Design, Analysis, and Optimization of Isogeny-Based Key Establishment Protocols

by

Jason Travis LeGrow

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Combinatorics and Optimization

Waterloo, Ontario, Canada, 2020

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner:	Christophe Petit Senior Lecturer, School of Computer Science, University of Birmingham
Supervisors:	David Jao Professor, Department of Combinatorics and Optimization, University of Waterloo
	Michele Mosca Professor, Department of Combinatorics and Optimization, University of Waterloo
Internal Member:	Alfred Menezes Professor, Department of Combinatorics and Optimization, University of Waterloo
Internal Member:	Douglas Stebila Associate Professor, Department of Combinatorics and Optimization, University of Waterloo
Internal-External Member:	Jerry Wang Assistant Professor, Department of Pure Mathematics, University of Waterloo

Author's Declaration

Statement of Contributions

I was the sole author of Chapters 1 and 7, which appear for the first time in this thesis.

I was the sole author of Chapter 2. This chapter is adapted from my Master's thesis [50, Chapter 1, Section 1.2]. This chapter contains no new contributions; only background material.

In Chapter 3, I was the sole author of Section 3.1, and it appears for the first time in this thesis. Section 3.2 is adapted from [39], which is joint work with Aaron Hutchinson, Brian Koziel, and Reza Azarderakhsh.

The results of Chapter 4 were joint work with David Jao, Christopher Leonardi, and Luis Ruiz-Lopez. Section 4.1 was written solely by me, and Section 4.3 contains results derived primarily by me and was written primarily by me, with input and edits from my co-authors. Section 4.2 was originally written primarily by Christopher Leonardi and Luis Ruiz-Lopez, and is included for continuity reasons. For the purposes of this thesis I added Sections 4.1 and 4.2.1 and expanded on the text of the remaining subsections. Sections 4.4 and 4.5 were written by me. This chapter is adapted from [40].

The results of Chapter 5 were joint work with Aaron Hutchinson, Brian Koziel, and Reza Azarderakhsh. Sections 5.1, 5.2 and 5.4 were written by me. Excluding Subsections 5.3.1 and 5.3.2, I was the primary author of Section 5.3, with edits from Aaron Hutchinson. With the exception of Subsection 5.5.3, the results of Section 5.5 were derived primarily by me and I was the primary author of this content, with input and edits from Aaron Hutchinson; for Subsection 5.5.3, Aaron Hutchinson stated and proved the main result and wrote the section with minor input and edits from me. In terms of implementations, my co-authors were responsible for implementing MCRim (discussed in Section 5.6), while I implemented CCCDRS-1, CCCDRS-2, and CCCDRS-3 (Section 5.6), as well as the stochastic search technique discussed in Section 5.5.5. Brian Koziel was responsible for the benchmarks reported in Table 5.2, while I was responsible for the benchmarks in Table 5.3. This chapter is adapted from [39].

The results of Chapter 6 were joint work with Aaron Hutchinson. Sections 6.1–6.6 were written by me. I devised the formalism of Section 6.7, and it was written primarily by Aaron Hutchinson with edits from me. I derived the results of Section 6.8; within that section, Sections 6.8.1 and 6.8.2 were written by Aaron Hutchinson, while Sections 6.8.3 and 6.8.4 were written primarily by me. Aaron Hutchinson conducted the simulations in Section 6.9 and wrote the majority of the text in that section, while I prepared the figures. Section 6.10 was written by me.

Abstract

We analyze the Commutative Supersingular Isogeny Diffie-Hellman protocol (CSIDH), a novel supersingular isogeny-based key establishment protocol. Our analysis is from three perspectives:

Quantum Cryptanalysis. Building upon quantum attacks on ordinary isogeny-based cryptography, we propose a subexponential-time quantum algorithm for inverting the complex multiplication group action for supersingular elliptic curves, which uses only polynomial quantum space. This improves upon previously-known algorithms which required subexponential quantum space.

Optimization. We develop more efficient algorithms for evaluating the class group action in the context of CSIDH. We consider "strategies"—formerly only considered for Supersingular Isogeny Diffie-Hellman (SIDH)—in the context of CSIDH, and develop systematic methods for optimizing "permutations" of the small primes used in CSIDH, which previously had been treated only in an *ad hoc* fashion. We also develop a systematic technique to optimize the CSIDH keyspace. These optimizations are complementary to prior work on optimizing CSIDH, including improved field arithmetic, Splitting Isogenies into Multiple Batches (SIMBA), and the two-point method. We apply our optimizations to the CSIDH-512 parameter set and give experimental results.

Fault Attacks. We consider physical attacks on static/ephemeral CSIDH in which limited information about which isogenies are "real" and which are "dummy" is revealed. We determine bounds on the number of fault injections required to recover the static secret key, and show that simply reordering the real and dummy isogenies from the ubiquitous "real-then-dummy" ordering to a dynamic random ordering dramatically increases the number of faults required, with negligible impact on the running time of the key exchange protocol (in contrast with prior fault attack countermeasures, which prevent fault attacks entirely at the cost of doubling the running time for key exchange).

Acknowledgements

This thesis was only made possible by the many people who provided their precious help and support.

To begin, I must sincerely thank my supervisors, Professors David Jao and Michele Mosca. David has supervised me from my entire time at the University of Waterloo and has been an invaluable source of advice, encouragement, and mathematical expertise every day. As well, Michele has not only supported my research with this top-notch expertise in quantum algorithms, but has also consistently shown a keen interest in my professional development, which I have always appreciated. This thesis would not have been possible without their support.

I would like to thank my other committee members—Professors Alfred Menezes and Douglas Stebila from the Department of Combinatorics and Optimization, Professor Jerry Wang from the Department of Pure Mathematics, and Dr. Christophe Petit from the University of Birmingham—for their discerning questions and suggestions, and for making my defence so enjoyable. This thesis was certainly made better by their suggestions.

I was extremely fortunate to have the opportunity to spend some time as a visiting researcher at the University of Auckland. I would like to thank Professor Steven Galbraith in particular for supervising me during that time.

I have had the great pleasure of working with a number of excellent colleagues during my time at the University of Waterloo and the University of Auckland, and I would like to thank them all for their hard work and insight. I must give particular thanks to Dr. Aaron Hutchinson, with whom I wrote the bulk of the novel results of this thesis after working closely for the last 18 months of my PhD.

Finally, I would like to thank my family and friends for supporting me throughout my time at the University of Waterloo. I couldn't have done it without you all.

Table of Contents

List of Figures

List of Tables

“Au milieu de l’hiver, j’apprenais enfin qu’il y avait en moi un été invincible.”

– Albert Camus, *Retour à Tipasa*

“In the midst of winter, I finally learned that within me there lay an invincible summer”

– Albert Camus, *Return to Tipasa*

Chapter 1

Introduction

Public-key cryptographic protocols whose security is based on the hardness of factoring or discrete logarithms—including the vast majority of protocols in use today—are susceptible to attacks using quantum algorithms. In order to ensure the continued security of digital communications, we must transition to protocols which cannot be broken by quantum computers, before large-scale quantum computers are readily available. To that end, cryptographers have developed *post-quantum* cryptosystems, which are conjectured to be secure against attacks by quantum computers. At present, the vast majority of post-quantum schemes fit into one of five categories:

1. Code-based cryptography;
2. Lattice-based cryptography;
3. Hash-based cryptography;
4. Multivariate cryptography; and,
5. Isogeny-based cryptography.

The focus of this thesis is isogeny-based cryptography. Isogeny-based protocols are based on the mathematics of elliptic curves: objects which are ubiquitous in *classical* (that is, not post-quantum) cryptography. In contrast with classical elliptic curve cryptography—where, typically, a single elliptic curve is used as a base group over which to instantiate a cryptographic protocol (for instance, Diffie-Hellman [25] or ElGamal [29])—isogeny-based cryptography uses maps *between* elliptic curves with certain algebraic and algebraic-geometric properties: isogenies. In particular, while the underlying hard problem of classical elliptic curve cryptography is typically the discrete logarithm problem (given two points

$P, Q \in E$, find $a \in \mathbb{Z}$ such that $Q = aP$), in isogeny-based cryptography the problem is, given two elliptic curves E and E', to find an isogeny $\phi \colon E \to E'$.

Elliptic curves over finite fields can be divided into two categories: ordinary and supersingular. In classical cryptography, supersingular elliptic curves are undesirable, since on such curves the discrete logarithm problem can be reduced to the discrete logarithm problem in the multiplicative group of a finite field, [73] where non-generic algorithms [70] can be used. In contrast, in isogeny-based cryptography, supersingular elliptic curves are preferred, since for ordinary curves the ring of endomorphisms is commutative, leading to subexponential attacks [16].

At present there are two prominent supersingular isogeny-based protocols: supersingular isogeny Diffie-Hellman (SIDH) [23] and commutative supersingular isogeny Diffie-Hellman (CSIDH) [14]. As their names imply, both are superficially-Diffie-Hellman-like key establishment protocols which use isogenies of supersingular elliptic curves. Both protocols have been the subject of many analyses leading to attacks, optimizations, and improved implementations. In this thesis we add to that body of work, with a particular focus on CSIDH. The thesis is organized as follows:

In Chapter 2 we cover the necessary algebraic-geometric background required to understand the protocols SIDH and CSIDH, which we detail in Chapter 3. In Chapter 4, we extend the results of [16] to the class of isogenies used in CSIDH, and reduce its quantum space complexity from subexponential to polynomial. In Chapter 5 we give some novel methods to optimize implementations of CSIDH, including improved algorithms for performing the fundamental operation (evaluation of the class group action) and for choosing parameter sets. In Chapter 6 we discuss fault attacks on SIDH and CSIDH, and study the efficacy of reordering the isogenies as a countermeasure for a class of attacks on CSIDH. Finally, we consider future work in Chapter 7.

Chapter 2

Algebraic Geometric Background

2.1 Fundamentals of Algebraic Geometry

To begin, we review the fundamental concepts of algebraic geometry and the theory of elliptic curves required for isogeny-based cryptography. The material is this section is taken primarily from [69, Chapters I–III].

2.1.1 Affine Varieties

Definition 2.1 (Affine n-Space). *Affine n-space* over a field K is the set

$$\mathbb{A}^n = \mathbb{A}^n(\overline{K}) = \{P = (x_1, x_2, \ldots, x_n) \,:\, x_i \in \overline{K} \;\; \forall\, 1 \le i \le n\}$$

where $\overline{K}$ is the algebraic closure of K.

The set of *K-rational points* of $\mathbb{A}^n$ is the set

$$\mathbb{A}^n(K) = \{P = (x_1, x_2, \ldots, x_n) \in \mathbb{A}^n \,:\, x_i \in K \;\; \forall\, 1 \le i \le n\}.$$

Denote by $\overline{K}[\mathbf{X}] = \overline{K}[X_1, X_2, \ldots, X_n]$ the polynomial ring over $\overline{K}$ in n indeterminates.

Definition 2.2 (Affine Algebraic set). A set $V \subseteq \mathbb{A}^n(\overline{K})$ is called *(affine) algebraic* if there is an ideal $I \subseteq \overline{K}[\mathbf{X}]$ such that

$$V = V_I := \{P \in \mathbb{A}^n(\overline{K}) \,:\, f(P) = 0 \;\; \forall\, f \in I\}.$$

Definition 2.3 (Ideal of an Affine Algebraic Set)**.** If $V \subseteq \mathbb{A}^n(\overline{K})$ is an affine algebraic set, we define its *ideal* as

$$I(V) = \{f \in \overline{K}[\mathbf{X}] \, : \, f(P) = 0 \;\; \forall \, P \in V\}.$$

Note that $I(V)$ really is an ideal in $\mathbb{A}^n[\overline{K}]$ since if $f(P) = g(P) = 0$ then $(f+g)(P) = 0$, and for any $h \in \overline{K}[\mathbf{X}]$, $(fh)(P) = 0$.

We say that an algebraic set V is *defined over* K if its ideal can be generated by polynomials in $K[\mathbf{X}]$; that is, if

$$\exists \, S \subseteq K[\mathbf{X}] \text{ such that } I(V) = (S);$$

notably, any ideal of $\overline{K}[\mathbf{X}]$ is finitely-generated by Hilbert's Basis Theorem, and so S can always be chosen to be finite.

When $V \subseteq \mathbb{A}^n$ is defined over K, we write V/K, and define the set of K-rational points of V as $V(K) = V \cap \mathbb{A}^n(K)$. We also define the ideal $I(V/K) = I(V) \cap K[\mathbf{X}]$.

Definition 2.4 (Affine Algebraic Variety)**.** We say that an affine algebraic set V is an *(affine) variety* if $I(V)$ is a prime ideal.

Definition 2.5 (Affine Coordinate Ring)**.** Let V/K be a variety. The *affine coordinate ring* of V/K is

$$K[V] = K[\mathbf{X}]/I(V).$$

The ring $K[V]$ is an integral domain, and its quotient field (the *function field of* V/K) is denoted $K(V)$.

Definition 2.6 (Dimension)**.** Let V be a $\overline{K}$-variety. The *dimension* of V—denoted $\dim(V)$—is defined to be the transcendence degree of the extension $\overline{K}(V)/\overline{K}$.

Definition 2.7 (Jacobian Matrix)**.** Let $\mathbf{f} = (f_1, f_2, \ldots, f_r)^T \in \overline{K}[X_1, X_2, \ldots, X_n]^r$ be a vector-valued polynomial function. Its *Jacobian matrix* at a point $P \in \mathbb{A}^n$ is

$$\mathbb{J}(\mathbf{f})(P) = \left[\frac{\partial f_i}{\partial X_j}(P)\right]_{\substack{1 \le i \le r \\ 1 \le j \le n}}.$$

Definition 2.8 (Nonsingular Variety)**.** Let $V \subseteq \mathbb{A}^n$ be a variety with $I(V) = (f_1, f_2, \ldots, f_r)$. Let $\mathbf{f} = (f_1, \ldots, f_r)^T$. We say that V is *nonsingular* (or *smooth*) at a point $P \in V$ if

$$\operatorname{rank}(\mathbb{J}(\mathbf{f})(P)) = n - \dim(V).$$

Notably, if $V \subseteq \mathbb{A}^n$ is a variety whose ideal is generated by a single polynomial f, then $\dim(V) = n-1$ and $\mathbb{J}(\mathbf{f})(P) = \nabla f(P)$ for all $P \in V$. Thus a point $P \in \mathbb{A}^n$ is a singular point of V if and only if

$$\begin{cases} f(P) = 0 \\ \nabla f(P) = \mathbb{0} \end{cases}.$$

Put another way, to check that such V is nonsingular, it suffices to verify that the above system of equations has no solution $P \in \mathbb{A}^n$.

For $P \in V$ define

$$M_P = \{f \in \overline{K}(V)) \,:\, f(P) = 0\}.$$

We note that $\overline{K}[V]/M_P \cong \overline{K}$ (with the isomorphism being the "evaluation-at-P" map) and so M_P is maximal. The quotient M_P/M_P^2 is a finite-dimensional vector space over $\overline{K}$. This maximal ideal can be used to give another characterization of nonsingularity:

Proposition 2.9 ([69, Chapter I, Proposition 1.7])**.** Let V be a $\overline{K}$-variety. A point $P \in V$ is nonsingular if and only if

$$\dim_{\overline{K}} M_P/M_P^2 = \dim(V).$$

Proof. See [36, Section I.5.1] □

Definition 2.10 (Local Ring of a Variety at a Point)**.** The *local ring of* V *at* P—denoted $\overline{K}[V]_P$—is the localization of $\overline{K}[V]$ at M_P; that is

$$\overline{K}[V]_P = \left\{F \in \overline{K}(V) \,:\, \exists\, f, g \in \overline{K}[V] \text{ such that } g(P) \neq 0 \text{ and } F = fg^{-1}\right\}.$$

If $F \in \overline{K}[V]_P$, then $F(P)$ is well-defined; we say that the functions in $\overline{K}[V]_P$ are *regular* or *defined* at P.

2.1.2 Projective Varieties

Definition 2.11 (Projective n-space)**.** *Projective n-space* over a field K is defined as $\mathbb{P}^n = \mathbb{P}^n(\overline{K}) = (\mathbb{A}^{n+1}\backslash\{\mathbb{0}\})/\sim$, where $\sim$ is the equivalence relation

$$(x_0, x_1, \ldots, x_n) \sim (y_0, y_1, \ldots, y_n) \iff \exists\, \lambda \in \overline{K}\backslash\{0\} \text{ such that } \lambda x_i = y_i \text{ for } 1 \leq i \leq n.$$

We use the notation

$$[x_0, x_1, \ldots, x_n] = \left\{(\lambda x_0, \lambda x_1, \ldots, \lambda x_n) \,:\, \lambda \in \overline{K}\backslash\{0\}\right\}.$$

The $x_0, x_1, \ldots, x_n$ are called *homogeneous coordinates* of the point $[x_0, x_1, \ldots, x_n] \in \mathbb{P}^n$.

As in the case of affine varieties, we define the set of K-rational points of $\mathbb{P}^n$ as

$$\mathbb{P}^n(K) = \left\{[x_0, x_1, \ldots, x_n] \in \mathbb{P}^n \,:\, \exists\, \lambda \in \overline{K}\backslash\{0\} \text{ such that } \lambda x_i \in K \;\; \forall\, 1 \le i \le n\right\}.$$

Note that $[x_0, x_1, \ldots, x_n] \in \mathbb{P}^n(K)$ does not imply that each $x_i \in K$; rather, some representative of the congruence class must have all its entries in K. In particular, for any i such that $x_i \neq 0$, we have $\frac{x_j}{x_i} \in K$ for all $1 \le j \le n$.

Definition 2.12 (Minimal Field of Definition)**.** The *minimal field of definition* (over K) for $P = [x_0, x_1, \ldots, x_n] \in \mathbb{P}^n(\overline{K})$ is

$$K(P) = K\left(\frac{x_0}{x_i}, \frac{x_1}{x_i}, \ldots, \frac{x_n}{x_i}\right) \text{ for any } i \text{ such that } x_i \neq 0.$$

Equivalently, $K(P)$ is the smallest extension of K for which $[x_0, x_1, \ldots, x_n] \cap K(P)^{n+1} \neq \varnothing$.

We want to translate the concept of an algebraic set to the projective setting. Since the coordinates of a projective point are defined only up to multiplication by a scalar, we need to consider a class of polynomials whose roots are preserved by maps of the form $\mathbf{X} \mapsto \lambda\mathbf{X}$. The most natural such class are *homogeneous polynomials*:

Definition 2.13 (Homogeneous Polynomial)**.** A polynomial $f \in \overline{K}[\mathbf{X}]$ is called *homogeneous of degree d* if

$$f(\lambda X_0, \lambda X_1, \ldots, \lambda X_n) = \lambda^d f(X_0, X_1, \ldots, X_n) \text{ for all } \lambda \in \overline{K}.$$

If f is homogeneous (of any degree) it makes sense to ask whether $f(P) = 0$ for some $P \in \mathbb{P}^n$, since the choice of homogeneous coordinates for P does not affect the answer.

Definition 2.14 (Projective Algebraic Set)**.** A set $V \subseteq \mathbb{P}^n$ is called *(projective) algebraic* if there exists some $I \subseteq \overline{K}[X_1, X_2, \ldots, X_n]$ which is homogeneous (*i.e.,* it has a generating set all of whose elements are homogeneous) and such that

$$V = V_I = \{P \in \mathbb{P}^n \,:\, f(P) = 0 \text{ for all homogeneous } f \in I\}.$$

Definition 2.15 (Homoegenous Ideal of a Projective Algebraic Set)**.** If V is a projective algebraic set we define its corresponding *(homogeneous) ideal* by

$$I(V) = \left(\left\{f \in \overline{K}[\mathbf{X}] \,:\, f \text{ is homogeneous and } f(P) = 0 \text{ for all } P \in V\right\}\right).$$

As in the affine case, we say that V is defined over K if $I(V)$ can be generated by a set of homogeneous polynomials in $K[\mathbf{X}]$, and in this case we write V/K. When V is defined over K its set of K-*rational points* is

$$V(K) = V \cap \mathbb{P}^n(K).$$

Definition 2.16 (Projective Variety)**.** A projective algebraic set V is called a *(projective) variety* if $I(V)$ is prime.

There are many naturally-isomorphic copies of $\mathbb{A}^n$ in $\mathbb{P}^n$; we describe the most natural such class here. Define the embeddings

$$\begin{aligned} \epsilon_i \colon \mathbb{A}^n &\to \mathbb{P}^n \\ (x_1, x_2, \ldots, x_n) &\mapsto [x_1, x_2, \ldots, x_{i-1}, 1, x_i, \ldots, x_n] \end{aligned}$$

We see that the image of ϵ_i is

$$\text{image}(\epsilon_i) = U_i = \{[x_0, x_1, \ldots, x_n] \,:\, x_i \neq 0\}$$

and, moreover, ϵ_i is injective; hence we can define an inverse on its image:

$$\begin{aligned} \epsilon_i^{-1} \colon U_i &\to \mathbb{A}^n \\ [x_0, x_1, \ldots, x_n] &\mapsto \left(\frac{x_0}{x_i}, \frac{x_1}{x_i}, \ldots, \frac{x_n}{x_i} \right). \end{aligned}$$

We identify $\mathbb{A}^n$ with $U_i \subseteq \mathbb{P}^n$ via ϵ_i when it is convenient. In particular, associated to each projective variety $V \subseteq \mathbb{P}^n$ is an affine variety—denoted $V \cap \mathbb{A}^n$—defined as $V \cap \mathbb{A}^n = \epsilon_i^{-1}(V)$ for a fixed i. Its ideal is

$$I(V \cap \mathbb{A}^n) = \{f(Y_1, Y_2, \ldots, Y_{i-1}, 1, Y_i, \ldots, Y_n) \,:\, f(X_0, X_1, \ldots, X_n) \in I(V)\} \subseteq \overline{K}[\mathbf{Y}]$$

The process of replacing $f(X_0, X_1, \ldots, X_n)$ with $f(Y_1, Y_2, \ldots, Y_{i-1}, 1, Y_i, \ldots, Y_n)$ is called *dehomogenization* with respect to X_i; the reverse process—*homogenization*—is defined as follows: the homogenization of $f(X_1, X_2, \ldots, X_n) \in \overline{K}[\mathbf{X}]$ with respect to X_i is

$$f^*(X_0, X_1, \ldots, X_n) = X_i^d f\left(\frac{X_0}{X_i}, \frac{X_1}{X_i}, \ldots, \frac{X_{i-1}}{X_i}, 1, \frac{X_{i+1}}{X_i}, \ldots, \frac{X_n}{X_i} \right)$$

where $d = \deg f$.

Definition 2.17 (Projective Closure)**.** Let $V \subseteq \mathbb{A}^n$ be an affine algebraic set, considered as a subset of $\mathbb{P}^n$ by applying ϵ_i. The *projective closure* of V—denoted $\overline{V}$—is the projective algebraic set whose homogeneous ideal is

$$I(\overline{V}) = (\{f^*(\mathbf{X}) \,:\, f \in I(V)\})\,.$$

Proposition 2.18 ([69, Chapter I, Proposition 2.6])**.**

(a) If V is an affine variety, then $\overline{V}$ is a projective variety and $V = \overline{V} \cap \mathbb{A}^n$.
(b) If V is a projective variety, either

  i. $V \cap \mathbb{A}^n = \varnothing$; or,
  ii. $V = \overline{V \cap \mathbb{A}^n}$.

(c) If an affine (respectively, projective) variety V is defined over K, then $\overline{V}$ (respectively, $V \cap \mathbb{A}^n$) is also defined over K.

Proof. See [36, Section I.2.3]. □

Proposition 2.18 allows us to identify each affine variety with a unique projective variety, and *vice versa*. It is usually easier to deal with affine coordinates rather than projective ones, and so we will always do so. In particular, when we write "V is a projective variety" defined by an inhomogeneous ideal, what we really mean is that V is the projective closure of the indicated variety W. The points in $V \backslash W$ are called the *points at infinity* of V, where in this last expression we identify W with $\epsilon_i(W)$ for some i.

Now we can define many concepts we defined for affine varieties again for projective varieties, by appealing to this correspondence. In particular:

Definition 2.19 (Dimension of a Projective Variety)**.** Let $V \in \mathbb{P}^n$ be a projective variety, and choose a copy of $\mathbb{A}^n \subseteq \mathbb{P}^n$ such that $V \cap \mathbb{A}^n \neq \varnothing$. The *dimension* of V defined to be

$$\dim(V) = \max_i \, \dim(\epsilon_i^{-1}(V \cap \operatorname{image}(\epsilon_i))).$$

Definition 2.20 (Nonsingular Projective Variety; Local Ring of a Projective Variety; Regular Map)**.** Let $V \subseteq \mathbb{P}^n$ be a projective variety, $P \in V$, and choose a copy of $\mathbb{A}^n \subseteq \mathbb{P}^n$ such that $P \in \mathbb{A}^n$. Then

1. V is *nonsingular* at P if $V \cap \mathbb{A}^n$ is nonsingular at P.
2. The *local ring* of V at P is $\overline{K}[V]_P = \overline{K}[V \cap \mathbb{A}]_P$.
3. A function $F \in \overline{K}(V)$ is *regular* at P if it is in $\overline{K}[V]_P$.

2.1.3 Maps Between Varieties

Definition 2.21 (Rational Map of Projective Varieties)**.** Let $V_1, V_2 \subseteq \mathbb{P}^n$ be varieties. A *rational map* from V_1 to V_2 is a sequence of functions $\phi = [f_0, f_1, \dots, f_n]$ such that

i. $f_0, f_1, \dots, f_n \in \overline{K}(V_1)$; and,
ii. For all $P \in V_1$ at which $f_0, f_1, \dots, f_n$ are defined, $[f_0(P), f_1(P), \dots, f_n(P)] \in V_2$.

We denote $\phi(P) = [f_0(P), f_1(P), \dots, f_n(P)]$.

We say that ϕ is *defined over* K if there is $\lambda \in \overline{K}\backslash\{0\}$ such that $\lambda f_0, \lambda f_1, \dots, \lambda f_n \in K(V_1)$.

Definition 2.22 (Regular Rational Map)**.** A rational map $\phi\colon V_1 \to V_2$ is *regular* (or *defined*) at a point $P \in V_1$ there is a function $g \in \overline{K}(V_1)$ such that

i gf_i is regular at P for $0 \leq i \leq n$; *i.e.*, $gf_i \in \overline{K}[V_1]_P$ for all i; and,
ii There is $0 \leq i \leq n$ such that $(gf_i)(P) \neq 0$.

Remark 2.23. For different $P \in V_1$, we may need different g in the definition of regular.

Definition 2.24 (Morphism)**.** A rational map which is regular everywhere is called a *morphism.*

Definition 2.25 (Isomorphism, Isomorphic Varieties)**.** A morphism $\phi\colon V_1 \to V_2$ is called an *isomorphism* if there is another morphism $\psi\colon V_2 \to V_1$ such that $\phi \circ \psi = \iota_{V_2}$ and $\psi \circ \phi = \iota_{V_1}$ (identity maps).

Two varieties V_1 and V_2 are *isomorphic* if there is a pair of isomorphisms between them.

2.2 Elliptic Curves

We begin by briefly discussing curves in general, and then restrict our attention to elliptic curves in particular.

Definition 2.26 (Curve)**.** A *curve* is a projective variety of dimension 1.

Proposition 2.27 ([69, Chapter II, Proposition 2.1])**.** Let C be a curve, and let $V \subseteq \mathbb{P}^n$ be a variety. Let P be a nonsingular point of C, and let $\phi\colon C \to V$ be a rational map. Then ϕ is defined at P.

In particular, if C is smooth, then ϕ is a morphism.

Proof. See [69, Section II.2]. □

Theorem 2.28 ([69, Theorem 2.3])**.** *Let C_1 and C_2 be smooth curves, and let $\phi\colon C_1 \to C_2$ be a morphism. Then either:*

i. *ϕ is constant; or,*
ii. *$\phi(C_1) = C_2$.*

Proof. See [36, Section II.6.8]. □

Definition 2.29 (Degree of a Morphism of Curves; Separable, Inseparable, and Purely Inseparable Morphisms)**.** Let $\phi\colon C_1 \to C_2$ be a morphism of curves. If ϕ is constant, we define the *degree* of ϕ as $\deg(\phi) = 0$. Otherwise ϕ is surjective, and ϕ induces an injection ϕ^* of function fields fixing K:

$$\begin{aligned}\phi^*\colon K(C_2) &\to K(C_1)\\ f &\mapsto f \circ \phi;\end{aligned}$$

in this case we define $\deg(\phi) = [K(C_1) : \phi^*(K(C_2))]$.

We say that ϕ is separable (respectively, inseparable; purely inseparable) if the extension $K(C_1)/\phi^*(K(C_2))$ is separable (respectively inseparable; purely inseparable).

We are finally equipped to discuss elliptic curves.

Definition 2.30 (Elliptic Curve)**.** An elliptic curve is a pair (E, O) where E is a nonsingular curve of genus[1] 1, and $O \in E$.

We typically omit O when it is understood from context.

We say that E is *defined over* K—written E/K—if E is defined over K as a projective curve, and O is K-rational.

In practice we use a particular representation of an elliptic curve, which the following proposition describes.

[1]Nonsingular curves in $\mathbb{P}^2$ (the only case we are concerned with) are defined by a single homogeneous polynomial $f(X, Y, Z)$. By the Plücker formula [57, Proposition 2.6], the genus of such a curve is $g = \binom{d-1}{2}$ where $d = \deg f$.

Proposition 2.31 (Weierstrass Coordinates, [69, Chapter III, Proposition 3.1])**.**

(a) Let E be an elliptic curve defined over K. There exist functions $x, y \in K(E)$ such that the map

$$\begin{aligned} \phi\colon E &\to \mathbb{P}^2 \\ \mathbf{X} &\mapsto [x(\mathbf{X}), y(\mathbf{X}), 1] \end{aligned}$$

is an isomorphism of E/K onto a curve given by a Weierstrass equation

$$C\colon Y^2Z + a_1XYZ + a_3YZ^2 = X^3 + a_2X^2Z + a_4XZ^2 + a_6Z^3$$

where $a_1, \ldots, a_6 \in K$ and $\phi(O) = [0, 1, 0]$. The functions x and y are called the *Weierstrass Coordinates* of E/K.

(b) Every smooth cubic curve C given by a Weierstrass equation

$$C\colon Y^2Z + a_1XYZ + a_3YZ^2 = X^3 + a_2X^2Z + a_4XZ^2 + a_6Z^3$$

is an elliptic curve defined over K with base point $O = [0, 1, 0]$.

Proof. See [69, Section III.3]. □

If the field of definition K of an elliptic curve has characteristic different from 2 and 3, a change of coordinates allows us to write it in the form

$$E : y^2 = x^3 + ax + b.$$

In this work, we will only work in fields of characteristic strictly larger than 3, and so we will always consider elliptic curves to have a defining equation of this form, which we call *short Weierstrass form.*

Associated to an elliptic curve are three fundamental quantities, which we define here for curves in short Weierstrass form:

1. The *discriminant*: $\Delta = -16(4a^3 + 27b^2)$.
2. The j*-invariant*: $j(E) = \frac{-1728(4a)^3}{\Delta}$.
3. The *invariant differential*: $\omega = \frac{dx}{(2y)}$.

For any elliptic curve E we can define an addition law $+\colon E \times E \to E$ which is a morphism of varieties and under which E and $E(K)$ are abelian groups. For an elliptic curve in short Weierstrass form, the law is particularly easy to write down, and so we present it here.

Definition 2.32 (Addition Law on Elliptic Curves in Short Weierstrass Form)**.** Let E/K be an elliptic curve with defining equation $E\colon y^2 = x^3 + ax + b$. If $P = (x_P, y_P), Q = (x_Q, y_Q) \in E$, we define $P + Q$ in the following way:

(a) If $P = O$, then $P + Q = Q$; otherwise,

(b) If $Q = O$, then $P + Q = P$; otherwise,

(c) If $y_Q = -y_P$, then $P + Q = O$; otherwise,

(d) If $P = Q$, then $P + Q$ is given by

$$\left(\frac{x_P^4 - 2ax_P^2 - 8bx_P + a^2}{4(x_P^2 + ax_P + b)}, \frac{(x_P^6 + 5ax_P^4 + 20bx_P^3 - 5a^2x_P^2 - 4abx_P - a - 8b)y_P}{8(x_P^3 + ax_P + b)^2}\right);$$

otherwise,

(e) If $P \neq Q$, then $P + Q$ is given by

$$\left(\frac{(y_Q - y_P)^2 - x_P^3 + x_P^2x_Q + x_Px_Q^2 - x_Q^3}{(x_Q - x_P)^2}, \frac{x_Py_Q - x_Qy_P + x_{P+Q}(y_P - y_Q)}{x_Q - x_P}\right).$$

2.2.1 Isogenies

Definition 2.33 (Isogeny)**.** Let E_1, E_2 be elliptic curves. An *isogeny* $\phi\colon E_1 \to E_2$ is a morphism satisfying $\phi(O) = O$.

We say that an elliptic curve E_2 is *isogenous* to an elliptic curve E_1 if there is a surjective isogeny $\phi\colon E_1 \to E_2$.

Definition 2.34 (Kernel of an Isogeny)**.** Let $\phi\colon E_1 \to E_2$ be an isogeny. Its *kernel* is

$$\ker(\phi) = \{P \in E_1 \,:\, \phi(P) = O\}.$$

Remark 2.35. It is known (see [69, Chapter III, Proposition 4.10(c)]) that whenever $\phi\colon E_1 \to E_2$ is a non-constant isogeny, $\ker(\phi)$ is a finite subgroup of E_1.

Conversely to Remark 2.35, all finite subgroups of E_1 are the kernels of isogenies. In particular, we have the following proposition:

Proposition 2.36 ([69, Chapter III, Proposition 4.12])**.** Let E be an elliptic curve and let $\Phi \subseteq E$ be a finite subgroup. Then there is a unique elliptic curve E' and a unique isogeny $\phi\colon E \to E'$ (up to isomorphism of isogenies) with $\ker(\phi) = \Phi$. The curve E' is denoted E/Φ.

Remark 2.37. Vélu's formulas [75] describe how to explicitly write the defining equation of E/Φ and the isogeny $\phi\colon E \to E/\Phi$.

We also have the following relationship between the degree of a separable isogeny and the size of its kernel:

Theorem 2.38 ([69, Chapter III, Theorem 4.10])**.** *Let ϕ be a non-constant separable isogeny. Then*

$$|\ker(\phi)| = \deg(\phi).$$

Associated to each non-constant isogeny $\phi\colon E_1 \to E_2$ is a unique second isogeny $\hat{\phi}\colon E_2 \to E_1$ satisfying

$$\hat{\phi} \circ \phi = [\deg \phi]$$

(see [69, Chapter III, Theorem 6.1(a)]), where for each $m \in \mathbb{Z}$, $[m]$ is the "multiplication-by-m" map:

$$\begin{aligned}[m]\colon E_1 &\to E_1\\ P &\mapsto mP\end{aligned}$$

We call $\hat{\phi}$ the *dual isogeny* of ϕ. Notably, the existence of the dual isogeny shows that the property of "being isogenous" is an equivalence relation on the set of elliptic curves.

The dual isogeny has the following properties:

Theorem 2.39 (Properties of the Dual Isogeny, [69, Chapter III, Theorem 6.2])**.** *Let $\phi\colon E_1 \to E_2$ be an isogeny of degree d. Then:*

(a) $\phi \circ \hat{\phi} = [d]$ (on E_2).
(b) If $\lambda\colon E_2 \to E_3$ is an isogeny then $\widehat{\lambda \circ \phi} = \hat{\phi} \circ \hat{\lambda}$.
(c) If $\psi\colon E_1 \to E_2$ is an isogeny then $\widehat{\phi + \psi} = \hat{\phi} + \hat{\psi}$.

(d) $\deg \hat{\phi} = d$.

(e) $\hat{\hat{\phi}} = \phi$.

We present three important examples of isogenies here:

Example 2.40 (Multiplication by m)**.** For an elliptic curve $E\colon y^2 = x^3 + ax + b$ defined over K, the multiplication-by-m map $[m]$ is an isogeny. We use properties of this isogeny to determine the isomorphism class of the m-torsion subgroup of E for m coprime to $\mathrm{char}(K)$:

$$E[m] = \ker([m]) = \{P \in E \,:\, mP = O\};$$

This group plays an important role in isogeny-based cryptography.

Note first that $\deg([0]) = 0$ by definition, and it is clear that $\deg([1]) = 1$. Assume for induction that for some $m \in \mathbb{Z}$, $\widehat{[m]} = [m]$; then by Theorem 2.39(c), for any $m \in \mathbb{Z}$ we have

$$\widehat{[m+1]} = \widehat{[m]} + \widehat{[1]} = [m] + [1] = [m+1]$$

and so by induction, $\widehat{[m]} = [m]$ for all $m \in \mathbb{Z}$. Then

$$[\deg[m]] = [m] \circ [m] = [m^2]$$

and so $[m^2 - \deg[m]]P = O$ for all $P \in E$. It can be shown (see [69, Chapter III, Proposition 4.2(b)]) that the only constant multiplication map on an elliptic curve is $[0]$, and so we must have $\deg[m] = m^2$.

Now, the assumption that $p \nmid m$ implies that $[m]$ is separable, and so $|E[m]| = \deg[m] = m^2$. We will show that

$$E[m] \cong \mathbb{Z}/m\mathbb{Z} \oplus \mathbb{Z}/m\mathbb{Z}.$$

For prime q, $|E[q]| = q^2$. Now, the only abelian groups of order q^2 are

$$\mathbb{Z}/q^2\mathbb{Z} \text{ and } \mathbb{Z}/q\mathbb{Z} \oplus \mathbb{Z}/q\mathbb{Z},$$

and clearly $E[q] \ncong \mathbb{Z}/q^2\mathbb{Z}$, since that group contains elements of order $q^2 > q$. So for primes q which are different from p, $E[q] \cong \mathbb{Z}/q\mathbb{Z} \oplus \mathbb{Z}/q\mathbb{Z}$.

Suppose for induction that for some $n \in \mathbb{N}$, $E[q^n] \cong \mathbb{Z}/q^n\mathbb{Z} \oplus \mathbb{Z}/q^n\mathbb{Z}$. Then $E[q^n] \subseteq E[q^{n+1}]$, and so $E[q^{n+1}]$ must be isomorphic to one of

i. $\mathbb{Z}/q^n\mathbb{Z} \oplus \mathbb{Z}/q^n\mathbb{Z} \oplus \mathbb{Z}/q\mathbb{Z} \oplus \mathbb{Z}/q\mathbb{Z}$,

ii. $\mathbb{Z}/q^n\mathbb{Z} \oplus \mathbb{Z}/q^n\mathbb{Z} \oplus \mathbb{Z}/q^2\mathbb{Z}$,
iii. $\mathbb{Z}/q^{n+1}\mathbb{Z} \oplus \mathbb{Z}/q^n\mathbb{Z} \oplus \mathbb{Z}/q\mathbb{Z}$, or
iv. $\mathbb{Z}/q^{n+1}\mathbb{Z} \oplus \mathbb{Z}/q^{n+1}\mathbb{Z}$.

