose swapping may lead to costly reductions. For example, this will happen when the number of degree-$\ell_j$ isogeny evaluations is smaller than the number of degree-$\ell_i$ isogeny evaluations. However, even in this scenario, the difference of costs between the two arrangements is expected to be very small.

Under the above 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

\[
\begin{align*}
    n + \sum_{j=2}^{n-1} (n + 1 - j)(j - 1) &= n + n \sum_{j=2}^{n-1} (j - 1) - \sum_{j=2}^{n-1} (j - 1)^2 \\
    &= n + n \sum_{j'=1}^{n-2} j' - \sum_{j'=1}^{n-2} (j')^2 \\
    &= n + n \left( \frac{(n - 2)(n - 1)}{2} \right) - \left( \frac{(n - 2)(n - 1)(2n - 3)}{6} \right) \\
    &= n + \frac{(n - 2)(n - 1)}{6} (3n - (2n - 3)) \\
    &= n + \frac{(n - 2)(n - 1)(n + 3)}{6}.
\end{align*}
\]

Notice also that any CSIDH strategy can be encoded 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 \(N_{k,j} := [\ell_{j+1}, \ell_{j+2}, \ldots, \ell_{j+k}] \subseteq L\). Then,

1. For each \(j := 0, 1, \ldots, n-1\), the optimal strategy for each \(N_{1,n-1-j}\) is \(S_1(N_{1,n-1-j}) = []\) and has a cost equal to \(C_1(S_1(N_{1,n-1-j})) = \nu((n-1-j,j))\).
2. For each \(k := 2, 3, \ldots, n\) and \(j := 0, 1, \ldots, n-k\), the optimal strategy is

\[
S_k(N_{k,j}) = [s] \text{ cat } S_{k-s}(N_{k-s,j+s}) \text{ cat } S_s(N_{s,j})
\]

and has a cost equal to \(C_k(S_k(N_{k,j})) = \min_h \alpha\), where \(s = \arg \min_h \alpha\), and

\[
\alpha = \{ C_{k-h}(S_{k-h}(N_{k-h,h+j})) + C_h(S_h(N_{h,j})) + \omega([(0,0),(h,0)]) + \omega([(0,0),(0,k-h)]) : h = 1, 2, \ldots, k-1 \}.
\]

Here, \(\omega([(0,0),(0,h)])\) and \(\omega([(0,0),(k-h,0)])\) 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 to the root of the sub-triangle \(\Delta_k\), associated with the sub-list of prime factors \(N_{k,j}\). See Figure 1b for an illustration of the first level of this recursive process with \(k = n\).

The remaining task is to figure out 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 on 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 [22], 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 \(T \in E(F_p)\), 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

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

In the event that the Elligator 2 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 \( \ell_i \) as defined above, into \( \sigma \) disjoint sets (batches) of size \( \frac{n}{\sigma} \). Afterwards, a multiplicative strategy is applied to each batch. Each multiplicative strategy is evaluated \( \kappa \) 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 \( 0 \leq e_i \leq m_i \), for \( i = 1, \ldots, n \). The missing prime factors are repaired using a multiplicative strategy, until all the \( m_i \) degree-\( \ell_i \) 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 \textit{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 [22]. 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 [22] performs approximately five isogeny constructions (as opposed to the ten constructions in [17]) and ten isogeny evaluations per \( \ell_i \).

Algorithm 3 of Appendix A summarizes the main idea proposed by Onuki et al. [22].

In the remaining of this paper we will refer to this approach, which uses two torsion points and dummy isogeny constructions, as the \textit{OAYT-style} CSIDH group action evaluation. We stress that OAYT-style studied in this work 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

\[
S(m_i) = \{ e \mid e = m_i \mod 2 \text{ and } |e| \leq m_i \},
\]

i.e., centered intervals containing only even or only odd integers. The action of vectors drawn from \( S(m)^n \) 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 $l_i^{\text{sign}}(e_i)$ for $e_i$ iterations, then alternates between $l_i$ and $l_i^{-1}$ for $m_i - e_i$ iterations. Algorithm 4 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-$\ell_i$ 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 2), 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 $\lambda$-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 $\lambda$-bits, but hopefully a lesser computational cost for the group action. If the modified vector has a 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 $\lambda$-bits of classical security can be summarized as follows:

0. Initial bound $(m_1, m_2, \ldots, m_n)$ that yields $\lambda$-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 $i := 1, 2, \ldots, n$:
   (a) Set $\vec{m} = (m_1, m_2, \ldots, m_n)$;
   (b) Decrease the $i$-th coordinate of $\vec{m}$ by one unit;
   (c) Compute
   $$\mu_i = \left\{ \vec{m} = \vec{m} + \Delta: \Delta \in (\mathbb{Z}_+ \cup \{0\})^n, \Delta_i = 0, \sum_{j=1}^{n} \log_2(\delta \cdot \vec{m}_j + 1) = 2\lambda \right\}$$
   (d) Select the local optimal element $\hat{m}$ of $\mu_i$ 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-$\ell_i$ isogeny construction will be performed for each small odd prime $\ell_i$ (i.e. that all the entries in the bound vector are strictly greater than 0), the above greedy method was applied iteratively $\left(\frac{m}{\ell_i} - 1\right)$ times. We heuristically found out that setting $m_n = \frac{3}{2} \cdot \frac{10}{3}$, tends to obtain better bound vectors. A Python-script implementation of the above greedy procedure found the following bounds,

$$\overline{m}_1 = (3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5, 5, 5, 5),$$

$$\overline{m}_2 = (1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3),$$

$$\overline{m}_3 = (3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5),$$

for MCR, OAYT, and dummy-free styles, respectively. Let us recall that each entry of these bound vectors corresponds with the number of degree-$(\ell_i)$ 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 $\gamma$ and $\Gamma$ 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 $p = 4 \cdot \prod_{i=1}^{\ell_i} - 1$ is a prime number. Then as discussed in 3.3, 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 $\gamma$ times. At this point with high probability all the degree-$\ell_i$ isogenies having entries $m_i = \delta$ 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 $\delta$. To proceed forward, all the entries of $m$ must be subtracted by $\delta$, disregarding the zero entries. Then, a new minimum entry $\delta'$ is computed and a new strategy $S_n(L')$ must be found and executed $\delta'$ times with $n' = \#L'$. This procedure is repeated until there are no more isogenies to be processed. In fact, after $\Gamma$ rounds, the estimated number of missing degree-$\ell_i$ 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.

\(^3\)In fact the probability of having completed all the degree-$\ell_i$ isogenies whose entries $m_i = \delta$ for $i = 1, \ldots, n$, depend on the order of the points output by the Elligator 2 procedure as discussed in §4.1.
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 \( \gamma_k \) times, where

\[
m^{(1)} = \{ m_1, m_2, \ldots, m_n \},
\]
\[
m^{(2)} = \{ m^{(1)}_1 - \gamma_1, \ldots, m^{(1)}_n - \gamma_1 \} \setminus \{0\},
\]
\[
m^{(3)} = \{ m^{(2)}_1 - \gamma_2, \ldots, m^{(2)}_n - \gamma_2 \} \setminus \{0\},
\]
\[
\vdots
\]
\[
m^{(t)} = \{ m^{(t-1)}_1 - \gamma_{t-1}, \ldots, m^{(t-1)}_n - \gamma_{t-1} \} \setminus \{0\}.
\]

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

\[
L_1 = [\ell_1, \ell_2, \ldots, \ell_n],
\]
\[
L_2 = [\ell_i \in L_1 : L^{(1)}_i > \gamma_1],
\]
\[
L_3 = [\ell_i \in L_2 : L^{(2)}_i > \gamma_2],
\]
\[
\vdots
\]
\[
L_t = [\ell_i \in L_{t-1} : L^{(t-1)}_i > \gamma_{t-1}].
\]

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

\[
m'_i := \left\lfloor m_i \cdot \frac{\ell_i}{\ell_i - 1} \right\rfloor \text{ for } i = 1, \ldots, n.
\]

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

The interested reader can see in Appendix D, the eleven optimal strategies that were found and used for performing the OAYT-style CSIDH-512 instantiation.

**Remark 5.** 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. We adopt the bound vectors presented in §4.4, and present a comparison of our results versus the SIMBA-based methods that use the exponent bounds \( m \) as reported in [7, §5.2]. Additionally, we replicated the experiments of Hutchinson et al. in [14], by using their publicly available files `addc.h`,

\[4\]The subsets of small odd primes and optimal strategies implemented in this work, can be easily extracted from our library.
Optimal strategies for CSIDH

simba_withdummy_2.h and action_simba_withdummy_2.c in combination with the software library by [7].

All of our experiments were executed 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.

5.1. A COMPARISON OF SIMBA MULTIPLICATIVE-BASED APPROACH VERSUS OPTIMAL STRATEGIES. In Table 3, we report 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:

1. SIMBA-σ-κ method with the configuration proposed in [17] and [22]. In other words, this corresponds to a Python-code version of the C-code implementation presented in [7]
2. 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 starting from the largest to the smallest.
3. The improvements presented in this work with the following bound vectors:
   (a) The ones proposed in Meyer-Campos-Reith [17] and Onuki et al. [22], and
   (b) The ones presented in section 4.4.

