Further Optimizations of CSIDH: A Systematic Approach to Efficient Strategies, Permutations, and Bound Vectors

Aaron Hutchinson[1], Jason LeGrow[1], Brian Koziel[2], and Reza Azarderakhsh[2]

[1] University of Waterloo, {a5hutchinson,jlegrow}@uwaterloo.ca
[2] Florida Atlantic University, {bkoziel2017, razarderakhsh}@fau.edu

Abstract. CSIDH, presented at Asiacrypt 2018, is a post-quantum key establishment protocol based on constructing isogenies between supersingular elliptic curves. Several recent works give constant-time implementations of CSIDH along with some optimizations of the ideal-class group action evaluation algorithm, including the SIMBA technique of Meyer, Campos, and Reith and the two-point method of Onuki, Aikawa, Yamazaki, and Takagi. A recent work of Cervantes-Vázquez, Chenu, Chi-Domínguez, De Feo, Rodríguez-Henríquez, and Smith details a number of improvements to the works of Meyer et al. and Onuki et al. Several of these optimizations—in particular, the choice of ordering of the primes, the choice of SIMBA partition and strategies, and the choice of bound vector which defines the secret keyspace—have been made in an *ad hoc* fashion, and so while they yield performance improvements it has not been clear whether these choices could be improved upon, or how to do so. In this work we present a framework for improving these optimizations using (respectively) linear programming, dynamic programming, and convex programming techniques. Our framework is applicable to any CSIDH security level, to all currently-proposed paradigms for computing the class group action, and to any choice of model for the underlying curves. Using our framework—along with another new optimization technique—we find improved parameter sets for the two major methods of computing the group action: in the case of the implementation of Meyer et al. we obtain a 16.85% speedup *without* applying the further optimizations proposed by Cervantes-Vázquez et al., while for that of Cervantes-Vázquez et al. under the two-point method we obtain a speedup of 5.08%, giving the fastest constant-time implementation of CSIDH to date.

1 Introduction

Isogenies between elliptic curves have gained increasing attention in the cryptographic world over the last several years. It is widely believed that the problem of constructing an isogeny between two given elliptic curves is hard, even with the power of quantum computing, and so it is natural to base cryptographic protocols around this problem. The use of isogenies in cryptography was initially

proposed by Couveignes in [5], and was independently rediscovered by Stolbunov and Rostovtsev in [15]. Perhaps the most well-known algorithm in isogeny-based cryptography is SIKE, one of the submissions to the National Institute for Standards and Technology's Post-Quantum Standardization process which is based on the Supersingular Isogeny Diffie-Hellman algorithm [6].

In 2018, Castryck, Lange, Martindale, Panny, and Renes proposed a similar key exchange algorithm titled Commutative Supersingular Isogeny Diffie-Hellman (CSIDH) in [1]. CSIDH uses the action of the ideal-class group on the set of isomorphism classes of supersingular elliptic curves defined over $\mathbb{F}_p$ to produce a key exchange algorithm reminiscent of the Diffie-Hellman method. Specifically, fix a prime of the form $p = 4\ell_1 \cdots \ell_n - 1$, where the ℓ_i are distinct small odd primes; in practice $\ell_1, \ldots, \ell_{n-1}$ are the first $n-1$ odd primes, and ℓ_n is chosen small while ensuring p is prime. Let $\mathcal{O}$ denote the $\mathbb{F}_p$-endomorphism ring of the supersingular Montgomery curve

$$E_0 : y^2 = x^3 + x$$

defined over $\mathbb{F}_p$. Then $\mathcal{O}$ has the property that each of the principal ideals $\ell_i\mathcal{O}$ split into the product of $\mathfrak{l}_i = (\ell_i, \pi - 1)$ and $\bar{\mathfrak{l}}_i = (\ell_i, \pi + 1)$, where π is the Frobenius endomorphism of E_0. Since $\ell_i\mathcal{O}$ is principal the elements of the ideal-class group represented by these ideals are inverses, and so $[\mathfrak{l}_i]^{-1} = [\bar{\mathfrak{l}}_i]$ in the ideal-class group.

To begin the key exchange protocol, Alice and Bob both select private keys of the form $(e_1^A, \ldots, e_n^A)$ and $(e_1^B, \ldots, e_n^B)$, respectively, where each e_i^A and e_i^B is an integer chosen from some fixed interval $[-b, b]$. Alice uses her key to compute a curve E_A, defined as applying the action of the ideal $[\mathfrak{l}_1]^{e_1^A} \cdots [\mathfrak{l}_n]^{e_n^A}$ on the initial curve E_0; Bob proceeds analogously, using his own key to compute a curve E_B:

$$E_A := [\mathfrak{l}_1]^{e_1^A} \cdots [\mathfrak{l}_n]^{e_n^A} * E_0, \qquad E_B := [\mathfrak{l}_1]^{e_1^B} \cdots [\mathfrak{l}_n]^{e_n^B} * E_0, \tag{1}$$

where $*$ denotes the ideal-class group action. Alice then sends E_A to Bob and Bob sends E_B to Alice. Each party then computes the action of the ideal corresponding to their own private key on the curve they received from the other person, in which Alice computes a curve E_{BA} and Bob computes a curve E_{AB}:

$$E_{BA} := [\mathfrak{l}_1]^{e_1^A} \cdots [\mathfrak{l}_n]^{e_n^A} * E_B, \qquad E_{AB} := [\mathfrak{l}_1]^{e_1^B} \cdots [\mathfrak{l}_n]^{e_n^B} * E_A. \tag{2}$$

The two curves E_{BA} and E_{AB} are isomorphic since they both correspond the the action of $[\mathfrak{l}_1]^{e_1^A + e_1^B} \cdots [\mathfrak{l}_n]^{e_n^A + e_n^B}$ on the initial curve E_0 by the commutativity of the ideal-class group. The shared key is then the j-invariant of the curve $E_{BA} \cong E_{AB}$.

The original method proposed in [1] for carrying out the actions in (1) and (2) is to first choose a random point $P \in E[\pi \pm 1]$, where E is the current curve and π denotes the Frobenius endomorphism. The point P will have some order $|P| = \ell_1^{c_1} \cdots \ell_n^{c_n}$, where $c_i \in \{0, 1\}$. The curve $\prod_{c_i=1} [\mathfrak{l}_i]^{c_i} * E_A$ can be computed by iteratively multiplying out all but one prime from P to yield a point Q,

constructing the isogeny $\varphi : E \to E/\langle Q \rangle$ via Vélu's formulas, and updating $P \leftarrow \varphi(P)$ and $E \leftarrow E/\langle Q \rangle$. One then repeats this procedure with a fresh point P, skipping any primes ℓ_i for which the action of the target ideal $[\mathfrak{l}_i]^{e_i}$ has been completed. Since the work of [1], there has much focus on making the evaluation of the group action more efficient.

1.1 Previous CSIDH Optimizations

CSIDH is a very new construction, but there have already been many contributions toward optimizing it. We focus here on works which optimize the overall structure of the group action evaluation itself, and put less emphasis on methods which improve curve arithmetic, isogeny computation, etc.

Meyer and Reith gave the first optimization [10] in 2018. After choosing a random point P the user has the freedom to choose the order in which the action of the $[\mathfrak{l}_i]$ are computing by selecting which primes ℓ_i to multiply out of $|P|$ first. The authors of [10] noticed that computing the action in descending order of primes results in a speedup over using an ascending order. They make other notable computational contributions as well, such as projectivizing the curve coefficients and deriving formulas for the codomain curves using twisted Edwards curves. See [10] for full details.

Meyer, Campos, and Reith gave a second optimization [9] in late 2018. First, they proposed to change the keyspace interval $[-b, b]$ so that each private key value e_i is selected from its own interval $[0, b_i]$ and the target security level is still achieved. Each private key value having the same sign is desirable since ideals $[\mathfrak{l}_i]$ and $[\mathfrak{l}_j]^{-1}$ cannot be computed using the same initial point P, i.e., once the field of definition of P is determined only the ideals of the corresponding sign can be considered. Furthermore the values b_i can be selected to achieve a speedup, and the authors use heuristics to find well-performing values for these parameters. Additionally the authors propose to use 'dummy' isogenies so that the same number of isogenies are always constructed, independent of the private key used. Specifically, e_i many 'real' isogenies and $b_i - e_i$ many dummy isogenies would be constructed, where the dummy computations would construct an isogeny but not update the points and curve coefficients to their new values. In essence, the isogenies are constructed but not used on dummy iterations. To our knowledge this was the first constant-time implementation of CSIDH.

One of the most notable contributions that Meyer, Campos, and Reith make in [9] is SIMBA (Splitting Isogenies into Multiple Batches). The SIMBA technique partitions the primes $\{\ell_1, \ldots, \ell_n\}$ into disjoint sets and evaluates the required group action on each smaller subset individually. See Section 2.4 for more details on the SIMBA technique. The authors of [9] use a simple method for determining the partition, but one might also ask how an optimal partition can be found.

A third optimization and constant-time version of CSIDH was performed by Onuki, Aikawa, Yamazaki, and Takagi in [12]. Here the authors retain signed key values e_i chosen from some interval $[-b_i, b_i]$. They track two randomly chosen points $P^+ \in E[\pi - 1]$ and $P^- \in E[\pi + 1]$ through the algorithm. For each prime

ℓ_i, the appropriate point is used to derive a kernel generator according to the sign of e_i by multiplying out all other primes as before. Both P^+ and P^- are then mapped through the isogeny to the next curve, and the point not used to derive the kernel generator is multiplied by ℓ_i. This allows both the $[\mathfrak{l}_i]$ and $[\mathfrak{l}_i]^{-1}$ to be considered on each iteration instead of being limited to only one.

There have been a few other improvements to CSIDH which optimize lower level aspects of the algorithm, and we only briefly note them here. In [11] the authors describe how to perform the CSIDH algorithm using Edwards curves instead of Montgomery curves, giving an algorithm comparable in operation cost. The authors of [8] implement CSIDH in embedded devices while optimizing the field arithmetic and group operations. In [2], XZ-coordinates are used on twisted Edwards curves with optimized addition chains for scalar multiplications, and two flaws in the constant-time implementations of [9] and [12] are repaired resulting in a speedup. The implementation of [2] is the fastest to date.

1.2 CSIDH Group Action Algorithm

In this section we look at the ideal-class group action evaluation algorithm performed in CSIDH as originally described in [1]. This algorithm takes input integers $(e_1, \ldots, e_n)$ and Montgomery curve coefficient $A \in \mathbb{F}_p$ and outputs the coefficient of the curve $[\mathfrak{l}_1]^{e_1} \cdots [\mathfrak{l}_n]^{e_n} * E_A$. The evaluation is given in Algorithm 1 as its written in [1].

Algorithm 1: CSIDH Group Action Evaluation

Input : $A \in \mathbb{F}_p$ and a list of integer $(e_1, \ldots, e_m)$.
Output: B such that $[\mathfrak{l}_1^{e_1} \cdots \mathfrak{l}_m^{e_m}]E_A = E_B$ (where $E_B : y^2 = x^3 + Bx^2 + x$).