In the first two cases, all elements of the groups have order dividing q^n, and so we would have $\mathbb{Z}/q^n\mathbb{Z} \oplus \mathbb{Z}/q^n\mathbb{Z} \oplus \mathbb{Z}/q\mathbb{Z} \oplus \mathbb{Z}/q\mathbb{Z} \subseteq E[q^n]$ or $\mathbb{Z}/q^n\mathbb{Z} \oplus \mathbb{Z}/q^n\mathbb{Z} \oplus \mathbb{Z}/q^2\mathbb{Z} \subseteq E[q^n]$, which is impossible because we know the group structure of $E[q^n]$. The third case is impossible because $\mathbb{Z}/q^{n+1}\mathbb{Z} \oplus \mathbb{Z}/q^n\mathbb{Z} \oplus \mathbb{Z}/q\mathbb{Z}$ contains q^3 elements of order q, but $|E[q]| = q^2$. Thus we must have

$$E[q^{n+1}] \cong \mathbb{Z}/q^{n+1}\mathbb{Z} \oplus \mathbb{Z}/q^{n+1}\mathbb{Z}$$

and so, by induction, $E[q^n] \cong \mathbb{Z}/q^n\mathbb{Z} \oplus \mathbb{Z}/q^n\mathbb{Z}$ for all $n \in \mathbb{N}$.

From here, it is easy to see that for $m = q_1^{n_1} q_2^{n_2} \ldots q_t^{n_t}$, with $p \nmid m$ we must have

$$E[m] \cong \bigoplus_{j=1}^{t} \left(\mathbb{Z}/q_j^{n_j}\mathbb{Z} \oplus \mathbb{Z}/q_j^{n_j}\mathbb{Z}\right) \cong \mathbb{Z}/m\mathbb{Z} \oplus \mathbb{Z}/m\mathbb{Z}.$$

Example 2.41 (The Frobenius Map)**.** Let $E\colon y = x^3 + ax + b$ be defined over $GF(p^n)$ for some prime p and $n \in \mathbb{N}$. The p^n-power Frobenius map is

$$\begin{aligned} \pi_{p^n}\colon E &\to E \\ (x, y) &\mapsto (x^{p^n}, y^{p^n}). \end{aligned}$$

It is clear that $\ker(\pi_{p^n}) = \{O\}$, and so $|\ker(\pi_{p^n})| = 1$. But $\deg(\pi_{p^n}) = p^n$, and so π_{p^n} must be inseparable.

Example 2.42 (Complex Multiplication by $\mathbb{Z}[\zeta_3]$)**.** Let $E\colon y^2 = x^3 + 1$ be defined over a field K of characteristic $p > 2$ which contains a primitive cube root of unity; that is

$$\exists\, \gamma \in K \text{ such that } \gamma^3 = 1 \text{ and } \gamma \neq 1.$$

Then the map $\phi\colon (x, y) \mapsto (\gamma x, y)$ maps E to E, since $(\gamma x)^3 + 1 = \gamma^3 x^3 + 1 = x^3 + 1 = y^2$. Moreover, $\phi(O) = O$, and so ϕ is an isogeny.

This map satisfies $\phi^3 = \iota_E$ and $\phi \neq \iota_E$, and can be thought of a way to extend the map $m \mapsto [m]$ from $\mathbb{Z}$ to the Eisenstein integers $\mathbb{Z}[\zeta_3]$, where $\zeta_3 = e^{\frac{2\pi i}{3}}$.

The following theorem tells us when there is an isogeny between two given curves defined over a given finite field.

Theorem 2.43 ([72, Section 3])**.** *Let E_1 and E_2 be elliptic curves defined over $GF(q)$. Then there exists an isogeny $\phi\colon E_1 \to E_2$ defined over $GF(q)$ if and only if*

$$|E_1(GF(q))| = |E_2(GF(q))|.$$

2.2.2 The Endomorphism Ring

The set of isogenies from an elliptic curve to itself is a ring under the operations of pointwise addition and function composition. We denote

$$\mathrm{End}(E) = \{\phi\colon E \to E \,:\, \phi \text{ is an isogeny}\};$$

this ring is called the *endomorphism ring* of E. We can consider the subring of isogenies which are defined over K; we denote this by

$$\mathrm{End}_K(E) = \{\phi\colon E \to E \,:\, \phi \text{ is an isogeny defined over } K\}.$$

As mentioned earlier, for any elliptic curve E, $\mathrm{End}(E)$ contains a copy of $\mathbb{Z}$ from the embedding $m \mapsto [m]$. In the case of elliptic curves defined over finite fields, we know more:

Theorem 2.44 (Rank of the Endomorphism Ring, [69, Chapter V, Theorem 3.1])**.** *Let E be an elliptic curve defined over $GF(q)$ for some prime power q. Then, as a $\mathbb{Z}$-module,* $\mathrm{End}(E)$ *has dimension* 2 *or* 4.

Proof. See [69, Section V.3]. □

We say that an elliptic curve defined over $GF(q)$ is *ordinary* if $\dim_{\mathbb{Z}} \mathrm{End}(E) = 2$ and that it is *supersingular* if $\dim_{\mathbb{Z}} \mathrm{End}(E) = 4$.

Traditional applications of elliptic curves in cryptography have been based on the hardness of the discrete logarithm problem. On supersingular elliptic curves, this problem reduces in probabilistic polynomial time to a discrete logarithm problem in the multiplicative group of a finite field [55, Sections 3–4], where non-generic algorithms can be used. Thus in this context ordinary curves are more secure than supersingular ones. On the contrary, in isogeny-based cryptography an attack due to Childs, Jao, and Soukharev [16] demonstrates that against quantum adversaries, ordinary curves offer only subexponential (rather than fully-exponential) security, and so supersingular elliptic curves are the standard. The algorithm proposed in [16] unfortunately requires subexponential quantum space; in Chapter 4 we present a generalization of their algorithm which works for a certain kind of isogeny on supersingular curves and which requires only polynomial quantum space.

The structure of the endomorphism rings of elliptic curves play a vital role in the security of isogeny-based key establishment protocols. In the following two sections we explain the structure of these endomorphism rings.

Structure of the Endomorphism Ring of an Ordinary Elliptic Curve

This material appears in [22, Sections 5.A and 7.A].

Definition 2.45 (Algebraic Number Field)**.** An *algebraic number field* K is an algebraic extension of $\mathbb{Q}$ of finite degree.

We say that K is a *quadratic* number field if $[K\colon \mathbb{Q}] = 2$; in this case K can be written as $K = \mathbb{Q}(\sqrt{N})$ for some square-free integer N. If $N > 0$ we say that K is a *real* quadratic field, and we say that it is *imaginary* otherwise.

Define the *discriminant* d_K of K as

$$d_K = \begin{cases} N & \text{if } N \equiv 1 \pmod 4 \\ 4N & \text{otherwise} \end{cases}.$$

Then d_K is congruent either to 0 or 1 (mod 4), and K can be written $K = \mathbb{Q}(\sqrt{d_K})$; that is, a quadratic field is determined uniquely by its discriminant.

The *ring of integers* $\mathcal{O}_K$ of $K = \mathbb{Q}(\sqrt{N})$ is

$$\mathcal{O}_K = \mathbb{Z}\left[\frac{d_K + \sqrt{d_K}}{2}\right] = \begin{cases} \mathbb{Z}[\sqrt{N}] & \text{if } N \not\equiv 1 \pmod 4 \\ \mathbb{Z}\left[\frac{1+\sqrt{N}}{2}\right] & \text{otherwise} \end{cases}.$$

Definition 2.46 (Order in a Quadratic Number Field)**.** An *order* $\mathcal{O}$ in a quadratic number field K is a subset satisfying:

1. $\mathcal{O}$ is a unital subring of K;
2. As a $\mathbb{Z}$ module, $\dim_{\mathbb{Z}}(\mathcal{O})$ is finite; and,
3. $\mathcal{O}$ contains a $\mathbb{Q}$-basis of K.

We note that

1. $\mathcal{O}$ is torsion-free (since K is torsion-free);
2. $\mathcal{O}$ is a free $\mathbb{Z}$-module of rank 2; and,
3. $K = \mathrm{Quot}(\mathcal{O})$.

Any order $\mathcal{O}$ of K is contained in $\mathcal{O}_K$, and so $\mathcal{O}_K$ is maximal. We can describe the structure of $\mathcal{O}$ in terms of $c = [\mathcal{O}_K : \mathcal{O}]$:

Lemma 2.47 ([22, Lemma 7.2])**.** Let $\mathcal{O}$ be an order in a quadratic field K of discriminant d_K, and let $c = [\mathcal{O}_K : \mathcal{O}]$. Then:

i. c is finite; and,
ii. $\mathcal{O} = \mathbb{Z} + c\mathcal{O}_K = \mathbb{Z}\left[c\frac{d_K+\sqrt{d_K}}{2}\right]$

We call $c = [\mathcal{O}_K : \mathcal{O}]$ the *conductor* of $\mathcal{O}$. Another important invariant of an order in a quadratic field is the *discriminant*, defined as follows. If $\mathcal{O} = \langle \alpha, \beta \rangle$, then its discriminant is

$$\Delta_{\mathcal{O}} = \begin{vmatrix} \alpha & \beta \\ \overline{\alpha} & \overline{\beta} \end{vmatrix}^2$$

where $\alpha \to \overline{\alpha}$ is the nontrivial automorphism of K fixing $\mathbb{Q}$. This expression for Δ is independent of the basis (α, β) chosen for $\mathcal{O}$; choosing the particular basis $(1, c\frac{d_K+\sqrt{d_K}}{2})$ gives

$$\Delta_{\mathcal{O}} = \left(\frac{c}{2}\right)^2 \begin{vmatrix} 1 & d_K + \sqrt{d_K} \\ 1 & d_K - \sqrt{d_K} \end{vmatrix}^2 = c^2 d_K$$

Since $c \in \mathbb{Z}$ and $K = \mathbb{Q}(\sqrt{d_K})$ we see that $K = \mathbb{Q}(\sqrt{\Delta_{\mathcal{O}}})$ as well for any order $\mathcal{O}$ of K. Moreover, $\Delta_{\mathcal{O}} > 0$ if K is real, and $\Delta_{\mathcal{O}} < 0$ if K is imaginary. In fact, $\Delta_{\mathcal{O}}$ determines $\mathcal{O}$ uniquely, and for any $\Delta \equiv 0, 1 \pmod 4$ there is an order $\mathcal{O}_\Delta \subset K$ with $\Delta_{\mathcal{O}_\Delta} = \Delta$.

We are finally prepared to described the endomorphism ring of an ordinary elliptic curve.

Theorem 2.48 (Structure of Ordinary Endomorphism Rings [69, Theorem V.3.1])**.** *Let E be an ordinary elliptic curve defined over $GF(q)$. Then there exists an imaginary quadratic field K and an integer $\Delta < 0$ such that*

$$\mathrm{End}(E) \cong \mathcal{O}_\Delta \subseteq K.$$

Structure of the Endomorphism Ring of a Supersingular Elliptic Curve

We first require two definitions.

Definition 2.49 (Quaternion Algebra)**.** A quaternion algebra is a $\mathbb{Q}$-algebra of the form

$$\mathcal{A} = \mathbb{Q} + \alpha\mathbb{Q} + \beta\mathbb{Q} + \alpha\beta\mathbb{Q}$$

where $\alpha, \beta \in \mathcal{A}$ satisfy $\alpha^2, \beta^2 \in \mathbb{Q}$, $\alpha^2, \beta^2 < 0$, and $\alpha\beta = -\beta\alpha$.

Definition 2.50 (Order in an Algebra)**.** An *order* $\mathcal{O}$ in a $\mathbb{Q}$-algebra $\mathcal{A}$ is a subset satisfying:

1. $\mathcal{O}$ is a unital subring of $\mathcal{A}$;
2. As a $\mathbb{Z}$-module, $\mathcal{O}$ is finite-dimensional; and,
3. $\mathcal{O}$ contains a $\mathbb{Q}$-basis of $\mathcal{A}$.

We have the following result characterizing the structure of endomorphism rings of supersingular elliptic curves:

Theorem 2.51 (Structure of Supersingular Endomorphism Rings [69, Corollary III.9.4])**.** *Let E be a supersingular elliptic curve defined over a finite field. Then there exists a quaternion algebra $\mathcal{A}$ and an order $\mathcal{O} \subseteq \mathcal{A}$ such that* $\mathrm{End}(E) \cong \mathcal{O}$*.*

Theorems 2.48 and 2.51 together tell us that $\mathrm{End}(E)$ is a commutative ring if and only if E is ordinary. This commutativity enables the construction of Couveignes and Rostovstev-Stolbunov [21, 66] (which use ordinary curves). When we restrict to $GF(p)$-rational endomorphisms of supersingular elliptic curves defined over $GF(p)$, however, we have the following result:

Theorem 2.52 (Supersingular $GF(p)$-rational Endomorphism Rings [24, Theorem 2.1])**.** *Let $E/GF(p)$ be supersingular. Then*

$$\mathrm{End}_{GF(p)}(E) \cong \mathbb{Z}\left[\sqrt{-p}\right];$$

in particular, $\mathrm{End}_{GF(p)}(E)$ *is an order in* $\mathbb{Q}[\sqrt{-p}]$ *and is thus commutative.*

Applying Theorem 2.52, Castryck, Lange, Martindale, Panny, and Renes construct a supersingular analogue of the scheme of Couveignes over supersingular curves, called CSIDH [14]. We detail this construction in Chapter 3.

2.2.3 The Ideal Class Group

CSIDH is built upon the action of a group—the ideal class group—on certain sets of elliptic curves. In this section we cover the necessary background to describe the scheme.

Before getting to the definition of ideal class group, we state the following lemma

Lemma 2.53 ([14, Lemma 6])**.** Let $E/GF(p)$ be an elliptic curve and $G \leq E$ be finite and $GF(p)$-rational. Then, up to $GF(p)$-isomorphism there exist exactly one curve $E'/GF(p)$ and separable isogeny $\psi\colon E \to E'$ defined over $GF(p)$ with $\ker \psi = G$.

Let $\mathcal{O}$ be an order in a quadratic number field K. If $\mathfrak{a} \subseteq \mathcal{O}$ is an ideal then it is of finite index, and we define its norm to be $N(\mathfrak{a}) = |\mathcal{O}/\mathfrak{a}|$. For any such ideal we necessarily have

$$\mathcal{O} \subseteq \{\beta \in K \,:\, \beta\mathfrak{a} \subseteq \mathfrak{a}\}$$

since $\mathfrak{a}$ is an ideal in $\mathcal{O}$. We say that $\mathfrak{a}$ is a *proper* ideal of $\mathcal{O}$ if

$$\mathcal{O} = \{\beta \in K \,:\, \beta\mathfrak{a} \subseteq \mathfrak{a}\}$$

Notably:

1. Any principal ideal is proper; and,
2. All ideals of $\mathcal{O}_K$ are proper.

This terminology extends to *fractional ideals*: sets of the form $\alpha\mathfrak{a}$ where $\alpha \in K\backslash\{0\}$. In particular, we call a fractional ideal $\mathfrak{b} \subseteq \mathcal{O}$ *proper* if

$$\mathcal{O} = \{\beta \in K \,:\, \beta\mathfrak{b} \subseteq \mathfrak{b}\}.$$

We call a fractional ideal $\mathfrak{b}$ *invertible* if there is another fractional ideal $\mathfrak{a}$ such that $\mathfrak{a}\mathfrak{b} = \mathcal{O}$. We say that a fractional ideal is *principal* if it takes the form $\mathfrak{b} = \beta\mathcal{O}$ for some $\beta \in K\backslash\{0\}$. Clearly all principal fractional ideal are invertible (take $\mathfrak{a} = \beta^{-1}\mathcal{O}$.). We have the following result related properness and invertibility:

Proposition 2.54 ([22, Proposition 7.4])**.** Let $\mathcal{O}$ be an order in a quadratic number field K, and let $\mathfrak{a}$ be a fractional $\mathcal{O}$-ideal. Then $\mathfrak{a}$ is proper if and only if it is invertible.

Let $I(\mathcal{O})$ denote the set of proper fractional ideals of $\mathcal{O}$. By Proposition 2.54, $I(\mathcal{O})$ is a group under ideal multiplication. The set $P(\mathcal{O})$ of principal ideals of $\mathcal{O}$ forms a subgroup of $I(\mathcal{O})$, and we define the *ideal class group* of $\mathcal{O}$ as

$$\mathrm{cl}(\mathcal{O}) = I(\mathcal{O})/P(\mathcal{O}).$$

It is known that each ideal class $[\mathfrak{a}] \in \mathrm{cl}(\mathcal{O})$ has an integral representative (*i.e.,* a representative $\mathfrak{a}$ which is truly an ideal in $\mathcal{O}$, rather than a fractional ideal); moreover, for each $M \in \mathbb{Z}$ there is such a representative with norm coprime to M ([22, Corollary 7.7]). This is a useful result since ideals of norm coprime to the conductor are invertible and factor uniquely into prime ideals [22, Proposition 7.20 and Exercise 7.26].

The following is taken from [14, Section 3]. Fix a prime $p \geq 5$ and an elliptic curve $E/GF(p)$. The Frobenius endomorphism satisfies

$$\pi_p^2 - t\pi_p + p = 0$$

where $t \in \mathbb{Z}$ is the trace of π_p. The curve E is supersingular if and only if $t = 0$ [69, Exercise 5.10(a)], in which case we obviously have $\pi_p^2 = -p$. Since $\pi_p \in \mathrm{End}_{GF(p)}(E) \cong \mathcal{O}$, we have $\mathbb{Z}[\pi_p] \subseteq \mathcal{O}$, since $\mathcal{O}$ is an order.

Any invertible ideal $\mathfrak{a} \subseteq \mathcal{O}$ factors as $\mathfrak{a} = (\pi_p\mathcal{O})^r\mathfrak{a}_s$ for some $r \in \mathbb{Z}$ and $\mathfrak{a}_s \not\subseteq \pi_p\mathcal{O}$. This allows us to define an isogeny

$$\phi_{\mathfrak{a}} \colon E \to E/\mathfrak{a}$$

of degree $N(\mathfrak{a})$ as $\phi_{\mathfrak{a}}^{(s)} \circ \pi_p^r$, where $\phi_{\mathfrak{a}}^{(s)}$ (the separable part of $\phi_{\mathfrak{a}}$) has kernel $\bigcap_{\alpha\in\mathfrak{a}_s} \ker\alpha$. It is known [76, Theorem 3.11] that if $\mathfrak{a}$ is a principal ideal then $\phi_{\mathfrak{a}}$ is an endomorphism, and so this construction defines an action of the ideal class group $\mathrm{cl}(\mathcal{O})$ on the set $\mathcal{E}\ell\ell_p(\mathcal{O})$of elliptic curves with $GF(p)$-rational endomorphism ring isomorphic to $\mathcal{O}$:

$$\begin{aligned} \mathrm{cl}(\mathcal{O}) &\circlearrowright \mathcal{E}\ell\ell_p(\mathcal{O}) \\ ([\mathfrak{a}], E) &\mapsto E/\mathfrak{a} =: [\mathfrak{a}] * E \end{aligned}$$

where $\mathfrak{a}$ is an integral representative of $[\mathfrak{a}]$. Moreover, this action is free and transitive.

Chapter 3

Isogeny-Based Key Establishment

Chapters 4, 5 and 6 build on and discuss previously-constructed isogeny-based key establishment protocols: Supersingular Isogeny Diffie-Hellman (SIDH) and Commutative Supersingular Isogeny Diffie-Hellman (CSIDH). We describe these protocols here, and discuss the computational problems that underlie their security.

3.1 Supersingular Isogeny Diffie-Hellman

Supersingular Isogeny Diffie-Hellman (SIDH) was introduced by De Feo, Jao, and Plût in 2011 [23]. Superficially the protocol resembles the classical Diffie-Hellman protocol [25], with the base group replaced by a set of elliptic curves, and the group operation replaced with isogeny codomain construction. We detail the protocol here and then discuss the computational problems that underlie its security.

3.1.1 The Protocol

The protocol description is as follows:

Setup: We require the following global parameters:

1. A prime $p = \ell_A^{e_A}\ell_B^{e_B}f \pm 1$ where ℓ_A and ℓ_B are small primes, and $\ell_A^{e_A} \approx \ell_B^{e_B}$;
2. A supersingular elliptic curve $E/GF(p^2)$; and,

3. Four points $P_A, P_B, Q_A, Q_B \in E(GF(p^2))$ such that $E[\ell_A^{e_A}] = \langle P_A, Q_A \rangle$ and $E[\ell_B^{e_B}] = \langle P_B, Q_B \rangle$.

One party (Alice) will use the $\ell_A^{e_A}$-torsion subgroup, and the other (Bob) will use the $\ell_B^{e_B}$-torsion subgroup.

Key Generation: Alice:

1. Selects $\alpha \in \mathbb{Z}/\ell_A^{e_A}\mathbb{Z}$ uniformly at random;
2. Constructs the isogeny $\phi_A \colon E \to E_A = E/\langle P_A + \alpha Q_A \rangle$; and,
3. Constructs the auxiliary points $R_A = \phi_A(P_B)$ and $S_A = \phi_A(Q_B)$.

Alice's private/public keypair is

$$\mathrm{sk}_A = \alpha \text{ and } \mathrm{pk}_A = (E_A, R_A, S_A).$$

Bob proceeds analogously.

Communication: The parties exchange their public keys.

Key Establishment: Alice computes

$$K_A = j\left(E_B/\langle R_B + \alpha S_B \rangle\right)$$

Bob proceeds analogously to find his key K_B. We have $K_A = K_B$.

The protocol is depicted in Figure 3.1.

3.1.2 Computational Problems for SIDH

As with all public-key cryptography, the security of SIDH is predicated on the difficulty of certain computational problems. As is typical, these problems have decisional and computational variants; we present these problems here.

Problem 3.1 (Supersingular Isogeny Problem)**.** Let $\phi_A \colon E \to E_A$ be an isogeny with kernel $\langle P_A + \alpha Q_A \rangle$ where α is chosen uniformly at random from $\mathbb{Z}/\ell_A^{e_A}\mathbb{Z}$. The *supersingular isogeny problem (SSI)* is, given E, E_A, $\phi_A(P_B)$, and $\phi_A(Q_B)$, to find a generator of $\ker \phi_A$.

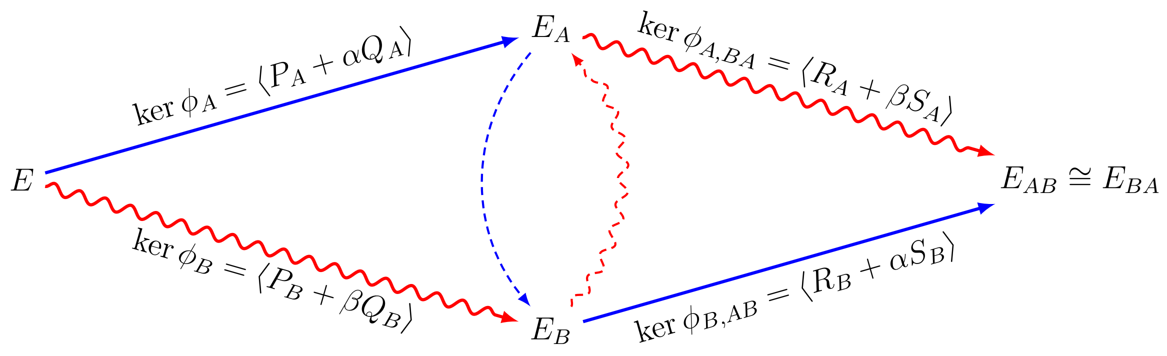

E
E_A
E_B
$E_{AB} \cong E_{BA}$
$\ker \phi_A = \langle P_A + \alpha Q_A \rangle$
$\ker \phi_B = \langle P_B + \beta Q_B \rangle$
$\ker \phi_{A,BA} = \langle R_A + \beta S_A \rangle$
$\ker \phi_{B,AB} = \langle R_B + \alpha S_B \rangle$

Figure 3.1: A depiction of the computations involved in SIDH. Alice follows the solid, plain blue arrows by finding the codomain curve of the indicated isogeny, and follows the dashed, plain blue arrow by reading the message she receives from Bob. Bob analogously follows the wavy red arrows.

The supersingular isogeny problem can be thought of as a supersingular isogeny analogue of the classical discrete logarithm problem, in the sense that it corresponds to key recovery in the SIDH scheme. In order to prove security of SIDH key establishment, we also require supersingular isogeny analogues of the computational and decisional Diffie-Hellman problems.

Problem 3.2 (Supersingular Computational Diffie-Hellman Problem)**.** Let $\phi_A\colon E \to E_A$ be an isogeny with kernel $\langle P_A + \alpha Q_A \rangle$ where α is chosen uniformly at random from $\mathbb{Z}/\ell_A^{e_A}\mathbb{Z}$. Similarly, let $\phi_B\colon E \to E_B$ be an isogeny with kernel $\langle P_B + \beta Q_B \rangle$ where β is chosen uniformly at random from $\mathbb{Z}/\ell_B^{e_B}\mathbb{Z}$. The *supersingular computational Diffie-Hellman problem (SSCDH)* is to find the j-invariant of

$$E_{AB} = E/\langle P_A + \alpha Q_A, P_B + \beta Q_B \rangle$$

given E, E_A, E_B, $\phi_A(P_B)$, $\phi_A(Q_B)$, $\phi_B(P_A)$, and $\phi_B(Q_A)$.

Problem 3.3 (Supersingular Decisional Diffie-Hellman Problem)**.** Let $\phi_A\colon E \to E_A$ be an isogeny with kernel $\langle P_A + \alpha Q_A \rangle$ where α is chosen uniformly at random from $\mathbb{Z}/\ell_A^{e_A}\mathbb{Z}$. Similarly, let $\phi_B\colon E \to E_B$ be an isogeny with kernel $\langle P_B + \beta Q_B \rangle$ where β is chosen uniformly at random from $\mathbb{Z}/\ell_B^{e_B}\mathbb{Z}$. Given a tuple

$$(E, E_A, E_B, \phi_A(P_B), \phi_A(Q_B), \phi_B(P_A), \phi_B(Q_A), E_C)$$

where either $E_C = E_{AB} = E/\langle P_A + \alpha Q_A, P_B + \beta Q_B \rangle$ or E_C is sampled uniformly at random from the set of all curves of the form

$$E/\langle P_A + y_A Q_A, P_B + y_B Q_B\rangle$$

where y_A and y_B are chosen with the same conditions as α and β, respectively, each with probability $\frac{1}{2}$, the *supersingular decisional Diffie-Hellman problem (SSDDH)* is to determine which is the case.

The security of SIDH is directly based on the assumed hardness of SSDDH; in particular, we have the following result:

Theorem 3.4 (Security of SIDH [23, Theorem 6.1])**.** *Under the assumption that the SSDDH problem cannot be solved in polynomial time, the SIDH key establishment protocol is secure in the authenticated links security model of Canetti and Krawczyk [13].*

Remark 3.5 (On the Security of SIKE)**.** While the security of SIDH relies on the SSDDH assumption, SIKE—the NIST post-quantum standardization candidate which is built upon SIDH—[2] requires only the weaker SSCDH assumption.

Hardness of Isogeny Problems Related to SIDH

To begin, we have the following polynomial-time reductions between the SSI, SSCDH, and SSDDH problems:

$$\text{SSI} \geq_P \text{SSCDH} \geq_P \text{SSDDH}.$$

Thus the assumption that SSDDH is a difficult problem (required for the security proof of Theorem 3.4) also requires that SSCDH and SSI also be hard problems. There are no known reductions in the opposite direction; however, the fastest known quantum algorithms for solving the SSDDH problem work by solving the underlying instances of SSI. The fastest known quantum algorithms for SSDDH run in fully-exponential time $\Theta(\sqrt[6]{p})$ [71], thus these isogeny problems seem well-suited to being cryptographic primitives. Moreover, recent analysis due to Jaques and Schnack [42] suggests that the classical control resources required to execute the claw-finding attack of [71] would make the parameter choices that allow it to achieve time $\Theta(\sqrt[6]{p})$ infeasible; instead, a feasible parameter choice or a different attack (such as that of [9]) should be considered—these alternatives have even larger time complexity of $\Theta(\sqrt[4]{p})$.

So far, algorithms for the SSDDH, SSCDH, and SSI problems have not made use of the auxiliary points $\phi_A(P_B), \phi_A(Q_B), \phi_B(P_A), \phi_B(Q_A)$. Since $E[\ell_A^{e_A}] = \langle P_A, Q_A\rangle$ and $E[\ell_B^{e_B}] = \langle P_B, Q_B\rangle$, any $\ell_A^{e_A}$-torsion point X can be written as $X = m_A P_A + n_A Q_A$ for some

$m_A, n_A \in \mathbb{Z}/\ell_A^{e_A}\mathbb{Z}$; moreover, since extended discrete logarithms are easy on supersingular elliptic curves [73, 70], these coefficients m_A, n_A are easy to compute; thus the auxiliary points give an adversary enough information to compute the image of any point in $E[\ell_A^{e_A}]$ under ϕ_B, and the image of any point in $E[\ell_B^{e_B}]$ under ϕ_A. While it is known (*e.g.,* [74]) that the images of random points of E under ϕ_A (respectively, ϕ_B) can be used to recover its kernel, it is not clear how the images of $\ell_B^{e_B}$-torsion (respectively, $\ell_A^{e_A}$-torsion) points can be used, since none of these points (other than the point at infinity) are annihilated by ϕ_A (respectively, ϕ_B)—in particular, "hidden subgroup" techniques (such as those of [47, 65]) cannot be used in this setting. However, in the static/ephemeral setting (where one party's key pair is fixed), active attacks have used invalid auxiliary points to recover the static secret key [31, 26], rendering static/ephemeral SIDH insecure. We discuss these attacks in Chapter 6.

Though the auxiliary points have not successfully been used to attack SIDH, they have been shown to be useful to attack protocols that resemble SIDH but use parameters which are, in a certain sense, "unbalanced." To discuss these attacks, we must consider the following generalization of Problem 3.1.

Problem 3.6 (Generalized Supersingular Isogeny Problem [49, Problem 1])**.** Given a prime p, smooth coprime integers A and B, two supersingular elliptic curves $E_0/GF(p^2)$ and $E/GF(p^2)$ connected by an A-isogeny ϕ, and the action of ϕ on $E_0[B]$, recover ϕ.

When $A = \ell_A^{e_A}$, $B = \ell_B^{e_B}$ for small primes ℓ_A, ℓ_B with $A \approx B$ and $p = AB \cdot f - 1$, Problem 3.6 is precisely Problem 3.1.

In [63], Petit is establishes (among other things) that, under certain heuristic assumptions, in the case that $B > A^4 > p^4$ and $j(E_0) = 1728$ (which was originally used as the base curve in SIKE, although the current version uses a different based curve which is 2-isogenous to that curve; this does not have any affect on the asymptotic complexity of the attack) there is a polynomial-time algorithm to solve Problem 3.6. In [49] this attack is extended to the cases $B > A^2 > p^2$ or $B > A^3 > p^{\frac{3}{2}}$. We explain the basic structure of the attack here; this material is based on [49, Section 2.3].

There are three main steps of the attack of [63]:

1. Compute a non-scalar $\theta \in \mathrm{End}(E_0)$ such that there exist $d, e \in \mathbb{Z}$ such that

$$\deg(\phi \circ \theta \circ \hat{\phi} + [d]) = Be$$

for e smooth and small.

2. Compute $\tau = \phi \circ \theta \circ \hat{\phi} + [d]$ using the (known) action of $\phi|_{E_0[B]}$.

3. Compute $\ker(\tau - [d]) \cap E[A]$, and hence compute ϕ.

By considering the endomorphisms $\pi\colon (x, y) \mapsto (x^p, y^p)$ and $\iota\colon (x, y) \mapsto (-x, \sqrt{-1}y)$, finding a θ as required in step 1 can be reduced to solving a Diophantine equation

$$A^2(pa^2 + pb^2 + c^2) = d^2 = Be$$

for a, b, c; then we can take $\theta = a\iota\pi + b\pi + c\iota$. Once this θ is found—say, by using Cornacchia's algorithm as proposed in [63]—we can decompose $\tau = \eta \circ \psi$, where $\deg(\psi) = B$ and $\deg(\eta) = e$. Since θ and $\phi|_{E_0[B]}$ (and hence $\hat{\phi}|_{E[B]}$) are known, η can then be found using a meet-in-the-middle approach, using a number of operations which is efficient, provided that e is smooth and small. Finally, in the third step we typically have $\ker(\hat{\phi}) = \ker(\tau - [d]) \cap E[A]$ (which is sufficient to recover ϕ); when this is not the case, a more sophisticated solution is sufficient (see [63, Section 4.3]). The attack of [49] uses much the same framework.

3.2 Commutative SIDH

In 2018, Castryck, Lange, Martindale, Panny, and Renes proposed a key exchange algorithm titled Commutative Supersingular Isogeny Diffie-Hellman (CSIDH) in [14]. CSIDH uses the action of the ideal class group on the set of isomorphism classes of supersingular elliptic curves defined over $GF(p)$ 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 as small as possible while ensuring p is prime. Let $\mathcal{O}$ denote the $GF(p)$-endomorphism ring of the supersingular Montgomery curve $E : y^2 = x^3 + x$ defined over $GF(p)$. Then $\mathcal{O}$ has the property that each of the principal ideals $\ell_i\mathcal{O}$ split into the product of $\mathfrak{l}_i = ([\ell_i], \pi_p - [1])$ and $\bar{\mathfrak{l}}_i = ([\ell_i], \pi_p + [1])$, where π_p is the Frobenius endomorphism of E. 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 $\mathrm{cl}(\mathcal{O})$.

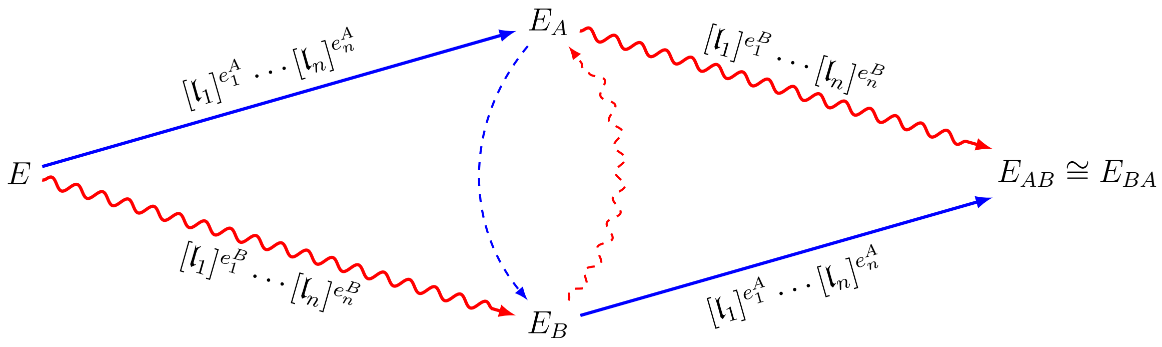
To begin the key exchange protocol, Alice and Bob both select private keys of the form $(e_1^A, \ldots, e_n^A)$ and $(e_1^B, \ldots, e_n^B)$, respectively, where each e_j^A and e_j^B is an integer chosen from some fixed interval $[-b_j, b_j] \cap \mathbb{Z}$. Alice uses her key to compute a curve E_A, 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; Bob proceeds analogously, using his own key to compute a curve E_B:

$$E_A := [\mathfrak{l}_1]^{e_1^A} \cdots [\mathfrak{l}_n]^{e_n^A} * E, \qquad E_B := [\mathfrak{l}_1]^{e_1^B} \cdots [\mathfrak{l}_n]^{e_n^B} * E, \tag{3.1}$$

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 particular, Alice computes E_{BA} and Bob computes E_{AB}, defined by:

$$E_{BA} := [\mathfrak{l}_1]^{e_1^A} \cdots [\mathfrak{l}_n]^{e_n^A} * E_B, \qquad E_{AB} := [\mathfrak{l}_1]^{e_1^B} \cdots [\mathfrak{l}_n]^{e_n^B} * E_A. \tag{3.2}$$

The two curves E_{BA} and E_{AB} are $GF(p)$-isomorphic since they both correspond to the action of $[\mathfrak{l}_1]^{e_1^A+e_1^B} \cdots [\mathfrak{l}_n]^{e_n^A+e_n^B}$ on the curve E, by the commutativity of the ideal class group. The shared key is the $GF(p)$-isomorphism class of $E_{BA} \cong E_{AB}$. The protocol is depicted in Figure 3.2.



E
E_A
E_B
$E_{AB} \cong E_{BA}$
$[\mathfrak{l}_1]^{e_1^A} \cdots [\mathfrak{l}_n]^{e_n^A}$
$[\mathfrak{l}_1]^{e_1^B} \cdots [\mathfrak{l}_n]^{e_n^B}$
$[\mathfrak{l}_1]^{e_1^B} \cdots [\mathfrak{l}_n]^{e_n^B}$
$[\mathfrak{l}_1]^{e_1^A} \cdots [\mathfrak{l}_n]^{e_n^A}$

Figure 3.2: A depiction of the computations involved in CSIDH. Alice follows the solid, plain blue arrows by evaluating the action of the indicated class group element, and follows the dashed, plain blue arrow by reading the message she receives from Bob. Bob analogously follows the wavy red arrows.

The original method proposed in [14] for carrying out the actions in (3.1) and (3.2) is to first choose a random point $P \in E[\pi_p \pm [1]]$, where E is the current curve and π_p denotes the Frobenius endomorphism. The point P will have some order $\mathrm{ord}(P) = \ell_1^{c_1} \cdots \ell_n^{c_n}$, where $c_i \in \{0, 1\}$ (after multiplication by 4). 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 [14], there has been much focus on making the evaluation of the group action more efficient; our contributions to optimizing the implementation of CSIDH appears in Chapter 5.

3.2.1 Computational Problems for CSIDH

As with SIDH, there are two computational problems and one decisional variant which are closely related to the security of CSIDH.