The last column in Table 3 gives the expected speedups for MCR- OAYT- and Dummy-free- styles using as a baseline the field arithmetic counts for multiplicative-based SIMBA-5-11 for MCR and dummy-free -styles, and SIMBA-3-8 for OAYT-style. The last three rows in Table 3 report the highest speedups. Notice that these three rows correspond with the last three rows in Table 4. Interestingly, applying optimal strategies for the SIMBA-based approach shows the same costs as a multiplicative-based SIMBA method. This similarity between these two approaches, nicely corresponds to the results reported by [14], where a SIMBA multiplicative-based strategy was found as the most economical.

A graphical view of several of these CSIDH strategies can be found in Figures 2 and 3 of Appendix C. Table 3 also showcases that the application of optimal strategies without the SIMBA approach, produces different integer vector bounds and competitive speedups compared with the work presented in [14].

5.2. EXPERIMENTAL RESULTS AND COMPARISON WITH RELATED WORK. Tables 4–5, report the field arithmetic counting and clock cycles timings obtained for three CSIDH-512 constant-time group action variants, 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 against the MCR and Dummy-free SIMBA-based implementation of [7].

5 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 2^σ. 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).

6 Notice that the cost of validating the public key was omitted from these estimates. However, as shown in the last row of Table 3, the computational cost of this task is negligible.
18 JESÚS-JAVIER CHI-DOMÍNGUEZ AND FRANCISCO RODRÍGUEZ-HENRÍQUEZ

| Algorithm      | Strategy     | Bounds: \( m \) | Group action evaluation | \( M \)   | \( S \)   | \( a \)   | Speedup (%) |
|----------------|--------------|-----------------|-------------------------|---------|---------|--------|------------|
| SIMBA-5-11     | multiplicative | as given in [17] | MCR-style               | 0.906   | 0.297   | 0.939  | —          |
|                | optimal      |                  | dummy-free              | 1.309   | 0.392   | 1.382  | 0.00       |
| SIMBA-3-8      | multiplicative | as given in [22] | OAYT-style              | 0.942   | 0.198   | 0.964  | 0.00       |
|                | optimal      |                  | dummy-free              | 1.342   | 0.392   | 1.345  | 0.00       |
| SIMBA-5-11     | Multiplicative | as given in section 4.4 | MCR-style               | 0.930   | 0.242   | 0.982  | 2.09       |
|                |              |                  | dummy-free              | 1.378   | 0.333   | 1.429  | -0.71      |
|                |              |                  | OAYT-style              | 0.960   | 0.234   | 0.987  | 0.00       |
| This work      | optimal      | as given in [17] | MCR-style               | 0.862   | 0.202   | 0.887  | 0.71       |
|                |              |                  | dummy-free              | 1.244   | 0.322   | 1.188  | 7.05       |
|                |              |                  | OAYT-style              | 0.840   | 0.172   | 0.810  | 3.10       |

**Table 3.** 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-\( \sigma-\kappa \) as a baseline, by only considering multiplication and squaring operations, and assuming \( M = S \). The last three rows in this table report the highest speedups. Adding the public key validation cost to these three rows, get the last three rows in Table 4. Public key validation was separately measured, and presented in the last row of the table.

| Implementation | Group action evaluation | \( M \)   | \( S \)   | \( a \)   | Speedup (%) |
|----------------|-------------------------|---------|---------|--------|------------|
| Cervantes-Vázquez et al. [7] | MCR-style               | 0.900   | 0.310   | 0.964  | —          |
|                | OAYT-style              | 0.658   | 0.210   | 0.691  | —          |
|                | dummy-free-style        | 1.319   | 0.423   | 1.389  | —          |
| Hutchinson et al. [14]         | OAYT-style              | 0.637   | 0.212   | 0.712  | 2.19       |
|                | dummy-free-style        | 0.862   | 0.250   | 0.866  | 7.69       |
| This work      | MCR-style               | 0.666   | 0.189   | 0.691  | 1.50       |
|                | OAYT-style              | 1.273   | 0.346   | 1.280  | 7.06       |
|                | dummy-free-style        | 1.273   | 0.346   | 1.280  | 7.06       |

**Table 4.** 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]. Only multiplication and squaring operations were considered and it was assumed that \( M = S \).

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- style implementations yield a 12.09%, 3.36% and 10.58% speedup over [7], respectively (See Table 5). Furthermore, Tables 4 and 5 show that our results are highly competitive with respect to the ones reported in [14]. We found that the field arithmetic operation and clock cycle associated to the script of [14] for the OAYT-style CSIDH-512 computation, is 0.7% cheaper and 0.43% faster than the ones corresponding to our work for this CSIDH variant, respectively.
Optimal strategies for CSIDH

Implementation | Group action evaluation | Mcycles | Speedup (%)
--- | --- | --- | ---
Cervantes-Vázquez et al. [7] | MCR-style | 339 | —
 | OAYT-style | 238 | —
 | dummy-free | 482 | —
Hutchinson et al. [14] | OAYT-style | 229 | 3.78
This work | MCR-style | 298 | 12.09
 | OAYT-style | 230 | 3.36
 | dummy-free-style | 431 | 10.58

Table 5. Clock cycle timings for constant-time CSIDH-512 group action evaluation, averaged over 1024 runs. The 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].