1 **while** *some* $e_i \neq 0$ **do**
2 Sample a random $x \in \mathbb{F}_p$.
3 Set $s \leftarrow +1$ if $x^3 + Ax^2 + x$ is a square in $\mathbb{F}_p$, else $s \leftarrow -1$.
4 Let $I = \{i | e_i \neq 0, \text{sign}(e_i) = s\}$. **If** $I = \emptyset$, **then** start over with a new x.
5 Let $t \leftarrow \prod_{i \in I} \ell_i$ and compute $Q \leftarrow [(p+1)/t]P$.
6 **for** *each* $i \in I$ **do**
7 Compute $R \leftarrow [t/\ell_i]Q$. **If** $R = \infty$, **then** skip this i.
8 Compute an isogeny $\varphi : E_A \to E_B : y^2 = x^3 + Bx^2 + x$ with $\ker \varphi = \langle R \rangle$.
9 Set $A \leftarrow B, Q \leftarrow \varphi(Q), t \leftarrow t/\ell_i$, and finally $e_i \leftarrow e_i - s$.
10 **end**
11 **end**
12 **Return** A

A given iteration of the loop on line (**6**) of Algorithm 1 would use a point Q to compute $[u]Q$ for some integer u, and then build an isogeny φ using $[u]Q$ as the generator for $\ker \varphi$. The following iteration will compute $[u/\ell_i]\varphi(Q)$ from $\varphi(Q)$. Writing u/ℓ_i as v, the effect from these two iterations is to compute $[v\ell_i]Q$

and $[v]\varphi(Q)$ given only the point Q. The algorithm as written accomplishes this by evaluating $[v\ell_i]$, evaluating φ, and finally evaluating $[v]$. If the integer v is large (as is often the case), this method potentially requires more effort than, say, computing $[v]Q$, then $[\ell_i][v]Q$, then $\varphi([v]Q)$.

A similar observation holds on a larger scale. For simplicity suppose line (**4**) of Algorithm 1 computes $I = \{1, \ldots, n\}$. The overall goal of the entire loop on line (**6**) is to use the initial point Q defined on line (**5**) to successively compute the points

$$\begin{array}{cc} (1.) & [\ell_1 \cdots \ell_{n-1}]Q \\ (2.) & [\ell_1 \cdots \ell_{n-2}]\varphi_1(Q) \\ (3.) & [\ell_1 \cdots \ell_{n-3}]\varphi_2\varphi_1(Q) \\ \vdots & \vdots \\ (n-1.) & [\ell_1]\varphi_{n-2} \cdots \varphi_1(Q) \\ (n.) & \varphi_{n-1}\varphi_{n-2} \cdots \varphi_1(Q) \end{array}$$

while also constructing the isogenies φ_i as needed. These n points can be computed from Q in a wide variety of different ways, and is entirely reminiscent of the problem of efficiently constructing an isogeny of degree ℓ^n detailed by De Feo, Jao, and Plût in [6]. In fact, if one takes all primes ℓ_i above to be some common prime ℓ, the problem of efficiently computing the n points defined above reduces to exactly the same problem solved in [6], which makes use of "optimal strategies".

We point out that the user has the freedom to iterate through the set I in any fashion desired due to the ideal-class group being abelian. If a different order of iteration is chosen, the corresponding points (as well as the curves themselves) computed by the algorithm will differ since the sequence of points $\{[\ell_1 \cdots \ell_{i-1}]\varphi_{n-i} \cdots \varphi_1(Q)\}$ depends on the ordering. Changing the ordering changes the computations involved, and so the computations for some orderings may require less effort than others. As far as we are aware, all previous implementations of CSIDH at the time of this writing use heuristics to select a well performing permutation of the primes ℓ_i, and a systematic method of determining an efficient permutation remains a relatively untouched problem.

1.3 Contributions and Paper Organization

The contributions of this work can be summarized as follows.

- We detail a general framework for analyzing and optimizing the CSIDH group-action evaluation algorithm. This framework is general enough to apply to any CSIDH parameter set and can be tailored to further optimize any other CSIDH implementation to date, such as those of [1, 9, 10, 12, 2]. Specifically, we use our framework to optimize three parameters used in any CSIDH instantiation:
  - We generalize the concept of the *measure* of a *strategy*, originally defined in [6]. Any strategy on n leaves provides a method for carrying out

the CSIDH algorithm. We analyze these strategies and are able to find globally optimal strategies when fixing the permutation parameter. A dynamic programming approach similar to that of [6] will easily find these optimal strategies for practical CSIDH parameters.
  - We frame the problem of finding an optimal permutation of the primes ℓ_i—for a fixed strategy—as a linear program; that is, an optimization problem in which the objective function and constraints are affine functions of the variables corresponding to the permutation. This allows us to use linear programming techniques (*e.g.*, the simplex method) to find a corresponding optimal permutation. This technique extends in a straightforward fashion to SIMBA, and can be used to find not only an optimal permutation of primes for each batch, but also an optimal distribution of primes to the SIMBA substrategies of a fixed SIMBA strategy.
  - We derive a mathematical program to produce a bound vector which approximately optimizes the running time for the class group action evaluation algorithms used in CSIDH. We approximate the solution to this program by relaxing to a convex program and applying an iterative rounding technique.
  - We further generalize the SIMBA technique of [9] to allow for different SIMBA strategies on each round of the algorithm, and eliminate each prime ℓ_i from all strategies after the b_i^{th} round.
- We used our optimization techniques to find parameter sets consisting of efficient SIMBA strategies, permutations, and bound vectors for two previous constant-time implementations of CSIDH-512: that of Meyer et al. in [9], and Cervantes-Vázquez et al. in [2]. Our optimized implementations achieve a speedup of 16.85% over the original code of [9] (without the optimizations proposed by [2]), and a speedup of 5.08% over the original code of [2] using the two-point method. To the best of our knowledge this gives the fastest constant-time implementation of CSIDH to date.

This paper is organized as follows. Section 2 details the framework which we use to optimize CSIDH, and discusses strategies, measures, permutations, the two-point method of [12], the SIMBA technique, and a general algorithm for evaluating the CSIDH group action under any strategy. Section 3 develops theoretical methods for finding efficient parameters for computing the ideal-class group action for CSIDH, including strategies, permutations, and bound vectors. The paper concludes in Section 4 which reports the results of our implementation of the best parameter sets we found.

2 Preliminaries

2.1 General Framework for Optimization

In this section we define the general framework that we use to approach the problem of optimizing CSIDH.

Strategies The idea of *strategy* has been explored in [6], but we use an alternative definition to better suit our needs and to hopefully simplify the exposition. For a positive integer n we let $T_n = (V, E)$ be the directed graph defined as follows. The vertices V of T_n are all points in the plane with integer coordinates which lie inside or on the boundary of the region bounded by the lines $x = 0, y = 0$, and $y = -x + n - 1$. The edges E of T_n consist of all line segments of unit length which connect two vertices in V. It follows that every edge is either horizontal or vertical. We turn T_n into a directed graph by orienting all horizontal edges to the right and all vertical edges upward.

Definition 1. *A **strategy** (in T_n) is a subgraph of T_n with the following properties:*

1. *The vertex $(0, 0)$ and all vertices on the line $y = -x + n - 1$ are vertices in S,*
2. *For each vertex v on the line $y = -x + n - 1$, there is a (not necessarily unique) path from $(0, 0)$ to v in S.*

We write $|S| = n$ to mean S is a strategy in T_n.

In order to define our version of canonical strategy, we define a binary operator $\#$ called *join* on the set of all strategies. For strategies S_1 and S_2, with $|S_1| = n_1$ and $|S_2| = n_2$, we define $S_1 \# S_2$ to be the strategy in $T_{n_1+n_2}$ constructed as follows:

1. $S_1 \# S_2$ contains the (unique) path connecting $(0, 0)$ to $(n_2, 0)$,
2. $S_1 \# S_2$ contains the (unique) path connecting $(0, 0)$ to $(0, n_1)$,
3. $S_1 \# S_2$ contains S_1 as a subgraph, shifted to the right n_2 units,
4. $S_1 \# S_2$ contains S_2 as a subgraph, shifted up n_1 units.

The join operator is both nonassociative and noncommutative. We say a strategy S in T_n is *canonical* if S can be expressed as $n-1$ many applications of the join operator on the strategy T_1; i.e., S is some parenthesization of

$$\underbrace{T_1 \# T_1 \# \cdots \# T_1}_{n}.$$

Each canonical strategy has a unique such expression, and so it follows that the number of canonical strategies in T_n is the number of parenthesizations of a binary operator on n terms. This is exactly the n^{th} Catalan number. An easy induction shows that every vertex in a canonical strategy has indegree at most 1 and outdegree at most 2, and a vertex has outdegree 0 precisely when it lies on the line $y = -x + n - 1$. This allows one to associate a binary tree structure to each canonical strategy S, and we therefore say that $(0, 0)$ is the *root* of S, and the vertices on the line $y = -x + n - 1$ are the *leaves* of S.

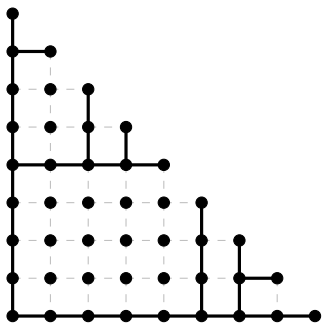
Suppose we merge together all but the outermost join operation to write a canonical strategy as $S = S_1 \# S_2$ for some canonical strategies S_1 and S_2; we define $S^L := S_1$ to be the *left substrategy* of S, and $S^R := S_2$ to be the *right*

substrategy of S. We emphasize that visually S^L lies to the right of the origin, and S^R lies above the origin. By definition of $\#$, we always have $|S_1 \# S_2| = |S_1| + |S_2|$.

The n^{th} *multiplication-based* strategy MB_n is defined recursively as $MB_1 = T_1$ and $MB_n = T_1 \# MB_{n-1}$. The n^{th} *isogeny-based* strategy IB_n is defined recursively as $IB_1 = T_1$ and $IB_n = IB_{n-1} \# T_1$. As far as we are aware, every implementation of CSIDH uses (various sizes of) a multiplication-based strategy to perform the ideal-class group action evaluation.

Our definition of strategy is entirely equivalent to that of a *full strategy* as defined in [6], and our canonical strategies are equivalent to those of [6]; we simply view the problem on a rectangular lattice as opposed to an equilateral triangular lattice, and the root of our strategies always correspond to the origin. We found this formulation very easy to work with, as emphasized in the following section.

Encoding Strategies It will be convenient, in both our analysis and for algorithmic purposes, to have a systematic method of writing down the edges which are present in a given strategy S. To do this we use two $\{0,1\}$-valued $(n-1) \times (n-1)$ sized matrices $H(S)$ and $V(S)$ (or simply H and V when S is clear), which respectively encode the horizontal and vertical edges of S. Specifically, $H_{ij} = 1$ if and only if the line segment connecting $(j-1, n-1-i)$ to $(j, n-1-i)$ is present in the strategy S, and $H_{ij} = 0$ otherwise. Similarly $V_{ij} = 1$ if and only if the line segment connecting $(j-1, n-i)$ to $(j-1, n-i+1)$ is present in S, and $V_{ij} = 0$ otherwise. Both H and V are lower triangular matrices since T_n is bounded by the line $y = -x + n - 1$. $H(T_n)$ and $V(T_n)$ are both lower triangular matrices in which every entry on and below the main diagonal is a 1. See Figure 1 for an example of a canonical strategy in T_9 together with its encoding matrices.



$$H = \begin{bmatrix} 1&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0 \\ 1&1&1&1&0&0&0&0 \\ 0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&1&0 \\ 1&1&1&1&1&1&1&1 \end{bmatrix}, V = \begin{bmatrix} 1&0&0&0&0&0&0&0 \\ 1&0&0&0&0&0&0&0 \\ 1&0&1&0&0&0&0&0 \\ 1&0&1&1&0&0&0&0 \\ 1&0&0&0&0&0&0&0 \\ 1&0&0&0&0&1&0&0 \\ 1&0&0&0&0&1&1&0 \\ 1&0&0&0&0&1&1&0 \end{bmatrix}$$

Fig. 1. A canonical strategy S (black lines) in T_9 (black and dashed gray lines), and the corresponding matrices $H(S)$ and $V(S)$. This strategy decomposes into copies of T_1 as $S = ((T_1 \#(T_1 \# T_1)) \# T_1) \#(((T_1 \# T_1) \# T_1) \#(T_1 \# T_1))$.