Problem 3.7 (Complex Multiplication Action Inversion [14, Problem 10: Key Recovery])**.** Given two supersingular elliptic curves $E/GF(p)$ and $E'/GF(p)$ with $\mathrm{End}_{GF(p)}(E) \cong \mathcal{O} \cong \mathrm{End}_{GF(p)}(E')$, find an ideal $\mathfrak{a} \in \mathcal{O}$—represented in a way which allows the action of $[\mathfrak{a}]$ to be evaluated efficiently—such that $E' = [\mathfrak{a}] * E$.

What is meant by "efficient" in Problem 3.7 is a matter of context—in Chapter 4, we solve the problem in time $2^{O(\sqrt{\log|\mathrm{cl}(\mathcal{O})| \log\log|\mathrm{cl}(\mathcal{O})|})}$, finding a representation of $[\mathfrak{a}]$ whose action can be evaluated in time $2^{O(\sqrt{\log|\mathrm{cl}(\mathcal{O})|})}$.

Problem 3.7 is the discrete logarithm analogue for CSIDH. The analogues of the computational and decisional Diffie-Hellman problems are as follows.

Problem 3.8 (CSIDH Shared Key Computation)**.** Given supersingular elliptic curves $E/GF(p)$, $E_A/GF(p)$, and $E_B/GF(p)$ with the same $GF(p)$-rational endomorphism ring $\mathcal{O}$, find $E_C = [\mathfrak{a}][\mathfrak{b}] * E$, where $[\mathfrak{a}]$ and $[\mathfrak{b}]$ satisfy

$$E_A = [\mathfrak{a}] * E \qquad\qquad E_B = [\mathfrak{b}] * E.$$

Problem 3.9 (CSIDH Shared Key Decision)**.** Given supersingular elliptic curves $E/GF(p)$, $E_A/GF(p)$, $E_B/GF(p)$, and $E_C/GF(p)$, where E_C is chosen either uniformly at random from $\mathcal{E\ell\ell}_{GF(p)}(\mathcal{O})$ or is equal to $[\mathfrak{a}][\mathfrak{b}] * E$, where $[\mathfrak{a}]$ and $[\mathfrak{b}]$ satisfy

$$E_A = [\mathfrak{a}] * E \qquad\qquad E_B = [\mathfrak{b}] * E,$$

each with probability $\frac{1}{2}$, the CSIDH Shared Key Decision problem is to determine which is the case.

So far, there are no proposed methods to tackle Problems 3.8 and 3.9 which do not involve solving Problem 3.7, though the problems are not known to be equivalent.

Chapter 4

A Quantum Algorithm for Inverting Complex Multiplication

In this chapter we describe a subexponential-time, polynomial quantum space algorithm which inverts the complex multiplication group action used in CSIDH. In particular, given a prime p of the form used in CSIDH and pair of supersingular elliptic curves $E/GF(p), E'/GF(p)$, our algorithm finds the element $[\mathfrak{a}] \in \mathrm{cl}(\mathcal{O})$ such that $E' = [\mathfrak{a}] * E$; moreover the representation of $[\mathfrak{a}]$ is such that its action can be computed in subexponential time. The algorithm uses Kuperberg's algorithm [47] and its extension due to Regev [65] to solve an instance of the dihedral hidden subgroup problem, and inherits its time and space complexity; in particular, it runs in subexponential time and polynomial quantum space.

In Section 4.1 we discuss the hidden subgroup problem in general, the dihedral case in particular, and Kuperberg's algorithm, and briefly mention the time and space complexity of Regev's extension. In the remaining sections we lay the groundwork for applying the algorithms in the context of CSIDH and analyze the time and space complexity of our algorithm. In particular, we give considerable attention to the question of how efficiently construct quantum states of the form

$$\frac{1}{\sqrt{|\mathrm{cl}(\mathcal{O})|}} \sum_{[\mathfrak{a}] \in \mathrm{cl}(\mathcal{O})} |[\mathfrak{a}]\rangle |[\mathfrak{a}] * E\rangle$$

which are required by Kuperberg's algorithm; underlying this is a careful analysis of a quantum implementation of the action of $\mathrm{cl}(\mathcal{O})$ on $\mathcal{E\ell\ell}_{GF(p)}(\mathcal{O})$. The fundamental technique that allows us to efficiently construct these states is a classical preprosessing stage whose output is used to convert between a “cyclic representation” of the class group elements—which are easy to enumerate, but whose action is not easy to evaluate—and a “prime decomposition” representation—whose action is easy to evaluate, but which are not easy to enumerate without knowing more about the structure of the class group.

4.1 Background

4.1.1 The Dihedral Hidden Subgroup Problem

To begin, we define the notion of a hidden subgroup problem.

Definition 4.1 (Hidden Subgroup Problem [58, Section 5.4.3])**.** Let X be a finite set, G be a finitely-generated group, and $H \leq G$. Let $f\colon G \to X$ be a function which satisfies $f(a) = f(b) \iff ab^{-1} \in H$. The *hidden subgroup problem* is to find H given X, G, and (quantum) black-box access to f.

The hidden subgroup framework encodes a very broad class of problems which are far from being solved in general. A number of well-known quantum algorithms can be cast in this framework by choosing particular groups X, G, H, and f, such as Shor's algorithm for discrete logarithms [68]. It is known (*q.v.* [52, Theorem 3.13]) that, when G is abelian, the problem can be solved in quantum polynomial time; moreover, it is known that the problem can always be solved using a polynomial number of quantum queries to f [30].

In this chapter we will be concerned with the case when G is a *dihedral group*; that is, when G is of the form

$$G \cong D_{2n} = \langle x, y | x^n, y^2, xyxy \rangle.$$

Equivalently, D_{2n} is the group $D_{2n} = \mathbb{Z}_N \rtimes_\theta \mathbb{Z}_2$ where $\theta(b)(k) = (-1)^b k$ for all $k \in \mathbb{Z}_N$.

We can encode the problem of recovering a CSIDH ephemeral secret key as an instance of the hidden subgroup problem in the group D_{2N}, where $N = |\mathrm{cl}(\mathcal{O})|$, assuming that $\mathrm{cl}(\mathcal{O})$ is cyclic (we justify this assumption in Note 4.2). In particular, let E be the global base curve and let E_A be Alice's ephemeral public key. Suppose that $\mathrm{cl}(\mathcal{O}) = \langle [\mathfrak{g}] \rangle$, and define the function $f\colon \mathbb{Z}_N \rtimes_\theta \mathbb{Z}_2 \to \mathcal{E\ell\ell}_p(\mathcal{O})$ by

$$f(k, b) = \begin{cases} [\mathfrak{g}^k] * E & \text{if } b = 1 \\ [\mathfrak{g}^k] * E_A & \text{if } b = 0 \end{cases}. \tag{4.1}$$

Since the action of $\mathrm{cl}(\mathcal{O})$ on $\mathcal{E}\ell\ell_p(\mathcal{O})$ is free and transitive, we have that

$$f(s_1, b) = f(s_2, b) \iff s_1 = s_2, \text{ and } f(s_1, 0) = f(s_2, 1) \iff E_A = [\mathfrak{g}^{s_2 - s_1}] * E.$$

Consequently, f hides the subgroup $H = \{(0,0), (s_A, 1)\}$, where s_A satisfies $E_A = [\mathfrak{g}^{s_A}] * E$. Thus if we can solve this instance of the dihedral hidden subgroup problem, we can recover Alice's ephemeral secret key.

4.1.2 Kuperberg's Algorithm

Kuperberg's algorithm is a quantum algorithm which solves the hidden subgroup problem in dihedral groups. As originally described [47] the algorithm requires subexponential time and space; later, Regev [65] reduced the quantum space complexity to polynomial. Kuperberg's algorithm is integral to the attack on CSIDH that we develop in this chapter, and so we describe it here and discuss Regev's extension in the next section.

We consider only the simplest case of the algorithm; in particular, we:

1. Suppose that N is a power of 2;
2. Suppose that the subgroup is of the form $H = \{(0,0), (s_A, 1)\}$ for some s_A; and,
3. Only find the lowest order bit of s_A.

We can generalize from these simplifying assumptions using [47, Section 5, Algorithm 2], [47, Proposition 2.1], and [47, Algorithm 1, Step 4], respectively.

The algorithm requires the construction of many copies of a certain type of one-qubit quantum state; before stating the algorithm we will detail how they can be constructed and manipulated. Given that we want to solve the hidden subgroup problem in D_{2N}, initialize the registers and ancilla to $\frac{1}{\sqrt{2N}} \sum_{(s,b) \in D_{2N}} |s, b\rangle |0\rangle$. Using the quantum oracle for f, write f to the zero register to obtain

$$\frac{1}{\sqrt{2N}} \sum_{(s,b) \in D_{2N}} |s, b\rangle |f(s, b)\rangle$$

Because f has a hidden subgroup $H = \langle (s_A, 1) \rangle$, measuring out the ancillary qubits will yield a random coset state $|(s, b) + H\rangle := \frac{1}{\sqrt{2}} \left(|s, b\rangle + \left| s + (-1)^b s_A, b \oplus 1 \right\rangle \right)$ for some

$(s,b) \in D_{2N}$. Without loss of generality we can take $b=1$, since every coset of H has a representative of this form. Then

$$|(s,1)+H\rangle = \frac{1}{\sqrt{2}}(|s,1\rangle + |s-s_A,0\rangle).$$

Applying the quantum Fourier transform on the second register transforms the state to

$$\frac{1}{\sqrt{2N}}\left(\sum_{k=0}^{N} e^{\frac{2\pi i k(s-s_A)}{N}}|k,0\rangle + \sum_{k=0}^{N} e^{\frac{2\pi i k s}{N}}|k,1\rangle\right)$$

which, after measuring all but the last qubit, becomes a state of the form

$$|\phi_k\rangle = e^{\frac{2\pi i k(s-s_A)}{N}}|0\rangle + e^{\frac{2\pi i k s}{N}}|1\rangle$$

where k is chosen uniformly at random. For computational purposes we can ignore global phase, and so we will instead say that we have a state

$$|\psi_k\rangle = |0\rangle + e^{\frac{2\pi i k s_A}{N}}|1\rangle.$$

These $|\psi_k\rangle$ are exactly the states required for the slope-finding algorithm. Given $|\psi_k\rangle$ and $|\psi_\ell\rangle$, their joint state is

$$|\psi_k\rangle \otimes |\psi_\ell\rangle = |00\rangle + e^{\frac{2\pi i \ell s_A}{N}}|01\rangle + e^{\frac{2\pi i k s_A}{N}}|10\rangle + e^{\frac{2\pi i (k+\ell) s_A}{N}}|11\rangle.$$

Applying CNOT to this state we obtain the state

$$\text{CNOT}|\psi_k\rangle \otimes |\psi_\ell\rangle = |00\rangle + e^{\frac{2\pi i \ell s_A}{N}}|01\rangle + e^{\frac{2\pi i (k+\ell) s_A}{N}}|10\rangle + e^{\frac{2\pi i k s_A}{N}}|11\rangle.$$

Measuring out the second qubit, we obtain either

$$\begin{aligned} |0\rangle + e^{\frac{2\pi i (k+\ell) s_A}{N}}|1\rangle &= |\psi_{k+\ell}\rangle, \text{ if 0 is measured, or} \\ e^{\frac{2\pi i \ell s_A}{N}}\left(|0\rangle + e^{\frac{2\pi i (k-\ell) s_A}{N}}|1\rangle\right) &= |\psi_{k-\ell}\rangle, \text{ if 1 is measured,} \end{aligned}$$

each with 50% probability. This "combining" operation is vital to the slope-finding algorithm, which is as follows:

1. Let $n = \left\lceil \sqrt{\log_2 N} \right\rceil$. Construct a list L_0 of $3 \cdot 2^{2n}$ states of the form $|\psi_k\rangle$, for k

uniformly randomly selected.

2. For j between 0 and $n-1$, suppose we have a list L_j of qubits $|\psi_k\rangle$ such that each k has at least nj trailing zeros. Divide the list into pairs $(|\psi_{k_1}\rangle, |\psi_{k_2}\rangle)$ such that k_1 and k_2 agree in their last $\min\{n(j+1), \log_2 N - 1\}$ bits. Combine these pairs and, if the state $|\psi_{k_1-k_2}\rangle$ is produced, add it to the new list L_{j+1}

3. The final list L_n contains qubits which are either in the state $|\psi_0\rangle$ or $|\psi_{\frac{N}{2}}\rangle$. With high probability, at least one will be of the form $|\psi_{\frac{N}{2}}\rangle$. Measure such a state in the Hadamard basis; the measured value is the value of the lowest-order unknown bit of s_A.

4.1.3 Regev's Extension

It is clear from the description of Kuperberg's algorithm that the quantum space required is $\Omega(2^{\sqrt{\log N}})$, which is subexponential in $\log N$. Regev [65] proposes a new "combining" operation which takes a larger number of states with uniformly random labels and outputs with high probability a state whose label has a certain number of trailing zeros. Though Regev states the algorithm only when N is a power of 2 which satisfies $\log_2 N = k_1 k_2 + 1$ with $k_1 = O\left(\sqrt{\frac{\log N}{\log\log N}}\right)$ and $k_2 = O(\sqrt{\log N \log\log N})$, this can be extended to arbitrary N using the results of [17, Appendix A]. Regev's algorithm trades time for quantum space, requiring time $\Theta(2^{\sqrt{\log N \log\log N}})$ but space only $O(\log^2 N \log^2 \log N)$.

4.2 Classical Preprocessing

Let $p = 4\ell_1\ell_2\cdots\ell_t - 1$ be a prime of the form used in CSIDH, $E/GF(p) : y^2 = x^3 + x$, and $\mathcal{O} \cong \mathrm{End}_{GF(p)}(E)$. Suppose $\mathrm{cl}(\mathcal{O})$ is a cyclic group (*q.v.* Note 4.2) of size N and with generator $[\mathfrak{g}]$. We describe a procedure to find, for each positive integer j, an expression $[\mathfrak{g}^{2^j}] = \prod_{i=1}^{t}[\mathfrak{l}_i^{e_i}]$, with $|e_i|$ subexponential with respect to $\log N$. Here, as in the description of CSIDH, the $[\mathfrak{l}_i]$, $i = 1, \ldots, t$ denote the ideals classes containing $\mathfrak{l}_i = ([\ell_i], \pi_p - 1)$. The idea of the algorithm is to find enough samples of the form $[\mathfrak{g}^k] = \prod_{i=1}^{t}[\mathfrak{l}_i^{e_i}]$ where the e_i are chosen at random and subexponentially large, and then use a BKW-like algorithm to express $[\mathfrak{g}^{2^j}]$ as a subexponentially-short product of the samples $[\mathfrak{g}^k]$.

Note 4.2 (On the cyclicity of $\mathrm{cl}(\mathcal{O})$). In general, the class group is not always cyclic, but heuristically it is cyclic in the vast majority of cases (97% of the time per Cohen-Lenstra [20]), and in the case of CSIDH-512 [6], which is the CSIDH parameter set we consider throughout this work. We conjecture that standard techniques such as [16, Appendix A] could be used to extend to the non-cyclic case.

4.2.1 A Heuristic Assumption

In order to construct the samples we require in the following sections, we require a heuristic assumption on the size of t relative to $\log N$, which ensures that there are sufficiently many random vectors of length t and entries bounded in magnitude by $\mathrm{poly}(\log N)$. This assumption can more intuitively be phrased in terms of the size of ℓ_t relative to $\frac{p+1}{\ell_t}$, under the heuristic assumption that $N = \sqrt{p}$ (as in [14]). We present the two assumptions here and prove the link between them; in Section 4.2.2, we make use of these assumptions.

Heuristic 4.3. For a prime p of the form $p = 4\ell_1\ell_2 \ldots \ell_t - 1$ where $\ell_1, \ell_2, \ldots, \ell_{t-1}$ are the first $t-1$ odd primes and ℓ_t is the smallest prime larger than ℓ_{t-1} for which the expression is prime, and $N = \sqrt{p}$, for all t large enough one has

$$t \geq 3\sqrt{\log N}.$$

Heuristic 4.4. Let $\ell_1, \ell_2, \ldots, \ell_{t-1}$ be the first $t-1$ odd primes. For all t large enough, there exists a prime $\ell_t > \ell_{t-1}$ for which $p = 4\ell_1\ell_2 \cdots \ell_t - 1$ is prime, and

$$\log \ell_t \leq \frac{2}{3}\sqrt{\log N} - 1.$$

Essentially, Heuristic 4.4 says that if the largest prime ℓ_t used in the construction of the prime p is chosen as small as possible, it cannot contribute "too much" to the bit length of N. We note that this heuristic is satisfied for all currently-proposed CSIDH parameter sets.

Claim 4.5. Heuristic 4.4 implies Heuristic 4.3.

Proof. Let t be large enough so that Heuristic 4.4 holds, and that $t \geq 2\log 2$. We have

$$N = \sqrt{4\ell_1\ell_2 \cdots \ell_t - 1} \leq \sqrt{4\ell_1\ell_2 \cdots \ell_t} \leq 2\ell_t^{\frac{t}{2}};$$

taking logarithms, we obtain

$$\log N \leq \frac{t}{2}\log \ell_t + \log 2 \leq \frac{t}{3}\sqrt{\log N}$$

(where we have used the fact that $t \geq 2\log 2$), so that $t \geq 3\sqrt{\log N}$, as required. $\square$

It is essentially impossible to complete the proof in this chapter without some heuristic assumption, though the ones we have chosen are not the only acceptable ones. In particular, very similar algorithms to ours appear in [8] and [11], each using a different heuristic argument. We note that our argument can be adapted to use [8, Heuristic 1] by replacing the ideal classes $[\mathfrak{l}_1], [\mathfrak{l}_2], \ldots, [\mathfrak{l}_t]$ in the sections that follow by the ideal classes $[\mathfrak{p}_1], [\mathfrak{p}_2], \ldots, [\mathfrak{p}_s]$ of that work.

4.2.2 Expander Graphs

It is known [41, Theorem 3.2] that isogeny graphs of (isomorphism classes of) elliptic curves with complex multiplication by an imaginary quadratic order $\mathcal{O}_\Delta$ with edges corresponding to isogenies with prime degree less than some fixed bound $(\log|\Delta|)^B$ are in fact expander graphs. The following well-known result about expander graphs then tells us about the distribution of elliptic curves chosen from this set by taking short random walks.

Lemma 4.6 ([41, Lemma 2.1])**.** Let Γ be a finite d-regular graph for which the non-trivial eigenvalues λ of the adjacency matrix are bounded by $|\lambda| \leq c$, for some $c < d$. Let S be any subset of the vertices of Γ, and v any vertex in Γ. A random walk of any length at least $\frac{\log 2|\Gamma|/|S|^{1/2}}{\log d/c}$ starting from v will land in S with probability between $\frac{1}{2}\frac{|S|}{|\Gamma|}$ and $\frac{3}{2}\frac{|S|}{|\Gamma|}$.

When creating the states required by Kuperberg's algorithm (*q.v.* Section 4.3.2) we must evaluate the subgroup-hiding function f on a uniform superposition over $\mathrm{cl}(\mathcal{O})$. In order to do so we must consider two representations of the elements of $\mathrm{cl}(\mathcal{O})$: the "cyclic representation" $[\mathfrak{g}^j]$, which is easy to enumerate, and the "prime decomposition" representation $\prod_i [\mathfrak{l}^{e_i}]$, whose action is easy to evaluate, provided that the e_i are sufficiently small. In this section we present an algorithm which allows us to switch between these presentations.

To begin our algorithm, we must sample values of $[\mathfrak{g}^j]$ for which $0 \leq j \leq |\mathrm{cl}(\mathcal{O})| - 1$ is (nearly) uniform, and the decomposition $[\mathfrak{g}^j] = \prod_i [\mathfrak{l}_i^{e_i^j}]$ has $e_i^j = \mathrm{poly}(\log N)$. First, we solve $[\mathfrak{g}^{a_i}] = [\mathfrak{l}_i]$ for each $1 \leq i \leq t$ in terms of our generator $[\mathfrak{g}]$ using Shor's algorithm [68] (see [67] for more a more detailed explanation of how Shor's algorithm applies in this setting). Then, choosing random $(e_1, \ldots, e_t)$ with each $|e_i| = \Theta(\log_2 N)$, we can determine $j := \log_{[\mathfrak{g}]} \prod_{i=1}^t [\mathfrak{l}_i^{e_i}] = \sum_{i=1}^t a_i e_i$. Lemma 4.6 tells us that these j are chosen nearly uniformly at random from the range $\{0, \ldots, N-1\}$. Moreover, the e_i^j are polynomial, and there are still $\Omega(2^{3\sqrt{\log N}})$ possible values of the discrete logarithm, given that $t \geq 3\sqrt{\log N}$.

Given that we can construct samples $[\mathfrak{g}^j]$ of the form described, we can use a version of the BKW algorithm [10] to compute $[\mathfrak{g}^{2^k}] = \prod_{i=1}^{t} [\mathfrak{l}_i^{e_i}]$, where the exponents are subexponential in $\log N$. Our algorithm uses two subroutines—upper and lower compression—which we describe now. Let $n = \left\lceil \sqrt{\log N} \right\rceil$. The idea of both subroutines is to take as input a collection of uniformly chosen positive integers bounded by $N \approx 2^{n^2}$, and reduce the number of non-zero coefficients of their expression in base 2^n.

Lemma 4.7 (Upper compression)**.** Let $k \in \{0, \dots, n-1\}$, let $c > 0$ and let $m = (c+1)2^n$. There exists an algorithm that takes as input $\mathbf{a} = (a_1, \dots, a_m) \in \{0, \dots, 2^{n(k+1)} - 1\}^m$ and outputs $\mathbf{b} = ((v_1, w_1, b_1), \dots, (v_{m'}, w_{m'}, b_{m'}))$ where $b_i = a_{w_i} - a_{v_i} < 2^{nk}$ and $m' \geq c(2^n)$.

Proof. For each $i \in \{0, \dots, 2^n - 1\}$ let B_i be the set of pairs (a_v, v) such that $i2^{nk} \leq a_v < (i+1)2^{nk}$, let $c_i = \max\{a \colon (a, v) \in B_i\}$ and w_i be such that $(c_i, w_i) \in B_i$. Notice that for any $(a_v, v) \in B_i$, $0 \leq c_i - a_v$, moreover, since $i2^{nk} \leq c_i, a_v < (i+1)2^{nk}$, the difference is bounded by 2^{nk}. The output of the algorithm is a vector consisting of the tuples $(v, w_i, c_i - a_v)$, with $(a_v, v) \in B_i \setminus \{(c_i, w_i)\}$, for each $i \in \{0, \dots, 2^n - 1\}$. $\square$

Lemma 4.8 (Lower compression)**.** Let $k \in \{0, \dots, n-1\}$, let $c > 0$ and let $m = (c+1)2^n$. There exists an algorithm that on input a vector $\mathbf{a} = (a_1, \dots, a_m) \in 2^{kn}\mathbb{Z}^m$ outputs a vector $\mathbf{b} = ((v_1, w_1, b_1), \dots, (v_{m'}, w_{m'}, b_{m'}))$ where $2^{nk+1} \mid b_i = a_{w_i} - a_{v_i}$ and $m' \geq c(2^n)$.

Proof. For each $i \in \{0, \dots, 2^n - 1\}$ let B_i be the set of pairs (a_v, v) such that $i2^{kn} \equiv a_v \mod 2^{(k+1)n}$, let $c_i = \max\{a \colon (a, v) \in B_i\}$ and w_i be such that $(c_i, w_i) \in B_i$. Note that for any $(a_v, v) \in B_i$, we have $0 \leq c_i - a_v \equiv 0 \mod 2^{(k+1)n}$. The output of the algorithm is a vector consisting of the tuples $(v, w_i, c_i - a_v)$, with $(a_v, v) \in B_i \setminus \{(c_i, w_i)\}$, for each $i \in \{0, \dots, 2^n - 1\}$. $\square$

For our purposes, we assume that the input of these algorithms is drawn from the uniform distribution (see Subsection 4.2.2). Suppose that one of the compression algorithms is called on an input $\mathbf{a}$ whose entries are sampled uniformly at random from $\{0, \dots, 2^{kn} - 1\}$. Then for any $i \in \{0, \dots, 2^n - 1\}$, the expected cardinality of B_i is $(c+1)$; therefore the expected value of $c_i = \max\{a \colon (a, v) \in B_i\}$ is $\frac{c+1}{c+2}2^n$. This implies that the expected statistical distance of the distribution $c_i - a$ and uniform is $2(1 - \frac{c+1}{c+2})$. By summing over all i, the expected statistical distance of the output distribution and uniform is at most $2^{n+1}(\frac{1}{c+2})$.

Now our aim is to write 2^ℓ, for $\ell \in \{0, \dots, n^2 - 1\}$, as a short linear combination of the given samples. The idea is to write $\ell = nq + r$, and call the lower compression algorithm q

times and the upper compression algorithm $n-q+1$ times, to obtain samples of the form $a2^{nq}$, and find $a=2^r$ among the samples.

Proposition 4.9 (Iteration)**.** Let $\ell \in \{0,\dots,n^2-1\}$ and let $m=2^{3n}$. There exists an algorithm $\mathcal{A}$ that takes $\mathbf{a}=(a_1,\dots,a_m)$ as input, and outputs a vector $\mathbf{s}\in\mathbb{Z}^m$ such that $\langle \mathbf{a},\mathbf{s}\rangle = 2^\ell$, and whose expected infinity norm $\|\mathbf{s}\|_\infty$ is bounded by 2^n.

Proof. Let $\ell = nq+r$ with $0\le r<n$. Let $\mathcal{A}'$, $\mathcal{A}''$ be the algorithms described in Lemmas 4.7 and 4.8, respectively. The algorithm $\mathcal{A}$ starts by initializing $\mathbf{a}^{(0)}=(a_1,\dots,a_m)$. For $i=1,\dots,q$, $\mathcal{A}$ calls $\mathcal{A}''$ on input $\mathbf{a}^{(i-1)}$ to obtain $\mathbf{b}^{(i)}=((v_1^{(i)},w_1^{(i)},b_1^{(i)}),\dots,(v_{m^{(i)}}^{(i)},w_{m^{(i)}}^{(i)},b_{m^{(i)}}^{(i)}))$ and sets $\mathbf{a}^{(i)}=(b_1^{(i)},\dots,b_{m^{(i)}}^{(i)})$. For $j=0,\dots,n-q+2$, the algorithm calls $\mathcal{A}'$ on input $\mathbf{a}^{(i+j)}$ to obtain $\mathbf{b}^{(i+j)}=((v_1^{(i+j)},w_1^{(i+j)},b_1^{(i+j)}),\dots,(v_{m^{(i+j)}}^{(i+j)},w_{m^{(i+j)}}^{(i+j)},b_{m^{(i+j)}}^{(i+j)}))$ and sets $\mathbf{a}^{(i+j)}=(b_1^{(i+j)},\dots,b_{m^{(i+j)}}^{(i+j)})$. By Lemmas 4.7 and 4.8, the length $m^{(n-1)}$ of $b^{(n-1)}$ is $(2^{2n}-n+1)2^n$, and its entries are of the form $a2^{nq}$, for $a\in[0,\dots,2^n-1]$. Moreover, following the discussion above, the distribution of a in this set is statistically close to uniform; therefore we can find 2^ℓ with high probability. Without loss of generality assume $a_0^{(n-1)}=2^\ell$; then by definition we have that 2^ℓ is written as a difference of two entries of $\mathbf{a}^{(n-2)}$. Following this recursively, after $n-1$ steps we can find 2^ℓ as a linear combination of 2^n (possibly repeated) entries of $\mathbf{a}$. Hence the largest coefficient of the linear combination is bounded by 2^n.

Each of the compression steps takes $O(2^{3n})$ time and $O(2^{3n})$ space; the overall complexity is $O\left((n-1)2^{3n}\right)=2^{O(\sqrt{\log N})}$ time and space. □

4.3 The Quantum Algorithm

4.3.1 Instantiating the Action of $\mathrm{cl}(\mathcal{O})$ in Polynomial Space

Since [14, Alg. 2] for computing the action of $\mathrm{cl}(\mathcal{O})$ on $\mathcal{E}\!\ell\!\ell_p(\mathcal{O})$ is not amenable to being instantiated quantumly, we present a modified algorithm here. While [14, Alg. 2] succeeds with probability 1 but has variable time, our algorithm has (tunable) fixed time but succeeds with (tunable) probability less than 1.

To begin, we give an algorithm for computing $E_B=[\mathfrak{l}^{\pm1}]*E_A$ for prime ℓ. We emphasize that this (classical) algorithm is designed with translation to a quantum algorithm—rather than efficiency—in mind.

Algorithm 1 succeeds if and only if there is $i^*\in\{1,2,\dots,r\}$ such that

Algorithm 1: A classical algorithm for computing $[\mathfrak{l}^{(-1)^s}] * E_A$ for prime ℓ, suitable for implementing on a quantum computer.

```
input : $A \in GF(p)$, and $s \in \{0,1\}$
output: $B \in GF(p)$ such that $[\mathfrak{l}^{(-1)^s}] * E_A = E_B$, where $E_B : y^2 = x^3 + Bx^2 + x$
1  $x_1, x_2, \ldots, x_r \xleftarrow{\$} GF(p)$
2  $c \leftarrow 0$
3  for $i$ from 1 to $r$ by 1 do
4      $y_i \leftarrow \sqrt{x_i^3 + Ax_i^2 + x_i}$          ▷ In the extension field $GF(p^2)$.
5      $P_i \leftarrow (x_i, y_i)$, $Q_i \leftarrow \left[\frac{p+1}{\ell}\right] P_i$
6      if $\left(\frac{x_i^3+Ax_i^2+x_i}{p}\right) = (-1)^s$ and $Q_i \neq \infty$ and $c = 0$ then
7          Compute $B$, where $\phi\colon E_A \to E_B : y^2 = x^3 + Bx^2 + x$ is an isogeny with
           $\ker \phi = \langle Q_i \rangle$
8      end
9      if $\left(\frac{x_i^3+Ax_i^2+x_i}{p}\right) = (-1)^s$ and $Q_i \neq \infty$ then
10         $c \leftarrow c + 1$
11     end
12 end
```

1. $\left(\frac{x_{i^*}^3+Ax_{i^*}^2+x_{i^*}}{p}\right) = (-1)^s$; and,
2. $\ell \nmid \operatorname{ord}_{E_A}(P_{i^*})$.

For uniformly random x, these conditions hold with probability $\frac{1}{2}$ and $\frac{\ell-1}{\ell}$, respectively, since $E(GF(p)) \cong \mathbb{Z}/4\mathbb{Z} \oplus \bigoplus_{k=1}^{t} \mathbb{Z}/\ell_k\mathbb{Z}$. Thus the total probability that Algorithm 1 succeeds is $1 - \left(\frac{\ell+1}{2\ell}\right)^r$. Later we shall choose a value of r so that our final quantum algorithm succeeds with sufficient probability.

Next we build upon Algorithm 1 to construct an algorithm which computes $E_B = [\mathfrak{l}^{\pm e}] * E_A$ for $e \in \mathbb{N}$. It is easy to see that Algorithm 2 succeeds with probability $\left(1 - \left(\frac{\ell+1}{2\ell}\right)^r\right)^e$.

Finally, Algorithm 3 computes $[\mathfrak{l}_1^{\pm e_1} \mathfrak{l}_2^{\pm e_2} \cdots \mathfrak{l}_t^{\pm e_t}] * E_A$.

Then

$$\mathbb{P}[\text{Algorithm 3 succeeds}] = \prod_{k=1}^{t} \left(1 - \left(\frac{\ell_k + 1}{2\ell_k}\right)^r\right)^{e_k} \geq \left(1 - \left(\frac{3}{4}\right)^r\right)^{\|\mathbf{e}\|_1}.$$

Algorithm 2: A classical algorithm for computing $[\mathfrak{l}^{(-1)^s e}] * E_A$ for prime ℓ, suitable for implementing on a quantum computer.

```
   input  : $A \in GF(p)$, $s \in \{0,1\}$, and $e \in \mathbb{N}$
   output: $B \in GF(p)$ such that $[\mathfrak{l}^{(-1)^s e}] * E_A = E_B$, where $E_B : y^2 = x^3 + Bx^2 + x$
1  $x_1, x_2, \ldots, x_r \xleftarrow{\$} GF(p)$
2  $B \leftarrow A$
3  for $i$ from 1 to $e$ by 1 do
4      $B \leftarrow C$, where $[\mathfrak{l}^{(-1)^s}] * E_B = E_C$: $y^2 = x^3 + Cx^2 + x$, computed using random
        values $x_1, x_2, \ldots, x_r$
5  end
```

Algorithm 3: An algorithm for computing $[\mathfrak{l}_1^{(-1)^{s_1} e_1} \mathfrak{l}_2^{(-1)^{s_2} e_2} \cdots \mathfrak{l}_t^{(-1)^{s_t} e_t}] * E_A$ for primes $\ell_1, \ell_2, \ldots, \ell_t$, suitable for implementing on a quantum computer.

```
   input  : $A \in GF(p)$, $\mathbf{s} \in \{0,1\}^t$, and $\mathbf{e} \in \mathbb{N}^t$
   output: $B \in GF(p)$ such that $[\mathfrak{l}_1^{(-1)^{s_1} e_1} \mathfrak{l}_2^{(-1)^{s_2} e_2} \cdots \mathfrak{l}_t^{(-1)^{s_t} e_t}] * E_A = E_B$, where
           $E_B : y^2 = x^3 + Bx^2 + x$
1  $x_1, x_2, \ldots, x_r \xleftarrow{\$} GF(p)$
2  $B \leftarrow A$
3  for $k$ from 1 to $t$ by 1 do
4      $B \leftarrow C$, where $[\mathfrak{l}_k^{(-1)^{s_k} e_k}] * E_B = E_C : y^2 = x^3 + Cx^2 + x$, computed using
       random values $x_1, x_2, \ldots, x_r$.
5  end
```

From here we briefly describe how to instantiate this quantumly. First we describe the quantum instantiation of Algorithm 1. For brevity of notation we consider an input $|s\rangle|E_A\rangle|\mathbf{0}\rangle$ which is not in superposition, but of course the algorithm extends linearly to superpositions. Before the quantum part of the algorithm begins, we sample $x_1, x_2, \ldots x_r \xleftarrow{\$} GF(p)$ classically and include them as part of the initial state. We will use them in the quantum instantiation of all three algorithms.

1. In the notation of Algorithm 1, write $(Q_i)_{i=1}^r$ to a new register to obtain

$$|s\rangle|E_A\rangle|x_1, x_2, \ldots, x_r\rangle|Q_1, Q_2, \ldots, Q_r\rangle|\mathbf{0}\rangle$$

2. In the notation of Algorithm 1, define $w_i = 1$ if $\left(\frac{x_i^3+Ax_i^2+x_i}{p}\right) = (-1)^s$ and $Q_i \neq \infty$, and $w_i = 0$ otherwise. Write $w_1, w_2, \dots w_r$ to a new register to obtain

$$|s\rangle|E_A\rangle|x_1, x_2, \dots, x_r\rangle|Q_1, Q_2, \dots, Q_r\rangle|w_1, w_2, \dots, w_r\rangle|\mathbf{0}\rangle.$$

   This can be done by writing the results of each of the Boolean functions $[Q_i \neq \infty]$ and $\left[\left(\frac{x^3+Ax^2+x}{p}\right) = (-1)^s\right]$ to new registers, applying a Toffoli gate from these registers onto another new register, and then uncomputing the results of the two Boolean functions.

3. Set aside a new 0-initialized register to contain c. For $i = 1, 2, \dots, r$ apply Vélu's formulas [75] conditioned on $w_i = 1$ and $c = 0$, and increment c conditioned on $w_i = 1$. Essentially, we look down the list of points until we find x_{i^*} which is "appropriate" (has $w_i = 1$) and, when we find the first appropriate point, we set a flag which indicates that we have computed $[\mathfrak{l}] * E_A$, and so we should not compute more. The resultant state is

$$|s\rangle|E_A\rangle|x_1, x_2, \dots, x_r\rangle|Q_1, Q_2, \dots, Q_r\rangle|w_1, w_2, \dots, w_r\rangle|\hat{c}\rangle|[\mathfrak{l}] * E_A\rangle|\mathbf{0}\rangle.$$

   where $\hat{c}$ counts the number of w_i which are equal to 1.

4. Uncompute $\hat{c}$ by subtracting 1 from it conditioned on $w_i = 1$, for $i = r, r-1, \dots, 1$. Then uncompute $(Q_i)_{i=1}^r$ and $(w_i)_{i=1}^r$ by reversing the circuit of step (ii). Rearranging the registers, the final state is

$$|s\rangle|E_A\rangle|[\mathfrak{l}] * E_A\rangle|x_1, x_2, \dots, x_r\rangle|\mathbf{0}\rangle,$$

   as required.