5.3. Expected field arithmetic costs for larger CSIDH instantiations.

Using our Python3-code implementation of the CSIDH protocol, we report in Table 6 the expected number of field operations for the OAYT-style CSIDH-1024 instantiation. It can be seen that the number of field operations is about the same as in the case of CSIDH-512. Likewise, Table 7 reports the expected cost of a group action evaluation of a OAYT-style CSIDH-1792 instantiation, where the 1790-bit prime \( p \), is determined by the product of the first 207 small odd primes \( \ell_i \)'s different from 149, equal to \( \prod_{i=1}^{207} \ell_i + 4 \). In both cases, we found suitable bounds based on the approach described in §4.4.

Notice that for the case of the OAYT-style CSIDH-1792, one requires exactly one degree-\( \ell_i \) isogeny construction for each \( \ell_i \), and thus only one optimal strategy must be applied. Moreover, by precomputing an integer \( u \in \mathbb{F}_p \) such that the Elligator 2 procedure with inputs \( u \) and Montgomery curve coefficient \( A \) returns two full torsion points, one can achieve a group action evaluation with a random-free fixed running time. This comes at the cost of doubling the public key size by adding the parameters \((A, u)\). In fact, CSIDH-1792 with OAYT-style ensures that Elligator 2 is always invoked using public parameters.

Table 6. Expected number of field operation for the constant-time CSIDH-1024 group action evaluation. Counts are given in millions of operations, averaged over 1024 random experiments. For computing the Cost column, it is assumed that \( M = S \) and all addition costs are ignored. Public key validation was separately measured, and presented in the last row of the table.
Table 7. Expected number of field operation for the constant-time CSIDH-1792 group action evaluation. Counts are given in millions of operations, averaged over 1024 random experiments. For computing the Cost column, it is assumed that \( M = S \) and all addition costs are ignored. Public key validation and full torsion point search were separately measured, and presented in the last rows of the table. The OAYT-style CSIDH group action reported in this table uses full torsion points and executes in a fixed running-time.

| Group action evaluation | M   | S   | a   | Cost |
|-------------------------|-----|-----|-----|------|
| MCR-style               | 1.040 | 0.239 | 0.910 | 1.279 |
| dummy-free              | 1.557 | 0.327 | 1.337 | 1.884 |
| OAYT-style              | 1.364 | 0.252 | 1.104 | 1.616 |
| Full torsion points search | 1.571 | 0.785 | 2.295 | 2.356 |
| Public key validation   | 0.089 | 0.044 | 0.130 | 0.133 |

6. Conclusions

The computational cost of the CSIDH group action evaluation, directly depends on the number and degree of the isogenies to be processed, which are determined by the \( n \) prime factors of \( \frac{p+1}{4} \). 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 group action 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.

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 5, our experimental results achieve performance speedups of 12.09%, 3.36% and 10.58% compared with the MCR, OAYT and dummy-free SIMBA-based implementations reported in [7]. Furthermore, Tables 4 and 5 show that our results are highly competitive with respect to the ones presented in [14].

As a future work, we would like to explore the approach presented in [21] for finding CSIDH optimized bound vectors.

Acknowledgments

This work was partially done while the second author was visiting the University of Waterloo. The second author received partial funds from the Mexican Science council CONACyT project 313572. 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, et al., Supersingular isogeny key encapsulation, Second Round Candidate of the NIST’s Post-quantum Cryptography Standardization Process, 2017, Available from: 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, 2013, 967–980.

[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 from: https://eprint.iacr.org/2020/341.

[5] W. Castryck and T. Decru, CSIDH on the surface, Post-Quantum Cryptography - 11th International Conference, LNCS, 12100, 2020, 111–129.

[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 B. Smith, Stronger and faster side-channel protections for CSIDH, Progress in Cryptology - LATINCRYPT 2019, LNCS, 11774, 2019, 173–193.

[8] D. Cervantes-Vázquez, 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 from: 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 from: https://eprint.iacr.org/2019/1373.

[10] C. Costello and H. Hisil, A simple and compact algorithm for SIDH with arbitrary degree isogenies, Advances in Cryptology - ASIACRYPT 2017 Part II, LNCS, 10625, 2017, 303–329.

[11] J.-M. Couveignes, Hard homogeneous spaces, Cryptology ePrint Archive, Report 2006/291 (2006), Available from: 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 from: 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, Topics in Cryptology - CT-RSA, LNCS, 12006, 2020, 512–537.
Appendix A. Constant-time Algorithms for computing the CSIDH Group Action

Algorithm 3 OAYT style from [22]. 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 \( \ell_i = (\ell_i, \pi - 1) \) and \( \ell_i^{-1} = (\ell_i, \pi + 1) \), where \( \pi \) maps to the \( p \)-th power Frobenius endomorphism on each curve. This algorithm computes exactly \( m \) isogenies for each ideal \( \ell_i \) (or \( \ell_i^{-1} \)).