When restricting to canonical strategies, H and V can alternatively be defined recursively. We define $H(T_1\#T_1) = V(T_1\#T_1) = [1]$ as an initial value, and compute the matrices of larger strategies as shown in Figure 2. These recursive definitions follow immediately from the definition of the join operator # given in Section 2.1. Note that in this case either one of the matrices H or V uniquely determines the other, and so S can be specified by giving only one of them. For computational purposes (in Sections 3.2 and 3.3 in particular) having both matrices is convenient, however.

$$H(S^L\#T_1) = \left[\begin{array}{c|c} 0 & \mathbf{0} \\ \hline \mathbf{e}_{n_1} & H(S^L) \end{array}\right] \qquad V(S^L\#T_1) = \left[\begin{array}{c|c} 1 & \mathbf{0} \\ \hline \mathbf{1} & V(S^L) \end{array}\right]$$

$$H(T_1\#S^R) = \left[\begin{array}{c|c} H(S^R) & \mathbf{0} \\ \hline \mathbf{1} & 1 \end{array}\right] \qquad V(T_1\#S^R) = \left[\begin{array}{c|c} V(S^R) & \mathbf{0} \\ \hline \mathbf{e}_1^T & 0 \end{array}\right]$$

$$H(S^L\#S^R) = \left[\begin{array}{c|c|c} H(S^R) & \mathbf{0} & \mathbf{0} \\ \hline \mathbf{0} & 0 & \mathbf{0} \\ \hline \mathbf{e}_{n_1-1}\mathbf{1}^T & \mathbf{e}_{n_2-1} & H(S^L) \end{array}\right] \quad V(S^L\#S^R) = \left[\begin{array}{c|c|c} V(S^R) & \mathbf{0} & \mathbf{0} \\ \hline \mathbf{e}_1^T & 0 & \mathbf{0} \\ \hline \mathbf{1}\mathbf{e}_1^T & \mathbf{0} & V(S^L) \end{array}\right]$$

Fig. 2. Recursive expressions for the encoding matrices H and V. S^L and S^R are canonical strategies with $|S^L| = n_1$ and $|S^R| = n_2$, with $n_1, n_2 > 1$. Every $H(S)$ and $V(S)$ matrix is square of dimension $|S| - 1$. $\mathbf{0}$ and $\mathbf{1}$ represent (often nonsquare) matrices of the appropriate size with all entries 0 and 1, respectively. 0 and 1 represent individual matrix entries. $\mathbf{e}_i$ is the unit basis column vector of the appropriate size with a 1 in the i^{th} position and 0's elsewhere. A^T denotes the transpose of A.

Measures We now generalize the concept of *measure* from [6] to account for differing weights for differing edges, which will be needed in analyzing strategies for CSIDH.

Definition 2. *A **measure** on T_n is a tuple $M = (\{p_i\}_{i=1}^n, f, g)$, where:*

- *$\{p_i\}_{i=1}^n$ is a sequence of positive real numbers,*
- *$f, g : \mathbb{R}^+ \to \mathbb{R}^+$ are some weight functions.*

We turn T_n into a weighted graph using the measure M as follows. For $1 \le i \le n-1$ we assign the weight $f(p_i)$ to any horizontal edge which connects a vertex on the line $x = i - 1$ to a vertex on the line $x = i$. For $1 \le i \le n-1$, we assign the weight $g(p_{n-i+1})$ to any vertical edge which connects a vertex on the line

$y = i - 1$ to a vertex on the line $y = i$. Any strategy in T_n inherits the weights from T_n.

Taking $\{p_i\}$ to be a constant sequence yields the original notion of measure defined in [6] when interpreted under our definition of T_n. The assignment of weights to the vertical edges may seem strange. We find motivation in CSIDH, where the cost of the i^{th} isogeny evaluation depends on the degree of the isogeny, which in turn depends on the $(n - i + 1)$-th prime used. One might use two separate sequences for the costs of horizontal and vertical edges respectively, but since both depend on the sequence of primes used we prefer the definition given above.

Throughout this paper, differing measures will all use common weight functions f and g. We will often identify a measure M with its sequence $\{p_i\}_{i=1}^n$ and omit mention of the functions f and g.

Definition 3. *The **cost** of a subgraph S of T_n for a given measure M is the sum of the weights of all edges in S. We write $(S)_M$ for the cost of S relative to M, or (S) when M is clear.*

The cost of a subgraph can be compactly expressed using the encoding matrices H and V introduced in Section 2.1 as

$$(S)_M = \sum_{i=1}^{n-1} f(p_i) \sum_{j=1}^{n-1} H_{j,i} + \sum_{i=1}^{n-1} g(p_{i+1}) \sum_{j=1}^{n-1} V_{i,j}. \tag{3}$$

Permutations In our original problem of optimizing CSIDH, we have the freedom to choose the order in which the primes ℓ_i are used. Choosing a different order will result in a permuted measure M, and so we need to take into account all possible permutations of M in our analysis.

Definition 4. *Let $\mathrm{Sym}(n)$ denote the symmetric group on $\{1, 2, \ldots, n\}$. We let $\sigma \in \mathrm{Sym}(n)$ act on a measure $M = \{p_i\}_{i=1}^n$ by defining $\sigma \cdot M$ to be the permuted measure $\{p_{\sigma(i)}\}_{i=1}^n$.*

The cost of a strategy S under the permuted measure $\sigma \cdot M$ is

$$(S)_{\sigma M} = \sum_{i=1}^{n-1} f(p_{\sigma(i)}) \sum_{j=1}^{n-1} H_{j,i} + \sum_{i=1}^{n-1} g(p_{\sigma(i+1)}) \sum_{j=1}^{n-1} V_{i,j}. \tag{4}$$

A *solution* for a measure $M = \{p_i\}_{i=1}^n$ is a pair (S, σ), where S is a canonical strategy with $|S| = n$, and $\sigma \in \mathrm{Sym}(n)$. A solution is *optimal* if $(S)_{\sigma M}$ is minimal among all solutions for M. Our ultimate goal is to find an algorithm which efficiently determines an optimal solution for M. Such a solution would determine an optimal method for performing an ideal-class group action evaluation as needed in CSIDH.

2.2 Mitigating Leakage Under Arbitrary Strategies

As first pointed out by Meyer et al. in [9] one may use dummy isogenies in CSIDH so that the number of isogenies constructed during the group action evaluation is independent of the private key. One issue that arises from using dummy isogenies is that additional multiplications are required on iterations that construct a dummy isogeny. This is because a real isogeny evaluation within the algorithm reduces the order of the point by a factor of the degree ℓ of the isogeny. If the isogeny is dummy, then the value of the point won't be updated and the factor ℓ will remain. In this situation we should instead multiply the point by ℓ to remove this factor.

Since strategies different from the multiplication-based strategy may require multiple isogeny evaluations on a given iteration, instead of multiplying all the points by ℓ we can simply multiply the initial randomly chosen point by any primes which will correspond to a dummy isogeny construction before the evaluation of the strategy begins. In this way we remove the 'bad' factors at the start by means of a single scalar multiplication per prime. This can be done in a secure fashion by using two copies of the point, multiplying one of them by each prime (not just the primes for dummies) while conditionally swapping the two points depending on the private key value for the current prime.

2.3 Two-Point Method and Parallelization

In [12], Onuki et al. find improved performance by tracking two points through each strategy: one from $E[\pi - 1]$ and one from $E[\pi + 1]$. When reaching an isogeny construction, the appropriate point is used depending on the sign of the private key in the corresponding position.

In the multiplication based strategy, having two points results in a negligible cost increase since only one of the two points needs to be multiplied to derive the kernel generator of the isogeny (though both points are still evaluated under the isogeny). When using other strategies this luxury is not an option since the path from the root to the leaf under consideration may pass through internal branch vertices, and so both points should be multiplied through nearly the entire strategy; the exception is horizontal paths within the strategy that end at a leaf and contain no branch vertices, in which case one can only multiply through whichever point is needed at the leaf node. In the regular model, this would result in highly increased cost since it uses roughly double the number of point multiplications.

As one remedy to this, we point out that the operations on the two points are entirely independent, and so if two processors are available all identical scalar multiplications and isogeny evaluations for the two points may be done in parallel. Therefore if one allows parallelizing operations, the method used by Onuki et al. could potentially be generalized to include strategies different from the multiplication based strategy.

If more than two processors are available, further parallelization is possible. An extensive analysis of parallelizing operations within the strategies of SIDH

was performed by Hutchinson and Karabina in [7], and we expect similar results to hold in this setting. We leave this for future examination and do not pursue this further in this work.

2.4 Splitting Isogenies into Multiple Batches (SIMBA)

In [9] Meyer et al. propose to partition the set of primes $\{\ell_1, \ldots, \ell_{74}\}$ into m many disjoint subsets to evaluate the group action on each smaller subset individually. The output curve from evaluating the action on one subset is fed as the input curve to the next, and a new initial point P is chosen for each iteration of each subset. They focus exclusively on positive private key values so that P is always chosen from $E[\pi - 1]$, and it's more likely that $|P|$ contains larger prime factors than smaller ones. Consequently, after a given number of rounds on a fixed key it's more likely that lower degree isogenies will still need to be constructed than higher ones. Meyer et al. therefore find it beneficial to merge the primes back into one set after μ many iterations and run CSIDH as originally proposed (but still using dummy isogenies) to construct the remaining isogenies. They call this technique Splitting Isogenies into Multiple Batches, or SIMBA-m-μ.

Within our framework, SIMBA can be summarized as: partition the primes $\{\ell_1, \ldots, \ell_n\}$ into m subsets, associate some strategy with each subset, and evaluate each strategy using the primes from each subset. Fresh points are randomly chosen for each strategy and must be multiplied by every prime not in the current subset, as well as by 4, prior to beginning the operations within the strategy.

We can generalize this further. First, there is no reason that the same strategy and permutation must be used for each of the subsets, so we are free to choose optimal parameters on each of them. Second, it's not required that the same partitioning be used each round. That is, once the strategies for each of the subsets have been evaluated once, we could optionally repartition the primes and use a different collection of strategies. This is quite advantageous since if any value b_i in the private key bound vector $\boldsymbol{b}$ is small in comparison to the rest of the vector, the prime ℓ_i can simply be removed from the partitioning after b_i number of rounds since all degree ℓ_i isogenies (both real and dummy) have likely been constructed by that point. This also eliminates the need of merging the batches after μ rounds since each batch is on a 'minimal' set of primes to begin with. Overall this has the effect of eliminating a significant number of redundant operations, although admittedly amounts to a much more complex algorithm.

This motivates the following definition. Recall that we identify a measure M with its sequence $\{p_i\}$, with some weight functions f and g hidden in the background.