For fixed ℓ, call the algorithm above $\mathcal{Q}_\ell^{(1)}$. We shall use it is a subroutine in the quantum instantiation of Algorithm 2. For input $|s\rangle|e\rangle|E_A\rangle|x_1, x_2, \dots, x_r\rangle|\mathbf{0}\rangle$:

1. Conditioned on register 2 being positive, apply $\mathcal{Q}_\ell^{(1)}$ to registers 1, 3, and 4 targeting a new register. Then decrement register 2, yielding

$$|s\rangle|e-1\rangle|E_A\rangle|x_1, x_2, \dots, x_r\rangle\left|[\mathfrak{l}^{(-1)^s}] * E_A\right\rangle|\mathbf{0}\rangle$$

2. Swap registers 3 and 5, and conditioned on register 2 being positive, apply $\mathcal{Q}_\ell^{(1)}$ again

to registers 1, 2, 3, and 4 targeting a new register. Decrement register 2 to obtain

$$|s\rangle|e-2\rangle\big|[\mathfrak{l}^{(-1)^s}]*E_A\big\rangle|x_1,x_2,\dots,x_r\rangle|E_A\rangle\big|[\mathfrak{l}^{(-1)^s 2}]*E_A\big\rangle|\mathbf{0}\rangle.$$

3. Given a state

$$|s\rangle|e-z\rangle\big|[\mathfrak{l}^{(-1)^s(z-1)}]*E_A\big\rangle|x_1,x_2,\dots,x_r\rangle|E_A\rangle\big|[\mathfrak{l}^{(-1)^s z}]*E_A\big\rangle|\mathbf{0}\rangle$$

swap registers 3 and 6, and apply the Pauli X to register 1. This yields

$$|1-s\rangle|e-z\rangle\big|[\mathfrak{l}^{(-1)^s z}]*E_A\big\rangle|x_1,x_2,\dots,x_r\rangle|E_A\rangle\big|[\mathfrak{l}^{(-1)^s(z-1)}]*E_A\big\rangle|\mathbf{0}\rangle.$$

Apply $\mathcal{Q}_\ell^{(1)}$ to registers 1, 3, and 4 targeting register 6 conditioned on register 2 being positive. Notice that the output of $\mathcal{Q}_\ell^{(1)}$ in this case is $[\mathfrak{l}^{(-1)^s(z-1)}]*E_A$ since we are applying $[\mathfrak{l}^{-(-1)^s}]$ in this case. This effectively erases the contents of register 6. We can then apply Pauli X to register 1 again, and, conditioned on register 2 being positive, we apply $\mathcal{Q}_\ell^{(1)}$ and decrement register 2 to obtain

$$|s\rangle|e-(z+1)\rangle\big|[\mathfrak{l}^{(-1)^s z}]*E_A\big\rangle|x_1,x_2,\dots,x_r\rangle|E_A\rangle\big|[\mathfrak{l}^{(-1)^s(z+1)}]*E_A\big\rangle|\mathbf{0}\rangle$$

4. Repeat step 3 $L-2$ times, where $L\geq e$. The result is

$$|s\rangle|e-L\rangle\big|[\mathfrak{l}^{(-1)^s(e-1)}]*E_A\big\rangle|x_1,x_2,\dots,x_r\rangle|E_A\rangle\big|[\mathfrak{l}^{(-1)^s e}]*E_A\big\rangle|\mathbf{0}\rangle$$

Copy register 6 onto a new register to obtain

$$|s\rangle|e-L\rangle\big|[\mathfrak{l}^{(-1)^s(e-1)}]*E_A\big\rangle|x_1,x_2,\dots,x_r\rangle|E_A\rangle\big|[\mathfrak{l}^{(-1)^s e}]*E_A\big\rangle\big|[\mathfrak{l}^{(-1)^s e}]*E_A\big\rangle|\mathbf{0}\rangle.$$

From here, we can simply reverse the iterations of step (iii) and steps (ii) and (i), erasing the ancillary registers. Rearranging registers yields

$$|s\rangle|e\rangle|E_A\rangle\big|[\mathfrak{l}^{(-1)^s e}]*E_A\big\rangle|x_1,x_2,\dots,x_r\rangle|\mathbf{0}\rangle,\text{ as required.}$$

For fixed $[\mathfrak{l}]$, call this algorithm $\mathcal{Q}_\ell^{(L)}$. To evaluate $[\mathfrak{l}_1^{(-1)^{s_1}e_1}\mathfrak{l}_2^{(-1)^{s_2}e_2}\cdots\mathfrak{l}_t^{(-1)^{s_t}e_t}]*E_A$, it suffices to apply each $\mathcal{Q}_{\ell_i}^{(L_i)}$ in turn to the appropriate registers of

$$|s_1,s_2,\dots,s_t\rangle|e_1,e_2,\dots,e_t\rangle|E_A\rangle|x_1,x_2,\dots,x_r\rangle|\mathbf{0}\rangle$$

where $L_i\geq e_i$. It is easy to see that this computes the correct value.

Remark 4.10. When we want to compute the action in superposition, we need to apply $\mathcal{Q}_\ell^{(L)}$ for L greater than *all* the e values supported in the superposition. For unknown states, this is not possible, but for our purposes it suffices to be able to compute the action for known superpositions; we can then choose L appropriately.

4.3.2 Constructing the States

In this subsection we show how to use the algorithms previously described to construct the states required to apply Kuperberg's algorithm.

Given curves $E = E_A : y^2 = x^3 + Ax^2 + x$ and $E' = E_B : y^2 = x^3 + Bx^2 + x$ where $E_B = [\mathfrak{a}] * E_A$, and $[\mathfrak{a}] = [\mathfrak{g}^a]$, Kuperberg's algorithm uses states of the form

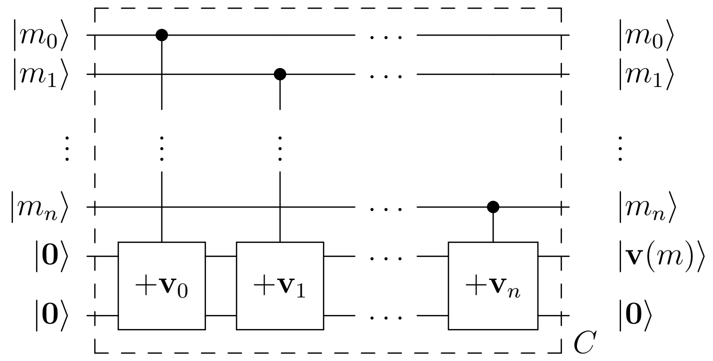
$$|\psi_k\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle + \exp\left(\frac{2\pi i a k}{N}\right)|1\rangle\right)$$

for k sampled uniformly at random from $\{0, 1, \ldots, N-1\}$.

Using the method described in Section 4.2 we can construct a table $\{(2^j, \mathbf{v}^{(j)}) \in \mathbb{N} \times \mathbb{Z}^t\}_{j=1}^n$ with $\|\mathbf{v}^{(j)}\|_\infty = 2^{O(n)}$ and

$$[\mathfrak{g}^{2^j}] = \left[\mathfrak{l}_1^{v_1^{(j)}} \mathfrak{l}_2^{v_2^{(j)}} \cdots \mathfrak{l}_t^{v_t^{(j)}}\right]$$

for $1 \leq j \leq N$. From this table, we construct the following quantum circuit, which converts from "cyclic notation" $[\mathfrak{g}^m]$ to "prime decomposition notation" $[\mathfrak{l}_1^{v_1}\mathfrak{l}_2^{v_2}\cdots\mathfrak{l}_t^{v_t}]$:



where $m = m_n \cdots m_1 m_0$ is the bit decomposition of m, and $\mathbf{v}(m)$ satisfies

$$[\mathfrak{g}^m] = [\mathfrak{l}_1^{v_1(m)}\mathfrak{l}_2^{v_2(m)}\cdots\mathfrak{l}_t^{v_t(m)}].$$

This circuit can be implemented in polynomial space using standard techniques. Using this circuit and the one from Section 4.3.1, we can give a complete algorithm for constructing the states $|\psi_k\rangle$.

1. Construct the state
$$|\Psi_0\rangle = \frac{1}{\sqrt{2N}} \sum_{m=0}^{N-1} |m\rangle|0\rangle|\mathbf{0}\rangle + |m\rangle|1\rangle|\mathbf{0}\rangle.$$

2. Apply C to the first and third registers above to obtain
$$|\Psi_1\rangle = \frac{1}{\sqrt{2N}} \sum_{m=0}^{N-1} |m\rangle|0\rangle|\mathbf{v}(m)\rangle|\mathbf{0}\rangle + |m\rangle|1\rangle|\mathbf{v}(m)\rangle|\mathbf{0}\rangle.$$

3. Apply the gate $|0,y\rangle \mapsto |0, y \oplus E_B\rangle, |1,y\rangle \mapsto |1, y \oplus E_A\rangle$ to the second and fourth registers above to obtain
$$|\Psi_2\rangle = \frac{1}{\sqrt{2N}} \sum_{m=0}^{N-1} |m\rangle|0\rangle|\mathbf{v}(m)\rangle|E_B\rangle|\mathbf{0}\rangle + |m\rangle|1\rangle|\mathbf{v}(m)\rangle|E_A\rangle|\mathbf{0}\rangle.$$

4. Apply the class group action gate to registers three, four, and five. If it is successful for each m, the resultant state is
$$|\Psi_3\rangle = \frac{1}{\sqrt{2N}} \sum_{m=0}^{N-1} \Big[|m\rangle|0\rangle|\mathbf{v}(m)\rangle|E_B\rangle|[\mathfrak{l}_1^{v_1(m)}\mathfrak{l}_2^{v_2(m)} \cdots \mathfrak{l}_t^{v_t(m)}] * E_B\rangle|\mathbf{0}\rangle$$
$$+ |m\rangle|1\rangle|\mathbf{v}(m)\rangle|E_A\rangle|[\mathfrak{l}_1^{v_1(m)}\mathfrak{l}_2^{v_2(m)} \cdots \mathfrak{l}_t^{v_t(m)}] * E_A\rangle|\mathbf{0}\rangle\Big].$$

5. Measure the fifth register. This gives a curve E_C, and the state will collapse to
$$|\Psi_4\rangle = \frac{1}{\sqrt{2}} \left(|m\rangle|0\rangle|\mathbf{v}(m)\rangle|E_B\rangle|\mathbf{0}\rangle + |m+a\rangle|1\rangle|\mathbf{v}(m+a)\rangle|E_A\rangle|\mathbf{0}\rangle\right),$$
and a satisfies $[\mathfrak{g}^a] * E_A = E_B$.

6. Apply C again to uncompute the $\mathbf{v}$-values, and discard the (now empty) third register to obtain
$$|\Psi_5\rangle = 2^{-\frac{1}{2}} \left(|m\rangle|0\rangle|E_B\rangle|\mathbf{0}\rangle + |m+a\rangle|1\rangle|E_A\rangle|\mathbf{0}\rangle\right)$$

for unknown random m.

7. Apply the Quantum Fourier Transform over $\mathbb{Z}/N\mathbb{Z}$ to the first register to obtain

$$|\Psi_6\rangle = \frac{1}{\sqrt{2N}} \sum_{k=0}^{N-1} \omega^{mk} |k\rangle|0\rangle|E_B\rangle|\mathbf{0}\rangle + \omega^{(m+a)k}|k\rangle|1\rangle|E_A\rangle|\mathbf{0}\rangle,$$

where $\omega = \exp\left(\frac{2\pi i}{N}\right)$.

8. Measure the first register to get a uniformly random k and the state

$$|\Psi_7\rangle = 2^{-\frac{1}{2}}\omega^{mk}\left(|0\rangle|E_B\rangle|\mathbf{0}\rangle + \omega^{ak}|1\rangle|E_A\rangle|\mathbf{0}\rangle\right).$$

9. Uncompute E_A and E_B using the gate from part (iii), and discard auxiliary qubits to yield

$$2^{-\frac{1}{2}}\omega^{mk}\left(|0\rangle + \omega^{ak}|1\rangle\right) \propto |\psi_k\rangle,$$

as required.

Remark 4.11. In step (iv) we evaluate the class group action on a uniform superposition over $\mathrm{cl}(\mathcal{O}) \times \mathbb{Z}/2\mathbb{Z}$ (with the second coordinate determining to which curve we apply the element of $\mathrm{cl}(\mathcal{O})$) with fixed randomness for each such input, using the prime decomposition presentation of the group elements. We find that the probability of evaluating the function correctly over the entire superposition is

$$\begin{aligned}\prod_{[\mathfrak{h}]\in\mathrm{cl}(\mathcal{O})} \prod_{k=1}^{t} \left(1 - \left(\frac{\ell_k+1}{2\ell_k}\right)^r\right)^{|v_k([\mathfrak{h}])|} &\geq \left(1 - \left(\frac{3}{4}\right)^r\right)^{N \max_{\mathfrak{h}\in\mathrm{cl}(\mathcal{O})} \|\mathbf{v}([\mathfrak{h}])\|_1} \\ &\geq 1 - \left(\frac{3}{4}\right)^r N \max_{\mathfrak{h}\in\mathrm{cl}(\mathcal{O})} \|\mathbf{v}([\mathfrak{h}])\|_1 \\ &\geq 1 - 2^{-\log_2 N + O\left(\sqrt{\log_2 N}\right)} \text{ for } r \geq 2(2-\log_2 3)\log_2 N;\end{aligned}$$

in particular, the success probability is negligibly different from 1 using only a polynomial number of random points $\{x_i\}_i$.

4.3.3 Using the States to Find the Hidden Shift

Now that we have a method to obtain states of the form $|\psi_k\rangle$ for uniformly random k, we can apply Regev's sieve [65] to extract the states $|\psi_1\rangle, |\psi_2\rangle, \ldots, |\psi_{2^{m-1}}\rangle$ where $m = \lceil \log_2 N \rceil$.

As discussed in Section 4.1.3, this method requires only polynomial quantum space. From here we proceed using [47, Remark 5.2]; we note that

$$\bigotimes_{k=0}^{m-1} |\psi_{2^k}\rangle = 2^{-\frac{m}{2}} \sum_{y=0}^{2^m-1} \omega^{ay}|y\rangle, \text{ and } F_N|a\rangle = N^{-\frac{1}{2}} \sum_{y=0}^{N-1} \omega^{ay}|y\rangle$$

where F_N is the quantum Fourier transform. Since these states have inner product $\frac{2^{\lfloor \log N \rfloor}}{N} = \Omega(1)$ and this inner product is preserved by the inverse Fourier transform, it follows that measuring $F_N^\dagger \bigotimes_{j=0}^{k-1} |\psi_{2^j}\rangle$ in the computational basis will yield a with probability $\Omega(1)$.

4.3.4 More Precise Time and Space Analysis

We briefly explain the time and space analysis of the algorithms.

The classical portions of the algorithm are:

1. Generating a subexponential number m of samples $\sum_{i=1}^{t} a_i e_i = k$, for small random e_i and known $a_i = \log_g[\mathfrak{l}_i]$.
2. Using a BKW-like algorithm to find an expression $\sum_{j=1}^{m} s_j k_j = 2^\ell$, with s_j subexponential and k_j from the given samples above. Its time and space complexity is $2^{O(\sqrt{\log N})}$.

As for the quantum portion, we have[1]:

1. Computing $N = |\mathrm{cl}(\mathcal{O})|$ requires time $O(\log_2 p)$ (assuming the Generalized Riemann Hypothesis) [7, Theorem 1.2].
2. For a given ℓ, Algorithm 1 runs in time $O(r\ell \cdot \mathrm{polylog}(p))$ and uses quantum space $O(r \cdot \mathrm{polylog}(p)) = O(\log N \cdot \mathrm{polylog}(p))$.
3. For Algorithm 2, we repeat Algorithm 1 $2^{O(\sqrt{\log N})}$ times. So this algorithm runs in time $O(r\ell_t \cdot \mathrm{polylog}(p))2^{O(\sqrt{\log N})} = 2^{O(\sqrt{\log N})}$ (for $r, \ell_t, \log p = 2^{O(\sqrt{\log N})}$) and space $O(\log N \cdot \mathrm{polylog}(p))$.
4. For Algorithm 3, apply Algorithm 2 t times. This requires time $2^{O(\sqrt{\log N})}$ (for $t = 2^{o(\sqrt{\log N})}$) and space $O(\log N \cdot \mathrm{polylog}(p))$.

[1]In each of these steps, the polylog(p) terms account for time required to implement field arithmetic and related algorithms (in the case of time complexity) and the associated scratch space required for those algorithms (in the case of space complexity).

5. Each sample in Regev's algorithm [65] invokes Algorithm 3 once, and so all our calls to Algorithm 3 take total time $2^{O(\sqrt{\log N \log\log N})}$. The space complexity is $O(\log^2 N \cdot \operatorname{polylog}(p) \cdot \log^2 \log N)$.

4.4 Related Work

There are a number of works that address the security of CSIDH, and give attacks similar to the one presented here. In particular, [8] and [11] were originally developed contemporaneously with our work, while [5] and [62] came slightly later and build upon our work, [8], and [11]. We discuss the differences between our work and those works, and explain how these works extend ours.

To begin, we consider [8]. As mentioned in Section 4.2 in this work the authors consider a different heuristic assumption on the structure of $\mathrm{cl}(\mathcal{O})$. This assumption, along with a different classical precomputation routine based on lattice reduction allows them to implement the hiding function defined in Equation 4.1 in time $2^{O(\sqrt[3]{\log N})}$ rather than $2^{O(\sqrt{\log N})}$. The asymptotic running time of their attack on CSIDH is essentially the same as in this chapter.

In [11], the authors give an attack with the same asymptotic running time, also based on Kuperberg's algorithm. They also perform a non-asymptotic analysis of their attack and conclude that none of the CSIDH parameter sets presented in [14] (CSIDH-512, CSIDH-1024, CSIDH-1792) achieve NIST level 1 security. They suggest that a 2260-bit prime (optimistically) or a 5280-bit prime (pessimistically) should be used to achieve that level of security.

In [62], the authors further develop the algorithm by generalizing the collimation sieve of [48] and applying it in the context of CSIDH. With this new development the authors suggest that CSIDH-512 can be broken using approximately 2^{60} T-gates (a standard measure of the complexity of quantum algorithms) which falls far short of even the most generous estimates required for NIST level 1 security; similarly, they show that CSIDH-1024 and CSIDH-1792 are only secure at NIST level 1 for optimistic values of MAXDEPTH.

Finally, the primary results of [5] are to develop improved quantum circuits for field arithemtic, isogeny evaluation, and other low-level components of the algorithm. In the course of this analysis, however, the authors claim that other works (particularly [11] and [62]) are too optimistic from the attacker's perspective, and thus that smaller primes than those recommended in [11] would be sufficient to provide NIST level 1 security.

4.5 Conclusion

We have presented a quantum algorithm for inverting the complex multiplication group action which requires only subexponential time and polynomial quantum space. Consequently we have shown that CSIDH—like the protocol of Couveignes—offers only subexponential security against quantum adversaries. In particular, while this attack has been known to exist in principle since the original publication of CSIDH [14], we have demonstrated how—under a mild heuristic asssumption on the distribution of primes of the form used in CSIDH—to implement the required subroutines efficiently on a quantum computer, and hence actually launch the attack.

Chapter 5

A Systematic Approach to Optimizing CSIDH

5.1 Introduction

In this chapter we present a framework for optimizing implementations of CSIDH, including both methods to optimize the algorithms used to compute the action of $\mathrm{cl}(\mathcal{O})$ on $\mathcal{E}\ell\ell_{GF(p)}(\mathcal{O})$, and to optimize system parameters—in particular, a vector of integers that defines the keyspace. We begin by covering required background from graph theory and mathematical optimization, followed by a discussion of previous works on CSIDH optimization—particularly SIMBA [56] and the two-point method [15]. Then we present our framework, and we give benchmarks for implementations that use our optimized algorithms and parameters. In Appendix B we give results in the direction of obtaining lower bounds on estimated running times for class group action evaluations which are derived from our optimization methods.

5.2 Non-Cryptographic Background

Our framework requires some fundamental knowledge of graph theory and mathematical optimization, which we present here.

5.2.1 Aspects of Graph Theory

A *directed graph* is a pair $G = (V, E)$, where $E \subseteq V^2$. The elements of V are called *vertices* and the elements of E are called *edges.* We use the notation $\mathcal{V}(G) = V$ and $\mathcal{E}(G) = E$. We frequently depict directed graphs by drawing a dot for each vertex and drawing an arrow from u to v if and only if $(u, v) \in \mathcal{E}(G)$. A directed graph G is *connected* if for any two vertices u, u' in the graph, there is a sequence $v_0, v_1, \ldots, v_n \in \mathcal{V}(G)$ such that

1. For $1 \leq i \leq n$, $(v_{i-1}, v_i) \in \mathcal{E}(G)$, and;
2. Either $u = v_0$ and $u' = v_n$, or $u' = v_0$ and $u = v_n$.

Such a sequence is called a $(u, u')-$*path* if $u = v_0$ and $u' = v_n$, or a $(u', u)-$*path* if $u' = v_0$ and $u = v_n$. Given a directed graph G, an *undirected cycle* in G is a list $v_0, v_1, \ldots, v_n$ of vertices of G satisfying

1. For $1 \leq i \leq n$, either $(v_{i-1}, v_i) \in \mathcal{E}(G)$ or $(v_i, v_{i-1}) \in \mathcal{E}(G)$;
2. With the exception of v_n, the vertices are pairwise distinct; and,
3. $v_0 = v_n$.

A *subgraph* H of a directed graph G is a directed graph satisfying $\mathcal{V}(H) \subseteq \mathcal{V}(G)$ and $\mathcal{E}(H) \subseteq \mathcal{E}(G)$. An *arborescence* S in a directed graph G is a connected subgraph which contains no undirected cycle. An arborescence is *spanning* if $\mathcal{V}(S) = \mathcal{V}(G)$. A *Steiner Arborescence* for root $r \in \mathcal{V}(G)$ and terminals $T \subseteq V(G)$ is one which contains a path from r to each $t \in T$.

For our work on lower bounds in Appendix B we will require the notion of a *flow network*:

Definition 5.1 (Flow Network)**.** A *flow network* is a tuple $(G = (V, E), \mathbf{c}, \mathbf{f})$ where:

1. G is a directed graph;
2. V has two distinguished elements s and t: the *source* and *terminal* of the flow, respectively;
3. $\mathbf{c} \in (\mathbb{R}_{\geq 0} \cup \{\infty\})^E$ is a vector of *capacities*;
4. $\mathbf{f} \in \mathbb{R}^E_{\geq 0}$ is a *flow* which is compatible with $\mathbf{c}$; that is, $\mathbf{f}$ satisfies
   (a) Compatibility: $f_e \leq c_e \;\forall\, e \in E$, and;

(b) Flow conservation:

$$\sum_{\substack{u \in V: \\ (u,v) \in E}} f_{(u,v)} = \sum_{\substack{w \in V: \\ (v,w) \in E}} f_{(v,w)} \ \forall\, v \in V \setminus \{s, t\}.$$

The following Lemma relates Steiner arborescences to flow networks.

Lemma 5.2. Let G be a directed graph, and let S be an arborescence in G. Let $r \in \mathcal{V}(S)$ and $T \subseteq \mathcal{V}(S)$. Construct S' from S by adding a new vertex t and edges (v, t) for all $v \in T$. Assign capacities to S' as

$$\mathbf{c}_{(u,v)} = \begin{cases} 1 & \text{if } u \in T \text{ and } v = t \\ \infty & \text{otherwise} \end{cases}.$$

Then, S is a Steiner arborescence for r and T if and only if S' admits a flow $\mathbf{f}$ with source r, sink t, which is compatible with $\mathbf{c}$ and for which

$$\sum_{v \in T} f_{(v,t)} \geq |T|.$$

Proof. First suppose that S is a Steiner arborescence with root r and terminals T. Begin by setting $\mathbf{f} = \mathbb{0}$; then for each $v \in T$, for each edge e in the unique path from r to v, set $f_e \leftarrow f_e + 1$, and set $f_{(v,t)} = 1$. It is clear that $\mathbf{f}$ is a flow with source r and sink t, since $\mathbb{0}$ is a flow, and for each non-root u vertex in each (r, v)-path, we always add 1 to the in-flow of u and 1 to its outflow. As well, $\mathbf{f}$ is compatible with $\mathbf{c}$ since each $f_{(v,t)} = 1 = c_{(v,t)}$, and

$$\sum_{v \in T} f_{(v,t)} = \sum_{v \in T} 1 = |T|$$

as required.

For the other direction, suppose that $\sum_{v \in T} f_{(v,t)} \geq |T|$. Since the capacity at each of the $|T|$ edges into t is exactly 1, it follows that $f_{(v,t)} = 1$ for all $v \in T$, since otherwise the total flow into t cannot reach $|T|$. Since r is the source, this immediately implies that there is a path from r to v for each $v \in T$. Since S is an arborescence by assumption, it follows that S is in fact a Steiner arborescence with root r and terminals T. □

5.2.2 Aspects of Mathematical Optimization

In this section we present standard mathematical optimization terminology and definitions.

Definition 5.3 (Mathematical Program, Decision Variables, Objective Function, Constraints)**.** A problem of the form

$$\begin{aligned} \text{Minimize} \quad & f(x) \\ \text{Subject to} \quad & g(x) = 0 \\ & h(x) \leq 0 \\ & x \in \mathcal{X} \subseteq \mathbb{R}^n \end{aligned} \tag{P}$$

is called a *mathematical program.* In a mathematical program of the form (P), we have the following terminology:

1. x are the *decision variables;*
2. f is the *objective function;*
3. $g(x) = 0$ are the *equality constraints;*
4. $h(x) \leq 0$ are the *inequality constraints;*
5. The equality constraints, inequality constraints, and $x \in \mathcal{X}$ are, together, the *constraints;* and,
6. The set of all x which satisfy the constraints is the *feasible region.*

Definition 5.4 (Linear Program, Standard Equality Form)**.** A mathematical program of the form (P) is a *linear program* if

1. f is a linear function of x;
2. g and h are affine functions of x; and,
3. $\mathcal{X} = \mathbb{R}^n$.

A generic linear program can be written in the form

$$\begin{aligned} \text{Minimize} \quad & \langle c, x \rangle \\ \text{Subject to} \quad & A_{\text{eq}} x = b_{\text{eq}} \\ & Ax \leq b \end{aligned} \tag{LP}$$

for some matrices $A_{\text{eq}} \in \mathbb{R}^{m_{\text{eq}} \times n}$, $A \in \mathbb{R}^{m \times n}$ and vectors $b_{\text{eq}} \in \mathbb{R}^{m_{\text{eq}}}$, $b \in \mathbb{R}^m$. When $A = -I_{n \times n}$ and $b = \mathbb{0}$, we say that the program is in *standard equality form (SEF-LP).*

5.3 A Framework for Group Action Algorithms

For $n \in \mathbb{N}$, let T_n denote the directed graph with

$$\mathcal{V}(T_n) = \{v \in \mathbb{N}^2 : \|v\|_1 \leq n\} \text{ and } \mathcal{E}(T_n) = \{(u, v) : \|u - v\|_1 = 1, \|u\|_1 < \|v\|_1\};$$

that is, the vertices of T_n are those integer points in the region of the plane bounded by the lines $x = 0$, $y = 0$, and $x + y = n$, with two vertices adjacent if and only if they differ by $(1, 0)$ or $(0, 1)$. We call edges of the first form *horizontal* and edges of the second form *vertical.* Each horizontal edge is directed to the right (*i.e.,* from u to $u + (1, 0)$) and each vertical edge is directed up (*i.e.,* from u to $u + (0, 1)$).

Definition 5.5 (Strategy; Well-formed Strategy)**.** A *strategy of size* n is a subgraph S of T_n such that:

1. $\mathcal{V}(S) = \mathcal{V}(T_n)$;
2. For each non-isolated vertex $v \in \mathcal{V}(S)$, there is a path from $(0, 0)$ to v; and,
3. S contains a path from $(0, 0)$ to each vertex in $L = \{v \in \mathbb{N}^2 : \|v\|_1 = n\}$ of $\mathcal{V}(T_n)$.

A strategy S is *well-formed* if for any directed edge $e \in \mathcal{E}(S)$, there is a vertex $v \in L$ such that the graph obtained from S by deleting e no longer contains a path from $(0, 0)$ to v.

Remark 5.6. Any well-formed strategy of size n is an arborescence; in particular, it is a Steiner arborescence for T_n with root $(0, 0)$ and terminals L.

As in [23], we are particularly interested in *canonical* strategies, which we define here. When S_1 and S_2 are strategies of size n_1 and n_2 respectively, define $S_1\#S_2$ to be the strategy of size $n_1 + n_2$ with

$$\begin{aligned}\mathcal{E}(S_1\#S_2) = \{((x, 0), (x + 1, 0)) : 0 \leq x \leq n_1 - 1\} &\cup \{((0, y), (0, y + 1)) : 0 \leq y \leq n_2 - 1\} \\ &\cup \{(u + (n_1, 0), v + (n_1, 0)) : (u, v) \in \mathcal{E}(S_2)\} \\ &\cup \{(u + (0, n_2), v + (0, n_2)) : (u, v) \in \mathcal{E}(S_1)\}.\end{aligned}$$

Intuitively, if we picture $S_1\#S_2$ as being embedded in the plane, then it consists of:

1. A path from $(0, 0)$ to $(n_1, 0)$;
2. A path from $(0, 0)$ to $(0, n_2)$;

3. A copy of S_1, shifted up n_2 units; and,

4. A copy of S_2, shifted right n_1 units.

The operation # is both non-commutative and non-associative. We use # in our definition of canonical strategy.

Definition 5.7 (Canonical Strategy)**.** A strategy S of size n is *canonical* if it is equal to some parenthization of $\underbrace{T_1 \# T_1 \# \cdots \# T_1}_{n \text{ times}}$.

It is clear that any two parenthizations yield different strategies, and so it is immediate that the number of canonical strategies of size n is $c_n = \frac{1}{n+1}\binom{2n}{n}$: the n^{th} Catalan number.

We have the following equivalent characterization of canonical strategies:

Lemma 5.8 (A Characterization of Canonical Strategies)**.** A strategy S of size n is *canonical* if and only if

1. $n = 1$; or,

2. $n > 1$ and $S = S_1 \# S_2$, where S_1 and S_2 are canonical.

In the second case we call S_1 and S_2 the *left* and *right substrategies* of S, respectively. We write $S^L = S_1$ and $S^R = S_2$.

Proof. In the case that $n = 1$ the result is clear. Suppose that $n > 1$ and that $S = S_1 \# S_2$ with S_1 and S_2 canonical and of sizes n_1 and n_2, respectively; clearly $n = n_1 + n_2$. Since S_1 and S_2 are parenthizations of $\underbrace{T_1 \# T_1 \# \cdots \# T_1}_{n_1 \text{ times}}$ and $\underbrace{T_1 \# T_1 \# \cdots \# T_1}_{n_2 \text{ times}}$, respectively, it immediately follows that $S = S_1 \# S_2$ is a parenthization of $\underbrace{T_1 \# T_1 \# \cdots \# T_1}_{n \text{ times}}$, as required.

For the converse direction, suppose that S is a parenthization of $\underbrace{T_1 \# T_1 \# \cdots \# T_1}_{n \text{ times}}$. If this expression does not begin with an open parenthesis, then it is necessarily of the form $T_1 \# S_2$ where S_2 is some parenthization of $\underbrace{T_1 \# T_1 \# \cdots \# T_1}_{n-1 \text{ times}}$, and the result follows immediately. Otherwise this expression begins with an open parenthesis; suppose that the corresponding close parenthesis occurs after n_1 instances of T_1. Then either $n_1 = n - 1$ and the result follows in much the same way as when $n_1 = 1$, or the remaining $n_2 = n - n_1$ occurrences of T_1 must occur in another pair of parentheses; in sum, we have an expression

$$S = (S_1)\#(S_2)$$

where S_1 and S_2 are parenthizations of $\underbrace{T_1\#T_1\#\cdots\#T_1}_{n_1 \text{ times}}$ and $\underbrace{T_1\#T_1\#\cdots\#T_1}_{n_2 \text{ times}}$, respectively; that is, $S = S_1\#S_2$ where S_1 and S_2 are canonical strategies, as required. □

This alternative characterization of canonical strategies is useful when we devise an algorithm which finds optimal canonical strategies. Figure 5.1 depicts a strategy of size 9 as a subgraph of T_9, and highlights its left and right substrategies.

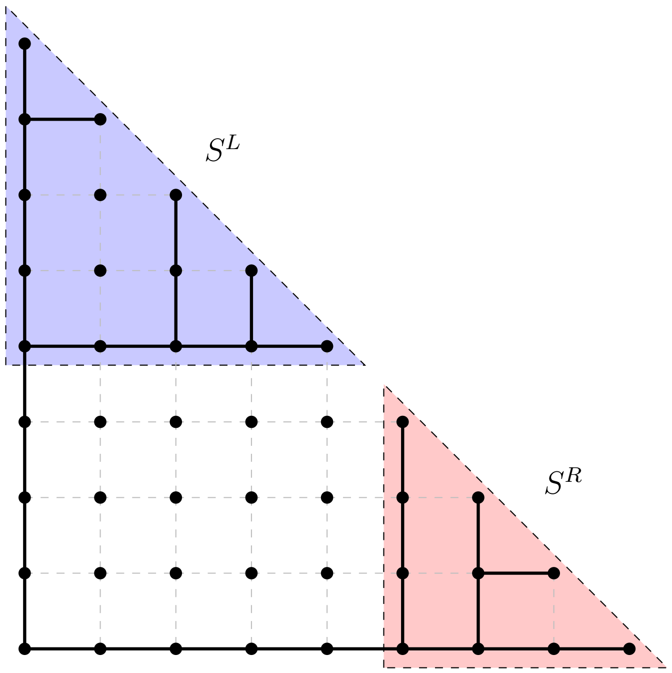


Figure 5.1: A strategy S of size 9 (black edges) and its left and right substrategies (blue and red shaded regions, respectively) embedded in T_9 (vertices, black edges, and dashed grey edges). For visual simplicity, we do not explicitly draw the direction of the edges; horizontal edges are directed to the right and vertical edges are directed up.

5.3.1 Generalized Measures

In [23], weights are assigned to graphs T_n by *measures*. We require a more general notion of measure than is present there, and so we introduce *generalized* measures.

Definition 5.9 (Generalized Measure)**.** A *generalized measure of size* n (or, simply *measure*) is a tuple $M = ((p_j)_{j=1}^n, f_H, f_V)$ where

1. $(p_j)_{j=1}^n$ is a sequence of non-negative real numbers, and;
2. $f_H, f_V \colon \mathbb{R}_+ \to \mathbb{R}_+$ are a pair of *weight functions.*

Definition 5.9 reduces to that of [23, Section 4.2] by taking $(p_j)_{j=1}^n$ to be a constant sequence or by taking f_H and f_V to be constant functions. As in [23], a measure $M = ((p_j)_{j=1}^n, f_H, f_V)$ of size n induces a weighting of T_n in the following way:

1. Each horizontal edge $((i,j),(i+1,j))$ gets weight $f_H(p_{i+1})$, and;
2. Each vertical edge $((i,j),(i,j+1))$ gets weight $f_V(p_{n-j+1})$.

We define the action of $\mathrm{Sym}(n)$ on the set $\mathcal{M}_n$ of generalized measures of size n as follows: if $\sigma \in \mathrm{Sym}(n)$ and $M = ((p_j)_{j=1}^n, f_H, f_V)$, then $\sigma \cdot M = ((p_{\sigma(j)})_{j=1}^n, f_H, f_V)$.

5.3.2 The Connection to CSIDH

Recall that the most basic operation required to execute CSIDH is to compute $[\mathfrak{l}_i] * E$ where ℓ_i is a small prime and $\mathfrak{l}_i = ([\ell_i], \pi_p - \iota_E)$. This computation is most efficiently done by finding a generator of the isogeny $\phi_{\mathfrak{l}_i}$ whose kernel is $\ker \phi_{\mathfrak{l}_i} = E(GF(p))[\ell_i]$; that is, a $GF(p)$-rational point $Q_{\mathfrak{l}_i}$ of E of order ℓ_i. The most straightforward method to recover such a $Q_{\mathfrak{l}_i}$ and hence compute $[\mathfrak{l}_i] * E$ is given in Algorithm 4.

If it were sufficient to compute $[\mathfrak{l}_i] * E$ for just one value of i, this would be the end of the story. However, in general we must compute $[\prod_{i=1}^n \mathfrak{l}_i^{e_i}] * E$ where each $e_i \in \{0,1\}$. The most obvious method to do so is to simply execute Algorithm 4 repeatedly, updating the curve each time. This approach requires multiplying points by each of the n primes $n-1$ times each. In [14] the authors improve on this method, instead using Algorithm 5.

In the j^{th} step of Algorithm 5, the point P is multiplied by the first $n-j$ primes to obtain a point of order ℓ_{n-j+1}, used to construct an ℓ_{n-j+1}-isogeny. Compared with n applications

Algorithm 4: An algorithm to compute $[\mathfrak{l}_i] * E$

```
input : $E$ a supersingular elliptic curve, $\ell_1, \ell_2, \ldots, \ell_n$ small primes, an index $1 \leq i \leq n$
output: $[\mathfrak{l}_i] * E$
1 $p = 4\ell_1\ell_2 \cdots \ell_n - 1$
2 $P \xleftarrow{\$} E(GF(p))$
3 $Q_{\mathfrak{l}_i} \leftarrow [4\ell_1\ell_2 \cdots \ell_{i-1}\ell_{i+1} \cdots \ell_n]P$
4 if $Q_{\mathfrak{l}_i} = \infty$ then
5     GOTO 2
6 end
7 else
8     return $E/\langle Q_{\mathfrak{l}_i} \rangle$
9 end
```

Algorithm 5: An algorithm to compute $[\mathfrak{l}_1^{e_1}\mathfrak{l}_2^{e_2} \cdots \mathfrak{l}_n^{e_n}] * E$[1]

```
input : $E$, $\ell_1, \ell_2, \ldots, \ell_n$ small primes, an index $1 \leq i \leq n$, a vector $\mathbf{e} \in \{0,1\}^n$
output: $[\mathfrak{l}_1^{e_1}\mathfrak{l}_2^{e_2} \cdots \mathfrak{l}_n^{e_n}] * E$
1 $p = 4\ell_1\ell_2 \cdots \ell_n - 1$
2 $P \xleftarrow{\$} E(GF(p))$ of order $\ell_1\ell_2 \cdots \ell_n$
3 for $i = n, n-1, \ldots, 1$ do
4     $Q_{\mathfrak{l}_i} \leftarrow [\ell_1\ell_2 \cdots \ell_{i-1}]P$
5     if $e_i = 1$ then
6         $\phi_{[\mathfrak{l}_i]} \leftarrow$ the isogeny with kernel $\langle Q_{\mathfrak{l}_i} \rangle$
7         $E \leftarrow \phi_{[\mathfrak{l}_i]}(E)$
8         $P \leftarrow \phi_{[\mathfrak{l}_i]}(P)$
9     end
10 end
11 return $E$
```

of Algorithm 4, Algorithm 5 trades $\binom{n}{2}$ point multiplications for n isogeny evaluations. Though asymptotically isogeny evaluations cost far more than point multiplications, for the proposed CSIDH parameter sets this tradeoff is favourable.

[1]Note that, in reality, P is chosen uniformly, and if some ℓ_i does not divide its order, that isogeny is skipped and done later. For brevity of exposition in this section, we make the simplifying assumption that P has the required order.

We can visualize the operations that are applied to P on a labelled copy of T_n, as depicted in Figure 5.2 (for the case $n = 9$). It is immediately apparent that these operations and their outputs—interpreted as edges and vertices in T_n, respectively—form a strategy; we call this strategy the *multiplication-based* strategy, since it uses the greatest number of multiplication edges among all strategies of a fixed size.

But there is no reason that the multiplication-based strategy is the only possibility for computing the required points; *any* strategy of size n yields an algorithm for computing $Q_{\mathfrak{l}_1}, Q_{\mathfrak{l}_2}, \ldots, Q_{\mathfrak{l}_n}$, and *vice versa.* This is the connection between abstract strategies as defined in Section 5.3, and CSIDH.