Require: 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.

Ensure: \( E_B = \ell_1^{e_1} \cdots \ell_n^{e_n} \ast E_A \).

\[ E_0 \leftarrow E_A \]

// Outer loop: Each \( \ell_i \) prime is processed \( m \) times
for \( i \in \{1, \ldots, m\} \) do
\[ T_+, T_- \leftarrow \text{ObtainFullTorsionPoint}(E_0) \]
\[ T_+, T_- \leftarrow [4]T_+, [4]T_- \]
// Now \( T_+, T_- \in E_n[\prod \ell_i] \)
// Inner loop: processing each prime factor \( \ell_i \mod (p+1) \)
for \( j \in \{0, 1, \ldots, n-1\} \) do
\[ s \leftarrow \text{isequal}(\text{sign}(e_j), -1) \]
\[ \text{cswap}(T_+, T_-, s) \]
// swap ideals \( \ell_{n-j} \) and \( \ell_{n-j}^{-1} \)
\[ G_j \leftarrow T_+ \]
for \( k \in \{1, \ldots, n-1-j\} \) do
\[ G_j \leftarrow [k]G_j \]
end for
\[ b \leftarrow \text{isequal}(e_{n-j}, 0) \]
\( (E_{j+1} \mod n, (G_j)) \leftarrow \text{QuotientIsogeny}(E_j, G_j, \ell_{n-j}) \)
\( T'_+ \leftarrow [\ell_{n-j}^{-1}]T_+ \)
\( T'_+ \leftarrow \text{xEVAL}(T'_+, (G_j)) \)
// Evaluate \( T'_+ \) on degree-\( \ell_{n-j}^{-1} \) isogeny
\( T_- \leftarrow \text{xEVAL}(T_-, (G_j)) \)
// Evaluate \( T_- \) on degree-\( \ell_{n-j}^{-1} \) isogeny
\[ \text{cswap}(E_j, E_{j+1} \mod n, b) \]
\[ \text{cswap}(T'_+, T_+, b) \]
// undo if \( e_{n-j} = 0 \)
\[ \text{cswap}(T'_+, T_-, b) \]
// undo if \( e_{n-j} = 0 \)
\[ T_+ \leftarrow [\ell_{n-j}]T_+ \]
\[ \text{cswap}(T_+, T_-, s) \]
// swap ideals \( \ell_{n-j} \) and \( \ell_{n-j}^{-1} \)
\[ e_{n-j} \leftarrow e_{n-j} - (b + 1 \mod 2) \]
end for
end for
return \( E_0 \)
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 $I_i = (\ell_i, \pi - 1)$ and $\zeta_{-1}^{-1} = (\ell_i, \pi + 1)$, where $\pi$ maps to the $p$-th power Frobenius endomorphism on each curve. This algorithm computes exactly $m$ isogenies for each ideal $I_i$ (or $\zeta_{-1}^{-1}$).

Require: A supersingular curve $E_A$ over $\mathbb{F}_p$, and an exponent vector $(e_1, \ldots, e_n)$ with each $e_i \in S(m)$, $m$ a positive number.
Ensure: $E_B = \ell_1^{e_1} \cdots \ell_n^{e_n} * E_A$.

$E_0 \leftarrow E_A$

//Outer loop: Each $\ell_i$ prime is processed $m$ times
for $i \in \{1, \ldots, m\}$ do

$T_{\pm}, T_{-} \leftarrow \text{ObtainFullTorsionPoint}(E_0)$ // $T_{\pm} \in E_n[\pi \mp 1]$

$T_{\pm}, T_{-} \leftarrow [4]T_{\pm}, [4]T_{-}$ // Now $T_{\pm}, T_{-} \in E_n[\prod \ell_i]$

Inner loop: processing each prime factor $\ell_i | (p+1)$;

for $j \in \{0, 1, \ldots, n-1\}$ do

$s \leftarrow \text{isequal}(\text{sign}(e_j), -1)$

cswap($T_{\pm}, T_{-}, s$) // swap ideals $I_{n-j}$ and $\zeta_{-1}^{-1}$

$G_j \leftarrow T_{\pm}$

for $k \in \{1, \ldots, n-1-j\}$ do

$G_j \leftarrow \ell_k[G_j]$

end for

($E_{(j+1) \mod n}, \langle G_j \rangle$) \leftarrow \text{QuotientIsogeny}($E_j, G_j, \ell_{n-j}$)

$T_{\pm} \leftarrow x\text{EVAL}(T_{\pm}, \langle G_j \rangle)$ // Evaluate $T_{\pm}$ on degree-$\ell_{n-j}$ isogeny

$T_{-} \leftarrow x\text{EVAL}(T_{-}, \langle G_j \rangle)$ // Evaluate $T_{-}$ on degree-$\ell_{n-j}$ isogeny

$T_{-} \leftarrow [\ell_{n-j}]T_{-}$

cswap($T_{\pm}, T_{-}, s$) // swap ideals $I_{n-j}$ and $\zeta_{-1}^{-1}$

$e_{n-j} \leftarrow e_{n-j} - 1$

end for

end for

return $E_0$

Appendix 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.