Definition 5. *For a collection of numbers $M = \{p_1, \ldots, p_n\}$, a **SIMBA strategy** S is a collection of pairs $(S_1, M_1), \ldots, (S_m, M_m)$ such that*

1. *S_i is a strategy (under Definition 1) for $i = 1, \ldots, m$,*
2. *M_i is a measure for S_i for $i = 1, \ldots, m$,*

3. M is the disjoint union of $M_1, \ldots, M_m$.

*The S_i are referred to as the **SIMBA substrategies**, and M_i the **SIMBA submeasures**, of S. We say $(|S_1|, \ldots, |S_m|)$ is the **SIMBA partition** of S.*

Encoding SIMBA Strategies Just as we did with full strategies, it is desirable to encode SIMBA strategies as a pair of matrices (H, V). If $S = (S_1, S_2)$ is a SIMBA strategy on two SIMBA substrategies, we define

$$H(S) = \left[\begin{array}{c:c:c} H(S_1) & \mathbf{0} & \mathbf{0} \\ \hdashline \mathbf{0} & 0 & \mathbf{0} \\ \hdashline \mathbf{0} & \mathbf{0} & H(S_2) \end{array}\right] \text{ and } \quad V(S) = \left[\begin{array}{c:c:c} V(S_1) & \mathbf{0} & \mathbf{0} \\ \hdashline \mathbf{0} & 0 & \mathbf{0} \\ \hdashline \mathbf{0} & \mathbf{0} & V(S_2) \end{array}\right]$$

where $H(S_i)$ and $V(S_i)$ are the encoding matrices of the S_i as defined in Section 2.1. Then, for a SIMBA strategy $S = (S_1, S_2, \ldots, S_m)$ on $m \geq 3$ substrategies, we define

$$H(S) = \left[\begin{array}{c:c:c} H(S') & \mathbf{0} & \mathbf{0} \\ \hdashline \mathbf{0} & 0 & \mathbf{0} \\ \hdashline \mathbf{0} & \mathbf{0} & H(S_m) \end{array}\right] \text{ and } \quad V(S) = \left[\begin{array}{c:c:c} V(S') & \mathbf{0} & \mathbf{0} \\ \hdashline \mathbf{0} & 0 & \mathbf{0} \\ \hdashline \mathbf{0} & \mathbf{0} & V(S_m) \end{array}\right]$$

where $S' = (S_1, S_2, \ldots, S_{m-1})$. This definition is used to make the optimization problems in Sections 3.2 and 3.3 compatible with SIMBA strategies.

2.5 General Algorithm

We give a general high level algorithm as follows. Let $\boldsymbol{b} = (b_1, \ldots, b_n)$ be the bound vector used so that private key values e_i are chosen from the interval $[-b_i, b_i]$ (or $[0, b_i]$ in the case of a non-negative approach such as [9]), and let $r = \max_i b_i$. The algorithm runs through r many rounds, with the i-th round considering an active subset $M_i \subset \{\ell_1, \ldots, \ell_n\}$ of primes. A SIMBA strategy $S_i = \{(S_{i,1}, M_{i,1}), \ldots, (S_{i,m_i}, M_{i,m_i})\}$ is chosen (in advance) for each M_i, as well as a permutation $\sigma_{i,j}$ for each $M_{i,j}$. The i-th round iterates from $j = 1$ to m_i, evaluating each of the SIMBA substrategies $S_{i,j}$ under its corresponding permuted measure $\sigma_{i,j} M_{i,j}$ one time. Once all r rounds are complete a final phase is executed, in which any isogenies which remain to be constructed due to a failure (either real or dummy) are built by means of a multiplication-based strategy with a descending-prime permutation.

The subsets M_i are defined to consist of all primes ℓ_j for which $b_j - i$ is positive. This choice of M_i eliminates a great deal of redundancy since a prime ℓ_j is eliminated from all future strategies (save for the final phase) exactly upon finishing the b_j-th round. Section 3 is dedicated to finding well-performing parameter values for $\boldsymbol{b}$, the SIMBA strategy S, and the permutations $\sigma_{i,j}$. The

final phase of the algorithm after the r rounds finish is entirely dependent upon the points randomly generated during execution, and so the strategy used for it cannot be optimized since it's not known in advance which primes will be present during this step and determining a strategy during run time is infeasible.

The group action evaluation algorithm is detailed in Algorithm 2, which calls Algorithms 3 and 4 as subroutines. The algorithms as written use the two point method of [12], but can be easily modified to use the non-negative key method of [10] by ignoring steps which involve the variables P_{-1} and $NegPts$. In this case some steps may be simplified, such as collapsing the loops (**5**) and (**11**) in Algorithm 3 down into one. As noted before, the conditional branching based on the private key values e_i seen in loops (**8**) in Algorithm 2 and (**4**) of Algorithm 4 can be performed securely by using constant time conditional swaps to dummy points. The branching on lines (**15**), (**20**) in Algorithm 3 and lines (**14**), (**19**) in Algorithm 4 can be handled similarly.

Algorithm 2: Our Ideal-Class Group Action Algorithm

Parameters: $b = (b_1, \ldots, b_n)$ with $b_i \in \mathbb{Z}$, strategies $S_{i,j}$ and permutations $\sigma_{i,j}$ on some set of primes $L_{i,j} \subset \{\ell_1, \ldots, \ell_n\}$ for $1 \leq j \leq m_i$ for some m_i and $1 \leq i \leq \max_i b_i$.

Input : $A \in \mathbb{F}_p$ and a list of integers $(e_1, \ldots, e_m)$, with $e_i \in [-b_i, b_i]$.

Output: $B \in \mathbb{F}_p$ such that $[\mathfrak{l}_1^{e_1} \cdots \mathfrak{l}_m^{e_m}]E_A = E_B$.

$t \leftarrow 0$.
$c \leftarrow b$.
for $i = 1$ *to* $\max b_i$ **do**
  for $j = 1$ *to* m_i **do**
    Choose random points $P_1 \in E_A(\pi - 1)$ and $P_{-1} \in E_A(\pi + 1)$ using Elligator.
    Compute $u \leftarrow 4 \cdot \prod_{\ell_k \notin L_{i,j}} \ell_k$.
    Update $P_1 \leftarrow [u]\, P_1$ and $P_{-1} \leftarrow [u]\, P_{-1}$.
    for $\ell_k \in L_{i,j}$ **do**
      If $e_k = 0$, **then** update $P_1 \leftarrow [\ell_k]P_1$ and $P_{-1} \leftarrow [\ell_k]P_{-1}$.
      If $e_k \neq 0$, **then** update $P_{-sign(e_k)} \leftarrow [\ell_k]P_{-sign(e_k)}$.
    end
    $A, e, c, t \leftarrow \textsc{Eval}(S_{i,j}, \sigma_{i,j}, L_{i,j}\,;\, A, P_1, P_{-1}, e, c, t)$.
  end
end
$A \leftarrow \textsc{BuildRemaining}(A, e, c, t)$
Return A

Algorithm 3: EVAL: Strategy Evaluation

Parameters: A permutation σ represented as a subsequence of $\{1, 2, \ldots, n\}$ of length m, a strategy S with m leaves represented by its vertical encoding matrix $V(S)$, defined on the primes $\{\ell_{\sigma(1)}, \ldots, \ell_{\sigma(m)}\}$.

Input : $A \in \mathbb{F}_p$ representing a curve $E_A : y^2 = x^3 + Ax^2 + x$, points $P_1 \in E_A(\pi - 1)$ and $P_{-1} \in E_A(\pi + 1)$, current value e of private key, vector $c = (c_1, \ldots, c_n)$ such that c_i many ℓ_i degree isogenies remain to be constructed, integer t denoting total number of isogenies already constructed.

Output: Updated values for A, e, c, t according to the isogenies constructed.

Initialize point arrays $PosPts$ and $NegPts$, with $PosPts[0] \leftarrow P_1$ and $NegPts[0] \leftarrow P_{-1}$.
for $i = m$ *down to* 1 **do**
  Update P_1 and P_{-1} to the last entries of $PosPts$ and $NegPts$, respectively, each occurring at position k.
  $k' \leftarrow \max\{x : V(S)_{i,x} = 1\}$.
  for $j = k + 1$ *to* $k' - 1$ **do**
    Update $P_1 \leftarrow \left[\ell_{\sigma(j)}\right] P_1$ and $P_{-1} \leftarrow \left[\ell_{\sigma(j)}\right] P_{-1}$
    if $V(S)_{i,j} = 1$ **then**
      Append P_1 to $PosPts$, append P_{-1} to $NegPts$
    end
  end
  for $j = k'$ *to* i **do**
    Update $P_{sign(e_{\sigma(j)})} \leftarrow [\ell_{\sigma(j)}] P_{sign(e_{\sigma(j)})}$.
  end
  if $k = i$, **then** delete the last entries of $PosPts$ and $NegPts$.
  if $e_{\sigma(i)} \neq 0$ and $P_{sign(e_{\sigma(i)})} \neq \infty$ **then**
    Construct isogeny $\phi : E_A \to E_B$ of degree $\ell_{\sigma(i)}$ with kernel $\langle P_{sign(e_{\sigma(i)})} \rangle$.
    Update all points in $PosPts$ and $NegPts$ with their images under ϕ.
    $A \leftarrow B$, $c_{\sigma(i)} \leftarrow c_{\sigma(i)} - 1$, $e_{\sigma(i)} \leftarrow e_{\sigma(i)} - sign(e_{\sigma(i)})$, $t \leftarrow t + 1$.
  end
  if $e_{\sigma(i)} = 0$ and RAND$(\{1, 2, \ldots, \ell_{\sigma(i)}\}) \neq 1$ **then**
    Construct dummy isogeny $\phi : E_A \to E_B$ of degree $\ell_{\sigma(i)}$ with kernel $\langle P_{sign(e_{\sigma(i)})} \rangle$.
    Perform dummy isogeny evaluations on all points in $PosPts$ and $NegPts$.
    $A \leftarrow A$, $c_{\sigma(i)} \leftarrow c_{\sigma(i)} - 1$, $e_{\sigma(i)} \leftarrow e_{\sigma(i)}$, $t \leftarrow t + 1$.
  end
end
Return A, e, c, t

3 Optimization Methods

For much of this section we work over an arbitrary set of primes $M = \{p_1, \ldots, p_n\}$, and all strategies, permutations, and measures will reference these primes. These primes can be thought of as some subset of the odd primes used in CSIDH. Sub-

Algorithm 4: BUILDREMAINING: Construct any remaining isogenies.

Input : $A \in \mathbb{F}_p$, current value e of private key, vector $c = (c_1, \ldots, c_n)$ such that c_i many ℓ_i degree isogenies remain to be constructed, integer t denoting total number of isogenies already constructed

while $t \neq \sum_{i=1}^{n} b_i$ **do**
  Choose random points $P_1 \in E_A(\pi - 1)$ and $P_{-1} \in E_A(\pi + 1)$ using Elligator.
  Update $P_1 \leftarrow [4]P_1$ and $P_{-1} \leftarrow [4]P_{-1}$.
  for $k = 1$ to n **do**
    if $e_k = 0$, **then** update $P_1 \leftarrow [\ell_k]P_1$ and $P_{-1} \leftarrow [\ell_k]P_{-1}$.
    if $e_k \neq 0$, **then** update $P_{-sign(e_k)} \leftarrow [\ell_k]P_{-sign(e_k)}$.
  end
  for $k = n$ down to 1 **do**
    if $c_k \neq 0$ **then**
      Assign $Q \leftarrow P_{sign(e_k)}$.
      for $i = 1$ to $k - 1$ **do**
        if $c_i \neq 0$, **then** update $Q \leftarrow [\ell_i]Q$.
      end
      if $e_{\sigma(i)} \neq 0$ and $P_{sign(e_k)} \neq \infty$ **then**
        Construct isogeny $\phi : E_A \to E_B$ of degree $\ell_{\sigma(i)}$ with kernel $\langle Q \rangle$.
        Update $P_1 \leftarrow \phi(p_1)$ and $P_{-1} \leftarrow \phi(P_{-1})$
        $A \leftarrow B$, $c_k \leftarrow c_k - 1$, $e_{\sigma(i)} \leftarrow e_{\sigma(i)} - sign(e_{\sigma(i)})$, $t \leftarrow t + 1$.
      end
      if $e_{\sigma(i)} = 0$ and RAND$(\{1, 2, \ldots, \ell_{\sigma(i)}\}) \neq 1$ **then**
        Construct dummy isogeny $\phi : E_A \to A_B$ of degree $\ell_{\sigma(i)}$ with kernel $\langle Q \rangle$.
        Perform dummy isogeny evaluations on P_1 and P_{-1}.
        $A \leftarrow A$, $c_{\sigma(i)} \leftarrow c_{\sigma(i)} - 1$, $e_{\sigma(i)} \leftarrow e_{\sigma(i)}$, $t \leftarrow t + 1$.
      end
    end
  end
end
Return A

sections 3.1, 3.2, and 3.3 respectively tackle optimizing the strategy, permutation, and bound vector variables, respectively. This section concludes in Subsection 3.4, which discusses how the three optimization algorithms come together to produce a full parameter set for CSIDH.