5.3.3 The Connection to Optimizing Implementations of CSIDH

So far, we have made no explicit connection to optimizing the runtime of implementations of CSIDH. Of course, to follow each edge in a strategy incurs some cost: the cost of evaluating some $[\ell_i]$ or some $\phi_{[\mathfrak{l}_i]}$. In principle this "cost" should be the running time of the algorithm for evaluating these maps; however, this is of course platform- and hardware-dependent. A reasonable proxy for running time is the number of underlying field operations (in $GF(p)$) required to implement the maps. We consider three basic field operations: multiplication $\mathbf{M}$, squaring $\mathbf{S}$, and addition $\mathbf{a}$. Table 5.1 contains the number of these operations required for evaluating the maps $[\ell_i]$ and $\phi_{\mathfrak{l}_i}$. We note that this table does not account for the improved isogeny evaluation algorithms of [4], which require less time asymptotically than is listed in Table 5.1. The reason for this is twofold: first, the values in Table 5.1 are representative of the algorithms used in prior implementations of CSIDH-512 (ours included), and thus are the relevant costs for our purposes. Second, the authors of [4] do not provide exact field operation counts for their improved algorithms, and so we can not easily model their cost in the way required for this work.

Using these formulae, given CSIDH primes $(\ell_i)_{i=1}^n$, we define a measure

$$\begin{aligned}
M_{\vec{\ell},\alpha,\beta} &= ((\ell_i)_{i=1}^n, f_H, f_V) \text{ where} \\
f_H(\ell) &= (8\lceil\log_2 \ell\rceil - 4) + \alpha(4\lceil\log_2 \ell\rceil - 2) + \beta(8\lceil\log_2 \ell\rceil - 6) \\
&= (8 + 4\alpha + 8\beta)\lceil\log_2 \ell\rceil - (4 + 2\alpha + 6\beta), \text{and} \\
f_V(\ell) &= 2\ell - 2 + 2\alpha + \beta(\ell + 1) \\
&= (2 + \beta)\ell - (2 - 2\alpha - \beta)
\end{aligned}$$

where α and β are positive real numbers which are the ratios of the cost of $\mathbf{S}$ and $\mathbf{a}$ against $\mathbf{M}$, respectively. If we assign the corresponding weights to T_n according to the

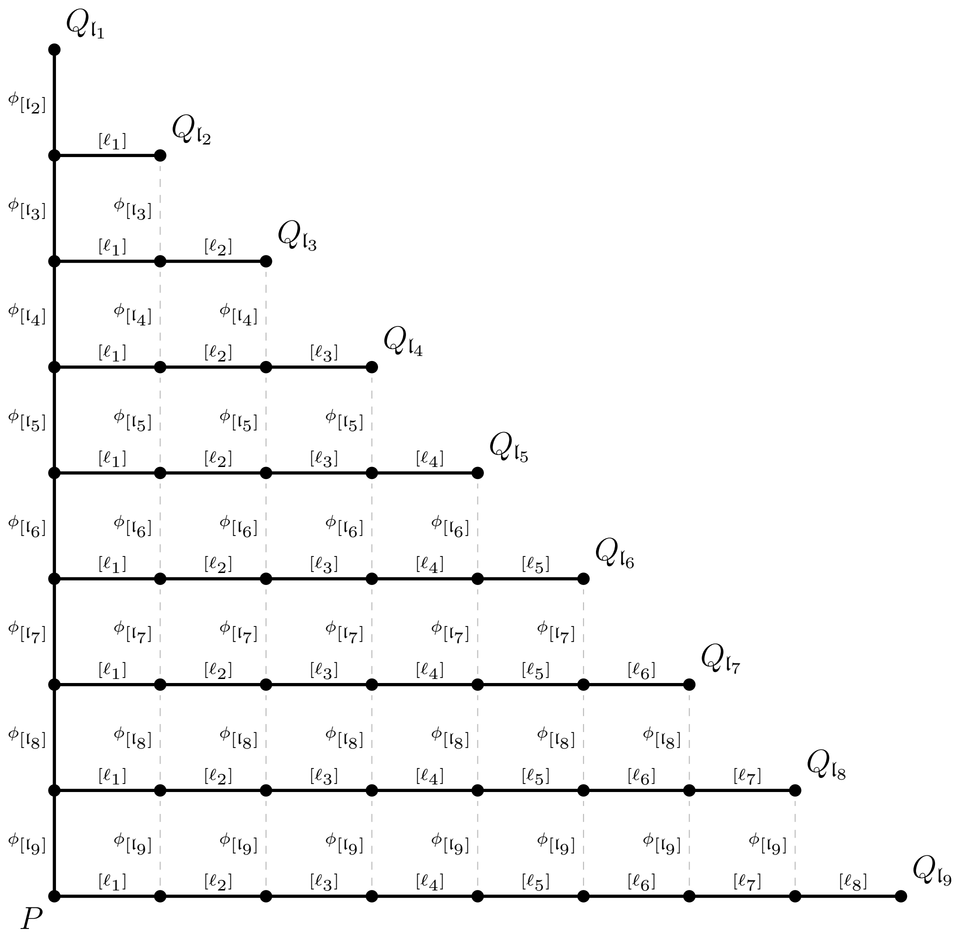


Figure 5.2: The multiplication-based strategy of size 9 (black edges) embedded in T_9 (vertices, black edges, and dashed grey edges). Vertices in the graph correspond to points on elliptic curves, horizontal edges to multiplication by a constant, and vertical edges to isogeny evaluations. The points P and $Q_{\mathfrak{l}_1}, Q_{\mathfrak{l}_2}, \ldots, Q_{\mathfrak{l}_9}$ from Algorithm 5 are labelled.

Operation:	**M**	**S**	**a**
Evaluating $[\ell_i]$	$8\lceil \log_2 \ell_i \rceil - 4$	$4\lceil \log_2 \ell_i \rceil - 2$	$8\lceil \log_2 \ell_i \rceil - 6$
Determining $\ker \phi_{\mathfrak{l}_i}$	$2\ell_i - 6$	$\ell_i - 3$	$3\ell_i - 11$
Determining $\operatorname{codom} \phi_{\mathfrak{l}_i}$	$\ell_i + \lceil \log_2 \ell_i \rceil - 1$	$2\lceil \log_2 \ell_i \rceil + 6$	6
Evaluating $\phi_{\mathfrak{l}_i}$	$2\ell_i - 2$	2	$\ell_i + 1$

Table 5.1: $GF(p)$ operation counts for some algorithms required to evaluate $*$. Note that the multiplication, squaring, and addition counts for the determining $\operatorname{codom} \phi_{[\mathfrak{l}_i]}$ are those that are required *beyond* determining $\ker \phi_{[\mathfrak{l}_i]}$, and that the counts for evaluating $\phi_{[\mathfrak{l}_i]}$ are those that are required *beyond* determining $\operatorname{codom} \phi_{[\mathfrak{l}_i]}$ and determining $\ker \phi_{[\mathfrak{l}_i]}$. We separate out these components because, for a given isogeny, we must determine its kernel and codomain only once, even if it is to be evaluated many times.

rules described in Section 5.3.1, then the total weight of any strategy in this weighted T_n is the number of $GF(p)$ multiplications (and equivalent operations) required to execute the strategy.

It remains to consider permuted measures. This assignment of weights to T_n implicitly takes a fixed ordering on the primes (as specified in [14], this ordering is from smallest to largest, so that the isogenies are constructed in order of largest to smallest degree), but there is no reason that the isogenies must be constructed in this order; *any* ordering of the primes is valid. To find the globally-optimal CSIDH implementation, one must optimize both the strategy and permutation of the primes simultaneously.

5.4 Prior Work

Two major optimizations for evaluating the complex multiplication action appear in the literature at present: splitting isogenies into multiple batches (SIMBA) [56] and the two-point technique [59]; we present them in this section. These two methods are complementary to one another and also to our optimizations—all three can be used simultaneously to best optimize a CSIDH implementation. We also detail how our optimizations complement the existing ones.

5.4.1 Splitting Isogenies into Multiple Batches—SIMBA

As the name implies, the fundamental idea of SIMBA is that we need not restrict ourselves to algorithms that use "full" strategies; that is, strategies which choose one point P at the beginning, and then map to each point $Q_{\mathfrak{l}_n}, Q_{\mathfrak{l}_{n-1}}, \ldots, Q_{\mathfrak{l}_1}$. Instead, we can split the primes into batches $S^{(i)} = \{\ell_1^{(i)}, \ldots, \ell_{n_i}^{(i)}\}$, for $1 \leq i \leq m$, and execute a strategy on each batch (starting with a fresh random point for each batch). In the nomenclature of [56], the specification of SIMBA-m-M[2] is as follows:

1. $S^{(i)} = \{\ell_j : j \equiv i \pmod{m}\}$ for $1 \leq i \leq m$.
2. At each of the first M steps, for each i, a fresh random point P is chosen and the point $P_i = [4 \prod_{\ell_k \notin S^{(i)}} \ell_k]P$ is constructed. Then, the multiplication-based strategy on S_i is executed.
3. For the remaining rounds, the multiplication-based strategy on the remaining primes is executed until all the required isogenies have been computed.

Notably, the overall order of the primes (and hence the partition sets and the order in which the primes appear in each multiplication-based strategy) is not specified. Moreover, the authors do not consider strategies other than the multiplication-based strategy. A natural generalization of our technique for finding the optimal permutation for full CSIDH strategies applies to SIMBA-m-M for any m and M. Moreover, for a given permutation our algorithm for finding an optimal strategy applies without modification to each batch of primes.

5.4.2 The Two-Point Technique

In the original specification of CSIDH, parties' private keys could include both positive and negative exponents. To evaluate $[\mathfrak{l}_i^{-1}] * E$ requires a point $\overline{Q}_{\mathfrak{l}_i} \in E$ which is of order ℓ_i and $GF(p^2)$-rational but *not* $GF(p)$-rational. In the original specification, $GF(p^2)$-rational points are chosen uniformly at random and, if they are $GF(p)$-rational, they are used in a multiplication-based strategy to compute isogenies of degrees corresponding to positive exponents, while if they are not, they are used in a multiplication-based strategy for those primes whose corresponding exponents are negative. In this regime, empirically it has been found that using only positive exponents is more efficient, as the same strategy and permutation of the primes can be used for *any* private key. In [59], Onuki *et al.* find improved performance by tracking two points through each strategy: one from $E[\pi_p - [1]]$

[2]The original notation is SIMBA-m-μ; we use alternate notation to avoid confusion with our use of $\boldsymbol{\mu}$: the vector of multiplication costs.

and one from $E[\pi_p + [1]]$. 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 [38], and we expect similar results to hold in this setting. We leave this for future examination and do not pursue this idea further in this work.

5.5 Optimization Techniques for CSIDH

We saw in Section 5.3 a framework for describing algorithms for evaluation of the class group action for CSIDH. In this section we present three results in the direction of actually optimizing CSIDH; in particular, we give

1. A linear programming formulation of the problem of finding the optimal permutation for a given set of primes and strategy;
2. A dynamic programming algorithm which finds the optimal strategy for a given set of primes and permutation; and,
3. An iterative convex programming approach to finding an approximately optimal bound vector.

We note that techniques 1 and 2 are used to optimize the algorithm for evaluating the action of $\mathrm{cl}(\mathcal{O})$ on $\mathcal{E}\ell\ell_{GF(p)}(\mathcal{O})$, and thus have applications outside of CSIDH, while technique 3 is a "protocol-level" optimization, and is essentially specific to CSIDH.

5.5.1 Encoding Strategies

In earlier works which explicitly deal with strategies [23, in particular], strategies are encoded as explicit parenthesization of $\underbrace{T_1 \# T_1 \# \cdots \# T_1}_{n_1 \text{ times}}$. While this encoding is convenient in certain applications, it is not particularly convenient for mathematical optimization. For our purposes we instead encode strategies S of size n as pairs of lower triangular matrices $H(S), V(S) \in \mathbb{R}^{(n-1)\times(n-1)}$. The matrices are defined as follows:

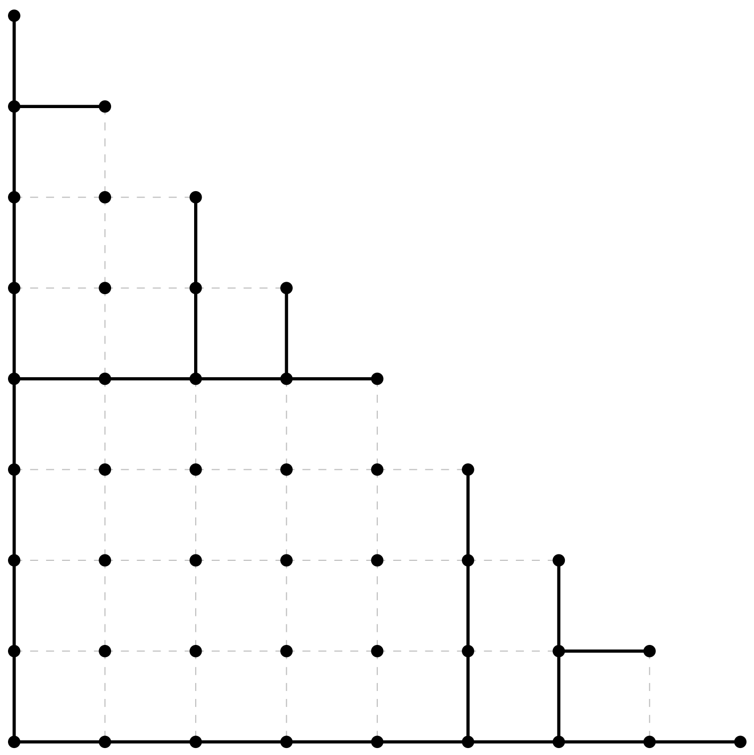
$$H(S)_{ij} = \begin{cases} 1 & \text{if } ((i-1, n-j-1), (i, n-j-1)) \in \mathcal{E}(S) \\ 0 & \text{otherwise} \end{cases}, \text{ and}$$

$$V(S)_{ij} = \begin{cases} 1 & \text{if } ((i-1, n-j), (i-1, n-j+1)) \in \mathcal{E}(S) \\ 0 & \text{otherwise} \end{cases}.$$

More intuitively, consider the subgraphs S_H and S_V of S, obtained from S by keeping only the horizontal and only the vertical edges, respectively. Consider the embedding of S_H in the plane; there are $n-1$ rows of $n-1$ "potential" horizontal edges, in which some edges are present. We then simply fill $H(S)$ with 1s in the positions where the corresponding edge is present, and 0s in the remaining positions, in the same orientation as in the embedding of S_H in the plane; we construct $V(S)$ from S_V analogously. An example of this construction is depicted in Figure 5.3.

If S is a canonical strategy, we can instead build up its strategy matrices $H(S)$ and $V(S)$ from $H(S_L)$ and $H(S_R)$, and $V(S_L)$ and $V(S_R)$, respectively. If we define $H(T_1)$ and $V(T_1)$ to be empty matrices, when $|S^L| = n_1$ and $|S^R| = n_2$, recursively we have

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



$$H(S) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \qquad V(S) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

Figure 5.3: A strategy S of size 9 (black edges) embedded in T_9 (vertices, black edges, and dashed grey edges) and its corresponding matrices $H(S)$ and $V(S)$.

$$H(T_1\#S^R) = \left[\begin{array}{c:c} 0 & \mathbb{0} \\ \hdashline \mathbf{e}_{n_2} & H(S^R) \end{array}\right] \qquad V(T_1\#S^R) = \left[\begin{array}{c:c} 1 & \mathbb{0} \\ \hdashline \mathbb{1} & V(S^R) \end{array}\right]$$

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

where $\mathbb{0}$ and $\mathbb{1}$ are matrices of 0s and 1s of the appropriate sizes, and $\mathbf{e}_j$ is the j^{th} standard basis vector of the appropriate length.

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

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

where $H(S_i)$ and $V(S_i)$ are the encoding matrices of the S_i as defined in Section 5.5.1. Then, for a SIMBA strategy $S = (S_1, S_2, \ldots, S_m)$ on $m \geq 3$ substrategies, we define

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

where $S' = (S_1, S_2, \ldots, S_{m-1})$. This definition is used to make the optimization problems in Sections 5.5.2 and 5.5.4 compatible with SIMBA strategies.

5.5.2 A Linear Program for an Optimal Permutation

Fix a measure $M = ((\ell_i)_{i=1}^n, f_H, f_V)$. Let $C_M(S, \sigma)$ denote the cost of strategy S under measure M when the primes are ordered according to σ. At the highest level, the problem of finding an optimal permutation for a given strategy S is

$$\begin{array}{rl} \text{Minimize} & C_M(S,\sigma) \\ \text{Subject to} & \sigma \in \mathrm{Sym}(n) \end{array} \tag{5.1}$$

where $\mathrm{Sym}(n)$ is the symmetric group on n symbols. To obtain an LP formulation of the problem, we first replace the permutation σ by a permutation matrix Σ given by

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

where $\{\mathbf{e}_i\}_i$ are the standard basis vectors. To write $C_M(S, \Sigma)$ as a linear function of Σ, first define

$$T_L = \left[I_{n-1} | \mathbb{0}\right] \text{ and } T_R = \left[\mathbb{0} | I_{n-1}\right]$$

and

$$\boldsymbol{\mu} = [f_H(\ell_i)]_{i=1}^n \text{ and } \boldsymbol{\iota} = [f_V(\ell_i)]_{i=1}^n.$$

Then

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

Finally we must enforce that Σ is a permutation matrix. In particular, a permutation matrix is an integer doubly-stochastic matrix, and so we have

$$\Sigma \text{ is a permutation matrix} \iff \begin{cases} \Sigma\mathbb{1} = \mathbb{1} \\ \mathbb{1}^T\Sigma = \mathbb{1}^T \\ \Sigma \geq 0 \\ \Sigma \in \mathbb{Z}^{n\times n} \end{cases}.$$

Altogether, we have the following formulation of the optimal permutation problem:

$$\begin{array}{rl} \text{Minimize} & \langle T_L^T H^T \mathbb{1}\boldsymbol{\mu}^T + T_R^T V \mathbb{1}\boldsymbol{\iota}^T, \Sigma\rangle_F \\ \text{Subject to} & \Sigma\mathbb{1} = \mathbb{1} \\ & \mathbb{1}^T\Sigma = \mathbb{1}^T \\ & \Sigma \geq 0 \\ & \Sigma \in \mathbb{Z}^{n\times n} \end{array} \tag{OPP-IP}$$

Of course, problem (OPP-IP) is an integer linear program rather than a linear program. We can relax away the integrality constraint to obtain a true linear program:

$$\begin{array}{rl} \text{Minimize} & \langle T_L^T H^T \mathbb{1}\boldsymbol{\mu}^T + T_R^T V \mathbb{1}\boldsymbol{\iota}^T, \Sigma\rangle_F \\ \text{Subject to} & \Sigma\mathbb{1} = \mathbb{1} \\ & \mathbb{1}^T\Sigma = \mathbb{1}^T \\ & \Sigma \geq 0 \end{array} \tag{OPP-LP}$$

The feasible region of (OPP-LP) is the *Birkhoff polytope* of (ambient) dimension $n \times n$:

$$B_n = \{\Sigma \in \mathbb{R}^{n\times n} \,:\, \Sigma\mathbb{1} = \mathbb{1}, \mathbb{1}^T\Sigma = \mathbb{1}^T, \Sigma \geq 0\}.$$

The Birkhoff polytope is *integral*: all its vertices have integer coordinates. By the Fundamental Theorem of Linear Programming [34, Theorems 2.11 and 2.12], (OPP-LP) has a solution which is a vertex of B_n, and hence integral, and hence a solution to (OPP-IP). Thus to solve problem (OPP-IP), we simply apply the simplex method [34, Chapter 2] or the Hungarian algorithm [46] to problem (OPP-LP) to obtain an optimal vertex.

An Extension to SIMBA Strategies

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

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

where m is the number of SIMBA substrategies of S. Notably the additional term $(m-1)\mathbb{1}^T \boldsymbol{\mu}$ is independent of the decision variables Σ, and so we can use Problem (OPP-LP) 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 (OPP-LP) 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 ith 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 ith 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 modified strategy matrices $\hat{H}(S)$ and $\hat{V}(S)$ by

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

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

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

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 (OPP-LP) 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.

5.5.3 An Algorithm for an Optimal Canonical Strategy

Let S^*_M denote the canonical strategy of smallest weight with respect to a measure M of size n. In [23, Section 4, Equation (5)], in the context of SIDH, the authors present the recurrence relation for the cost of S^*_M for measures M of the form $M = ((\ell_i)_{i=1}^n, f_H, f_V)$ where $(\ell_i)_{i=1}^n$ is a constant sequence $(\ell)_{i=1}^n$:

$$C_M(S^*_M) = \min_{1\le k<n} \left\{ C_{M_k}(S^*_{M_k}) + C_{M'_k}(S^*_{M'_k}) + (n-k)f_H(\ell) + kf_V(\ell) \right\} \tag{5.2}$$

where for each k, $M_k = ((\ell_i)_{i=1}^k, f_H, f_V)$ and $M'_k = ((\ell_i)_{i=k+1}^n, f_H, f_V)$. Intuitively, this says that the cost of a minimal strategy is the minimum over all possible divisions into left and right of the cost of a minimal left substrategy ($C_{M'_k}(S^*_{M'_k})$) plus the cost of a minimal right substrategy ($C_{M_k}(S^*_{M_k})$) plus the cost of joining them ($(n-k)f_H(\ell)+kf_V(\ell)$). Indeed, this intuition is precisely how the Equation (5.2) is proved.

This intuition extends to the case that f_H and f_V are non-constant, as is the case in CSIDH. The result is the following recurrence for the cost of an optimal CSIDH strategy for a measure M:

$$C_M(S^*_M) = \min_{1\le k<n} \left\{ C_{M_k}(S^*_{M_k}) + C_{M'_k}(S^*_{M'_k}) + \sum_{i=1}^{k} f_H(\ell_i) + \sum_{i=k+1}^{n} f_V(\ell_i) \right\} \tag{5.3}$$

Equation (5.3) is an immediate consequence of the following Lemma:

Lemma 5.10. For any measure $M = ((\ell_i)_{i=1}^n, f_H, f_V)$, let S^*_M denote a canonical strategy of smallest weight with respect to M. Then $S^*_M = S^*_{M_{k^*}} \# S^*_{M'_{k^*}}$, where for any k we define $M_k = ((\ell_i)_{i=1}^k, f_H, f_V)$ and $M'_k = ((\ell_i)_{i=k+1}^n, f_H, f_V)$; $S^*_{M_{k^*}}$ and $S^*_{M'_{k^*}}$ are optimal canonical strategies for M_{k^*} and M'_{k^*}, respectively, and

$$k^* \in \operatorname*{argmin}_{1\le k<n} \left\{ C_{M_k}(S^*_{M_k}) + C_{M'_k}(S^*_{M'_k}) + \sum_{i=1}^{k} f_H(\ell_i) + \sum_{i=k+1}^{n} f_V(\ell_i) \right\}$$

Proof. By Lemma 5.8, $S^*_M = (S^*_M)^L \# (S^*_M)^R$ where $(S^*_M)^L$ and $(S^*_M)^R$ are canonical strategies. If $(S_M^*)^L$ is of size k, then of course $(S_M^*)^R$ is of size $n-k$, and we have

$$C_M(S_M^*) = C_{M_k}((S_M^*)^L) + C_{M_k'}((S_M^*)^R) + \sum_{i=1}^{k} f_H(\ell_i) + \sum_{i=k+1}^{n} f_V(\ell_i) \tag{5.4}$$

Now, it must be that $(S_M^*)^L$ is an optimal canonical strategy for M_k, for if it is not, we can construct a strategy $\hat{S}$ on M of strictly smaller weight by setting $\hat{S} = S_{M_k}^* \# (S_M^*)^R$, where S_M^* is an optimal canonical strategy for M_k; similarly, $(S_M^*)^R$ must be optimal for M_k'.

So the only candidate optimal canonical strategies for M are those of the form $S_M^* = S_{M_k}^* \# S_{M_k'}^*$ where $S_{M_k}^*$ and $S_{M_k'}^*$ are optimal canonical strategies for M_k and M_k', respectively; all the remains is to choose k. By Equation (5.4), any k^* which minimizes the expression $C_{M_k}((S_M^*)^L) + C_{M_k'}((S_M^*)^R) + \sum_{i=1}^{k} f_H(\ell_i) + \sum_{i=k+1}^{n} f_V(\ell_i)$ among all $1 \leq k < n$ yields an optimal canonical strategy, and no other k does; thus we indeed have

$$k^* \in \operatorname*{argmin}_{1 \leq k < n} \left\{ C_{M_k}(S_{M_k}^*) + C_{M_k'}(S_{M_k'}^*) + \sum_{i=1}^{k} f_H(\ell_i) + \sum_{i=k+1}^{n} f_V(\ell_i) \right\}$$

and the proof is complete. $\square$

5.5.4 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 [14] propose to select the values of the private key $(e_1, \ldots, e_n)$ from some common interval $[-b, b]$. Meyer *et al.* in [56] instead consider sampling each value e_i from its own interval $[0, b_i]$, where the vector $\mathbf{b} = (b_1, \ldots, b_n)$ is to be chosen so that a speedup is gained while still maintaining a target security level. In [56] 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 $\mathbf{b}$ 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 $\mathbf{b}$, we must determine the relationship between $\mathbf{b}$ 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 $\mathbf{b}$ in order to ensure security.

The requirement to maintain security in the case of non-negative exponents (*à la* [56]) 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 [59] 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 $\mathbf{b}$ 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 [19] 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

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

This reformulated constraint is convex, which is computationally convenient.

In the case of exponents which are not restricted to be non-negative (*à la* [14, 59]) the argument of [59] applies without modification, and we arrive at the similarly-reformulated convex constraint.

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

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 $(\mathbb{1}^T H)_i \mu_{\sigma(i)}$; and,

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 $(V\mathbb{1})_{i-1}\iota_{\sigma(i)}$.

$\ell_{\sigma(i)}$ **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 $\langle \mathbf{c} - \mathbf{d}, \mathbf{b} \rangle + \max_j\{b_j\}\mathbb{1}^T\mathbf{d}$ where, by the above arguments

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

where $\circ$ is the Hadamard product.

So far, we have accounted only for the cost incurred in the first $\max_j\{b_j\}$ strategy executions. If each execution of a strategy successfully lets 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. $P_0 \in E[\pi_p - [1]]$ (in the case $b_{\sigma(i)} > 0$), or $P_0 \in E[\pi_p + [1]]$ (in the case $b_{\sigma(i)} < 0$); and,

2. $\ell_{\sigma(i)}$ divides the order of P_0.

If we choose $\mathbf{b} \geq \mathbb{0}$ (as proposed in [56]), or use the two-point technique of [59], 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 $E[[\ell_i], \pi_p \pm [1]] \cong \mathbb{Z}/\ell_i\mathbb{Z}$. 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 $\mathbf{b}$, this cost is not a convex function of $\mathbf{b}$, 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 $\mathbb{1}^T\boldsymbol{\mu}$.

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:

$$\begin{array}{rl} \text{Minimize} & \langle \mathbf{c}-\mathbf{d}, \mathbf{b}\rangle + \max_j\{b_j\}\mathbb{1}^T\mathbf{d} + \max_j\left\{\frac{b_j}{\ell_j-1}\right\}\mathbb{1}^T\boldsymbol{\mu} \\ \text{Subject to} & \sum_{i=1}^n \log_2(b_i+1) \geq \lambda \\ & \mathbf{b} \geq 0 \\ & \mathbf{b} \in \mathbb{Z}^n \end{array} . \tag{5.5}$$

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 $(m-1)\cdot\mathbb{1}^T\boldsymbol{\mu}$, where m is the number of SIMBA substrategies. Notably, this term is constant as a function of the decision variables $\mathbf{b}$ 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 $\mathbf{b} \in \mathbb{Z}^n$ before solving. Let the relaxed problem be denoted (CP_0) and let its solution $\hat{\mathbf{b}}^{(0)}$. We construct a new program (CP_1) by adding the constraint $b_{i_0} = \left\lceil \hat{b}_{i_0}^{(0)} \right\rfloor$, where i_0 is the index of the entry of $\hat{\mathbf{b}}^{(0)}$ 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 $\mathbf{b}$ which is nearest to an integer in $\hat{\mathbf{b}}^{(k)}$. 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 $\mathbf{b}$ (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 $\mathbf{b}$ which has the largest index among all the entries of $\mathbf{b}$ which take on the value $\max_j\{b_j\}$, until no entry of $\mathbf{b}$ can be decreased while maintaining the required security level; we make this heuristic choice because larger indices correspond to larger primes and hence more expensive isogeny computations, and thus decreasing the bound vector entry of the largest possible.

An Extension to the Two-Point Technique. 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 the mathematical program uses a different bound to ensure security. In particular, the coefficient vectors are given by

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

(where $\hat{H}$ and $\hat{V}$ are as defined in Section 5.5.2), and the new mathematical program is

$$\begin{array}{rl}\text{Minimize} & \langle \mathbf{c}-\mathbf{d},\mathbf{b}\rangle + \max_j\{b_j\}\mathbb{1}^T\mathbf{d} + 2\max_j\left\{\frac{b_j}{\ell_j-1}\right\}\mathbb{1}^T\boldsymbol{\mu}\\ \text{Subject to} & \sum_{i=1}^n \log_2(2b_i+1) \geq \lambda\\ & \mathbf{b}\geq 0\\ & \mathbf{b}\in\mathbb{Z}^n\end{array}. \tag{5.6}$$

As in Section 5.5.2, 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 $\mathbf{c}$ to use a modified cost model with $2\boldsymbol{\iota}$ in place of $\boldsymbol{\iota}$.

5.5.5 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* [56]) and then discuss modifications we make when optimizing for the two-point technique. We note that in this section the "big-picture" optimization problem is too intractable to solve exactly, and we must unfortunately rely on more *ad hoc* techniques; nevertheless, our procedure does result in permutations, strategies, and bound vectors that give a noticable speedup in the context of CSIDH-512.

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 6 on measure $M = (\{\ell_i\}_{i=1}^n, f, g)$. We chose $T = 1000$ (here, T is the number of iterations of the outer loop),$m_{\min} = 1, m_{\max} = 5$. In initial searches, we did not bound the sizes of the SIMBA substrategies; going forward, we chose to bound the size of each SIMBA substrategy by
$$\max\left\{2, \left\lfloor \frac{n}{m+2} \right\rfloor\right\} \le |S_j| \le \left\lceil \frac{n}{m} \right\rceil + 15 \ \ \forall 1 \le j \le m.$$
(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 (5.5) using the iterative rounding technique described in Section 5.5.4 to obtain a bound vector $\mathbf{b}$.

3. For $2 \le k \le \max_j\{b_j\}$, let $M_k^{(\mathbf{b})} = (\{\ell_i\}_{i\colon b_i \ge k}, f, g)$. To obtain a permutation and SIMBA strategy for the k^{th} round of computation, we run Algorithm 6 on the measure $M_k^{(\mathbf{b})}$. 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
$$\max\left\{2, \left\lfloor \frac{n}{m+2} \right\rfloor\right\} \le |S_j| \le \left\lceil \frac{n}{m} \right\rceil + 15 \ \ \forall 1 \le j \le m.$$

Algorithm 6: A stochastic search algorithm for an optimal strategy and permutation.

```
   input : A measure $M$ of size $n$. Natural numbers $T, m_{\min}, m_{\max}$. A permutation
           $\sigma^*$.
   output: A permutation $\sigma$ and SIMBA strategy $S$
 1 Choose $m^* \leftarrow \{m_{\min}, m_{\min}+1, \ldots, m_{\max}\}$ uniformly at random
 2 Choose $P^* = (n_1, n_2, \ldots, n_{m^*})$, a partition of $n$, uniformly at random
 3 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 $\sigma^* M$
 4 Set $C^* = (S^*)_{\sigma^* M}$
 5 for $i$ from 1 to $T$ do
 6     Set $(\sigma, C) \leftarrow (\sigma^*, C^*)$
 7     Choose $m \leftarrow \{m_{\min}, m_{\min}+1, \ldots, m_{\max}\}$ uniformly at random
 8     while $C < C'$ do
 9         Set $C' \leftarrow C$
10         Choose $P = (n_1, n_2, \ldots, n_m)$, a partition of $n$, uniformly at random
11         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 $\sigma M$
12         Set $\sigma$ to be the optimal permutation for $S$ and $M$
13         Set $C \leftarrow (S)_{\sigma M}$
14     end
15     if $C < C^*$ then
16         Set $(\sigma^*, m^*, P^*, S^*, C^*) \leftarrow (\sigma, m, P, S, C)$
17     end
18 end
```

SIMBA and the Two-Point Technique

The algorithm above applies essentially unchanged when the two-point technique of [59] 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 multiplication-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 (5.6) using the iterative rounding technique described in Section 5.5.4 to obtain a bound vector $\mathbf{b}$.

3. For $2 \leq k \leq \max_j\{b_j\}$, let $M_k^{(\mathbf{b})} = (\{\ell_i\}_{i:\, b_i \geq k}, f, g)$. Applying the same technique as in step 1., find the optimal permutation and partition for each $M_k^{(\mathbf{b})}$.

Considerations when Optimizing for Submeasures. Suppose M' is a proper submeasure of M. We note that the cost of evaluating a strategy S under M' is $(S)_{M'} + m\mathbb{1}^T\boldsymbol{\mu}_{M''}$, 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 for the multiplications that must be performed to remove the prime factors present in M'' from the order of the initial points of the SIMBA substrategies. 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 6, we must modify lines (**4**) and (**13**) to include this term. When using the two-point method, the additional term is instead $2m\mathbb{1}^T\boldsymbol{\mu}_{M''}$, since the primes of M'' must be eliminated from the orders of both torsion points at the beginning of each SIMBA substrategy.

5.6 Results

In this section we detail the performance results of our optimizations. In particular, we adapt two implementations from the literature—that of Meyer, Campos and Reith [56], which uses SIMBA but *not* the two-point method, and which we refer to as the MCR implementation; and that of Cervantes-Vázquez, Chenu, Chi-Domínguez, De Feo, Rodríguez-Henríquez and Smith [15], which uses SIMBA and the two-point method, and which we refer to as the CCCDRS implementation—to use optimized permutations and bound vectors. We also implement arbitrary SIMBA strategies in the MCR implementation; however, for the two-point technique we could not find strategies that improve on multiplication-based SIMBA strategies, so we do not implement arbitrary SIMBA strategies there. We have four major implementations based on these:

1. MCR-improved (MCRim): We use the MCR code as a base, and implement an

optimized SIMBA strategy, permutation, and bound vector. We also employ nested strategies.

2. CCCDRS-1: We use the CCCDRS code as a base, and implement optimal permutations, with no other modifications.
3. CCCDRS-2: We build upon the CCCDRS-1 code and additionally use nested strategies (for the bound vector of [15]).
4. CCCDRS-3: We build upon the CCCDRS-2 code and additionally use an optimized bound vector (and corresponding nested strategies).

We benchmarked all four algorithms using two different field arithmetic libraries: one "generic" and one "optimized." Table 5.2 contains details of the running times and operation counts for our implementations and for implementations from the literature (including the implementation of [14], which is not constant-time) using generic arithmetic, while Table 5.3 contains running times for the six relevant constant-time implementations with optimized field arithmetic.

Implementation	**M**	**S**	**a**	Latency (Mcycles)	Speedup (%)
CSIDH [14]	463 287	136 654	416 891	610.2	-
MCR [56]	1 036 675	425 377	1 020 712	1483.9	-
This work (MCRim)	905 200	312 483	859 759	1233.9	16.85
CCCDRS [15] (Two point)	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 5.2: Field operation counts and latency for seven relevant implementations of CSIDH-512 using generic field arithmetic. Note that the implementation labelled "CSIDH" is not constant-time, and is included only for reference.

5.7 Conclusions

We developed systematic techniques for optimizing three parameters used in the CSIDH group action evaluation algorithm: the strategy, permutation of the primes ℓ_i, and bound

Implementation	Latency (Mcycles)	Speedup (%)
MCR [56]	407.9	-
This work (MCRim)	355.8	12.78
CCCDRS [15] (Two point)	251.2	-
This work (CCCDRS-1)	248.5	1.09
This work (CCCDRS-2)	241.7	3.77
This work (CCCDRS-3)	238.5	5.06

Table 5.3: Latency for six relevant implementations of CSIDH-512 using optimized field arithmetic.

vector from which private key values are chosen. Prior works in this area have used *ad hoc* methods to determine 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, [62]), new CSIDH parameter sets may have to be derived to meet NIST security levels. 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.

Chapter 6

An Analysis of Fault-Injection Attacks on Static/Ephemeral CSIDH

6.1 Introduction

This chapter concerns key establishment protocols in the static/ephemeral setting; that is, in applications where Alice's secret key is fixed. In this setting an adversary can try to learn information about Alice's fixed secret by repeatedly engaging in key establishment sessions with her. In the simplest possible kind of attack, sufficiently-many honestly-generated shared secrets in which Alice's secret key is fixed allow the adversary to learn (partial information about) that secret; more sophisticated attacks may use dishonestly-generated public keys, or even physical attacks, such as side-channel attacks (attacks in which physical information about an implementation of a protocol, such as timing or power consumption, is used to attack the protocol) or fault injection (attacks in which external means are used to introduce errors into a protocol that can be exploited to break security).

Static/Ephemeral SIDH has been the target of attacks using dishonestly-generated public keys, while both SIDH and CSIDH have been the target of side-channel and fault-injection attacks. In this chapter we discuss these attacks on SIDH and CSIDH and the presently-existing countermeasures, and propose new countermeasures for fault-injection attacks on CSIDH. We consider our countermeasures in the context of a "weak" model, where currently-existing countermeasures are perfect (that is, they completely prevent this kind of fault attack) but expensive (in particular, they approximately double the running time of the protocol) whereas ours is imperfect (that is, they merely increase the number of fault attacks required for a high success probability, rather than preventing the attack entirely) but cheap (leaving the running time essentially unchanged).

To begin, we discuss a number of background topics required to understand our attack and analysis, including Gray codes, some topics in statistics, and some concepts in optimization. Then, we give some background on attacks on static/ephemeral isogeny-based key establishment protocols; in particular, those attacks on SIDH which use dishonestly-generated auxiliary points [31, 26], and then a number of fault attacks on SIDH [74, 32]. We discuss fault attacks on the "dummy" isogenies of CSIDH, and the currently-known countermeasure [15], and then discuss a new potential countermeasure in which the real and dummy isogenies are reordered. We examine the efficacy of this countermeasure both by simulating an attack based on maximum *a posteriori* learning and by deriving upper and lower bounds on the number of faults required (as a function of the bound vector and desired certainty level) of a slightly more naïve attack. Our analyses apply in both the signed key and unsigned key settings, and our simulations include six bound vectors (three in each setting) from the literature and consider certainty thresholds from 0.1% to 99.9%. Finally, we discuss how, in the signed setting, the key signs can be recovered from the key magnitudes using a standard meet-in-the-middle approach. To improve the efficiency, we use Gray codes to determine an optimal order in which to compute the entries of the tables used in the meet-in-the-middle attack, and prove its optimality.