B.1. MCR-style: using one torsion point and dummy isogeny constructions. 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-$(\prod_{k=1}^{n-j} \ell_k)$ 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 $\ell_i$ let us define

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

The navigation rules to walk across $\Delta_n$ are described as follows:

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

\[
T_{0,j}, T_{n-1-j,j} \leftarrow \text{cswap}(T_{0,j}, T_{n-1-j,j}, b_{n-1-j}),
\]

\[
E_{j+1}, (T_{n-1-j,j}) \leftarrow \text{QuotientIsogeny}(E_{j}, T_{n-1-j,j}, \ell_{n-j}),
\]

\[
E_{j}, E_{j+1} \leftarrow \text{cswap}(E_{j}, E_{j+1}, b_{n-1-j}), \quad \text{and}
\]

\[
T_{0,j}, T_{n-1-j,j} \leftarrow \text{cswap}(T_{0,j}, T_{n-1-j,j}, b_{n-1-j}).
\]

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

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

(a) If \(i = 0\):

\[
R_{d_{n-j}+1} \leftarrow R_{d_{n-j}} + R_1 \quad \text{and} \quad T_{i,j} \leftarrow R_{d_{n-j}} + R_{d_{n-j}+1} = \left[\ell_{n-j}\right]R_1.
\]

(b) Otherwise:

\[
T_{i,j} \leftarrow \left[\ell_{n-j}\right]T_{i,j}.
\]

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

\[
T_{i,j+1} \leftarrow \text{EvaluateIsogeny}(T_{i,j}, (T_{n-1-j,j})), \quad \text{and}
\]

\[
T_{i,j}, T_{i,j+1} \leftarrow \text{cswap}(T_{i,j}, T_{i,j+1}, b_{n-j-1}).
\]

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

B.2. OAYT-style: Using two torsion points and dummy isogeny constructions. The vertices of \(S_n(L)\) are labeled by an integer pair \((i, j)\) where \(0 \leq j < n\) and \(0 \leq i < (n-j)\). The vertex \((i, j)\) determines a pair of torsion-\((\prod_{k=1}^{\ell_i} \ell_k)\) points \(T_{+,i,j} \in E_j[\pi - 1]\) and \(T_{-,i,j} \in E_j[\pi + 1]\). 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 \((k, j)\) determine rational elliptic curve points on the same curve \(E_j\). Now, for each \(\ell_i\) 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 \(\Delta_n\) are described as follows:

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

\[
T_{+,0,j}, T_{-,0,j} \leftarrow \text{cswap}(T_{+,0,j}, T_{-,0,j}, s_{n-1-j}),
\]
\[
T_{+,n-1-j,j}, T_{-,n-1-j,j} \leftarrow \text{cswap}(T_{+,n-1-j,j}, T_{-,n-1-j,j}, s_{n-1-j}),
\]
\[
T_{+,0,j}, T_{n-1-j,j} \leftarrow \text{cswap}(T_{+,0,j}, T_{+,n-1-j,j}, b_{n-1-j}),
\]
\[
E_{j+1}, (T_{+,n-1-j,j}) \leftarrow \text{QuotientIsogeny}(E_j, T_{+,n-1-j,j}),
\]
\[
E_j, E_{j+1} \leftarrow \text{cswap}(E_j, E_{j+1}, b_{n-1-j}),
\]
\[
T_{+,0,j}, T_{n-1-j,j} \leftarrow \text{cswap}(T_{+,0,j}, T_{+,n-1-j,j}, b_{n-1-j}),
\]
\[
T_{+,n-1-j,j}, T_{-,n-1-j,j} \leftarrow \text{cswap}(T_{+,n-1-j,j}, T_{-,n-1-j,j}, s_{n-1-j}),
\]
\[
T_{+,0,j}, T_{-,0,j} \leftarrow \text{cswap}(T_{+,0,j}, T_{-,0,j}, s_{n-1-j}).
\]

Here, \(\text{QuotientIsogeny}(E_j, T_{+,n-1-j,j})\) is performed by assuming \(T_{+,n-1-j,j}\) has order-\(\ell\), and its second output, \(\langle T_{+,n-1-j,j} \rangle\), 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 \(\ell_{n-j}\). To be more precise, the edge \([(i, j), (i, j + 1)]\) means that the following computations have been performed:

\[
T_{+,i,j}, T_{-,i,j} \leftarrow \text{cswap}(T_{+,i,j}, T_{-,i,j}, s_{n-1-j}),
\]