3.1 Optimizing the Strategies

Let M be a measure. In this section we fix an arbitrary permutation σ and describe a method for determining a strategy which is optimal under the permuted measure σM. That is, we optimize $(S)_{\sigma M}$ over S for fixed σ and M. For this section by replacing M with σM we may assume that σ is the identity permutation. This reduces the problem to simply finding an optimal strategy for the

measure M. This is done nearly identically to the method described in [6] for constant measures.

Theorem 1. *Fix a measure $M = \{p_i\}_{i=1}^n$. Suppose S is a canonical strategy for which $(S)_M$ is minimal over all canonical strategies for M. If $k = |S^L|$, then S^L and S^R are canonical strategies for which $(S^L)_{M^L}$ and $(S^R)_{M^R}$ are minimal over all canonical strategies for M^L and M^R, respectively, where $M^L := \{p_i\}_{i=n-k+1}^n$ and $M^R := \{p_i\}_{i=1}^{n-k}$.*

The theorem is a generalization of Lemma 4.5 from [6]. The proof is essentially the same, with the appropriate generalizations made.

Proof. If S is fixed, we first notice that the cost that S^L contributes to $(S)_M$ depends only on the last k entries of M. This can be seen through Equation (3). Alternatively, by the construction of $S = S^L \# S^R$ given in Section 2.1, S^L is contained within the lines $x = n-k, y = 0$, and $y = -x + n - 1$, and by the weight assignment given in Definition 2 each horizontal (resp. vertical) edge in this region is assigned some weight from the set $\{f(p_{n-k+1}), \ldots, f(p_n)\}$ (resp. the set $\{g(p_{n-k+1}), \ldots, g(p_n)\}$). By a similar argument, the cost that S^R contributes to $(S)_M$ depends only on the first $n-k$ entries of M. We can therefore view S^L as a strategy in T_k under the measure M^L, and S^R as a strategy in T_{n-k} under the measure M^R.

Let a be the (unique) path in T_n connecting the root $(0,0)$ to the vertex $(n-k,0)$ (seen as the root of S^L), and let b be the (unique) path in T_n connecting the root $(0,0)$ to the vertex $(0,k)$ (seen as the root of S^R). Then the strategy S decomposes as a disjoint union $S = S^L \cup S^R \cup a \cup b$, and by the preceeding paragraph the cost of S under M can therefore be written as

$$(S)_M = (S^L)_{M^L} + (S^R)_{M^R} + \sum_{i=1}^{n-k} f(p_i) + \sum_{i=n-k+1}^{n} g(p_i),$$

where the first and second summations represent the cost of the paths a and b, respectively, under M. We notice that these two summations depend only on M and k, and not on the substrategies S^L and S^R themselves. If either S^L or S^R is suboptimal under M^L or M^R, respectively, then we may get a lower cost strategy S by replacing the suboptimal strategy with an optimal one under its corresponding measure. This concludes the proof.

Definition 6. *For a measure $M = \{p_i\}_{i=1}^n$ with $n > 1$, for $1 \le k \le n-1$ we define the k-th **left** and **right submeasures** of M as*

$$M_k^L = \{p_i\}_{i=n-k+1}^n \qquad\qquad M_k^R = \{p_i\}_{i=1}^{n-k}.$$

Let $C(M)$ be the cost of an optimal strategy under the measure $M = \{p_i\}_{i=1}^n$. As a consequence of Theorem 1, $C(M)$ can be computed recursively as

$$C(M) = \min_{k=1,\ldots,n-1} \left\{ C(M_k^L) + C(M_k^R) + \sum_{i=1}^{n-k} f(p_i) + \sum_{i=n-k+1}^{n} g(p_i) \right\}. \qquad (5)$$

Just as in the case of finding an optimal strategy for SIDH in [6], the above equality again suggests a dynamic programming approach for finding an optimal strategy in our generalized setting. That is, we compute $C(\{p_i\}_{i=1}^n)$ by using a sliding window submeasure which increases in size: we iterate $k = 1, \ldots, n$ and $j = 1, \ldots, n-k+1$ and compute $C(\{p_i\}_{i=j}^{j+k-1})$ using equation (5) with the length-one measure initial values $C(p_i) = 0$ for all i. Here, k represents the window size and j represents the window position. This gives an $\tilde{O}(n^2)$ algorithm computing the cost of the best strategy, and an optimal strategy can be constructed by keeping track of an index at which the minimum occurs at each step. Alternatively, one may construct the matrices H and V for the optimal strategy recursively as defined in Section 2.1.

In the two-point setting of [12], a similar result holds by doubling most of the above summations. The discussion in Section 2.3 suggests that every edge in the strategy should have double weight, except those which lie on a horizontal path ending in a leaf and containing no branch node. This occurs precisely when the left substrategy is T_1. For the two-point scenario we therefore have the recursive equality

$$C(M) = \min\left(\left\{C(M_1^R) + \sum_{i=1}^{n-1} f(p_i) + 2g(p_n)\right\} \cup \left\{C(M_k^L) + C(M_k^R) + \sum_{i=1}^{n-k} 2f(p_i) + \sum_{i=n-k+1}^{n} 2g(p_i) : k = 2, \ldots, n-1\right\}\right).$$

3.2 Optimizing the Permutations

We now fix a full strategy S and measure M, and take a mathematical programming approach to the problem of finding a permutation σ so that $(S)_{\sigma M}$ is minimal. Write $M = (\{p_i\}_{i=1}^n, f, g)$, and define column matrices

$$\boldsymbol{\mu} = [f(p_i)]_{i=1}^n \text{ and } \boldsymbol{\iota} = [g(p_i)]_{i=1}^n.$$

Let H and V be the matices that encode the edges of S, as defined in Section 2.1. If the primes are permuted according to σ, then by Equation (4) the cost of S with respect to the permuted measure is

$$(S)_{\sigma M} = \sum_{i=1}^{n-1}\sum_{j=1}^{n-1} H_{i,j}\mu_{\sigma(j)} + \sum_{i=1}^{n-1}\sum_{j=1}^{n-1} V_{i,j}\iota_{\sigma(i+1)}.$$

In order to simplify this expression and write it in a form that is amenable to standard optimization techniques, we will use the permutation matrix representation of $\mathrm{Sym}(n)$. For any $\sigma \in \mathrm{Sym}(n)$, let $\rho(\sigma) \in \{0,1\}^{n\times n}$ be defined by

$$\rho(\sigma) = \sum_{i=1}^{n} \boldsymbol{e}_{\sigma(i)}\boldsymbol{e}_i^T$$

where $\{\boldsymbol{e}_i\}_{i=1}^n$ are the standard basis column vectors. Then, letting

$$T_L = \left[I_{n-1}|\mathbf{0}\right],\ T_R = \left[\mathbf{0}|I_{n-1}\right], \text{ and } \Sigma = \rho(\sigma)$$

with I_{n-1} an identity matrix of size $n-1$, we can write the cost of strategy S under measure σM as

$$\begin{aligned}
(S)_{\sigma M} &= \sum_{i=1}^{n-1}\sum_{j=1}^{n-1} H_{i,j}\mu_{\sigma(j)} + \sum_{i=1}^{n-1}\sum_{j=1}^{n-1} V_{i,j}\iota_{\sigma(i+1)} \\
&= \sum_{i=1}^{n-1}\sum_{j=1}^{n-1} H_{i,j}(\Sigma\boldsymbol{\mu})_j + \sum_{i=1}^{n-1}\sum_{j=1}^{n-1} V_{i,j}(\Sigma\boldsymbol{\iota})_{i+1} \\
&= \sum_{i=1}^{n-1}\sum_{j=1}^{n-1} H_{i,j}(T_L\Sigma\boldsymbol{\mu})_j + \sum_{i=1}^{n-1}\sum_{j=1}^{n-1} V_{i,j}(T_R\Sigma\boldsymbol{\iota})_i \\
&= \mathbf{1}^T H T_L \Sigma\boldsymbol{\mu} + \mathbf{1}^T V^T T_R \Sigma\boldsymbol{\iota}
\end{aligned}$$

When solving for the optimal permutation for a given strategy and measure, all the quantities above are problem data, except for Σ, which is the decision variable. We thus rewrite $(S)_{\sigma M}$ as

$$\begin{aligned}
(S)_{\sigma M} &= \mathbf{1}^T H T_L \Sigma\boldsymbol{\mu} + \mathbf{1}^T V^T T_R \Sigma\boldsymbol{\iota} \\
&= \langle T_L^T H^T \mathbf{1}\boldsymbol{\mu}^T + T_R^T V \mathbf{1}\boldsymbol{\iota}^T, \Sigma\rangle_F
\end{aligned}$$

where $\langle\cdot,\cdot\rangle_F$ is the Frobenius inner product. Then the problem of finding the optimal permutation for a given strategy and measure is to minimize the above quantity subject to Σ being a permutation matrix; this can more succinctly be written

$$\begin{aligned}
\text{Minimize}\quad & \langle T_L^T H^T \mathbf{1}\boldsymbol{\mu}^T + T_R^T V \mathbf{1}\boldsymbol{\iota}^T, \Sigma\rangle_F \\
\text{Subject to}\quad & \Sigma\mathbf{1} = \mathbf{1} \\
& \mathbf{1}^T\Sigma = \mathbf{1}^T \\
& \Sigma \geq 0 \\
& \Sigma \in \mathbb{Z}^{n\times n}
\end{aligned} \tag{6}$$

Problem (6) is an integer linear program in standard equality form. If we relax away the integrality constraint, we arrive at Problem (7):

$$\begin{aligned}
\text{Minimize}\quad & \langle T_L^T H^T \mathbf{1}\boldsymbol{\mu}^T + T_R^T V \mathbf{1}\boldsymbol{\iota}^T, \Sigma\rangle_F \\
\text{Subject to}\quad & \Sigma\mathbf{1} = \mathbf{1} \\
& \mathbf{1}^T\Sigma = \mathbf{1}^T \\
& \Sigma \geq 0
\end{aligned} \tag{7}$$

which is a linear program in standard equality form.