6.2 Combinatorial Background: Gray Codes

In discussing algorithms to recover the sign of a key after recovering its magnitude, we will need to be familiar with the concept of a Gray code, so we define it here.

Definition 6.1 (Gray Code)**.** A *Gray code* of length k is an ordering of the elements of $\mathbb{Z}_2^k$ such that:

1. Each element appears exactly once in the list; and,
2. Consecutive elements in the list differ in only one index.

The canonical Gray code of length k is the *reflected binary Gray code* RBGC_k, constructed recursively by the following process:

1. $\mathrm{RBGC}_1 = (0, 1)$

2. RBGC$_k$ is obtained by listing the elements of RBGC$_{k-1}$ in order, with a 0 appended at the beginning of each, followed by the elements of RBGC$_{k-1}$ in reverse order, with a 1 appended at the beginning of each.

6.3 Statistical Background

In this section we cover some basic statistical concepts that will be required to understand the fault attacks presented in later sections and their analysis. In particular, we need some introductory concepts in parameter estimation, as well as some concentration and anticoncentration inequalities.

6.3.1 Parameter Estimation

Given a probability distribution $p_\theta(x)$ for an unknown parameter θ, it is natural to ask how we can learn the value of θ by drawing samples of a random variable X distributed according to p_θ. In this work we will use maximum *a posteriori* estimation, which we describe here.

Suppose that we have an *a priori* belief that the true parameter θ was chosen according to a distribution $q(\theta)$. Then, after drawing samples $x_1, x_2, \ldots, x_m$ independently according to the distribution p_θ, from the Bayesian perspective, the *a posteriori* belief on the distribution of θ is

$$q_{\mathrm{MAP}}(\theta | x_1, x_2, \ldots, x_m) = \frac{q(\theta) \prod\limits_{j=1}^{m} p_\theta(x_i)}{\displaystyle\int_\Omega q(\omega) \prod_{i=1}^{m} p_\omega(x_i)\, \mathrm{d}\omega}$$

where Ω is the space from which θ is drawn. Given this, the most likely value of θ—the maximum *a posteriori* estimate—is

$$\theta_{\mathrm{MAP}} = \operatorname*{argmax}_{\theta \in \Omega}\ q_{\mathrm{MAP}}(\theta | x_1, x_2, \ldots, x_m) = \operatorname*{argmax}_{\theta \in \Omega}\ q(\theta) \prod_{j=1}^{m} p_\theta(x_i)$$

where the second equality comes from the fact that $\int_\Omega q(\omega) \prod_{i=1}^{m} p_\omega(x_i)\, \mathrm{d}\omega$ is independent of θ. Intuitively, θ_{MAP} is the best guess at the true value of the parameter, given all the samples we have drawn from p_θ and the *a priori* distribution of θ.

6.3.2 Some Inequalities

To begin, we define the following distributions, each parameterized by a single parameter $p \in [0,1]$:

$$\text{Bernoulli: } X \sim \mathrm{B}(p) \iff \mathbb{P}[X = b] = \begin{cases} 1-p & \text{if } b = 0 \\ p & \text{if } b = 1 \end{cases}$$

$$\text{Modified Rademacher: } X \sim \mathrm{B}_{\pm}(p) \iff P[X = b] = \begin{cases} 1-p & \text{if } b = \frac{-1}{2(1-p)} \\ p & \text{if } b = \frac{1}{2p} \end{cases}$$

These random variables will encode the outcome of a fault attack, where the parameter p depends on the bound vector entry b_j and key entry e_j.

Concentration and Anticoncentration Inequalities

Providing bounds on the number of faults required to recover a static secret key with a certain guaranteed success probability requires us to analyze the behaviour of certain random variables. In particular, we need results which bound the probability that a random variable takes on values near its mean.

A lower bound on the probability that a random variable takes a value sufficiently close to its mean is commonly called a *concentration inequality.* In our analysis, we use the following bound to obtain an upper bound on the number of faults required to guarantee any given success probability of a certain kind of fault attack.

Theorem 6.2 (Hoeffding's Inequality (for *iid* Bernoulli Random Variables) [37])**.** *Let $p \in [0,1]$, and let $\zeta_i \sim \mathrm{B}(p)$ for $i \in \mathbb{N}$. For $m \in \mathbb{N}$, let $Z_m = \frac{1}{m}\sum_{i=1}^{m} \zeta_i$. Then for any $t \in \mathbb{R}$,*

$$\mathbb{P}\big[|Z_m - p| < t\big] \geq 1 - 2e^{-2mt^2}$$

On the other hand, an upper bound on the probability that a random variable takes a value sufficiently close to its mean is commonly called an *anticoncentration inequality.* We apply the following bound to obtain a lower bound on the number of faults required to guarantee a given success probability of a fault attack:

Theorem 6.3 (The Paley-Zygmund Inequality [61])**.** *Let $X \geq 0$ be a random variable with* *finite variance, and let $\vartheta \in [0,1]$. Then*

$$\mathbb{P}\big[X \geq \vartheta \mathbb{E}[X]\big] \geq (1-\vartheta)^2 \frac{\mathbb{E}[X]^2}{\mathbb{E}[X^2]}.$$

Mean Estimates for Absolute Sums of Modified Rademacher Random Variables

We will apply the Paley-Zygmund inequality to random variables of the form $\Xi_m = \left|\frac{1}{m}\sum_{i=1}^m \xi_i\right|$ where the ξ_i are *iid* $\mathrm{B}_\pm(p)$. This will require estimates on $\mathbb{E}\left[\Xi_m\right]$, which we obtain using the following inequalities:

Theorem 6.4 (The Khintchine Inequality [44])**.** *Let $\mathbf{a} \in \mathbb{R}^m$, and let $\xi_i \sim \mathrm{B}_\pm(\frac{1}{2})$ be a sequence of iid random variables. Then for any $p > 0$ there exist absolute constants A_p, B_p such that*

$$A_p \|\mathbf{a}\|_2^p \leq \mathbb{E}\left[\left|\sum_{i=1}^m a_i \xi_i\right|^p\right] \leq B_p \|\mathbf{a}\|_2^p$$

Theorem 6.5 (The Marcinkiewicz-Zygmund Inequality [53])**.** *Let ξ_i be a sequence of independent random variables with mean 0, and let $p \geq 1$ be such that $\mathbb{E}[|\xi_i|^p] < \infty$. Then there exist absolute constants A'_p, B'_p such that*

$$A'_p \,\mathbb{E}\left[\left(\sum_{i=1}^m \xi_i^2\right)^{\frac{p}{2}}\right] \leq \mathbb{E}\left[\left|\sum_{i=1}^m \xi_i\right|^p\right] \leq B'_p \,\mathbb{E}\left[\left(\sum_{i=1}^m \xi_i^2\right)^{\frac{p}{2}}\right]$$

Of course, in order to provide concrete estimates, we will need the values of A_1, B_1, A'_1, and B'_1. From [35] (for A_1, B_1) and [18, Theorem 10.3.2] (for A'_1, B'_1) we find that we can take

$$A_1 = \frac{1}{\sqrt{2}} \qquad B_1 = 1 \qquad A'_1 = \frac{1}{2\sqrt{2}} \qquad B'_1 = 2.$$

6.4 Optimization Background

Here we discuss some fundamental concepts in optimization that come up in our arguments in Sections 6.8.3 and 6.8.4; particularly the Lagrangian, the KKT conditions, and linear programming duality. For a more in-depth discussion of these topics, see, for instance, [12].

Throughout this section, let (P) refer to the following mathematical program:

$$\begin{array}{rl} \text{Minimize} & f(\mathbf{x}) \\ \text{Subject to} & g(\mathbf{x}) = \mathbb{0}_m \\ & h(\mathbf{x}) \leq \mathbb{0}_t \end{array} \tag{P}$$

To begin, we define the Lagrangian:

Definition 6.6 (Lagrangian)**.** The *Lagrangian* of problem (P) is defined as

$$\mathcal{L}(\mathbf{x}; \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(x) + \boldsymbol{\lambda}^T g(\mathbf{x}) + \boldsymbol{\mu}^T h(\mathbf{x})$$

for vectors $\boldsymbol{\lambda} \in \mathbb{R}^m$ and $\boldsymbol{\mu} \in \mathbb{R}^t$ with $\boldsymbol{\mu} \geq 0$.

Note that for all feasible $\mathbf{x}$, $\boldsymbol{\lambda} \in \mathbb{R}^m$ and $\boldsymbol{\mu} \in \mathbb{R}^t$ with $\boldsymbol{\mu} \geq 0$ we have

$$\mathcal{L}(\mathbf{x}; \boldsymbol{\lambda}, \boldsymbol{\mu}) \leq f(\mathbf{x}).$$

The Lagrangian has many useful properties and applications; we will use the KKT conditions and the concept of a dual program, which we define here.

Definition 6.7 (KKT Conditions)**.** The *KKT conditions* for the problem (P) and vectors $\mathbf{x} \in \mathbb{R}^n$, $\boldsymbol{\lambda} \in \mathbb{R}^m$, and $\boldsymbol{\mu} \in \mathbb{R}^t$ are:

Stationarity: $\nabla_{\mathbf{x}} \mathcal{L}(\mathbf{x}; \boldsymbol{\lambda}, \boldsymbol{\mu}) = \mathbb{0}_n$;

Primal Feasibility: $g(\mathbf{x}) = \mathbb{0}_m$, $h(\mathbf{x}) \leq \mathbb{0}_t$;

Dual Feasibility: $\boldsymbol{\mu} \geq \mathbb{0}_t$; and,

Complementary Slackness: $\mu_i h(\mathbf{x})_i = 0$ for all $1 \leq i \leq t$.

The connection between the KKT conditions and optimality are given in the following two theorems:

Theorem 6.8 (KKT Conditions—Sufficiency)**.** *Suppose that $f, g,$ and h are continuously differentiable (componentwise, in the case of g and h) at a point $\mathbf{x}$, and moreover that there exist $\boldsymbol{\lambda} \in \mathbb{R}^m$ and $\boldsymbol{\mu} \in \mathbb{R}^t$ which, together, satisfy the KKT conditions for (P). Then $\mathbf{x}$ is optimal for (P).*

Theorem 6.9 (KKT Conditions—Necessity in the Presence of a Constraint Qualification)**.** *Suppose that f, g, and h are continuously differentiable (componentwise, in the case of g and h) at a point $\mathbf{x}$, and that $\mathbf{x}$ is optimal for (P). Moreover, suppose that either*[1]*:*

- *g and h are affine; or,*
- *f and h are convex, g is affine, and there exists a point $\mathbf{x}^{(s)}$ which satisfies $g(\mathbf{x}^{(s)}) = \mathbb{0}_m$ and $h(\mathbf{x}^{(s)}) < \mathbb{0}_t$ (a* Slater Point*).*

Then there exist $\boldsymbol{\lambda} \in \mathbb{R}^m$ and $\boldsymbol{\mu} \in \mathbb{R}^t$ which, together with $\mathbf{x}$, satisfy the KKT conditions for (P).

In particular, Theorem 6.9 says that in the presence of an appropriate constraint qualification, in order to solve (P), it suffices to solve for $\mathbf{x}, \boldsymbol{\lambda}$ and $\boldsymbol{\mu}$ which satisfy the KKT conditions.

The Lagrangian can be used to derive the *dual program* to a linear program. Here we present only the special case of linear programming duality that we will require in this chapter, along with a fundamental result concerning dual linear programs.

For a linear program of the form

$$\begin{aligned} \text{Minimize} \quad & \langle \mathbf{c}, \mathbf{x} \rangle \\ \text{Subject to} \quad & A\mathbf{x} \geq \mathbf{b} \end{aligned} \tag{P}$$

where $\mathbf{c}, \mathbf{x} \in \mathbb{R}^n$, $\mathbf{b} \in \mathbb{R}^m$, and $A \in \mathbb{R}^{m \times n}$ its corresponding *dual program* is

$$\begin{aligned} \text{Maximize} \quad & \langle \mathbf{b}, \boldsymbol{\lambda} \rangle \\ \text{Subject to} \quad & A^T\boldsymbol{\lambda} = \mathbf{c} \\ & \boldsymbol{\lambda} \geq \mathbb{0} \end{aligned} \tag{D}$$

We have the following result concerning these linear programs:

Theorem 6.10. *Let $\mathbf{x}$ be feasible for (P) and let $\boldsymbol{\lambda}$ be feasible for (D). The following two statements are equivalent:*

1. *$\mathbf{x}$ is optimal for (P) and $\boldsymbol{\lambda}$ is optimal for (D)*
2. *$\lambda_i \cdot (A\mathbf{x} - \mathbf{b})_i = 0$ for all $1 \leq i \leq n$.*

We will use Theorem 6.10 in our discussion of optimal Gray codes in Section 6.8.4.

[1]A condition that ensures that the KKT conditions are necessary for optimality is called a *constraint qualification.* Here we give the two constraint qualifications that are relevant in this chapter; they are far from the only such conditions.

6.5 Multiple Protocol Instances for Static/Ephemeral Key Establishment

In response to attacks on static/ephemeral SIDH (described in Section 6.6), the use of multiple simultaneous protocol instances was proposed as a countermeasure [3]. The idea is general enough to be applied to any key establishment protocol: to establish a key, each party chooses k private keys, construct and exchanges the corresponding public keys, and constructs k^2 shared secrets: one for each combination of the parties' keys. To construct the shared key, both parties hash together the k^2 shared secrets constructed this way.

For clarity, we describe explicitly the result of applying this construction to SIDH.

Example 6.11 (k-SIDH)**.**

Setup: We require the following global parameters:

1. A prime $p = \ell_A^{e_A} \ell_B^{e_B} f \pm 1$ where ℓ_A and ℓ_B are prime and $\ell_A^{e_A} \approx \ell_B^{e_B}$;
2. A supersingular elliptic curve $E/GF(p^2)$;
3. Four points $P_A, P_B, Q_A, Q_B \in E(GF(p^2))$ such that $E[\ell_A^{e_A}] = \langle P_A, Q_A \rangle$ and $E[\ell_B^{e_B}] = \langle P_B, Q_B \rangle$; and,
4. A preimage-resistant hash function H.

One party (Alice) will use the $\ell_A^{e_A}$-torsion subgroup, and the other (Bob) will use the $\ell_B^{e_B}$-torsion subgroup.

Key Generation: For $1 \leq i \leq k$, Alice:

1. Selects $\alpha_i \in \mathbb{Z}/\ell_A^{e_A}\mathbb{Z}$ uniformly at random;
2. Constructs the isogeny $\phi_A^{(i)} \colon E \to E_A^{(i)} = E/\langle P_A + \alpha_i Q_A \rangle$; and,
3. Constructs the auxiliary points $R_A^{(i)} = \phi_A^{(i)}(P_B)$ and $S_A^{(i)} = \phi_A^{(i)}(Q_B)$.

Alice's private/public keypair is

$$\begin{aligned} \mathrm{sk}_A &= (\alpha_1, \ldots, \alpha_k) \\ \mathrm{pk}_A &= \Big((E_A^{(1)}, R_A^{(1)}, S_A^{(1)}), \ldots, (E_A^{(k)}, R_A^{(k)}, S_A^{(k)}) \Big) . \end{aligned}$$

Bob proceeds analogously.

Communication: The parties exchange their public keys.

Key Establishment: Alice computes

$$z_{i,i'} = j\left(E_B^{(i')}/\langle R_B^{(i')} + \alpha_i S_B^{(i')}\rangle\right)$$

for each $1 \le i, i' \le k$. She computes the shared key

$$K_A = H(z_{1,1}, z_{1,2}, \ldots, z_{1,k}, z_{2,1}, z_{2,2}, \ldots, z_{2,k}, \ldots, z_{k,1}, z_{k,2}, \ldots, z_{k,k}).$$

Bob proceeds analogously.

Typically k-SIDH also includes key validation; in particular, Alice verifies that:

1. Each $E_B^{(i')}$ is a supersingular elliptic curve defined over $GF(p^2)$; and,
2. $e_{\ell_A^{e_A}}(R_B^{(i')}, S_B^{(i')}) = e_{\ell_A^{e_A}}(P_A, Q_A)^{\ell_B^{e_B}}$ for each i', where $e_{\ell_A^{e_A}}$ is the Weil pairing on $E[\ell_A^{e_A}]$ [69, Section III.8].

and Bob performs the analogous verifications.

6.6 Prior Work

Before discussing our fault attack techniques and countermeasures, we briefly discuss prior attacks on static/ephemeral isogeny-based key establishment protocols—particularly SIDH (and variants) and CSIDH. The attacks fall into two major categories: fault attacks, and attacks on SIDH-like schemes using invalid auxiliary points (which are *not* fault attacks)—these latter attacks do not extend to CSIDH since CSIDH does not use auxiliary points.

6.6.1 Attacks Using Invalid Auxiliary Points

Galbraith, Petit, Shani, and Ti [31] presented two adaptive attacks on static/ephemeral SIDH: a "simple" attack when no key validation is present, and a "difficult" attack which succeeds even when the best-known direct key validation measure due to Kirkwood, Lackey, McVey, Motley, Solinas, and Tuller [45] is employed. We present both attacks here and explain why the simple attack works; for the complete details of the difficult attack, see [31]. We also briefly discuss Dobson, Galbraith, LeGrow, Ti, and Zobernig's extension of this attack to k-SIDH [26].

The Simple Attack. We suppose that the SIDH prime is of the form $p = 2^n 3^m f \pm 1$, that Alice is using the 2-torsion subgroup, and that Alice's key is of the form $(1, \alpha)$—the extension to other scenarios is straightforward. In the simple attack, we learn the value of α bit-by-bit by sending invalid auxiliary points; in particular, one point is of small order. To learn the least significant bit of α, we first select an ephemeral SIDH key $(1, \beta)$ and construct the corresponding valid SIDH message (E_B, X_B, Y_B); then, we send the message $(E_B, X_B, [2^{n-1}]Y_B)$ to Alice. Notice that

$$X_B + [\alpha][2^{n-1}]Y_B = \begin{cases} X_B & \text{if } \alpha_0 = 0 \\ X_B + [2^{n-1}]Y_B & \text{if } \alpha_0 = 1 \end{cases}$$

since Y_B has order 2^n. Now, Alice computes the secret

$$E_B/\langle X_B + [\alpha][2^{n-1}]Y_B \rangle = \begin{cases} E_B/\langle X_B \rangle & \text{if } \alpha_0 = 0 \\ E_B/\langle X_B + [2^{n-1}]Y_B \rangle & \text{if } \alpha_0 = 1 \end{cases}$$

Since X_B and Y_B are known to us, we can simply compute both possible curves, reveal Alice's computed key[2], and compare to recover the value of α_0.

Inductively, suppose that the first i bits $\alpha_0, \alpha_1, \ldots, \alpha_{i-1}$ of α are known. Define $K_i = \sum_{k=0}^{i-1} \alpha_k 2^k$ and note that this quantity is known to us; to learn α_i, we will proceed as for α_0, except sending the message $(E_B, X_B - [2^{n-i-1}K_i]Y_B, [2^{n-i-1}]Y_B)$. Then Alice will compute

$$\begin{aligned} E_B/\langle X_B - [2^{n-i-1}K_i]Y_B + [\alpha][2^{n-i-1}]Y_B \rangle &= E_B/\langle X_B + [2^{n-1}\alpha_i + 2^{n-i-1}K_i - 2^{n-i-1}K_i]Y_B \rangle \\ &= \begin{cases} E_B/\langle X_B \rangle & \text{if } \alpha_i = 0 \\ E_B/\langle X_B + [2^{n-1}]Y_B \rangle & \text{if } \alpha_i = 1 \end{cases} \end{aligned}$$

and we can use the same technique as for α_0 to recover α_i. Altogether, we recover α in its entirety.

[2]In a practical setting, an adversary is unlikely to be able to simply "reveal" Alice's key. Rather, we imagine that Alice would send a message m and corresponding MAC tag $\sigma = \mathrm{MAC}(\mathrm{sk}_A, m)$, and the adversary can simply test her two candidate values of sk_A against σ to determine Alice's key.

The Difficult Attack. The simple attack is thwarted in a straightforward fashion; if Alice computes the Weil pairing of the points she is sent, she will find that the output is

$$e_{2^n}(X_B - [2^{n-i-1}K_i]Y_B, [2^{n-i-1}]Y_B) = e_{2^n}(X_B, Y_B)^{2^{n-i-1}} \neq e_{2^n}(X_B, Y_B)$$

since $e_{2^n}(X_B, Y_B)$ is a primitive $(2^n)^{\text{th}}$ root of unity. However, the simple attack can be modified to thwart this countermeasure; that is, it is possible to send invalid auxiliary points X'_B, Y'_B which satisfy $e_{2^n}(X'_B, Y'_B) = e_{2^n}(X_B, Y_B)$ and from which we can learn the bits of α. In particular, to learn α_i it is sufficient to send the points

$$\begin{aligned} X'_B &= [\vartheta](X_B - [2^{n-i-1}K_i]Y_B) \\ Y'_B &= [\vartheta(1 + 2^{n-i-1})]Y_B \end{aligned}$$

where ϑ satisfies $\vartheta^2 \equiv 1 + 2^{n-i-1} \pmod{2^n}$.

A Partial Extension to *k*-SIDH. It is folklore that the attack of [31] can be thwarted using multiple protocol instances. Azarderakhsh, Jao, and Leonardi [3] proposed to use 92-SIDH to thwart the attack, while Kayacan [43] asserted that 2-SIDH was sufficient. Dobson, Galbraith, LeGrow, Ti, and Zobernig [26] demonstrate that the attack of [31] can be extended to an attack on k-SIDH with asymptotic cost $O(16^k)$ times the original cost; in particular, it yields a feasible attack on 2-SIDH. They conjecture that it would be infeasible to attack k-SIDH using the same techniques as long as $k \geq 49$.

6.6.2 Fault Attacks

In this section, we discuss the basic kinds of faults that we consider, and attacks that use these faults to recover static secret key information for SIDH and CSIDH.

Fundamentals of Fault Injection. At the highest level, a fault attack uses physical means to tamper with the execution of a cryptographic algorithm in order to learn about a static secret. For some examples of the kinds of physical means used to induce faults (in the context of smart cards), see [51].

There are essentially three kinds of faults that can be injected at the bit level (see, for instance, [51, Chapter 6]):

1. Bit randomization: A given bit's value is changed to a value chosen at random from $\{0, 1\}$.

2. Bit set/reset: A given bit's value is forced to become 0 (bit resetting) or 1 (bit setting).

3. Bit flip: A given bit's value is flipped, from 0 to 1 or from 1 to 0.

Some authors (*e.g.*, [74, 32]) consider contexts in which fault attacks are less precise; in particular, they consider byte-level randomization, where the value of a given byte of memory is randomized. In practice this is a more achievable outcome than a bit-level attack, but the lack of precision may make it more difficult to use or interpret the outcome of the attack, or require more individual fault injections to learn enough about the static secret material.

Attacks on SIDH and Variants. The first fault attack on static/ephemeral SIDH was due to Ti [74]. The attack uses bit or byte randomization to obtain the image of a random point under Alice's secret isogeny; if the point has order divisible by $\ell_A^{e_A}$ then the adversary can use this information to find the image of an $\ell_A^{e_A}$-torsion point under Alice's secret $\ell_A^{e_A}$-isogeny. By [74, Section 4], this point generates a subgroup which is the kernel of an isogeny which is "close to" the dual isogeny of Alice's secret isogeny, and the correct dual can be found efficiently by a brute-force search with high probability.

Subsequently, Gélin and Wesolowski [32] propose "loop-abort" fault attack using bit set/reset and bit randomization; that is, a kind of fault attack that targets a loop counter, with the hope of causing the loop to end before the necessary iterations are completed. Loop abort fault attacks have a long history in elliptic curve and post-quantum cryptography; they were first introduced in the context of pairing-based cryptography [60] and have also been considered for lattice-based schemes employing the Fiat-Shamir protocol (in particular, BLISS [27] and Ring-TESLA [1]) and the hash-and-sign paradigm (in particular, GPV [33, 28]).

In SIDH, Alice constructs her secret iteratively; in particular, after the k^{th} round of computation, Alice has computed

$$E_A^{(k)} := E/\langle \ell_A^{e_A-k}(P_A + \alpha Q_A)\rangle$$

where α is her secret key. To learn the least significant bit of α, Gélin and Wesolowski use loop abort faults to terminate the computation after the first round of computation; since Q_A has order $\ell_A^{e_A}$, we know that $\ell_A^{e_A-1}(P_A + \alpha Q_A)$ must be one of $\ell_A^{e_A-1}P_A + t \cdot \ell_A^{e_A-1} \cdot Q_A$ for $t \in \{0, 1, \ldots, \ell_A - 1\}$, and an attacker can determine which is the case using Alice's output shared secret (access to which is modelled in a number of ways) and trial and error.

From there, once the $k-1$ low order bits of α are known to the attacker, the k^{th} can be learned using a similar method; the attacker repeats the process until they learn all n bits of Alice's secret.

Attacks on CSIDH and a Countermeasure. Recall, as discussed in Chapter 5 that in order for CSIDH to run in constant-time, a constant number (as a function of the secret key) of isogenies of each degree must be computed. In order to make this happen, in each round of key establishment exactly b_j ℓ_j-isogenies are constructed, of which $|e_j|$ are "real" (that is, they actually contribute to the construction of the shared secret) and $b_j - |e_j|$ are "dummy" (that is, they do not contribute to the construction of the shared secret). Though it has not been described in great detail, the fundamental idea that has been explored (particularly in [15]) is to inject faults during isogeny computations and use an honest party's output to differentiate real isogenies from dummy isogenies. In all prior implementations of CSIDH that use these dummy isogenies, the isogenies are performed in a fixed order: all real isogenies, followed by all dummy isogenies. This allows a binary search for the key value e_j (up to sign) and so, in principle the key can be recovered using very few faults—on the order of 300 for CSIDH-512 (see Table 6.1 in Section 6.9).

In [15, Section 5] the authors propose a "dummy-free" but still constant-time variant of CSIDH, arguing that if every isogeny is real, there is nothing to be learned from a fault attack of the kind discussed above. To maintain the constant-time nature of the algorithm, the authors first modify how the keyspace is derived from the bound vector; in particular, if the j^{th} bound vector entry is b_j, then the j^{th} key entry is chosen as

$$e_j \xleftarrow{\$} \mathcal{K}_j(b_j) = \{e \in [-b_j, b_j] \cap \mathbb{Z} \,:\, |e| \equiv b_j \pmod 2\}\,.$$

Then, to evaluate $[\mathfrak{l}_j^{e_j}] * E$, the authors propose to first evaluate the action of $[\mathfrak{l}_j^{\mathrm{sgn}(e_j)}]$ $|e_j|$ times, and then to alternate evaluating the action of $[\mathfrak{l}_j^{\mathrm{sgn}(e_j)}]$ and $[\mathfrak{l}_j^{-\mathrm{sgn}(e_j)}]$ a total of $b_j - |e_j|$ times. Since $|e_j| \equiv b_j \pmod 2$, these alternating applications of $[\mathfrak{l}_j^{\mathrm{sgn}(e_j)}]$ and $[\mathfrak{l}_j^{-\mathrm{sgn}(e_j)}]$ will cancel one another out, and so the output curve will be correct. In order to maintain the security level, the bound vector must be doubled, and so this "dummy-free" technique leads to a roughly 100% increase in the time required for key establishment.

6.7 Preliminaries for Our Work

6.7.1 Our Contributions

Our contributions in this work can be summarized as follows:

1. We demonstrate how current implementations of CSIDH which use a "real-then-dummy" approach can be vulnerable to fault injection attacks, in which an attacker can achieve a complete break of the system under ideal conditions by recovering the private key using $\sum_{j=1}^{n} \lceil \log_2(b_j) + 1 \rceil$ many faults via a binary search attack in a static key setting.

2. As a remedy to the above attack, for a fixed key $\mathbf{e}$ we propose randomly mixing the constructions of the $|e_j|$ many degree ℓ_j real isogenies with the dummy isogenies. At the time of evaluation of the group action, we choose a binary-valued *decision vector* $\mathbf{x}^j = (x_1^j, \ldots, x_{b_j}^j)$ with weight $|e_j|$ uniformly at random; we then construct the i^{th} isogeny of degree ℓ_j as real if $x_i^j = 1$ and dummy otherwise.

3. We analyze attacking the randomized protocol described above using an oracle $\mathcal{O}$, in which $\mathcal{O}(j, i)$ reveals x_i^j. We derive formulas for the distribution on the key e_j given a string of outputs from $\mathcal{O}(j, \cdot)$ from pairwise different instances of $\mathbf{x}^j$ under the same key $\mathbf{e}$. We derive an upper bound on the number of faults required to achieve any desired error threshold ϵ for any given bound vector in this setting.

4. In the setting of signed keys, we discuss how to determine the signs of the key entries once their magnitudes have been determined using fault attacks. We introduce an approach based on Gray codes which we estimate (based on numerical experiments) to be, on average, 88% more efficient than the naïve approach, even when the naïve approach uses per-key-magnitude optimized permutations and strategies.

5. We simulated attacking this randomized version of CSIDH under ideal conditions with 6 different bound vectors $\mathbf{b}$ used in previous implementations of CSIDH-512. We found that an attacker can learn the key up to sign with absolute certainty in the real-then-dummy setting using between 260–300 fault attacks in the signed setting and between 340–370 attacks in the unsigned setting depending on the bound vector. In contrast, by using a uniformly random decision vector the number of attacks needed to learn the key up to sign with only 1% certainty on average increases to a range of 2 400–3 600 in the signed setting and 10 500–16 000 in the unsigned

setting. To achieve 99% certainty, the number of attacks increases to 7 600–12 700 in the signed setting and 30 700–45 600 for the unsigned setting. Based on our data and the minimal overhead required for this modification, we recommend that future implementations of CSIDH utilizing dummy isogeny constructions randomize the constructed isogenies in this manner.

6.7.2 The Decision Vector

Let $p = 4\ell_1 \cdots \ell_n - 1$ be prime with $\ell_1, \ldots, \ell_n$ pairwise distinct small odd primes. For each prime ℓ_j, we encode the choice of constructing the i^{th} degree ℓ_j isogeny $\varphi_{i,j}$ as either a real or dummy into a binary *decision vector* $\mathbf{x}^j = (x_1^j, \ldots, x_{b_j}^j)$, in which $x_i^j = 1$ denotes that the ith degree ℓ_j isogeny shall be constructed as real, and $x_i^j = 0$ denotes that the ith degree ℓ_j isogeny shall be constructed as a dummy. For correctness of the algorithm, the Hamming weight $H(\mathbf{x}^j)$ of $\mathbf{x}^j$ must be equal to $|e_j|$. This vector x^j represents only the choice of the type of isogeny constructed and may be explicitly or implicitly stored in memory for a given implementation of the group action, and it is this vector which our attacks target. As an example Algorithm 7 depicts the constant time algorithm given by Onuki *et al.* in [59]. Line 12 of Algorithm 7 computes the boolean value "$e_i \neq 0$", which is used as a mask bit to determine the type of isogeny to be constructed. We consider this boolean as one of the values in the decision vector $\mathbf{x}^i$. The decision vector for other dummy-based constant time algorithms for CSIDH, such as those given in [39, 56], are defined similarly.

6.7.3 General Structure of the Attack

We target our attacks on the second round of the key exchange when one party is computing the action of their private key on the curve they received from the other party. Through most of this work we consider only the scenario where an attacker targets a single isogeny constructed for a specific prime and iteration; i.e., the attacker carries out a bit-flip fault attack which changes the value of x_i^j for one particular i and j. We assume an ideal setting in which the attacker can precisely target this decision vector using the methods described in the next two sections for their choice of any single pair (j, i). If an adversary performs multiple attacks on pairs $(j_1, i_1), \ldots, (j_m, i_m)$, we assume that each pair references a distinct evaluation of $(\mathbf{e}, E) \mapsto [\prod_i \mathfrak{l}_i^{e_i}] * E$, with the private key $\mathbf{e}$ static and E allowed to vary. This would be the case for example when a static key is used over multiple sessions.

Algorithm 7: Constant time version of CSIDH group action evaluation.

```
Input  : $A \in GF(p), b \in \mathbb{N}$, a list of integers $(e_1, \ldots, e_n)$ s.t. $-b \leq e_i \leq b$ for
         $i = 1, \ldots, n$, and distinct odd primes $\ell_1, \ldots, \ell_n$ s.t. $p = 4\prod_i \ell_i - 1$.
Output: $B \in GF(p)$ s.t. $E_B = (\mathfrak{l}_1^{e_1} \cdots \mathfrak{l}_n^{e_n}) * E_A$, where $\mathfrak{l}_i = (\ell_i, \pi - 1)$ for
         $i = 1, \ldots, n$, and $\pi$ is the $p^{\text{th}}$ power Frobenius endomorphism of $E_A$.
1  Set $e'_i = b - |e_i|$.
2  while some $e_i \neq 0$ or some $e'_i \neq 0$ do
3      Set $S = \{i \mid e_i \neq 0$ or $e'_i \neq 0\}$.
4      Set $k = \prod_{i \in S} \ell_i$.
5      Generate points $P_0 \in E_A[\pi - 1]$ and $P_1 \in E_A[\pi + 1]$ by Elligator.
6      Let $P_0 = [(p+1)/k]P_0$ and $P_1 = [(p+1)/k]P_1$.
7      for $i \in S$ do
8          Set $s$ be the sign bit of $e_i$.
9          Set $Q = [k/\ell_i]P_s$.
10         Let $P_{1-s} = [\ell_i]P_{1-s}$.
11         if $Q \neq \infty$ then
12             if $e_i \neq 0$ then
13                 Compute an isogeny $\varphi : E_A \to E_B$ with $\ker(\varphi) = \langle Q \rangle$.
14                 Let $A \leftarrow B$, $P_0 \leftarrow \varphi(P_0)$, $P_1 \leftarrow \varphi(P_1)$, and $e_i \leftarrow e_i - 1 + 2s$.
15             else
16                 Dummy computation.
17                 Let $A \leftarrow A$, $P_s \leftarrow [\ell_i]P_s$, and $e'_i \leftarrow e'_i - 1$.
18             end
19         end
20     end
21     Let $k \leftarrow k/\ell_i$.
22 end
23 Return $A$
```

6.7.4 Fault Attack Method and Oracle: Unsigned Setting

We now define an oracle which models fault attacks in the setting of unsigned private keys (that is, $e_j \in [0, b_j] \cap \mathbb{Z}$). An intuitive attack to consider is flipping the value of x_j^i. In Algorithm 7, this can be accomplished by influencing the value of the boolean computed in line 12 on a particular iteration. If x_i^j is changed from 0 (dummy construction) to 1 (real construction), then the output curve will likely have an extra factor of $[\mathfrak{l}_j]$ applied compared to the correct shared key because keys here are unsigned. In the event the construction is skipped due to the point lacking the proper order, the correct output curve will be computed. If x_i^j is modified from 1 to 0, then the output curve will be lacking a factor of $[\mathfrak{l}_j]$. Upon a key reveal query the attack can determine which situation they are in and learn the true value of x_i^j.

Other implementations of CSIDH, such as that of [39], use non-multiplication based strategies for evaluating the group action, and in such an implementation the point multiplication performed in line 17 of Algorithm 7 is moved for efficiency to be included in line 6 (in a constant time fashion) for any indices for which a dummy computation is performed. As a consequence, if the attacker attempts to change the value of x_i^j from 0 to 1 at the time of isogeny construction, the construction will automatically be skipped because the point to be used as the kernel generator is actually trivial and the correct shared key will be computed. On the other hand if x_i^j is changed from 1 to 0 in such a generalized algorithm, the point to be used to normally construct the isogeny will retain a factor of ℓ_j in its order (assuming it was present to begin with) throughout the remainder of the algorithm until a fresh point is chosen. Therefore all further isogeny constructions derived from this point will be corrupted and the final output of the algorithm will be incorrect. One option for the attacker is to change both x_i^j and the decision on multiplying the point by ℓ_j, but this may or may not be practical to achieve.

In both settings above, the attacker can target x_i^j and subsequently learn its value. We therefore define an oracle $\mathcal{O}$ which reveals the value of $\mathbf{x}$ at the desired pair of indices: $\mathcal{O}(j,i) = x_i^j$ for $1 \leq j \leq n$ and $0 \leq i \leq b_j$.

6.7.5 Fault Attack Method and Oracle: Signed Setting

Here we describe an attack and an oracle for the setting of signed private keys (so $e_j \in [-b_j, b_j] \cap \mathbb{Z}$). As in the previous subsection, we formulate an attack which reveals the value of x_i^j.

For Algorithm 1, changing the value of the boolean in line 12 will produce an output curve that is off by either a factor of $[\mathfrak{l}_j]$ or $[\mathfrak{l}_j]^{-1}$ depending on both x_i^j and the sign of e_j. To get an outcome with more information, we instead target the sign bit s computed in line 8. Let E be the curve determined at the end of the algorithm when no attack is performed and let $\hat{E}_{i,j}$ be the final resulting curve when the sign bit s used for constructing isogeny (j,i) is flipped. If x_i^j is 0, then a dummy construction is performed regardless of how s is modified and we have $E = \hat{E}_{i,j}$. If x_i^j is 1, we will have $\hat{E}_{i,j} = [\mathfrak{l}_j]^2 * E$ if $e_j > 0$ or

$\hat{E}_{i,j} = [\mathfrak{l}_j]^{-2} * E$ if $e_j < 0$. After a key reveal query, the attacker can check which equality holds, learn the value of x_i^j, and additionally learn the sign of e_j when $x_i^j = 1$.

In the context of a generalized algorithm using a non-multiplication based strategy such as the algorithm of [39], the attack described in the previous subsection still applies here and one may formulate the same oracle as shown there. The attack described above which modifies s will only work if the attacker modifies both s and prevents ℓ_j from being multiplied to the points after they are randomly generated, for otherwise the resulting kernel generator will be trivial and the isogeny construction skipped.

Based on this discussion, the attacker has the ability to learn the value of x_i^j using a fault attack, but not necessarily the sign of e_j. To address the most general case, we assume the oracle does not reveal the sign of the key and define $\mathcal{O}(j,i) = x_i^j$ as before.

6.8 Attack Analysis

In this section, we analysis how individual attacks from Section 6.7 which target particular isogenies $\varphi_{i,j}$ can be performed together for varying i and j to gain information about the private key vector $\mathbf{e}$. Going forward, we will use $\mathcal{O}$ to refer to the proper oracle from Section 6.7; that is, $\mathcal{O}$ refers to $\mathcal{O}_{\texttt{Signed}}$ when considering the setting of signed private key values, and $\mathcal{O}$ will refer to $\mathcal{O}_{\texttt{Unsigned}}$ when considering the unsigned setting. The analysis is essentially the same in both cases since in the signed case a single query $\mathcal{O}_{\texttt{Signed}}(j,\cdot)$ which results in 0 can be used to determine the sign of e_j, and so effectively the signed setting reduces to the unsigned setting.