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

\[
T_{-,i,j} \leftarrow [\ell_{n-j}]T_{-,i,j},
\]
\[
T_{+,i,j+1} \leftarrow \text{EvaluateIsogeny}(T_{+,i,j}, \langle T_{+,n-1-j,j} \rangle),
\]
\[
T_{-,i,j+1} \leftarrow \text{EvaluateIsogeny}(T_{-,i,j}, \langle T_{+,n-1-j,j} \rangle),
\]
\[
T_{+,i,j}, T_{+,i,j+1} \leftarrow \text{cswap}(T_{+,i,j}, T_{+,i,j+1}, b_{n-1-j}),
\]
\[
T_{-,i,j}, T_{-,i,j+1} \leftarrow \text{cswap}(T_{-,i,j}, T_{-,i,j+1}, b_{n-1-j}),
\]
\[
T_{+,i,j+1}, T_{-,i,j+1} \leftarrow \text{cswap}(T_{+,i,j+1}, T_{-,i,j+1}, s_{n-1-j}).
\]

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

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

\[
T_{+,i,j}, T_{-,i,j} \leftarrow \text{cswap}(T_{+,i,j}, T_{-,i,j}, s_{n-1-j}),
\]
\[
T_{+,i+1,j} \leftarrow [\ell_i]T_{+,i,j},
\]
\[
T_{+,i+1,j}, T_{-,i+1,j} \leftarrow \text{cswap}(T_{+,i+1,j}, T_{-,i+1,j}, s_{n-1-j}), \text{ and}
\]
\[
T_{+,i,j}, T_{-,i,j} \leftarrow \text{cswap}(T_{+,i,j}, T_{-,i,j}, s_{n-1-j}).
\]

(b) Otherwise:
\[
T_{+,i+1,j} \leftarrow [\ell_i]T_{+,i,j}, \quad \text{and} \\
T_{-,i+1,j} \leftarrow [\ell_i]T_{-,i,j}.
\]

B.3. **Dummy-free style: using two torsion point without dummy isogeny constructions.** The vertices of \(S_n(L)\) are labeled by an integer pair \((i, j)\) where \(0 \leq j < n\) and \(0 \leq i < (n - j)\). The vertex \((i, j)\) determines a pair of torsion-

\[
(\prod_{k=1}^{n-j} \ell_k)
\]

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 \(\ell_i\), 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 \(\Delta_n\) are described as follows:

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

\[
T_{+,n-1-j,j}, T_{-,n-1-j,j} \leftarrow \text{cswap}(T_{+,n-1-j,j}, T_{-,n-1-j,j}, s_{n-1-j}), \\
E_{j+1}, (T_{+,n-1-j,j}) \leftarrow \text{QuotientIsogeny}(E_j, T_{+,n-1-j,j}), \\
E_j, E_{j+1} \leftarrow \text{cswap}(E_j, E_{j+1}, b_{n-1-j}), \quad \text{and} \\
T_{+,n-1-j,j}, T_{-,n-1-j,j} \leftarrow \text{cswap}(T_{+,n-1-j,j}, T_{-,n-1-j,j}, s_{n-1-j}).
\]

Here, \(\text{QuotientIsogeny}(E_j, T_{+,n-1-j,j})\) is performed by assuming \(T_{+,n-1-j,j}\) has order-\((\ell_{n-j})\), and its second output, \((T_{+,n-1-j,j})\), 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 \(\ell_{n-j}\). To be more precise, the edge \([(i, j), (i, j + 1)]\) means that the following computations and constant-time swaps have been performed:

\[
T_{+,i,j}, T_{-,i,j} \leftarrow \text{cswap}(T_{+,i,j}, T_{-,i,j}, s_{n-1-j}), \\
T_{-,i,j} \leftarrow [\ell_{n-j}]T_{-,i,j}, \\
T_{+,i,j+1} \leftarrow \text{EvaluateIsogeny}(T_{+,i,j}, \langle T_{+,n-1-j,j} \rangle), \\
T_{-,i,j+1} \leftarrow \text{EvaluateIsogeny}(T_{-,i,j}, \langle T_{+,n-1-j,j} \rangle), \quad \text{and} \\
T_{+,i,j}, T_{-,i,j} \leftarrow \text{cswap}(T_{+,i,j}, T_{-,i,j}, s_{n-1-j}).
\]