By the Fundamental Theorem of Linear Programming, there is an optimal solution to Problem (7) which is a vertex of the feasible region $\{\Sigma \in \mathbb{R}^{n\times n} : \Sigma\mathbf{1} = \mathbf{1}, \mathbf{1}^T\Sigma = \mathbf{1}^T, \Sigma \geq 0\}$. This feasible region is B_n—the Birkhoff polytope

in $\mathbb{R}^{n^2}$—and its vertices are precisely the $n \times n$ permutation matrices; consequently, the optimal values of Problems (6) and (7) are equal, and there exists an optimizer of Problem (7) which is also an optimizer of Problem (6). Moreover, for a 'generic' linear objective function the optimizer of Problem (7) is unique, and so to solve Problem (6) it typically suffices to solve Problem (7). Indeed, in practice the solution obtained to Problem (7) for data arising from CSIDH parameters and strategies is always feasible for Problem (6). Even when the solution is not unique, we can rely on algorithms for linear programming problems which always return solutions which are vertices of the feasible region (*e.g.,* the simplex method) in order to find solutions to Problem (6).

An Extension to SIMBA Strategies The same arguments apply in a straightforward fashion to the case SIMBA strategies; in this scenario, however, we have

$$(S)_{\sigma M} = \langle T_L^T H^T \mathbf{1}\boldsymbol{\mu}^T + T_R^T V \mathbf{1}\boldsymbol{\iota}^T, \Sigma \rangle_F + (m-1)\mathbf{1}^T\boldsymbol{\mu}$$

where m is the number of SIMBA substrategies of S. Notably the additional term $(m-1)\mathbf{1}^T\boldsymbol{\mu}$ is independent of the decision variables Σ, and so we can use Problem (7) without modification when optimizing the permutation for a given SIMBA strategy.

An Extension to the Two-Point Method When optimizing permutations for the two point method, we cannot use Program (7) without a minor modification. In this setting, all vertical edges require two isogeny evaluations and, for the horizontal edges, some require two point multiplications while some require only one. In particular, when considering the i^{th} row of H, let $k_i = \max_{1 \le k \le n-1}\{k \colon V_{i,k} = 1\}$. Then in order to be able to compute the isogeny evaluations specified by V, for each 1 among the first k entries of the i^{th} row of H, we must multiply both torsion points by the corresponding prime, while for the remaining 1s in that row, we only need to multiply one torsion point (the one which corresponds to the sign of e_i).

To construct the appropriate linear program for this setting, we define a modified strategy matrices $\hat{H}(S)$ and $\hat{V}(S)$ by

$$\hat{H}(S)_{ij} = \begin{cases} H_{i,j} & \text{if } j \ge k_i \\ 2H_{i,j} & \text{if } j \le k_i - 1 \end{cases} \qquad \hat{V}(S) = 2V(S)$$

where k_i is as defined above. We can then use Problem (7) with the following modified objective function:

$$(S)_{\sigma M} = \langle T_L^T \hat{H}^T \mathbf{1}\boldsymbol{\mu}^T + T_R^T \hat{V} \mathbf{1}\boldsymbol{\iota}^T, \Sigma \rangle_F.$$

We note that when S is the multiplication-based strategy (or a SIMBA strategy all of whose SIMBA substrategies are multiplication-based), $\hat{H}(S) = H(S)$, and so we can apply Problem (7) by instead modifying the cost model, using $2\boldsymbol{\iota}$ in place of $\boldsymbol{\iota}$; since the best SIMBA substrategies we have found for the two point method have all been multiplication-based, we employ this modification in our parameter-finding scripts.

3.3 Optimizing the Bound Vector

We now leave behind the setting of full generality and return to CSIDH, where we consider the primes $M = \{\ell_1, \ldots, \ell_n\}$. Castryck et al. in [1] propose to select the values of the private key $(e_1, \ldots, e_n)$ from some common interval $[-b, b]$. Meyer et al. in [9] instead consider sampling each value e_i from its own interval $[0, b_i]$, where the vector $\boldsymbol{b} = (b_1, \ldots, b_n)$ is to be chosen so that a speedup is gained while still maintaining a target security level. In [9] the authors state that trying to find optimal values of b_i leads to a large integer optimization problem which is not likely to be solvable exactly. They give some vectors $\boldsymbol{b}$ that they found heuristically, but did not give details on the method used to find the provided values. We give details on our version of this optimization problem now.

Informally, the optimization problem can be written

$$\begin{array}{rl} \text{Minimize} & \text{The expected cost of computing the required isogenies} \\ \text{Subject to} & \text{The protocol is sufficiently secure} \end{array}.$$

To rewrite this as a mathematical program for the optimal exponent bound vector $\boldsymbol{b}$, we must determine the relationship between $\boldsymbol{b}$ and the cost of computing the (real and dummy) isogenies for the group action, using a given strategy, as well as the constraints that must be enforced on $\boldsymbol{b}$ in order to ensure security.

The requirement to maintain security in the case of non-negative exponents (*à la* [9]) is that ideals of the form $\mathfrak{l}_1^{e_1} \cdots \mathfrak{l}_n^{e_n}$ for $0 \leq e_i \leq b_i$ cover the class group nearly uniformly. An analysis was performed in [12] when selecting e_i from the intervals $[-b_i, b_i]$, which can be easily adapted to the case $[0, b_i]$. Under this adaptation, the requirement for the vector $\boldsymbol{b}$ when selecting each e_i from the interval $[0, b_i]$ is that $\prod(b_i + 1)$ is at least the size of the class group. By the heuristics in [3] the size of the class group is approximately $\sqrt{p}$ (recall that $p = 4\ell_1 \cdots \ell_n - 1$), and so we need $\prod(b_i + 1) \geq \sqrt{p}$ as a constraint in the optimization problem. Then, sufficient security can be guaranteed by enforcing

$$\prod_{i=1}^{n}(b_i + 1) \geq \sqrt{p} \qquad \Longleftrightarrow \qquad \sum_{i=1}^{n} \log_2(b_i + 1) \geq \frac{1}{2}\log_2 p = \lambda.$$

This reformulated constraint is convex, which is computationally convenient.

In the case of exponents which are not restricted to be non-negative (*à la* [1, 12]) the argument of [12] applies without modification, and we arrive at the similarly-reformulated convex constraint.

$$\prod_{i=1}^{n}(2b_i + 1) \geq \sqrt{p} \qquad \Longleftrightarrow \qquad \sum_{i=1}^{n} \log_2(2b_i + 1) \geq \frac{1}{2}\log_2 p = \lambda.$$

All that remains is to determine the cost of computing the isogenies when executing a given strategy. As before, let $\mu_{\sigma(i)}$ and $\iota_{\sigma(i)}$ denote the cost of evaluating multiplication-by-$\ell_{\sigma(i)}$ maps and evaluating $\ell_{\sigma(i)}$-isogenies, respectively. As well, let $\kappa_{\sigma(i)}$ be the combined cost of computing the kernel points from a given generator and computing the codomain curve of an $\ell_{\sigma(i)}$-isogeny.

We must consider two cases: rounds in which $\ell_{\sigma(i)}$ is 'active' (that is, there are still $\ell_{\sigma(i)}$-isogenies to be computed), and rounds in which $\ell_{\sigma(i)}$ is 'inactive' (that is, there are no more $\ell_{\sigma(i)}$-isogenies to compute).

$\ell_{\sigma(i)}$ is active. In this case, we must:

1. Compute one $\ell_{\sigma(i)}$-isogeny kernel and codomain curve, incurring cost $\kappa_{\sigma(i)}$.
2. Evaluate $[\ell_{\sigma(i)}]$ for each 1 entry of the i^{th} column of H, if $i \leq n-1$, incurring cost $(\mathbf{1}^T H)_i \mu_{\sigma(i)}$
3. Evaluate an $\ell_{\sigma(i)}$-isogeny for each 1 entry of the $(i-1)^{\text{th}}$ row of V, if $i \geq 2$, incurring cost $(V\mathbf{1})_{i-1} \iota_{\sigma(i)}$.

$\ell_{\sigma(i)}$ is inactive. In this case, we must evaluate $[\ell_{\sigma(i)}]$ once at the beginning of the strategy, incurring cost $\mu_{\sigma(i)}$.

Let c_i denote the cost associated with prime ℓ_i in an active round, and d_i denote the cost associated with prime ℓ_i in an inactive round. In the event that the starting point in every round is of full order (so that an isogeny of each order can be computed in each round), there are b_i active rounds for ℓ_i and $\max_j\{b_j\} - b_i$ inactive rounds for ℓ_i. Thus the total cost associated with ℓ_i is

$$c_i \cdot b_i + d_i \cdot (\max_j\{b_j\} - b_i) = (c_i - d_i) \cdot b_i + \max_j\{b_j\} d_i$$

so that the total cost across all i is $\langle \boldsymbol{c} - \boldsymbol{d}, \boldsymbol{b} \rangle + \max_j\{b_j\} \mathbf{1}^T \boldsymbol{d}$ where, by the above arguments

$$\begin{aligned} \boldsymbol{c} &= \Sigma^{-1} \left((\mathbf{1}^T H T_L)^T \circ (\Sigma \boldsymbol{\mu}) + (T_R^T V \mathbf{1}) \circ (\Sigma \boldsymbol{\iota}) + \Sigma \boldsymbol{\kappa} \right) \text{ and} \\ \boldsymbol{d} &= \boldsymbol{\mu} \end{aligned}$$

where $\circ$ is the Hadamard product.

So far, we have accounted only for the the cost incurred in the first $\max_j\{b_j\}$ strategy executions. If each execution of a strategy successfully let us evaluate isogenies of each active degree ℓ_i this would be sufficient; however, when selecting our initial points P_0, we are not guaranteed that they will be of full order, and thus it is possible that there will be some active primes for which we are not able to construct the required isogenies. When this happens, we necessarily must perform additional rounds of computation. In order to account for the cost of these additional rounds, we must estimate the number of additional rounds required and the cost of one round.

For each i, the point P_0 will allow us to compute the required $\ell_{\sigma(i)}$-isogeny if and only if:

1. $P_0 \in E[\pi - 1]$ (in case $b_{\sigma(i)} > 0$), or $P_0 \in E[\pi + 1]$ (in case $b_{\sigma(i)} < 0$); and,
2. $\ell_{\sigma(i)}$ divides the order of P_0.

If we choose $\boldsymbol{b} \geq \mathbf{0}$ (as proposed in [9]), or use the two-point technique of [12], at the beginning of each strategy round these conditions are satisfied with probability $\frac{\ell_{\sigma(i)}-1}{\ell_{\sigma(i)}}$, since for each i we have $E[\ell_i, \pi \pm 1] \cong \mathbb{Z}/\ell_i\mathbb{Z}$. For large $\ell_{\sigma(i)}$ the success probability is relatively high, and so we expect most of the isogenies will be computed during the $\max_j\{b_j\}$ rounds. Though we can in principle compute the expected cost of each additional round for a given bound vector $\boldsymbol{b}$, this cost is not a convex function of $\boldsymbol{b}$, and its inclusion in the mathematical program would make it difficult to solve. Instead, acknowledging that few isogenies need to be computed, and that these isogenies will likely correspond to small primes for which isogeny evaluations are cheap, we approximate the expected cost of an additional round by $\mathbf{1}^T\boldsymbol{\mu}$. This is clearly an underestimate of the expected cost (we have to multiply by each prime at least once in order to obtain points of prime order to generate the isogeny kernels) but works well enough in practice to yield a runtime improvement.

It remains to determine the expected number of required additional rounds. The expected total number of rounds required to complete the required $\ell_{\sigma(i)}$ isogenies is $\frac{\ell_{\sigma(i)}}{\ell_{\sigma(i)}-1}b_{\sigma(i)}$, and $b_{\sigma(i)}$ rounds which include the prime $\ell_{\sigma(i)}$ are completed. Thus the number of additional rounds required for $\ell_{\sigma(i)}$ is expected to be $\frac{b_{\sigma(i)}}{\ell_{\sigma(i)}-1}$. The maximum of this quantity over all i is then the number of additional rounds expected to be required to finish the algorithm.