We consider cases based on how $\mathbf{x}$ is generated. Section 6.8.1 examines the setting in which all real isogeny are constructed first, followed by any remaining dummy isogenies. The remainder of the section analyzes when each $\mathbf{x}_j$ is chosen uniformly at random with the correct Hamming weight. In Sections 6.8.2 and 6.8.3, we look at subcases based on whether each $\mathbf{x}_j$ is fixed or dynamic.

6.8.1 "Real-then-dummy" Decision Vector

In this section we consider the "real-then-dummy" method, which every instantiation of CSIDH in the literature has used so far at the time of this writing. Here, $\mathbf{x}^j$ has exactly the form

$$\mathbf{x}^j = (\underbrace{1, 1, \ldots, 1}_{|e_j|}, 0, 0 \ldots, 0).$$

For this scenario the attack is extremely simple: the private key e_j corresponds exactly to the position in which the last 1 appears, and so a simple binary search can determine e_j with absolute certainty in exactly $\lceil \log_2(b_j) \rceil + 1$ many queries to the oracle $\mathcal{O}(j, \cdot)$. It follows that the entire key $\mathbf{e}$ can be determined exactly using $\sum\limits_{j=1}^{n} (\lceil \log_2(b_j) \rceil + 1)$ calls to $\mathcal{O}$.

As the above shows, the real-then-dummy case is susceptible to a very simple attack. The most obvious change to make to attempt to counter the binary search attack is to randomize the value of each $\mathbf{x}^j$. In the following subsections, we consider the case when $\mathbf{x}^j$ is drawn from the set $X_j := \{\mathbf{x}^j \in \{0, 1\}^{b_j} : H(\mathbf{x}^j) = |e_j|\}$ uniformly at random, where H denotes Hamming weight.

6.8.2 Fixed Uniformly Random Decision Vector

We briefly remark on the approach of generating $\mathbf{x}^j$ randomly from the set X_j at the time of key generation, and using this same $\mathbf{x}^j$ for every evaluation of the action $[\prod_i \mathfrak{l}_i^{e_i}] * E$. Effectively $\mathbf{x}^j$ becomes part of the key in this scenario. The most straightforward attack to learn $|e_j|$ would query $\mathcal{O}$ at (j, i) for $i = 1, 2, \ldots, b_j$ to find each value of $\mathbf{x}^j$, which is possible since the value of $\mathbf{x}^j$ never changes among subsequent group actions. Therefore in this setting $|e_j|$ can be learned using b_j calls to $\mathcal{O}(j, \cdot)$, and the total key $\mathbf{e}$ can be learned up to sign using $\sum_{j=1}^{n} b_j$ many calls to $\mathcal{O}$. Asymptotically this is better than the real-then-dummy approach, but in practice offers little extra security for actual values of $\mathbf{b}$. See Section 6.9 for a comparison.

6.8.3 Dynamic Uniformly Random Decision Vector

We now consider the primary focus of this work, which is when the decision vector $\mathbf{x}^j$ is chosen from $X_j = \{\mathbf{x}^j \in \{0, 1\}^{b_j} : H(\mathbf{x}^j) = |e_j|\}$ uniformly at random during every evaluation $(\mathbf{e}, E) \mapsto [\prod_i \mathfrak{l}_i^{e_i}] * E$ of the group action. We refer to this setting as having a *dynamic* decision vector. If one views the decision vector $\mathbf{x}^j$ as a means of permuting the constructions of the real and dummy isogenies, then the oracle calls $\mathcal{O}(j, i_1)$ and $\mathcal{O}(j, i_2)$ for $i_1 \neq i_2$ on different computations of the group action may actually correspond to the construction of the *same* isogeny, and so multiple calls to $\mathcal{O}(j, \cdot)$ yield less information than in the previous settings.

Unsigned exponents

For $\ell \in \mathbb{N}$, let $\boldsymbol{\beta}^{(\ell)}$ denote the string of outputs of the first ℓ queries of $\mathcal{O}$, and let $\mathbf{q}_j^{(\boldsymbol{\beta}^{(\ell)})}$ denote the adversary's *a posteriori* distribution on e_j, having seen $\boldsymbol{\beta}^{(\ell)}$. That is,

$$q_{j,k}^{(\beta_\ell)} := \mathbb{P}[e_j = k | \boldsymbol{\beta}^{(\ell)}].$$

Theorem 6.12. *In the unsigned exponent setting, for every $1 \leq j \leq n$, $0 \leq k \leq b_j$, and binary string β^ℓ of length $\ell \geq 1$ we have*

$$q_{j,k}^{(\beta^\ell)} = \frac{\mathbb{P}[x_i^j = \boldsymbol{\beta}_\ell^{(\ell)} | e_j = k] \cdot q_{j,k}^{(\boldsymbol{\beta}^{(\ell-1)})}}{\sum\limits_{t=0}^{b_j} \mathbb{P}[x_j^j = \boldsymbol{\beta}_\ell^{(\ell)} | e_j = t] \cdot q_{j,t}^{(\boldsymbol{\beta}^{(\ell-1)})}}, \tag{6.1}$$

where $\boldsymbol{\beta}^{(0)}$ is the empty string and $q_{j,k}^{\boldsymbol{\beta}^{(0)}} := \mathbb{P}[e_j = k] = 1/(b_j + 1)$ for every k. A non-recursive form of $q_{j,k}^{(\boldsymbol{\beta}^{(\ell)})}$ for any $\boldsymbol{\beta}^{(\ell)}$ with $\ell \geq 0$ is

$$q_{j,k}^{(\boldsymbol{\beta}^\ell)} = \frac{(b_j - k)^{\ell - H(\boldsymbol{\beta}^{(\ell)})} k^{H(\boldsymbol{\beta}^{(\ell)})}}{\sum_{t=0}^{b_j} (b_j - t)^{\ell - H(\boldsymbol{\beta}^{(\ell)})} t^{H(\boldsymbol{\beta}^{(\ell)})}}, \tag{6.2}$$

where H denotes Hamming weight.

Proof. We first prove equality (6.1) using Bayes' Theorem, the Law of Total Probability, and the fact that the e_j are chosen uniformly at random.

$$\begin{aligned} q_{j,k}^{(\boldsymbol{\beta}^{(\ell)})} &= \mathbb{P}[e_j = k | \boldsymbol{\beta}^{(\ell)}] \\ &= \frac{\mathbb{P}[\boldsymbol{\beta}^{(\ell)} | e_j = k] \cdot \mathbb{P}[e_j = k]}{\mathbb{P}[\boldsymbol{\beta}^{(\ell)}]} && \text{(Bayes' theorem)} \\ &= \frac{\mathbb{P}[\boldsymbol{\beta}^{(\ell)} | e_j = k] \cdot \mathbb{P}[e_j = k]}{\sum\limits_{t=0}^{b_j} \mathbb{P}[\boldsymbol{\beta}^{(\ell)} | e_j = t] \cdot \mathbb{P}[e_j = t]} && \text{(Law of total probability)} \end{aligned}$$

$$= \frac{\mathbb{P}[\boldsymbol{\beta}^{(\ell)}|e_j = k]}{\sum\limits_{t=0}^{b_j} \mathbb{P}[\boldsymbol{\beta}^{(\ell)}|e_j = t]} \qquad (e_j \text{ is } \textit{a priori} \text{ uniform})$$

$$= \frac{\mathbb{P}[x_i^j = \beta_\ell^{(\ell)}|e_j = k] \cdot \mathbb{P}[\boldsymbol{\beta}^{(\ell-1)}|e_j = k]}{\sum\limits_{t=0}^{b_j} \mathbb{P}[x_i^j = \beta_\ell^{(\ell)}|e_j = t] \cdot \mathbb{P}[\boldsymbol{\beta}^{(\ell-1)}|e_j = t]} \qquad \begin{pmatrix} \beta_\ell^{(\ell)}|e_j \text{ and } \boldsymbol{\beta}^{(\ell-1)}|e_j \\ \text{are independent} \end{pmatrix}$$

Equality (6.1) follows from the last line above since for all t (including k) we have

$$\begin{aligned} \mathbb{P}[\boldsymbol{\beta}^{(\ell-1)}|e_j = t] &= q_{j,t}^{(\boldsymbol{\beta}_{(\ell-1)})} \cdot \mathbb{P}[\boldsymbol{\beta}^{(\ell-1)}]/\mathbb{P}[e_k = t] \\ &= q_{j,t}^{(\beta_{\ell-1})} \cdot \mathbb{P}[\boldsymbol{\beta}^{(\ell-1)}]/\mathbb{P}[e_k = k]. \end{aligned}$$

again by Bayes' Theorem.

We now prove equality (6.2) by induction on ℓ using equality (6.1). When $\ell = 0$ it is easy to see that the expression in Equation (6.2) simplifies to $1/(b_j + 1) = \mathbb{P}[e_j = k]$ with the understanding that $0^0 = 1$. Assume now that $\ell \geq 1$ and Equation (6.2) holds for any binary string of length less than ℓ. In the case that $\boldsymbol{\beta}_\ell^{(\ell)} = 1$, we have that

$$\begin{aligned} q_{j,k}^{(\boldsymbol{\beta}^{(\ell)})} &= \frac{\mathbb{P}[x_i^j = 1|e_j = k] \cdot q_{j,k}^{(\boldsymbol{\beta}^{(\ell-1)})}}{\sum_{t=0}^{b_j} \mathbb{P}[x_i^j = 1|e_j = t] \cdot q_{j,t}^{(\boldsymbol{\beta}^{(\ell-1)})}} && \text{(by Equation (6.1))} \\ &= \frac{\frac{k}{b_j} \cdot \frac{(b_j - k)^{\ell-1-H(\boldsymbol{\beta}^{(\ell-1)})} k^{H(\boldsymbol{\beta}^{(\ell-1)})}}{\sum_{t=0}^{b_j} (b_j - t)^{\ell-1-H(\boldsymbol{\beta}^{(\ell)})} t^{H(\boldsymbol{\beta}^{(\ell-1)})}}}{\sum_{t=0}^{b_j} \frac{t}{b_j} \frac{(b_j - t)^{\ell-1-H(\boldsymbol{\beta}^{(\ell-1)})} t^{H(\boldsymbol{\beta}^{(\ell-1)})}}{\sum_{t'=0}^{b_j} (b_j - t')^{\ell-1-H(\boldsymbol{\beta}^{(\ell)})} t'^{H(\boldsymbol{\beta}^{(\ell-1)})}}} && \text{(inductive hypothesis)} \\ &= \frac{(b_j - k)^{\ell-(H(\boldsymbol{\beta}^{(\ell-1)})+1)} k^{H(\boldsymbol{\beta}^{(\ell-1)})+1}}{\sum_{t=0}^{b_j} (b_j - t)^{\ell-(H(\boldsymbol{\beta}^{(\ell-1)})+1)} t^{H(\boldsymbol{\beta}^{(\ell-1)})+1}} && \text{(denominators cancel)} \\ &= \frac{(b_j - k)^{\ell-H(\boldsymbol{\beta}^{(\ell)})} k^{H(\boldsymbol{\beta}^{(\ell)})}}{\sum_{t=0}^{b_j} (b_j - t)^{\ell-H(\boldsymbol{\beta}^{(\ell)})} t^{H(\boldsymbol{\beta}^{(\ell)})}} \end{aligned}$$

since $H(\boldsymbol{\beta}^{(\ell)}) = H(\boldsymbol{\beta}^{(\ell-1)}) + 1$ in this case. The case $\beta_\ell^{(\ell)} = 0$ is similar and we omit the details.

□

Attack Model

Here we detail an attack on CSIDH in the setting of dynamic decision vectors in both the signed and unsigned settings which makes use of the probabilities previously computed in this section. In the attack, referred to as *least certainty*, the attacker chooses a key index $1 \leq j^* \leq n$ in which to inject a fault on each iteration, where in the unsigned setting the index is chosen through the formula

$$j^* = \underset{1 \leq j \leq n}{\operatorname{argmin}} \left(\max_{0 \leq k \leq b_j} q_{j,k}^{\boldsymbol{\beta}_j^{(\ell)}} \right).$$

where $\boldsymbol{\beta}_j^{(\ell)}$ is the string of oracle outputs for the index j (with ℓ also depending on j). That is, the attacker targets the index for which they are least certain about the value of the key. The variables q_j are initialized as the uniform distribution on $b_j + 1$ elements.

For the signed setting, our oracle $\mathcal{O}(j, \cdot)$ only gives information on e_j up to sign, and so our attack attempts only to learn the key up to sign. For $1 \leq k \leq b_j$ we therefore define

$$r_{j,k}^{\boldsymbol{\beta}_j^{(\ell)}} = q_{j,k}^{\boldsymbol{\beta}_j^{(\ell)}} + q_{j,-k}^{\boldsymbol{\beta}_j^{(\ell)}} = 2q_{j,k}^{\boldsymbol{\beta}_j^{(\ell)}}$$

and $r_{j,0}^{\boldsymbol{\beta}_j^{(\ell)}} = q_{j,0}^{\boldsymbol{\beta}_j^{(\ell)}}$, and the attacker chooses the index

$$j^* = \underset{1 \leq j \leq n}{\operatorname{argmin}} \left(\max_{0 \leq k \leq b_j} r_{j,k}^{\boldsymbol{\beta}_j^{(\ell)}} \right).$$

to attack on each round. Here we initialize q_j as the uniform distribution on $2b_j + 1$ elements.

In both settings the index i for which to call $\mathcal{O}(j, \cdot)$ on is chosen uniformly at random, but the index may be selected in any other manner without affecting the overall efficiency of the attack.

In both the signed and unsigned setting, the attacker performs some desired number of iterations, with each iteration choosing the index j^* to attack based on the above formulas. Once these iterations are complete, the attacker is left with a probability distribution on the (absolute value of the) key, in which the most likely value for $|e_j|$ is given by $\underset{0 \leq k \leq b_j}{\operatorname{argmax}}\, r_{j,k}^{\boldsymbol{\beta}_j^{(\ell)}}$, with $r_{j,k}^{\boldsymbol{\beta}_j^{(\ell)}} = q_{j,k}^{\boldsymbol{\beta}_j^{(\ell)}}$ in the unsigned setting.

These attacks are detailed in Algorithms 8 and 9 in the unsigned and signed settings, respectively. Both algorithms keep variables $\ell[j]$ and $w[j]$ tracking the number of attacks on index j and number of 1's seen from the oracle $\mathcal{O}(j, \cdot)$, respectively (so, the information contained in $\boldsymbol{\beta}_j^{(\ell)}$ above). Variables $q[j][k]$ store the probabilities $r_{j,k}^{\boldsymbol{\beta}_j^{(\ell)}}$ for $1 \leq j \leq n$ and $0 \leq k \leq b_j$ and the current value of ℓ. In the signed context of Algorithm 9, the probability $q_{j,0}^{\boldsymbol{\beta}^{(\ell)}} = \mathbb{P}[e_j = 0 | \boldsymbol{\beta}^{(\ell)}]$ is computed as a special case since all other $q_{j,k}^{\boldsymbol{\beta}^{(\ell)}}$ have their values doubled as discussed previously.

Both Algorithms 8 and 9 perform attacks until some desired certainty threshold on the total key is obtained. The certainty on the key is given by the quantity

$$\prod_{j=1}^{n} \max_{0 \leq k \leq b_j} q[j][k],$$

which represents the probability that the key $\mathbf{e}$ (up to sign) is equal to

$$\left(\operatorname*{argmax}_{0 \leq k \leq b_j} q[1][k], \ldots, \operatorname*{argmax}_{0 \leq k \leq b_j} q[n][k] \right).$$

Bounds on Naïve Attacks.

In this section we seek to determine bounds on the number of faults required to guarantee a given success rate $1 - \epsilon$ in a fault attack. To that end, we consider a particular kind of attack—which we call the "naïve" method—which is simultaneously effective, intuitive, and easy enough to analyze. Throughout this section we implicitly assume that we are working in the setting of unsigned keys; the techniques in this section extend in a very straightforward fashion to determining the *magnitude* of a given key entry in the signed setting. We consider the problem of determining the sign of the key given its magnitude in Section 6.8.4.

Our naïve attack in this setting is as follows: choose a vector $\mathbf{m} \in \mathbb{N}^n$ in advance[3], and for $1 \leq j \leq n$, apply m_j fault attacks on isogenies of degree ℓ_j. Then, suppose that we observe a sequence $\boldsymbol{\beta}^{(m_j)}$ of outputs in which w_j such isogenies are revealed to be real; we then guess that

$$e_j = e_j^* := \left\lceil b_j \frac{w_j}{m_j} \right\rfloor;$$

[3]This is in contrast with Algorithms 8 and 9, where the number of attacks is not determined in advance, and instead we stop once we are sufficiently certain about the key.

Algorithm 8: Least Certainty Attack (Unsigned Dynamic)

```
Parameters: Bound vector $\mathbf{b} = (b_1, \dots, b_n)$ from which keys $e_j \in [0, b_j]$ are drawn.
Input      : Certainty bound $\epsilon \in (0,1)$, unknown private key $\mathbf{e} = (e_1, \dots, e_n)$
             accessed through oracle $\mathcal{O}$.
Output     : Probability distribution $q_{j,k}$ on key $\mathbf{e}$.
1  $\ell \leftarrow [0 : j = 1, 2, \dots, n]$.
2  $w \leftarrow [0 : j = 1, 2, \dots, n]$.
3  $q \leftarrow [\,[1/(b_j + 1) : k = 0, 1, \dots, b_j] : j = 1, 2, \dots, n]$.
4  certainty $\leftarrow 0$.
5  while certainty $< \epsilon$ do
       /* Choose index and attack.                                    */
6      $j^* \leftarrow \operatorname*{argmin}_{1 \le j \le n} \left( \max_{0 \le k \le b_j} q[j][k] \right)$.
7      $w[j^*] \leftarrow w[j^*] + \mathcal{O}(j^*, \text{Random}(1, 2, \dots, b_j))$.
8      $\ell[j^*] \leftarrow \ell[j^*] + 1$.

       /* Update probabilities for index $j^*$.                        */
9      den $\leftarrow \sum_{k=0}^{b_{j^*}} (b_{j^*} - k)^{\ell[j^*] - w[j^*]} \cdot k^{w[j^*]}$.
10     $q[j^*] \leftarrow \left[ (b_{j^*} - k)^{\ell[j^*] - w[j^*]} \cdot k^{w[j^*]} / \text{den} : k = 0, 1, \dots, b_{j^*} \right]$.
11     certainty $\leftarrow \prod_{j=1}^{n} \max_{0 \le k \le b_j} q[j][k]$.
12 end
13 Return $q$
```

that is, we guess the e_j which minimizes $\left| \frac{e_j}{b_j} - \frac{w_j}{m_j} \right|$. This value of e_j is what we would obtain by rounding the maximum likelihood estimate for e_j, *if the* a priori *distribution of e_j were uniform on* $[0, b_j]$ *rather than* $[0, b_j] \cap \mathbb{Z}$—note that this is *not* always the same as the maximum likelihood estimate of e_j for our *a priori* distribution. This is most easily seen by considering the case when $0 < w_j < \frac{m_j}{2b_j}$, in which the naïve method recommends guessing $e_j = 0$, while the maximum likelihood method knows that $e_j = 0$ is impossible.

A natural question is this: how large should each m_j be chosen to ensure that we have a reasonable probability (say, $\frac{1}{2}$; more generally, ϵ) of guessing the final key correctly?

We approach this problem from two directions: how large must m_i be to have a sufficiently large guaranteed success probability, and how small can m_i be before the guaranteed

Algorithm 9: Least Certainty Attack (Signed Dynamic)

```
Parameters: Bound vector $\mathbf{b} = (b_1, \ldots, b_n)$ from which keys $e_j \in [-b_j, b_j]$ are
            drawn.
Input     : Certainty bound $\epsilon \in (0, 1)$, unknown private key $\mathbf{e} = (e_1, \ldots, e_n)$
            accessed through oracle $\mathcal{O}$.
Output    : Probability distribution $q_{j,k}$ on key $\mathbf{e}$.
1  $\ell \leftarrow [0 : j = 1, 2, \ldots, n]$.
2  $w \leftarrow [0 : j = 1, 2, \ldots, n]$.
3  $q \leftarrow [\,[1/(2b_j + 1)] \;||\; [2/(2b_j + 1) : k = 1, 2, \ldots, b_j] : j = 1, 2, \ldots, n]$.
4  for certainty $< \epsilon$ do
       /* Choose index and attack.                                        */
5      $j^* \leftarrow \operatorname*{argmin}_{1 \le j \le n} \left( \max_{0 \le k \le b_j} q[j][k] \right)$.
6      $w[j^*] \leftarrow w[j^*] + \mathcal{O}(j^*, \mathrm{Random}(1, 2, \ldots, b_j))$.
7      $\ell[j^*] \leftarrow \ell[j^*] + 1$.

       /* Update probabilities for index $j^*$.                            */
8      $\mathrm{den} \leftarrow b_{j^*}^{\ell[j^*] - w[j^*]} \cdot 0^{w[j^*]} + \sum_{k=1}^{b_{j^*}} 2(b_{j^*} - k)^{\ell[j^*] - w[j^*]} \cdot k^{w[j^*]}$.
9      $q[j^*][0] \leftarrow b_{j^*}^{\ell[j^*] - w[j^*]} \cdot 0^{w[j^*]} / \mathrm{den}$
10     $q[j^*][k] \leftarrow 2(b_{j^*} - k)^{\ell[j^*] - w[j^*]} \cdot k^{w[j^*]} / \mathrm{den}$ for $k = 1, 2, \ldots, b_{j^*}$.
11     certainty $\leftarrow \prod_{j=1}^{n} \max_{0 \le k \le b_j} q[j][k]$.
12 end
13 Return $q$
```

failure probability is too large.

In order to develop these arguments, we must first determine a more convenient expression for when our naïve guess is correct. To begin, our naïve guess is correct exactly when $e_j = \left\lceil b_j \frac{w_j}{m_j} \right\rfloor$. Written more explicitly, we require

$$e_j - \frac{1}{2} \le b_j \frac{w_j}{m_j} < e_j + \frac{1}{2}$$

or, equivalently

$$\frac{e_j}{b_j} - \frac{1}{2b_j} \le \frac{w_j}{m_j} < \frac{e_j}{b_j} + \frac{1}{2b_j};$$

that is, we guess e_j correctly if the empirical probability of detecting a real isogeny ($\frac{w_j}{m_j}$) differs from the true probability ($\frac{e_j}{b_j}$) by at most $\frac{1}{2b_j}$ (on the left) or by strictly less than $\frac{1}{2b_j}$ on the right. We can loosen this bound slightly to obtain the following convenient inequalities:

$$\mathbb{P}\left[\left|\frac{e_j}{b_j} - \frac{w_j}{m_j}\right| < \frac{1}{2b_j}\right] \le \mathbb{P}\left[e_j = \left\lceil b_j \frac{w_j}{m_j} \right\rfloor\right] \le \mathbb{P}\left[\left|\frac{e_j}{b_j} - \frac{w_j}{m_j}\right| \le \frac{1}{2b_j}\right] \tag{6.3}$$

These sandwiching probabilities are of convenient forms for applying generic probability theoretic results.

Guaranteeing sufficient success probability. We begin by determining an upper bound on the number of attacks required in order to guarantee success probability $1 - \epsilon$. We have the following theorem:

Theorem 6.13. *Let* $\mathbf{b}$ *be a bound vector. For any* $\epsilon \in (0, 1)$*, in order to guarantee success probability at least* $1 - \epsilon$ *in a naïve attack on a key chosen from the keyspace defined by* $\mathbf{b}$*, it suffices to inject*

$$\sum_{j=1}^{n} m_j \le \min \left\{ \sum_{j=1}^{n} \left\lceil 2b_j^2 \log_e \frac{2}{1 - \sqrt[n]{1-\epsilon}} \right\rceil, \sum_{j=1}^{n} \left\lceil 2b_j^2 \log_e \left(2 + \frac{\frac{2}{\epsilon}\|\mathbf{b}\|^2 - 2\min_k\{b_k\}^2}{b_j^2} \right) \right\rceil \right\}$$

individual faults.

Proof. We note that in order for the attack on the full key $\mathbf{e}$ to succeed with this probability it suffices that the attack on each individual key entry e_j succeeds with probability $\sqrt[n]{1-\epsilon}$, and so we proceed to determine the number of fault attacks required on e_j to have this success probability.

When performing fault attacks on e_j, the outcome of the i^{th} attack is modelled by a Bernoulli random variable $\zeta_i \sim \mathrm{B}(\frac{e_j}{b_j})$. For $m_j \in \mathbb{N}$, let $Z_{m_j} = \frac{1}{m_j}\sum_{i=1}^{m_j} \zeta_i$. Note first that $\mathbb{E}[Z_{m_j}] = \frac{e_j}{b_j}$; then, by Equation (6.3), we have

$$\mathbb{P}\left[\left|Z_{m_j} - \mathbb{E}[Z_{m_j}]\right| < \frac{1}{2b_j}\right] = \mathbb{P}\left[\left|\frac{e_j}{b_j} - \frac{w_j}{m_j}\right| < \frac{1}{2b_j}\right] \leq \mathbb{P}\left[e_j = \left\lceil b_j \frac{w_j}{m_j}\right\rfloor\right]$$

We will apply a Hoeffding bound [37] to the lefthand side; in particular, for any $t \geq 0$ we have

$$\mathbb{P}\left[\left|Z_{m_j} - \mathbb{E}[Z_{m_j}]\right| < t\right] \geq 1 - 2e^{-2m_j t^2}.$$

Substituting $\mathbb{E}[Z_{m_j}] = \frac{e_j}{b_j}$ and $t = \frac{1}{2b_j}$ we find that

$$\mathbb{P}\left[\left|Z_{m_j} - \frac{e_j}{b_j}\right| < \frac{1}{2b_j}\right] \geq 1 - 2e^{-\frac{m_j}{2b_j^2}}.$$

Thus to ensure sufficient success probability, it suffices to ensure that for each j we have $1 - 2e^{-\frac{m_j}{2b_j^2}} \geq \sqrt[n]{1-\epsilon}$; that is, that

$$m_j \geq 2b_j^2 \log_e \frac{2}{1 - \sqrt[n]{1-\epsilon}}.$$

Since we must inject an integer number of faults, we round the righthand side up to obtain a lower bound.

Of course, there is no reason that the probability of success of each key entry be equal; indeed, it may be possible to achieve the required success probability using fewer total fault injections by achieving higher certainty on some key entries and lower certainty on others. We can formulate an integer convex program which minimizes the total number of fault attacks required to ensure probability $1-\epsilon$ of guessing the key correctly, using the Hoeffding bound to lower bound the success probability of guessing each key entry. Note that

$$\mathbb{P}\left[\mathbf{e} = (e_1^*, \ldots, e_n^*)\right] = \prod_{j=1}^{n} \mathbb{P}[e_j = e_j^*] \geq \prod_{j=1}^{n}\left(1 - 2e^{-\frac{m_j}{2b_j^2}}\right)$$

and so to guarantee success probability $1-\epsilon$ it suffices to have

$$\prod_{j=1}^{n}\left(1-2e^{-\frac{m_j}{2b_j^2}}\right) \geq 1-\epsilon, \text{ or, equivalently, } \sum_{j=1}^{n}\log_e\left(1-2e^{-\frac{m_j}{2b_j^2}}\right) \geq \log_e(1-\epsilon).$$

This constraint is convex; our final integer convex program is

$$\begin{array}{rl} \text{Minimize} & \sum_{j=1}^{n} m_j \\ \text{Subject to} & \sum_{j=1}^{n}\log_e\left(1-2e^{-\frac{m_j}{2b_j^2}}\right) \geq \log_e(1-\epsilon) \\ & \mathbf{m} \in \mathbb{Z}^n \end{array} \tag{P}$$

This program is difficult to solve in general, but we can use duality to find an upper bound on its optimal value.

The Lagrangian for the convex relaxation of (P) is

$$\mathcal{L}(\mathbf{m};\lambda) = \lambda\log_e(1-\epsilon) + \sum_{i=1}^{n}\left(m_i - \lambda\log_e\left(1-2e^{-\frac{m_j}{2b_j^2}}\right)\right)$$

We know from the KKT conditions [12, Section 5.5.3] that at the optimal $\mathbf{m} = \mathbf{m}^*$, there will exist a corresponding multiplier $\lambda^* \geq 0$ which satisfies $\nabla_{\mathbf{m}}\mathcal{L}(\mathbf{m}^*;\lambda^*) = \mathbb{0}$. We compute

$$\frac{\partial}{\partial m_j}\mathcal{L}(\mathbf{m}^*;\lambda^*) = 1 + \frac{\frac{\lambda^*}{b_j^2}}{e^{\frac{m_j^*}{2b_j^2}} - 2}$$

so that

$$\nabla_{\mathbf{m}}\mathcal{L}(\mathbf{m}^*;\lambda^*) = \mathbb{0} \iff m_j^* = 2b_j^2\log_e\left(2+\frac{\lambda^*}{b_j^2}\right) \text{ for all } j.$$

We have thus reduced the problem to determining λ^*. Moreover, noting that if $\mathbf{m} \geq \mathbf{m}^*$ then $\mathbf{m}$ is also feasible for (P), in fact it suffices to find a lower bound on λ^*.

Substituting the form of m_j^* into the constraint, we find that λ^* must satisfy

$$\sum_{j=1}^{n}\log_e\left(1-2e^{-\frac{2b_j^2\log_e\left(2+\frac{\lambda^*}{b_j^2}\right)}{2b_j^2}}\right) = \sum_{j=1}^{n}\log_e\left(1-\frac{2b_j^2}{\lambda^*+2b_j^2}\right) \geq \log_e(1-\epsilon). \tag{6.4}$$

(indeed, it is the smallest solution to the above). Exponentiating both sides of the inequality above, we find that λ^* satisfies

$$\prod_{j=1}^{n} \left(1 - \frac{2b_j^2}{\lambda^* + 2b_j^2}\right) \geq 1 - \epsilon$$

From here we must bound the quantity on the left from below in order to find a lower bound for λ^*. We apply the following straightforward lemma:

Lemma 6.14. Let $n \geq 2$ and $\alpha_1, \alpha_2, \ldots, \alpha_n \in [0, 1]$. Then

$$1 - \sum_{j=1}^{n} \alpha_j \leq \prod_{j=1}^{n} (1 - \alpha_j)$$

Proof. We proceed by induction on n. When $n = 2$ we have

$$(1 - \alpha_1)(1 - \alpha_2) = 1 - \alpha_1 - \alpha_2 + \alpha_1\alpha_2 \geq 1 - (\alpha_1 + \alpha_2)$$

as required. For induction suppose the statement holds up to some $n - 1$; then

$$\begin{aligned}
\prod_{j=1}^{n} (1 - \alpha_j) &= (1 - \alpha_n) \prod_{j=1}^{n-1} (1 - \alpha_j) \\
&\geq (1 - \alpha_n) \left(1 - \sum_{j=1}^{n-1} \alpha_j\right) && \text{(By the inductive hypothesis)} \\
&= 1 - \sum_{j=1}^{n} \alpha_j + \alpha_n \sum_{j=1}^{n-1} \alpha_j \\
&\geq 1 - \sum_{j=1}^{n} \alpha_j
\end{aligned}$$

as required. □

Now for $\lambda \geq 0$ we have $\frac{2b_j^2}{\lambda + 2b_j^2} \in [0, 1]$ for all $b_j \in \mathbb{R}$, and so we can write

$$\prod_{j=1}^{n} \left(1 - \frac{2b_j^2}{\lambda + 2b_j^2}\right) \geq 1 - \sum_{j=1}^{n} \frac{2b_j^2}{\lambda + 2b_j^2} \geq 1 - \sum_{j=1}^{n} \frac{2b_j^2}{\lambda + 2\min_k\{b_k\}^2} = 1 - \frac{2\|\mathbf{b}\|^2}{\lambda + 2\min_k\{b_k\}^2}$$

Thus to guarantee sufficient success probability it suffices to enforce

$$1 - \frac{2\|\mathbf{b}\|^2}{\lambda + 2\min_k\{b_k\}^2} \geq 1 - \epsilon$$

for which we can take $\lambda = \frac{2}{\epsilon}\|\mathbf{b}\|^2 - 2\min_k\{b_k\}^2$, and hence

$$m_j = 2b_j^2 \log_e \left(2 + \frac{\frac{2}{\epsilon}\|\mathbf{b}\|^2 - 2\min_k\{b_k\}^2}{b_j^2}\right) \quad \forall\, j.$$

Of course, the number of faults targeted at each key entry must be integer, so for the purposes of the bound we round up each m_j.

Taking the smaller of the two upper bounds we have derived yields the result of Theorem 6.13. □

Remark 6.15. In numerical experiments the second of the two bounds we derived tends to yield a smaller upper bound at higher certainty levels and when the bound vector has a few large entries and many small entries; in contrast, the first bound performs better at smaller certainty levels and when the bound vector is more uniform.

Guaranteeing sufficient failure probability. We now move on to the following question: how many fault attacks are provably *not* enough to allow a desired success probability? We first consider the "worst case" variant of this question; in particular, for any desired success probability $\frac{1}{2} \leq 1 - \epsilon \leq 1$, we determine a key $\mathbf{e}$ and a number of fault attacks m for which the success probability of the naïve attack is less than $1 - \epsilon$ for key $\mathbf{e}$.

We first consider the special case when each entry of the bound vector $\mathbf{b}$ is even, since this case allows us to apply tighter estimates, leading to stronger overall results. We first state a theorem regarding fault attacks on a single key entry, which we later extend to full keys.

Theorem 6.16. *Let $1 \leq j \leq n$, and let $\mathbf{b}$ be a bound vector with b_j even. Consider a key $\mathbf{e}$ whose j^{th} key entry is $e_j = \frac{1}{2}b_j$. Suppose that the naïve attack is launched, and that m_j faults are targeted at the j^{th} entry of the key. Suppose further that*

$$m_j < \frac{1}{2}b_j^2\left(1 - \sqrt{2\epsilon}\right)^2$$

for some $\epsilon \leq \frac{1}{2}$. Then the attack will return the incorrect value for the j^{th} key entry with probability at least ϵ.

Proof. Let $\{\xi_i\}_{i=1}^{\infty}$ be a sequence of *iid* random variables following the $\mathrm{B}_{\pm}(\frac{1}{2})$ distribution; that is, for all $i \in \mathbb{N}$

$$\mathbb{P}[\xi_i = 1] = \mathbb{P}[\xi_i = -1] = \frac{1}{2}.$$

For each $m \in \mathbb{N}$, define the random variable $\Xi_m = \left|\frac{1}{m}\sum_{i=1}^{m} \xi_i\right|$. In particular, Ξ_{m_j} is the quantity by which the fraction of real isogenies or dummy isogenies detected (whichever is greater) exceeds the fraction of dummy isogenies or real isogenies (whichever is smaller) after m fault attacks are performed; equivalently, it is *twice* the quantity by which fraction of real isogenies or dummy isogenies (whichever is greater) exceeds the expected fraction, $\frac{1}{2}$. We know that after m fault attacks, the probability of failure is at least the probability that Ξ_{m_j} differs from 0 by more than $\frac{1}{b_j}$; we will bound this probability from below.

We first require bounds on $\mathbb{E}[\Xi_m]$. Applying Khintchine's Inequality [44] with optimal constants [35], we find that

$$\frac{1}{\sqrt{2m_j}} \leq \mathbb{E}[\Xi_{m_j}] \leq \frac{1}{\sqrt{m_j}}.$$

We are now in a position to apply the Paley-Zygmund inequality [61]. In particular, for any $\vartheta \in [0, 1]$ we have

$$(1-\vartheta)^2 \frac{\mathbb{E}[\Xi_{m_j}]^2}{\mathbb{E}[\Xi_{m_j}^2]} \leq \mathbb{P}\left[\Xi_{m_j} > \vartheta \mathbb{E}[\Xi_{m_j}]\right] \leq \mathbb{P}\left[\Xi_{m_j} > \frac{\vartheta}{\sqrt{2m_j}}\right].$$

Taking $\vartheta = \frac{\sqrt{2m_j}}{b_j}$ (which is valid for the Paley-Zygmund inequality as long as $m_j \leq \frac{b_j^2}{2}$, which is guaranteed from the hypotheses of the theorem) yields

$$\mathbb{P}[\text{We guess incorrectly}] \geq \mathbb{P}\left[\Xi_{m_j} > \frac{1}{b_j}\right] \geq \left(1 - \frac{\sqrt{2m_j}}{b_j}\right)^2 \frac{\mathbb{E}[\Xi_{m_j}]^2}{\mathbb{E}[\Xi_{m_j}^2]}. \tag{6.5}$$

Since we know $\frac{1}{2m_j} \leq \mathbb{E}[\Xi_{m_j}]^2 \leq \frac{1}{m_j}$, to obtain a lower bound on our failure probability, it suffices to determine $\mathbb{E}[\Xi_{m_j}^2]$. We have

$$\Xi_{m_j}^2 = \frac{1}{m_j}\left(\sum_{i=1}^{m_j} \xi_i\right)^2 = \frac{1}{m_j^2}\sum_{i=1}^{m_j} \xi_i^2 + \frac{1}{m_j^2}\sum_{i=1}^{m_j}\sum_{i'=i+1}^{m_j} \xi_i \xi_{i'}$$

so that

$$\mathbb{E}[\Xi_{m_j}^2] = \frac{1}{m_j} + \frac{1}{m_j^2}\mathbb{E}\left[\sum_{i=1}^{m_j}\sum_{j=i+1}^{m_j}\xi_i\xi_j\right] = \frac{1}{m_j}.$$

since the ξ_i are independent with mean 0 and satisfy $\xi_i^2 = 1$. Substituting this into Equation (6.5) we obtain the following lower bound on the failure probability of the naïve attack on the j^{th} entry of the given key:

$$\mathbb{P}\left[\Xi_{m_j} > \frac{1}{b_j}\right] \geq \frac{1}{2}\left(1 - \frac{\sqrt{2m_j}}{b_j}\right)^2. \tag{6.6}$$

Thus to ensure that the failure probability is sufficiently high for this particular key entry, it suffices to solve $\frac{1}{2}\left(1 - \frac{\sqrt{2m_j}}{b_j}\right)^2 > \epsilon$, whose solution is readily seen to be

$$m_j < \frac{1}{2}b_j^2\left(1 - \sqrt{2\epsilon}\right)^2$$

provided that $\epsilon \leq \frac{1}{2}$, as required.