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

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

\[
T_{+,i,j}, T_{-,i,j} \leftarrow cswap(T_{+,i,j}, T_{-,i,j}, s_{n-1-j}),
\]
\[
T_{+,i+1,j} \leftarrow [\ell_i]T_{+,i,j},
\]
\[
T_{+,i+1,j}, T_{-,i+1,j} \leftarrow cswap(T_{+,i+1,j}, T_{-,i+1,j}, s_{n-1-j}), \text{ and}
\]
\[
T_{+,i,j}, T_{-,i,j} \leftarrow cswap(T_{+,i,j}, T_{-,i,j}, s_{n-1-j}).
\]

(b) Otherwise:

\[
T_{+,i+1,j} \leftarrow [\ell_i]T_{+,i,j}, \text{ and}
\]
\[
T_{-,i+1,j} \leftarrow [\ell_i]T_{-,i,j}.
\]

Appendix C. A graphical view of CSIDH strategies

---

**Figure 2.** Two variants of the CSIDH group action evaluation: MCR style as proposed in [17] and OAYT style as proposed in [22]. Each one of the two approaches depicted in this figure, computes a group action using the SIMBA-\(\sigma\)-\(\kappa\) method, constructing isogenies of prime degree grouped in \(\sigma\) batches. Each round must be repeated \(\kappa\) times. A final repair round applies a multiplicative strategy to process the prime factors not covered during the \(\kappa\) rounds. Horizontal edges and vertical edges represent isogeny evaluations \(q_{\ell_i}\), and scalar multiplications \(p_{\ell_i}\), respectively.

---

Appendix D. Strategies à la SIDH adapted to OAYT-style CSIDH 512

Here, we present eleven optimal strategies that were found for performing the OAYT-style CSIDH 512 instantiation. We first found an optimal strategy for all the seventy four prime factors of \(p+1\). This strategy completed the required number of group actions for just one of those seventy-four prime factors. Thereafter, we found a second strategy for the remaining seventy-three prime factors. In the third
iteration, a strategy for sixty-three remaining prime factors was found. And so on so forth, until all the last remaining twenty-one prime factors were processed by the eleventh strategy.

1. Strategy à la SIDH for 74 primes:
   \[ [43, 16, 7, 4, 3, 2, 1, 3, 2, 1, 6, 5, 4, 3, 2, 1, 10, 5, 4, 3, 2, 1, 9, 8, 7, 6, 5, 4, 3, 2, 1, 31, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 22, 8, 7, 6, 5, 4, 3, 2, 1, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1] ; \]
2. Strategy à la SIDH for 73 primes:
   [42, 16, 7, 4, 3, 2, 1, 3, 2, 1, 6, 5, 4, 3, 2, 1, 10, 5, 4, 3, 2, 1, 9, 8, 7, 6, 5, 4, 3, 2, 1, 30, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 21, 8, 7, 6, 5, 4, 3, 2, 1, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1];

3. Strategy à la SIDH for 63 primes:
   [38, 11, 7, 4, 2, 1, 3, 2, 1, 6, 5, 4, 3, 2, 1, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 27, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1];

4. Strategy à la SIDH for 45 primes:
   [24, 10, 6, 4, 3, 2, 1, 5, 4, 3, 2, 1, 9, 8, 7, 6, 5, 4, 3, 2, 1, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1];

5. Strategy à la SIDH for 37 primes:
   [20, 8, 5, 3, 2, 1, 4, 3, 2, 1, 7, 6, 5, 4, 3, 2, 1, 13, 6, 5, 4, 3, 2, 1, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1];

6. Strategy à la SIDH for 33 primes:
   [18, 7, 4, 3, 2, 1, 3, 2, 1, 6, 5, 4, 3, 2, 1, 11, 6, 5, 4, 3, 2, 1, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1];

7. Strategy à la SIDH for 31 primes:
   [16, 7, 4, 3, 2, 1, 3, 2, 1, 6, 5, 4, 3, 2, 1, 10, 5, 4, 3, 2, 1, 9, 8, 7, 6, 5, 4, 3, 2, 1];

8. Strategy à la SIDH for 27 primes:
   [14, 6, 4, 2, 1, 3, 2, 1, 5, 4, 3, 2, 1, 9, 4, 3, 2, 1, 8, 7, 6, 5, 4, 3, 2, 1];

9. Strategy à la SIDH for 25 primes:
   [ 1, 1, 6, 4, 2, 1, 3, 2, 1, 5, 4, 3, 2, 1, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1];

10. Strategy à la SIDH for 22 primes:
    [10, 6, 3, 2, 1, 2, 1, 5, 4, 3, 2, 1, 9, 8, 7, 6, 5, 4, 3, 2, 1];

11. Strategy à la SIDH for 21 primes:
    [10, 5, 3, 2, 1, 2, 1, 4, 3, 2, 1, 9, 8, 7, 6, 5, 4, 3, 2, 1];

Received May 2020; revised August 2020.

E-mail address: jesus.chidomiguez@tuni.fi
E-mail address: francisco@cs.cinvestav.mx