From the above, given a pair (H, V) of strategy matrices and a permutation matrix Σ, we use the following program to estimate the optimal bound vector when using SIMBA with only one torsion point:

$$\begin{array}{rl} \text{Minimize} & \langle \boldsymbol{c}-\boldsymbol{d}, \boldsymbol{b}\rangle + \max_j\{b_j\}\mathbf{1}^T\boldsymbol{d} + \max_j\left\{\frac{b_j}{\ell_j-1}\right\}\mathbf{1}^T\boldsymbol{\mu} \\ \text{Subject to} & \sum_{i=1}^n \log_2(b_i+1) \geq \lambda \\ & \boldsymbol{b} \geq 0 \\ & \boldsymbol{b} \in \mathbb{Z}^n \end{array} . \tag{8}$$

We note that the program does *not* take into account the additional multiplications required to construct the starting points for the SIMBA substrategies. This term is $(m-1)\cdot\mathbf{1}^T\boldsymbol{\mu}$, where m is the number of SIMBA substrategies. Notably, this term is constant as a function of the decision variables $\boldsymbol{b}$ and so the optimal solution does not change. Though the objective function and constraints are convex, it is not obvious how to solve this program because the variables are constrained to be integer. To approximate the solution, we begin by relaxing to a continuous convex program by removing the constraint $\boldsymbol{b} \in \mathbb{Z}^n$ and solve. Let the relaxed problem be denoted (CP_0) and let its solution $\hat{\boldsymbol{b}}^{(0)}$. We construct a new program (CP_1) by adding the constraint $b_{i_0} = \left\lceil \hat{b}^{(0)}_{i_0} \right\rfloor$, where i_0 is the index of the entry of $\hat{\boldsymbol{b}}^{(0)}$ which is closest to integer. Then for $1 \leq k \leq n-1$, we repeat this process: solve (CP_k) and fix the entry of $\boldsymbol{b}$ which is nearest to an integer in $\hat{\boldsymbol{b}}^{(k)}$. Finally, in (CP_n), all but one variable is fixed; we solve the problem and round the only unfixed variable up to ensure sufficient security.

At the end of this process the bound vector may be 'too secure;' that is, it may be possible to reduce some entries of $\boldsymbol{b}$ (and hence reduce the expected running time of key exchange) while maintaining the required security level. Our *ad hoc* solution in this scenario is to repeatedly reduce the entry of $\boldsymbol{b}$ which has the largest index among all the entries of $\boldsymbol{b}$ which take on the value $\max_j\{b_j\}$, until no entry of $\boldsymbol{b}$ can be decreased while maintaining the required security level.

When using two torsion points in each strategy, the process is the essentially the same, except that the coefficient vectors change slightly (because we sometimes have to perform two computations—one for each torsion point—rather than one) and that the mathematical program uses a different bound to ensure security. In particular, the coefficient vectors are given by

$$\boldsymbol{c} = \Sigma^{-1}\left((\mathbf{1}^T\hat{H}T_L)^T \circ (\Sigma\boldsymbol{\mu}) + (T_R^T\hat{V}\mathbf{1}) \circ (\Sigma\boldsymbol{\iota}) + \Sigma\boldsymbol{\kappa}\right) \text{ and}$$
$$\boldsymbol{d} = 2\boldsymbol{\mu}$$

(where $\hat{H}$ and $\hat{V}$ are as defined in Section 3.2), and the new mathematical program is

$$\begin{array}{ll} \text{Minimize} & \langle \boldsymbol{c}-\boldsymbol{d}, \boldsymbol{b}\rangle + \max_j\{b_j\}\mathbf{1}^T\boldsymbol{d} + 2\max_j\left\{\frac{b_j}{\ell_j-1}\right\}\mathbf{1}^T\boldsymbol{\mu} \\ \text{Subject to} & \sum_{i=1}^n \log_2(2b_i+1) \geq \lambda \\ & \boldsymbol{b} \geq 0 \\ & \boldsymbol{b} \in \mathbb{Z}^n \end{array} . \tag{9}$$

As in Section 3.2, in the special case that S is the multiplication-based strategy or a SIMBA strategy all of whose SIMBA substrategies are multiplication-based, we can instead alter the definition of $\boldsymbol{c}$ to use a modified cost model with $2\boldsymbol{\iota}$ in place of $\boldsymbol{\iota}$.

3.4 The Complete Optimization Methodology

So far we have defined the optimization methodology only piecewise; in this section we present the full optimization 'pipeline,' starting from a measure $M = (\{\ell_i\}_{i=1}^n, f, g)$ and ending with a complete parameter set: a bound vector which defines the keyspace, and a collection of SIMBA strategies and permutations to use for each round of the algorithm. We first present the routine we used for plain SIMBA (*à la* [9]) and then discuss modifications we make when optimizing for the two-point technique.

Plain SIMBA

1. We first search for a SIMBA strategy $S = (S_1, S_2, \ldots, S_m)$ and corresponding permutation Σ. In particular, we apply Algorithm 5 on measure $M = (\{\ell_i\}_{i=1}^n, f, g)$. We chose $T = 1000, m_{\min} = 1, m_{\max} = 5$. In initial searches,

we did not bound the sizes of the SIMBA substrategies; going, we chose to bound the size of each SIMBA substrategy by

$$\max\left\{2, \left\lfloor \frac{n}{m+2} \right\rfloor\right\} \leq |S_j| \leq \left\lceil \frac{n}{m} \right\rceil + 15 \;\; \forall 1 \leq j \leq m.$$

(where m is the number of SIMBA substrategies in consideration), because initial searches suggested that this range was most promising. This S will be the SIMBA strategy that is used in the first round of computing the class group action.

2. Using the strategy and permutation obtained in step 1., we approximately solve the program (8) using the iterative rounding technique described in Section to obtain a bound vector $\boldsymbol{b}$.
3. For $2 \leq k \leq \max_j\{b_j\}$, let $M_k^{(\boldsymbol{b})} = (\{\ell_i\}_{i:\, b_i \geq k}, f, g)$. To obtain a permutation and SIMBA strategy for the k^{th} round of computation, we run Algorithm 5 on the measure $M_k^{(\boldsymbol{b})}$. We used $T = 100, m_{\min} = 1, m_{\max} = 5$. As in Step 1., for each number m of substrategies, we bound the size of each SIMBA substrategy by

$$\max\left\{2, \left\lfloor \frac{n}{m+2} \right\rfloor\right\} \leq |S_j| \leq \left\lceil \frac{n}{m} \right\rceil + 15 \;\; \forall 1 \leq j \leq m.$$

SIMBA and the Two-Point Technique The algorithm of Section 3.4 applies essentially unchanged when the two-point technique of [12] is used. In this setting the best SIMBA substrategies we found were consistently multiplication-based. Seeing this, we decided to do an exhaustive search for the optimal SIMBA decomposition and permutation when all SIMBA substrategies are multuplication-based. In particular, our process was:

1. For each m between 1 and 5 and each partition $P = (n_1, n_2, \ldots, n_m)$ of n with parts of size at least 2, compute the optimal permutation σ for the SIMBA strategy S whose substrategies are the multiplication-based strategies of size $n_1, n_2, \ldots, n_m$. Choose the partition, permutation, and strategy with the lowest cost among these.
2. Using the strategy and permutation obtained in step 1., we approximately solve the program (9) using the iterative rounding technique described in Section to obtain a bound vector $\boldsymbol{b}$.
3. For $2 \leq k \leq \max_j\{b_j\}$, let $M_k^{(\boldsymbol{b})} = (\{\ell_i\}_{i:\, b_i \geq k}, f, g)$. Applying the same technique as in step 1., find the optimal permutation and partition for each $M_k^{(\boldsymbol{b})}$.

Optimzing for Submeasures. Suppose M' is a proper submeasure of $M = \{\ell_i\}$. We note that the cost of evaluating a strategy S under M' is $(S)_{M'} + m\mathbf{1}^T\boldsymbol{\mu}_{M''}$, where M'' is the complement of M' in M and $\boldsymbol{\mu}_{M''}$ is the subvector of $\boldsymbol{\mu}$ corresponding to the indices present in M''. This additional term accounts

Algorithm 5: Our stochastic search algorithm for an optimal strategy and permutation.

Input : A measure M of size n. Natural numbers $T, m_{\min}, m_{\max}$. An initial permutation σ^*.
Output: A permutation σ and SIMBA strategy S

Choose $m^* \leftarrow \{m_{\min}, m_{\min}+1, \ldots, m_{\max}\}$ uniformly at random
Choose $P^* = (n_1, n_2, \ldots, n_{m^*})$, a partition of n, uniformly at random
Set $S^* = (S_1^*, S_2^*, \ldots S_{m^*}^*)$ to be the optimal SIMBA strategy with SIMBA substrategies of size $(n_1, n_2, \ldots, n_m)$ for the measure $\sigma^* M$
Set $C^* = (S^*)_{\sigma^* M}$
for i **from** 1 **to** T **do**
    Set $(\sigma, C) \leftarrow (\sigma^*, C^*)$
    Choose $m \leftarrow \{m_{\min}, m_{\min}+1, \ldots, m_{\max}\}$ uniformly at random
    do
        Set $C' \leftarrow C$
        Choose $P = (n_1, n_2, \ldots, n_m)$, a partition of n, uniformly at random
        Set $S = (S_1, S_2, \ldots S_m)$ to be the optimal SIMBA strategy with SIMBA substrategies of size $(n_1, n_2, \ldots, n_m)$ for the measure σM
        Set σ to be the optimal permutation for S and M
        Set $C \leftarrow (S)_{\sigma M}$
    while $C < C'$
    if $C < C^*$ **then**
        Set $(\sigma^*, m^*, P^*, S^*, C^*) \leftarrow (\sigma, m, P, S, C)$
    end
end
Return (P^*, σ^*, S^*)

for the multiplications that must be performed to remove the prime factors present in M'' from the order of the initial points of the strategy (see line (**7**) of Algorithm 2). It is important to correctly account for this additional cost during algorithms which will compare the cost of SIMBA strategies that consist of different numbers of SIMBA substrategies—in particular, when computing costs in Algorithm 5, we must modify lines (**4**) and (**13**) to include this term. When using the two-point method, the additional term is instead $2m\mathbf{1}^T\boldsymbol{\mu}_{M''}$, since the primes of M'' must be eliminated from the orders of both torsion points at the beginning of each SIMBA substrategy.

4 Implementation

4.1 Cost Model

In terms of formulating a cost model, there are essentially two scenarios: using Montgomery curves with the formulas of [10], or using twisted Edwards curves with the formulas of [2]. The costs for various operations are summarized in Table 1. We use **M** to denote $\mathbb{F}_p$ multiplications, **S** to denote $\mathbb{F}_p$ squarings, and

Operation	**M**	**S**	**a**	
			Montgomery	Edwards
`LADDER`	$8t-4$	$4t-2$	$8t-6$	$8t-6$
`EVAL`	$2p-2$	2	$p+1$	$p+3$
`KER`	$2p-6$	$p-3$	$4p-12$	$3p-11$
`CODOM`	$p+2t^*-1$	$2t+6$	6	2

Table 1. Costs for various operations. $\mathbf{M}, \mathbf{S}$, and $\mathbf{a}$ respectively represent multiplications, squarings, and additions in $\mathbb{F}_p$. p is an odd prime, $t = \lceil \log_2(p) \rceil$, and t^* is the Hamming weight of p. For the purposes of our cost model, we use the estimate $t^* \approx \frac{1}{2}\lceil \log_2 p \rceil$.

$\mathbf{a}$ to denote $\mathbb{F}_p$ additions/subtractions. In the table, p is interpreted as an odd prime. `LADDER` refers to computing $[p]P$ for a given point P using the Montgomery ladder. The operation `KER` denotes the cost of computing the kernel points $P, [2]P, \ldots, [\frac{p-1}{2}]P$ of an isogeny φ from a given generator P of order p. In the Montgomery setting, the `KER` table entry includes the cost of the computing the points $[i]P$, as well as the $p-1$ additions required for computing the sums and differences of these coordinates described in Algorithm 4 of [4]. `CODOM` considers constructing the codomain of a degree p isogeny φ given its kernel points $\langle P \rangle$. `EVAL` computes $\varphi(Q)$ for a given point Q, assuming the kernel points are already computed. We point out that each operation requires the same number of multiplications and squarings independent of the setting (*e.g.*, Montgomery or Edwards), but the number of additions and subtractions vary.

In the context of a measure $M = (\{\ell_i\}, f, g)$ on a strategy for CSIDH, $f(\ell_i)$ represents the cost of performing the operation $(\ell_i, P) \mapsto [\ell_i]P$, while $g(\ell_i)$ represents the cost to evaluate an isogeny of degree ℓ_i at some point (assuming the kernel points have been computed already). In practice, we therefore take f as the sum over the `LADDER` row of Table 1 and g as the sum over the `EVAL` row, including only one of the 'Montgomery' or 'Edwards' columns according to the appropriate context. We set $\mathbf{S} = 0.8\mathbf{M}$. We tested each of the conversions $\mathbf{a} = 0\mathbf{M}$, $\mathbf{a} = 0.05\mathbf{M}$, and $\mathbf{a} = 0.2\mathbf{M}$, and found that that first two approximations gave parameter sets with nearly identical runtime, while the third was slightly slower; in choosing our final parameter sets we used the results arising from $\mathbf{a} = 0\mathbf{M}$.

4.2 Implementation Details

We applied the results from Section 3 to two separate settings:

1. The work of Meyer, Campos, and Reith in [9] (based on previous work of Meyer and Reith in [10]). Here, Montgomery curves are used with points represented in XZ-coordinates. To compute the codomain curve of an isogeny, a conversion to a Twisted-Edwards model is used. This method uses non-

negative private key values, and so only one point is traced through a strategy at a time. We refer to this as the "MCR method".

2. The work of Cervantes-Vázquez, Chenu, Chi-Domínguez, De Feo, Rodríguez-Henríquez, and Smith in [2]. Here, twisted Edwards curves are used exclusively with points represented using YZ-coordinates. The authors apply formulas for the Edwards setting to both the MCR method and the two-point technique of [12], along with a projectivized Elligator map and optimized addition chains for scalar multiplication. We call this the "CCCDRS" method.

In each setting we used the optimization techniques of Section 3.4 to find full CSIDH parameter sets at the 128-bit security level, where we take the primes ℓ_i suggested by [1] for CSIDH-512. It should be pointed out that Peikert in [13] suggests that the parameters given by [1] for CSIDH-512 may not actually provide 128 bits of security, but we consider this parameter set in order to directly compare with previous optimizations of CSIDH; all of the results in this work are compatible with any collection of distinct odd primes used for CSIDH. We implemented Algorithm 5 in a combination of Octave and Matlab in order to construct SIMBA strategies, permutations, and bound vectors for each of the implementations described below.

Table 2 summarizes our results for each of the implementations we consider. The values of the table reflect the median over 200 iterations of a single group action evaluation, including validation of supersingularity of the output curve. All of the tests were executed on a i7-4790k clocked at 4 GHz with Turbo boost disabled and running on a single core only. Some of the constant-time implementations did not have generic arithmetic and so were updated to work with generic arithmetic in order to work on our device.

The first row of Table 2 gives the original implementation of CSIDH due to Castryck et al. in [1]. We stress that their implementation is *not* constant-time and include it in the table only for reference.

For the MCR method we used the publicly available code of [9], modified to fit our optimized parameter set (which includes an optimized SIMBA strategy for each round, a permutation for each strategy, and a bound vector). Our implementation of the MCR method used a custom Sage script which takes a strategy and permutation as input and outputs C code which efficiently executes the given strategy and permutation. In particular, we merged together consecutive point multiplications (horizontal paths in the strategy for which no internal leaf in the path is a branch node). This Sage script allowed us to test a wide variety of strategies without having to write custom code for each one. The implementation did not use the optimizations suggested by [2]. Compared with [9], our results yielded a 16.85% speedup.

For the CCCDRS implementations we only considered the two-point version, and we did not find any SIMBA substrategies that outperform the multiplication-based strategy. Consequently our C code generation script for this implementation only produces code for custom SIMBA substrategy *sizes*, permutations, and bound vectors. We used an Octave script to produce C header files that can

be used as drop-in replacements for corresponding header files in the implementation of [2] to implement our custom parameters.

In order to better demonstrate how optimizing each parameter using our techniques affects the efficiency of the implementations, we provide benchmarks for three CCCDRS method implementations. The first we denote as CCCDRS-1, in which we use the bound vector of [2] and a single SIMBA strategy S and corresponding permutations for the full measure $M = \{\ell_i\}$ found using Algorithm 5; here, the same strategy S is used in each round. For CCCDRS-1, we achieve a speedup of only 0.53% over the original implementation of [2]. Our second implementation is denoted CCCDRS-2, in which we modify CCCDRS-1 to use optimized permutations and a SIMBA strategy on the submeasure $M_i^{(\boldsymbol{b})}$ in the i^{th} round, for $1 \leq i \leq 7 = \max_j\{b_j\}$, rather than a SIMBA strategy and permutation on the full measure M. For CCCDRS-2 we attained a speedup of 3.73% over [2]. Finally we considered CCCDRS-3, where we use a bound vector obtained by the technique of Section 3.3 on top of the other optimizations of CCCDRS-2. CCCDRS-3 applies all of the optimizations of Section 3, and with it we achieved a speedup of 5.05%.

Implementation	**M**	**S**	**a**	Latency (Mcycles)	Speedup (%)
CSIDH [1]	463 287	136 654	416 891	610.2	-
MCR [9]	1 036 675	425 377	1 020 712	1483.9	-
This work (MCR)	905 200	312 483	859 759	1233.9	16.85
CCCDRS [2] (Two pt.)	664 936	224 081	750 992	879.1	-
This work (CCCDRS-1)	659 816	223 793	745 710	874.4	0.53
This work (CCCDRS-2)	637 352	218 635	724 958	846.3	3.73
This work (CCCDRS-3)	632 444	209 310	704 576	834.4	5.08

Table 2. Field operation counts and latency for seven relevant implementations of CSIDH-512.

5 Conclusions

We developed systematic techniques for optimizing three parameters used in the CSIDH group action evaluation algorithm: the strategy used, the permutation of the primes ℓ_i, and the bound vector from which private key values are chosen. Prior works in this area have used *ad hoc* methods for determining these parameters, and as far as we are aware this work is the first step in the direction of determining an optimal parameter set. Our implementation results show that significant speedups can be achieved when using our techniques to find efficient parameter sets. In light of recent cryptanalysis (in particular, [14]), new CSIDH parameter sets will have to be derived to meet NIST security levels while still being efficient. The optimization methods presented here can be used to contribute to these parameter sets (in the form of the bound vector) and to efficient class group action evaluation algorithms in this context, rather than using *ad hoc* techniques.

References

1. Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. CSIDH: An Efficient Post-Quantum Commutative Group Action. In Thomas Peyrin and Steven Galbraith, editors, *Advances in Cryptology – ASIACRYPT 2018*, pages 395–427, Cham, 2018. Springer International Publishing.
2. Daniel Cervantes-Vázquez, Mathilde Chenu, Jesús-Javier Chi-Domínguez, Luca De Feo, Francisco Rodríguez-Henríquez, and Benjamin Smith. Stronger and Faster Side-Channel Protections for CSIDH. In Peter Schwabe and Nicolas Thériault, editors, *Progress in Cryptology – LATINCRYPT 2019*, pages 173–193, Cham, 2019. Springer International Publishing.
3. H. Cohen and H. W. Lenstra. Heuristics on Class Groups of Number Fields. In Hendrik Jager, editor, *Number Theory Noordwijkerhout 1983*, pages 33–62, Berlin, Heidelberg, 1984. Springer Berlin Heidelberg.
4. Craig Costello and Huseyin Hisil. A Simple and Compact Algorithm for SIDH with Arbitrary Degree Isogenies. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017*, pages 303–329, Cham, 2017. Springer International Publishing.
5. Jean-Marc Couveignes. Hard Homogeneous Spaces. Cryptology ePrint Archive, Report 2006/291, 2006. `https://eprint.iacr.org/2006/291`.
6. Luca De Feo, David Jao, and Jérôme Plût. Towards Quantum-Resistant Cryptosystems from Supersingular Elliptic Curve Isogenies. *J. Mathematical Cryptology*, 8(3):209–247, 2014.
7. Aaron Hutchinson and Koray Karabina. Constructing Canonical Strategies for Parallel Implementation of Isogeny Based Cryptography. In *19th International Conference on Cryptology in India, New Delhi, India, December 9–12, 2018, Proceedings*, pages 169–189. Springer, 12 2018.
8. Amir Jalali, Reza Azarderakhsh, Mehran Mozaffari Kermani, and David Jao. Towards Optimized and Constant-Time CSIDH on Embedded Devices. Cryptology ePrint Archive, Report 2019/297, 2019. `https://eprint.iacr.org/2019/297`.
9. Michael Meyer, Fabio Campos, and Steffen Reith. On Lions and Elligators: An Efficient Constant-Time Implementation of CSIDH. In Jintai Ding and Rainer Steinwandt, editors, *Post-Quantum Cryptography*, pages 307–325, Cham, 2019. Springer International Publishing.
10. Michael Meyer and Steffen Reith. A Faster Way to the CSIDH. In Debrup Chakraborty and Tetsu Iwata, editors, *Progress in Cryptology – INDOCRYPT 2018*, pages 137–152, Cham, 2018. Springer International Publishing.
11. Tomoki Moriya, Hiroshi Onuki, and Tsuyoshi Takagi. How to Construct CSIDH on Edwards Curves. Cryptology ePrint Archive, Report 2019/843, 2019. `https://eprint.iacr.org/2019/843`.
12. Hiroshi Onuki, Yusuke Aikawa, Tsutomu Yamazaki, and Tsuyoshi Takagi. (Short Paper) A Faster Constant-Time Algorithm of CSIDH Keeping Two Points. In Nuttapong Attrapadung and Takeshi Yagi, editors, *Advances in Information and Computer Security*, pages 23–33, Cham, 2019. Springer International Publishing.
13. Chris Peikert. He Gives C-Sieves on the CSIDH. Cryptology ePrint Archive, Report 2019/725, 2019. `https://eprint.iacr.org/2019/725`.
14. Chris Peikert. He Gives C-Sieves on the CSIDH. *IACR Cryptology ePrint Archive*, 2019:725, 2019.
15. Alexander Rostovtsev and Anton Stolbunov. Public-Key Cryptosystem Based on Isogenies. Cryptology ePrint Archive, Report 2006/145, 2006. `https://eprint.iacr.org/2006/145`.