□

Theorem 6.16 has the following immediate corollary:

Corollary 6.17. Let $\mathbf{b}$ be a bound vector in which the entries labelled by I_{even} are even, and let $T \subseteq I_{\text{even}}$ with $|T| = t$. Then for any naïve attack which targets m_j fault attacks at the j^{th} key entry for each $j \in T$ and succeeds in recovering the full key with probability at least $1 - \epsilon$ for some $\epsilon \leq 1 - 2^{-t}$ when the true key $\mathbf{e}$ satisfies $e_j = \frac{1}{2}b_j \;\; \forall j \in T$, we must have

$$\exists j \in T \text{ such that } m_j \geq \frac{1}{2}b_j^2\left(1 - \sqrt{2\left(1 - \sqrt[t]{1-\epsilon}\right)}\right)^2.$$

Proof. Suppose for contradiction that

$$m_j < \frac{1}{2}b_j^2\left(1 - \sqrt{2\left(1 - \sqrt[t]{1-\epsilon}\right)}\right)^2 \quad \forall j \in T.$$

Applying Theorem 6.16 (noting that $\epsilon \leq 1-2^{-t}$ implies $1-\sqrt[t]{1-\epsilon} \leq \frac{1}{2}$), for each $j \in T$, the probability that the j^{th} key entry is not guessed correctly is strictly greater than $1-\sqrt[t]{1-\epsilon}$.

Noting that in order to recover the full key correctly we must recover the key entries from T correctly, we find that the probability of recovering the full key correctly is at most

$$\mathbb{P}[\text{The attack recovers } \mathbf{e}] < \prod_{j \in T} \left(1 - (1 - \sqrt[t]{1-\epsilon})\right) = 1 - \epsilon$$

as required. $\square$

Taking sets of the form $T = \{j\}$ in Corollary 6.17, we obtain the following result:

Corollary 6.18. Let $\mathbf{b}$ be a bound vector in which the entries labelled by I_{even} are even. Then in any naïve attack which targets m_j fault attacks at the j^{th} key entry for each $j \in I_{\text{even}}$ and succeeds in recovering the full key with probability at least $1 - \epsilon$ for some $\epsilon \leq \frac{1}{2}$ when the true key $\mathbf{e}$ satisfies $e_j = \frac{1}{2} b_j \ \ \forall j \in I_{\text{even}}$, we must have

$$m_j \geq \frac{1}{2} b_j^2 \left(1 - \sqrt{2\epsilon}\right)^2 \ \ \forall j \in I_{\text{even}}.$$

Corollary 6.18 yields the following straightforward lower bound on the number of faults required to ensure success probability $1 - \epsilon$, for any $\epsilon \leq \frac{1}{2}$, when the key satisfies $e_j = \frac{1}{2} b_j \ \ \forall j \in I_{\text{even}}$:

$$\sum_{j \in I_{\text{even}}} m_j \geq \frac{1}{2} \left(1 - \sqrt{2\epsilon}\right)^2 \sum_{j \in I_{\text{even}}} b_j^2.$$

However, with more care, Corollary 6.17 can also be used to obtain the following, stronger result:

Corollary 6.19. Let $\mathbf{b}$ be a bound vector in which the entries labelled by $I_{\text{even}} = \{1, 2, \ldots, t\}$ are even. Then in any naïve attack which targets m_j fault attacks at the j^{th} key entry for each $j \in I_{\text{even}}$ and succeeds in recovering the full key with probability at least $1 - \epsilon$ for some $\epsilon \leq \frac{1}{2}$ when the true key $\mathbf{e}$ satisfies $e_j = \frac{1}{2} b_j \ \ \forall j \in I_{\text{even}}$, we must have

$$\sum_{j \in I_{\text{even}}} m_j \geq \frac{1}{2} \sum_{j=1}^{t} b_j^2 \left(1 - \sqrt{2\left(1 - \sqrt[j]{1-\epsilon}\right)}\right)^2$$

where $\mathbf{b}$ is ordered such that $b_1 \geq b_2 \geq \cdots \geq b_t$.

Proof. To begin, consider the following mathematical program:

$$\begin{array}{ll} \text{Minimize} & \sum_{j\in I_{\text{even}}} m_j \\ \text{Subject to} & \max_{j\in T}\left\{m_j - \tfrac{1}{2}b_j^2\left(1-\sqrt{2\left(1-\sqrt[|T|]{1-\epsilon}\right)}\right)^2\right\} \geq 0 \ \ \forall T \subseteq I_{\text{even}} \end{array} \tag{6.7}$$

By Corollary 6.17, for any fault attack which targets m_j faults at the j^{th} key entry for each $j \in I_{\text{even}}$ and which satisfies the assumptions of Corollary 6.19, the tuple $(m_j)_{j\in I_{\text{even}}}$ is feasible for (6.7). We will prove Corollary 6.19 by solving (6.7) and demonstrating that its optimal value is precisely $\frac{1}{2}\sum_{j=1}^{t} b_j^2\left(1-\sqrt{2\left(1-\sqrt[j]{1-\epsilon}\right)}\right)^2$.

First, this objective value is clearly achievable: simply take

$$m_j = \frac{1}{2}b_j^2\left(1-\sqrt{2\left(1-\sqrt[j]{1-\epsilon}\right)}\right)^2. \tag{6.8}$$

This is feasible for (6.7) since for each $1 \leq k \leq t$, each subset T of I_{even} of size k contains at least one element of $\{t, t-1, \ldots, k\}$, say j^*. Then,

$$m_{j^*} = \frac{1}{2}b_{j^*}^2\left(1-\sqrt{2\left(1-\sqrt[j^*]{1-\epsilon}\right)}\right)^2 \geq \frac{1}{2}b_{j^*}^2\left(1-\sqrt{2\left(1-\sqrt[k]{1-\epsilon}\right)}\right)^2$$

since $j^* \geq k$, establishing the feasibility of (6.8).

To demonstrate that (6.8) is optimal for (6.7), we first have the following claim.

Claim 6.20. Let $\hat{\mathbf{m}}$ be feasible for (6.7). Then there exists $\tilde{\mathbf{m}}$ which:

- Is feasible for (6.7);
- Has objective value no larger than $\hat{\mathbf{m}}$, and;
- Satisfies

  $$\tilde{m}_{k_j'} = \frac{1}{2}b_{k_j}^2\left(1-\sqrt{2\left(1-\sqrt[j]{1-\epsilon}\right)}\right)^2$$

  for $j = 1, 2, \ldots t$, where $(k_1, k_2, \ldots, k_t)$ is a permutation of $\{1, 2, \ldots, t\}$ such that

  $$\frac{2\hat{m}_{k_1}}{b_{k_1}^2} \leq \frac{2\hat{m}_{k_2}}{b_{k_2}^2} \leq \cdots \leq \frac{2\hat{m}_{k_t}}{b_{k_t}^2}$$

Proof. Let $(k_1, k_2, \dots, k_t)$ be a permutation of $\{1, 2, \dots, t\}$ such that

$$\frac{2\hat{m}_{k_1}}{b_{k_1}^2} \le \frac{2\hat{m}_{k_2}}{b_{k_2}^2} \le \cdots \le \frac{2\hat{m}_{k_t}}{b_{k_t}^2}.$$

Let $\alpha_j = \frac{2\hat{m}_{k_j}}{b_{k_j}^2}$ for each $j \in \{1, 2, \dots, t\}$. If $\alpha_j = \left(1 - \sqrt{2\left(1 - \sqrt[j]{1-\epsilon}\right)}\right)^2$ for all j, we are done; if not, let

$$J = \left\{ j^* \, : \, \alpha_{j*} \neq \left(1 - \sqrt{2\left(1 - \sqrt[j^*]{1-\epsilon}\right)}\right)^2 \right\} \neq \varnothing.$$

There are then two cases:

Case 1: $\exists j^* \in J$ such that $\alpha_{j*} < \left(1 - \sqrt{2\left(1 - \sqrt[j^*]{1-\epsilon}\right)}\right)^2$.

Choose the smallest such j^* and let $T^* = \{k_1, k_2, \dots, k_{j^*}\}$. By our choice of j^*, we have $\alpha_j < \left(1 - \sqrt{2\left(1 - \sqrt[j^*]{1-\epsilon}\right)}\right)^2$ for all $j \le j^*$; that is,

$$\hat{m}_{k_j} < \frac{1}{2} b_{k_j}^2 \left(1 - \sqrt{2\left(1 - \sqrt[j^*]{1-\epsilon}\right)}\right)^2$$

for all $j \in \{1, 2, \dots, j^*\}$. Thus $\hat{\mathbf{m}}$ violates the constraint

$$\max_{j \in T^*} \left\{ m_j - \frac{1}{2} b_j^2 \left(1 - \sqrt{2\left(1 - \sqrt[|T^*|]{1-\epsilon}\right)}\right)^2 \right\} \ge 0$$

so that $\hat{\mathbf{m}}$ was not feasible to begin with.

Case 2: $\alpha_j > \left(1 - \sqrt{2\left(1 - \sqrt[j]{1-\epsilon}\right)}\right)^2 \quad \forall j \in J$.

In this case, consider the new vector $\tilde{\mathbf{m}}$ defined by

$$\tilde{m}_{k_j} = \begin{cases} \frac{1}{2} b_{k_j}^2 \left(1 - \sqrt{2\left(1 - \sqrt[j]{1-\epsilon}\right)}\right)^2 & \text{if } j \in J \\ \hat{m}_j & \text{otherwise} \end{cases}.$$

It is clear that $\tilde{\mathbf{m}} \le \hat{\mathbf{m}}$ and so $\tilde{\mathbf{m}}$ achieves a better objective value, and so it only remains to show that $\tilde{\mathbf{m}}$ is feasible. Let $T \subseteq \{1, 2, \dots, n\}$, and consider the constraint of (6.7) indexed by T. We consider three cases:

Case a: $T \cap J = \varnothing$.

In this case the constraint is clearly satisfied since $\tilde{m}_{k_j} = \hat{m}_{k_j}$ as long as $j \notin J$.

Case b: $T \cap J \neq \varnothing$ and $|T| \leq \max_{j \in T \cap J}\{j\}$.

Let $j^* = \max_{j \in T \cap J}\{j\}$. In this case the constraint is satisfied because $j^* \in T$ and

$$\tilde{m}_{j^*} = \frac{1}{2} b_{k_{j^*}}^2 \left(1 - \sqrt{2\left(1 - \sqrt[j^*]{1-\epsilon}\right)}\right)^2 \geq \frac{1}{2} b_{k_{j^*}}^2 \left(1 - \sqrt{2\left(1 - \sqrt[|T|]{1-\epsilon}\right)}\right)^2$$

since $|T| \leq j^*$.

Case c: $k_{j^*} \in T$ and $|T| > \max_{j \in T \cap J}\{j\}$.

Again, let $j^* = \max_{j \in T \cap J}\{j\}$. There are two possibilities:

Case i: $\alpha_{j^*} < \left(1 - \sqrt{2(1 - \sqrt[|T|]{1-\epsilon})}\right)^2$.

Since $\hat{\mathbf{m}}$ was feasible for (6.7), we know that $\exists k_{j_T} \in T$ such that

$$\hat{m}_{k_{j_T}} \geq \frac{1}{2} b_{k_{j_T}}^2 \left(1 - \sqrt{2(1 - \sqrt[|T|]{1-\epsilon})}\right)^2.$$

Since this inequality is not satisfied when $j_T = j^*$ (by the assumption of the case), we have $j_T \neq j^*$. Then, since $\tilde{m}_{k_{j_T}} = \hat{m}_{k_{j_T}}$, the constraint indexed by T is indeed satisfied.

Case ii: $\alpha_{j^*} \geq \left(1 - \sqrt{2(1 - \sqrt[|T|]{1-\epsilon})}\right)^2$.

Since $|T| > j^*$, $\exists j_T > j^*$ with $k_{j_T} \in T$. By the ordering of the α_j, this implies that

$$\hat{m}_{k_{j_T}} \geq \frac{1}{2} b_{k_{j_T}}^2 \left(1 - \sqrt{2(1 - \sqrt[|T|]{1-\epsilon})}\right)^2$$

so that the constraint indexed by T is indeed satisfied.

The $\tilde{\mathbf{m}}$ constructed in case 2 is precisely what is hypothesized. □

In light of Claim 6.20, it suffices to demonstrate that our proposed solution $\mathbf{m}$ has the smallest objective value among all vectors of the form $\tilde{\mathbf{m}}$. We use the following claim:

Claim 6.21. Let $\tilde{\mathbf{m}}$ satisfy

$$\tilde{m}_{k_j} = \frac{1}{2} b_{k_j}^2 \left(1 - \sqrt{2\left(1 - \sqrt[j]{1-\epsilon}\right)}\right)^2$$

for $j = 1, 2, \dots t$, where $(k_1, k_2, \dots, k_t)$ is a permutation of $\{1, 2, \dots, t\}$ such that

$$\frac{2\hat{m}_{k_1}}{b_{k_1}^2} \leq \frac{2\hat{m}_{k_2}}{b_{k_2}^2} \leq \dots \leq \frac{2\hat{m}_{k_t}}{b_{k_t}^2}.$$

Suppose that $(k_1, k_2, \dots, k_t)$ has at least one inversion (that is, there exists $j_1 < j_2$ with $k_{j_1} > k_{j_2}$). Then there exists $\mathbf{m}'$ which

- Has objective value no greater than that of $\tilde{\mathbf{m}}$, and;
- Satisfies the above condition, with a new permutation $(k_1', k_2', \dots, k_t')$ which has fewer inversions than $(k_1, k_2, \dots, k_t)$.

Proof. Let j_1, j_2 be an inversion in $(k_1, k_2, \dots, k_t)$. Define

$$k_j' = \begin{cases} k_{j_2} & \text{if } j = j_1 \\ k_{j_1} & \text{if } j = j_2 \\ k_j & \text{otherwise} \end{cases}.$$

and $m_{k_j'}' = \frac{1}{2} b_{k_j'} \left(1 - \sqrt{2\left(1 - \sqrt[j]{1-\epsilon}\right)}\right)^2$ for $j = 1, 2, \dots, t$. Clearly $(k_1', k_2', \dots, k_t')$ has fewer inversions than $(k_1, k_2, \dots, k_t)$, since we have swapped two entries of an inversion and left the other entries alone. As for their objective values, we have

$$\begin{aligned}
&\sum_{j \in I_{\text{even}}} \tilde{m}_j - \sum_{j \in I_{\text{even}}} m_j' \\
&\qquad = \frac{1}{2} \underbrace{(b_{k_{j_1}}^2 - b_{k_{j_2}}^2)}_{\geq 0} \underbrace{\left(\left(1 - \sqrt{2(1-(1-\epsilon)^{1/k_{j_1}})}\right)^2 - \left(1 - \sqrt{2(1-(1-\epsilon)^{1/k_{j_2}})}\right)^2\right)}_{\geq 0} \\
&\qquad \geq 0
\end{aligned}$$

so that $\mathbf{m}'$ has objective value at most that of $\tilde{\mathbf{m}}$, as required. □

Thus the optimal solution $\mathbf{m}$ to (6.7) must satisfy

$$m_{k_j} = \frac{1}{2} b_{k_j}^2 \left(1 - \sqrt{2\left(1 - \sqrt[j]{1-\epsilon}\right)}\right)^2$$

for $j = 1, 2, \ldots t$, where $(k_1, k_2, \ldots, k_t)$ is a permutation of $\{1, 2, \ldots, t\}$ such that

$$\frac{2\hat{m}_{k_1}}{b_{k_1}^2} \leq \frac{2\hat{m}_{k_2}}{b_{k_2}^2} \leq \cdots \leq \frac{2\hat{m}_{k_t}}{b_{k_t}^2};$$

moreover, there exists such an optimal solution for which $(k_1, k_2, \ldots, k_t)$ has no inversions; that is, it is the identity permutation. This is precisely our proposed solution, and hence the proof is complete.

□

The hypothesis that the error probability ϵ satisfies $\epsilon \leq \frac{1}{2}$ is required for Corollary 6.19 in order for the lower bound of Corollary 6.18 to be valid. When $\epsilon > \frac{1}{2}$, however, we can still obtain a lower bound on the number of faults required in the worst case by considering the inequalities of Corollary 6.17 which remain valid. In particular, we have the following result.

Corollary 6.22. Let $\mathbf{b}$ be a bound vector in which the entries labelled by $I_{\text{even}} = \{1, 2, \ldots, t\}$ are even. Then in any naïve attack which targets m_j fault attacks at the j^{th} key entry for each $j \in I_{\text{even}}$ and succeeds in recovering the full key with probability at least $1-\epsilon$ for some $\epsilon \leq 1 - 2^{-s}$ for some integer $1 \leq s \leq t$ when the true key $\mathbf{e}$ satisfies $e_j = \frac{1}{2} b_j \;\; \forall j \in I_{\text{even}}$, we must have

$$\sum_{j=s}^{t} m_j \geq \frac{1}{2} \sum_{j=s}^{t} b_j^2 \left(1 - \sqrt{2\left(1 - \sqrt[j]{1-\epsilon}\right)}\right)^2$$

where $\mathbf{b}$ is ordered such that $b_1 \geq b_2 \geq \cdots \geq b_t$.

Proof. By Corollary 6.17, for each subset T of I_{even} of size at least s, we must have

$$\exists j \in T \text{ such that } m_j \geq \frac{1}{2} b_j^2 \left(1 - \sqrt{2\left(1 - \sqrt[|T|]{1-\epsilon}\right)}\right)^2.$$

For sets T of size less than s we cannot apply Corollary 6.17 because ϵ may be too large; the lower bound of this corollary compared with Corollary 6.19 is a consequence of this fact.

From here, the result follows by applying the technique of the proof of Corollary 6.19 to the candidate solution $\mathbf{m}$ defined by

$$m_j = \begin{cases} 0 & \text{if } j < s \\ \frac{1}{2}b_j^2\left(1 - \sqrt{2\left(1 - \sqrt[j]{1-\epsilon}\right)}\right)^2 & \text{if } j \geq s \end{cases}.$$

□

From here, we continue our "worst-case" analysis by extending to bound vectors whose entries may be even or odd, and arbitrary key values. In particular, we have the following result:

Theorem 6.23. *Let* $\mathbf{b}$ *be a bound vector entry and let* $e_j \in \{1, 2, \dots, b_j - 1\}$ *be a possible value of* j^{th} *key entry. Let* $\hat{e}_j = \min\{e_j, b_j - e_j\}$*, and* $0 \leq \epsilon \leq \frac{\hat{e}_j}{8(b_j - \hat{e}_j)}$*. Then for a naïve attack which targets* m_j *faults at the* j^{th} *key entry and which correctly recovers its value with probability at least* $1 - \epsilon$ *when it is equal to* e_j*, we have*

$$m_j \geq \frac{1}{2}\hat{e}_j^2\left(1 - 2\sqrt{\frac{2(b_j - \hat{e}_j)}{\hat{e}_j}\epsilon}\right)^2,$$

Proof. Let $\Xi_{m_j} = |\sum_{i=1}^{m_j} \xi_i|$ for $\xi_i \sim \mathrm{B}_{\pm}(p)$ independent and identically distributed. Note that each $\mathbb{E}[\xi_i] = 0$, so we are able to apply the Marcinkiewicz-Zygmund inequality [53] with optimal constants [18, Theorem 10.3.2], which states that

$$\frac{1}{2\sqrt{2}}\mathbb{E}\left[\sqrt{\sum_{i=1}^{m_j}\xi_i^2}\right] \leq \mathbb{E}[\Xi_{m_j}] \leq 2\mathbb{E}\left[\sqrt{\sum_{i=1}^{m_j}\xi_i^2}\right]$$

for all $m_j \geq 1$. We note that these constants are precisely half (on the left) and twice (on the right) the corresponding constants from the Khintchine inequality; this will lead us to looser bounds, which is why we considered the case of $e_j = \frac{b_j}{2}$ separately. We have that

$$\mathbb{E}\left[\sqrt{\sum_{i=1}^{m_j}\xi_i^2}\right] = \sum_{k=0}^{m_j}\binom{m_j}{k}p^k(1-p)^{m_j-k}\sqrt{\frac{k}{4p^2} + \frac{m_j - k}{4(1-p)^2}}.$$

For simplicity, assume that $p \leq \frac{1}{2}$ (the $p > \frac{1}{2}$ case can be treated similarly); then from the above we can write

$$\frac{\sqrt{m_j}}{2(1-p)} \leq \mathbb{E}\left[\sqrt{\sum_{i=1}^{m_j} \xi_i^2}\right] \leq \frac{\sqrt{m_j}}{2p}$$

from which it follows by the Marcinkiewicz-Zygmund inequality that

$$\frac{\sqrt{m_j}}{4\sqrt{2}(1-p)} \leq \mathbb{E}[\Xi_{m_j}] \leq \frac{\sqrt{m_j}}{p}.$$

Now, we would like to apply the Paley-Zygmund inequality again; to do so, we must compute $\mathbb{E}[\Xi_{m_j}^2]$. We have

$$\begin{aligned}
\Xi_{m_j}^2 &= \sum_{i=1}^{m_j} \xi_i^2 + \sum_{i=1}^{m_j} \sum_{k=i+1}^{m_j} \xi_i \xi_k \\
\implies \mathbb{E}[\Xi_{m_j}^2] &= \sum_{i=1}^{m_j} \mathbb{E}[\xi_i^2] + \sum_{i=1}^{m_j} \sum_{k=i+1}^{m_j} \mathbb{E}[\xi_i]\mathbb{E}[\xi_k] \\
&= m_j \cdot \left(\frac{p}{4p^2} + \frac{1-p}{4(1-p)^2}\right) + \binom{m_j}{2} \cdot 0^2 = \frac{m_j}{4p(1-p)}.
\end{aligned}$$

Thus for any $\vartheta \in [0,1]$ we have

$$\begin{aligned}
\mathbb{P}\left[\Xi_{m_j} > \vartheta \frac{\sqrt{m_j}}{4\sqrt{2}(1-p)}\right] &\geq \mathbb{P}\left[\Xi_{m_j} > \vartheta \mathbb{E}[\Xi_{m_j}]\right] \\
&\geq (1-\vartheta)^2 \frac{\mathbb{E}[\Xi_{m_j}]^2}{\mathbb{E}[\Xi_{m_j}^2]} \\
&\geq (1-\vartheta)^2 \frac{\frac{m_j}{32(1-p)^2}}{\frac{m_j}{4p(1-p)}} = (1-\vartheta)^2 \frac{p}{8(1-p)}. \qquad (6.9)
\end{aligned}$$

How does this connect to fault attacks on CSIDH? Suppose that the j^{th} key entry is $e_j \in [1, \frac{b_j}{2}]$; letting $p = \frac{e_j}{b_j}$, we can think of ξ_i as reporting the result of the i^{th} fault attack on e_j, returning the positive result if the corresponding isogeny is real, and the negative result if the isogeny is dummy.

Let w_j be the number of real isogenies detected; if we can express $\mathbb{P}[|\frac{w_j}{m_j} - \frac{e_j}{b_j}| > \frac{1}{2b_j}]$ in terms of the behaviour of Ξ_{m_j}, we can lower bound our failure probability. Note that

$$\Xi_{m_j} = \left|\frac{w_j}{2p} - \frac{m_j - w_j}{2(1-p)}\right| = \left|\frac{w_j}{2p(1-p)} - \frac{m_j}{2(1-p)}\right| = \frac{m_j}{2p(1-p)}\left|\frac{w_j}{m_j} - p\right|$$

and so

$$\left|\frac{w_j}{m_j} - \frac{e_j}{b_j}\right| > \frac{1}{2b_j} \iff \Xi_m > \frac{m_j}{4p(1-p)b_j} = \frac{m_j b_j}{4e_j(b_j - e_j)}.$$

Now, taking $\vartheta = \frac{\sqrt{2m_j}}{e_j}$ in Equation (6.9) we obtain

$$\mathbb{P}\left[\left|\frac{w_j}{m_j} - \frac{e_j}{b_j}\right| > \frac{1}{2b_j}\right] = \mathbb{P}\left[\Xi_{m_j} > \frac{m_j b_j}{4(b_j - e_j)e_j}\right] \geq \left(1 - \frac{\sqrt{2m_j}}{e_j}\right)^2 \frac{e_j}{8(b_j - e_j)}$$

whenever $e_j \leq \frac{b_j}{2}$ and $1 \leq m_j \leq \frac{1}{2}e_j^2$. Similarly we find that when $e_j > \frac{b_j}{2}$ we have

$$\mathbb{P}\left[\left|\frac{w_j}{m_j} - \frac{e_j}{b_j}\right| > \frac{1}{2b_j}\right] = \mathbb{P}\left[\Xi_{m_j} > \frac{m_j b_j}{4e_j(b_j - e_j)}\right] \geq \left(1 - \frac{\sqrt{2m_j}}{b_j - e_j}\right)^2 \frac{b_j - e_j}{8e_j}$$

whenever $1 \leq m_j \leq \frac{1}{2}(b_j - e_j)^2$. We can unify these statements as

$$\mathbb{P}\left[\left|\frac{w_j}{m_j} - \frac{e_j}{b_j}\right| > \frac{1}{2b_j}\right] \geq \left(1 - \frac{\sqrt{2m_j}}{\min\{e_j, b_j - e_j\}}\right)^2 \frac{\min\{e_j, b_j - e_j\}}{8\max\{e_j, b_j - e_j\}}$$

for all $e_j \in \{1, \ldots, b_j - 1\}$, provided that $1 \leq m_j \leq \frac{1}{2}\min\{e_j^2, (b_j - e_j)^2\}$. Thus, if

$$1 \leq m_j < \frac{1}{2}\min\{e_j^2, (b_j - e_j)^2\}\left(1 - 2\sqrt{\frac{2\max\{e_j, b_j - e_j\}}{\min\{e_j, b_j - e_j\}}\epsilon}\right)^2$$

we are guaranteed that the attack will fail to correctly recover the j^{th} key entry with probability greater than ϵ, as required. $\square$

We can extend these results to the "average-case" (over uniformly random key entries) by noting that the *a priori* probability that the j^{th} key entry is any particular value

$e_j^* \in \{1, 2, \ldots, b_j - 1\}$ is $\frac{1}{b_j+1}$; then, we have the lower bound

$$\mathbb{P}[\text{We guess } e_j \text{ incorrectly}] \geq \mathbb{P}\left[e_j = e_j^*\right] \cdot \mathbb{P}\left[\text{We guess } e_j \text{ incorrectly } \;\middle|\; e_j = e_j^*\right]$$
$$\geq \begin{cases} \frac{1}{2(b_j+1)}\left(1 - \frac{\sqrt{2m_j}}{b_j}\right)^2 & \text{if } e_j^* = \frac{b_j}{2} \\ \frac{\min\{e_j^*, b_j - e_j^*\}}{8(b_j+1)\max\{e_j^*, b_j - e_j^*\}}\left(1 - \frac{\sqrt{2m_j}}{\min\{e_j^*, b_j - e_j^*\}}\right)^2 & \text{otherwise} \end{cases}$$

whenever $m_j \leq \frac{1}{2}\min\{e_j^*, b_j - e_j^*\}^2$. From there, one could perform an analogous analysis to the worst-case case.

6.8.4 Determining the Signs of the Key

When the key entries are signed, the techniques of the previous section can be used to determine the key up to sign. To determine the sign of the key entries, we use a standard meet-in-the-middle approach; in particular, we split the primes into two batches

$$B_L = \{\ell_1, \ell_2, \ldots, \ell_k\} \text{ and } B_R = \{\ell_{k+1}, \ell_{k+2}, \ldots, \ell_n\}$$

(where $k = \lceil \frac{n}{2} \rceil$) and populate two tables T_L and T_R with curves of the form

$$E_L = [\mathfrak{l}_1^{(-1)^{s_1}|e_1^*|} \cdots \mathfrak{l}_k^{(-1)^{s_k}|e_k^*|}] * E_0 \text{ and } E_R = [\mathfrak{l}_{k+1}^{(-1)^{s_{k+1}}|e_{k+1}^*|} \cdots \mathfrak{l}_n^{(-1)^{s_n}|e_n^*|}] * E_A$$

respectively, for $s_1, s_2, \ldots, s_n \in \{0, 1\}$. When a match between the tables is found at $s_1^*, s_2^*, \ldots, s_n^*$, the correct key is

$$\mathbf{e}^* = ((-1)^{s_1^*}|e_1^*|, \ldots, (-1)^{s_k^*}|e_k^*|, -(-1)^{s_{k+1}^*}|e_{k+1}^*|, \ldots, -(-1)^{s_n^*}|e_n^*|).$$

Naïvely, populating these tables requires evaluating the class group action $2^k + 2^{n-k}$ times, using ideals whose product decomposition contains $\sum_{i=1}^k |e_i^*|$ terms (for T_L) or $\sum_{i=k+1}^n |e_i^*|$ terms (for T_R). However, we can make this more efficient by constructing the entries of the tables in a better order.

In particular, if we order our choices of $s_1, s_2, \ldots, s_k$ according to a length-k binary Gray code C, we need only apply a class group element of the form $[\mathfrak{l}_j]^{\pm 2|e_j^*|}$ when s_j changes.

This reduces the cost to

$$\sum_{i=1}^{k} 2\tau_i |e_i^*| \kappa_i \tag{6.10}$$

where $\boldsymbol{\tau}$ are the *transition numbers* of C—that is, τ_i is the number of times that the i^{th} bit flips in C—and κ_i is the cost of evaluating $(E, \ell_i) \mapsto [\mathfrak{l}_i] * E$.

It remains to determine the Gray code C that, for a given $\boldsymbol{\kappa}$ and $\mathbf{e}^*$, minimizes the cost of constructing the required curves. We have the following result:

Lemma 6.24. Suppose the primes $\ell_1, \ell_2, \ldots, \ell_k$ are ordered such that

$$|e_1^*|\kappa_1 \leq |e_2^*|\kappa_2 \leq \cdots \leq |e_k^*|\kappa_k.$$

Then, the reflected binary code minimizes the quantity of Equation (6.10) over Gray codes of length k.

Proof. Recall that the reflected binary code has transition numbers

$$\boldsymbol{\tau}^* = (2^{k-1}, 2^{k-2}, \ldots, 1).$$

We have the following result regarding transition numbers:

Claim 6.25. Let $\boldsymbol{\tau}$ be the transition numbers of a binary Gray code of length k. Then for any $T \subseteq \{1, 2, \ldots, k\}$ we have

$$\sum_{t \in T} \tau_t \geq 2^{|T|} - 1.$$

Proof (of claim). Consider the "code" $C|_T$ obtained from C by:

1. Removing $\mathbf{c}_i$ whenever $\mathbf{c}_{i-1} \oplus \mathbf{c}_i = \hat{\mathbf{e}}_j$ for some $j \notin T$; and,
2. Replacing $\mathbf{c}_i$ by $\mathbf{c}_i|_T$ for $1 \leq i \leq 2^k$.

We note that $C|_T$ has exactly $2^k - \sum_{j \notin T} \tau_j = \sum_{t \in T} \tau_t + 1$ codewords, and that each word $\mathbf{v} \in \{0,1\}^{|T|}$ is a codeword of $C|_T$, since C was a Gray code. Thus we have

$$\sum_{t \in T} \tau_t + 1 \geq 2^{|T|} \implies \sum_{t \in T} \tau_t \geq 2^{|T|} - 1$$

as required. □

Now define the linear program

$$\begin{aligned} &\text{Minimize} \quad \sum_{i=1}^{k}(2|e_i^*|\kappa_i)\tau_i \\ &\text{Subject to} \quad \sum_{t\in T}\tau_t \geq 2^{|T|}-1 \ \text{ for all } T \subseteq \{1,2,\ldots,k\} \end{aligned} \tag{GC}$$

By the claim above, any $\boldsymbol{\tau}$ which is the transition numbers of a Gray code will be feasible for (GC); moreover, its objective value is precisely will be precisely the expression (6.10). Thus to prove that the reflected binary code is optimal, it suffices to show that $\boldsymbol{\tau}^*$ is optimal for (GC).

The dual program to (GC) is

$$\begin{aligned} &\text{Maximize} \quad \sum_{T\subseteq\{1,2,\ldots,k\}}(2^{|T|}-1)y_T \\ &\text{Subject to} \quad \sum_{\substack{T\subseteq\{1,2,\ldots,k\} \\ \text{s.t. } t\in T}} y_T = 2|e_i^*|\kappa_i \ \text{ for all } t \in \{1,2,\ldots,k\} \\ &\qquad\qquad\qquad \mathbf{y} \geq \mathbb{0} \end{aligned} \tag{GCD}$$

In order to prove that $\boldsymbol{\tau}^*$ is optimal for (GC) it suffices to find $\mathbf{y}^*$ which is feasible for (GCD) and which satisfies the complementary slackness conditions

$$y_T^* \cdot \left(\sum_{t\in T}\tau_t^* - (2^{|T|}-1)\right) = 0 \text{ for all } T \subseteq \{1,2,\ldots,k\}. \tag{6.11}$$

Claim 6.26. The vector $\mathbf{y}^*$ defined by

$$y_T^* = \begin{cases} 2|e_1^*|\kappa_1 & \text{if } T = \{1,2,\ldots,k\} \\ 2|e_i^*|\kappa_i - 2|e_{i-1}^*|\kappa_{i-1} & \text{if } T = \{i,i+1,\ldots,k\} \text{ with } i \geq 2 \\ 0 & \text{otherwise} \end{cases}$$

is feasible for (GCD) and satisfies Equation (6.11), and hence proves the optimality of $\boldsymbol{\tau}^*$ in (GC).

Proof (of claim). To begin, we note that $\mathbf{y} \geq \mathbb{0}$ since $2|e_i^*|\kappa_i \geq 2|e_{i-1}^*|\kappa_{i-1}$ for all $i \geq 2$ by our choice of ordering of the ℓ_i, and $2|e_1^*|\kappa_1 > 0$. As well, for all t we have

$$\sum_{\substack{T \subseteq \{1,2,\ldots,k\} \\ \text{s.t. } t \in T}} y_T = 2|e_1^*|\kappa_1 + \sum_{i=2}^{t} (2|e_i^*|\kappa_i - 2|e_{i-1}^*|\kappa_{i-1}) = 2|e_t^*|\kappa_t$$

as required, and so $\mathbf{y}^*$ really is feasible for (GCD). As for the complementary slackness conditions, it is clear that they hold for each T with $y_T^* = 0$, and so we only need to consider $T = \{t, t+1, \ldots, k\}$ for $t = 1, 2, \ldots, k$.

For each such T we have

$$\sum_{j \in T} \tau_t^* = \sum_{j=t}^{k} 2^{k-j} = 2^{k-t+1} - 1 = 2^{|T|} - 1$$

so that the complementary slackness condition is indeed satisfied, completing the proof of the claim. □

Since $\boldsymbol{\tau}^*$ is optimal for (GC), the reflected binary code is indeed the optimal code for populating the tables, as required. □

The reflected binary code yields an optimal order in which to populate the table for the claw-finding attack if we restrict ourselves to algorithms which must construct the next entry of the table using the current entry as the starting point. However, at least for the left batch of primes, we are storing all curves in memory simultaneously; thus we can in principle start the computation of the next table entry from *any* of the previously-computed curves. Such an algorithm does not necessarily correspond to a Gray code; rather, it corresponds to a spanning tree in the hypercube graph $H_k = (V, E)$ defined by

$$V = \mathbb{Z}_2^k \text{ and } E = \{\{\mathbf{u}, \mathbf{v}\} : \mathbf{u} \oplus \mathbf{v} = \hat{\mathbf{e}}_j \text{ for some } j\}.$$

(where $\{\hat{\mathbf{e}}_j\}_j$ is the standard basis of $\mathbb{R}^k$) with edge weights defined by

$$\mathbf{u} \oplus \mathbf{v} = \hat{\mathbf{e}}_j \implies w_{\{\mathbf{u},\mathbf{v}\}} = 2|e_j^*|\kappa_j.$$

Gray codes of length k are Hamilton paths in H_k, which are a particular kind of spanning tree; however, in principle we can do better by considering other kinds of spanning trees. In reality, we have the following result.

Lemma 6.27. Suppose the primes $\ell_1, \ell_2, \ldots, \ell_k$ are ordered such that

$$|e_1^*|\kappa_1 \leq |e_2^*|\kappa_2 \leq \cdots \leq |e_k^*|\kappa_k.$$

Then, the Hamilton path constructed from the reflected binary code is a minimal spanning tree for H_k, with the edge weights defined above.

Proof. We begin by defining the analogue of the transition numbers $\boldsymbol{\tau}$ for a spanning tree $\mathcal{T} = (V_{\mathcal{T}}, E_{\mathcal{T}})$ of H_k:

$$\tau_j = |\{\{\mathbf{u}, \mathbf{v}\} \in E_{\mathcal{T}} \,:\, \mathbf{u} \oplus \mathbf{v} = \hat{\mathbf{e}}_j\}|.$$

We have the following analogue of the first claim from Lemma 6.24

Claim 6.28. Let $\boldsymbol{\tau}$ be the transition numbers of a spanning tree $\mathcal{T}$ in H_k. Then for any $T \subseteq \{1, 2, \ldots, k\}$ we have

$$\sum_{t \in T} \tau_t \geq 2^{|T|} - 1.$$

Proof (of claim). Consider the graph $\mathcal{T}|_T$ constructed from $\mathcal{T}$ by identifying each collection of vertices that differ only in their entries indexed by $\{1, 2, \ldots, k\} \backslash T$, relabelling each vertex $\mathbf{v}$ by $\mathbf{v}|_T$, and eliminating any loops (edges incident at only one vertex). The graph $\mathcal{T}|_T$:

- Has vertex set $\mathbb{Z}_2^{|T|}$, and hence has $2^{|T|}$ vertices;
- Is connected, because $\mathcal{T}$ was connected; and,
- Has exactly $|E_{\mathcal{T}}| - \sum_{j \notin T} \tau_j = \sum_{t \in T} \tau_t$ edges.

Since any connected graph on $2^{|T|}$ vertices has at least $2^{|T|} - 1$ edges, it follows immediately that

$$\sum_{t \in T} \tau_t \geq 2^{|T|} - 1$$

as required. $\square$

It follows then that the transition numbers $\boldsymbol{\tau}$ of any spanning tree in H_k are feasible for the program (GC); we already know from Lemma 6.24 that $\boldsymbol{\tau}^*$ (the transition numbers for the reflected binary code) is optimal for (GC); thus the corresponding Hamilton path in H_k is indeed a minimum spanning tree. $\square$

Given that for a given partition B_L, B_R of the primes into two sets, we know how to order the primes in each set, it remains to determine how to determine the sets themselves. It is clear from Lemma 6.24 that, if the full set of primes is ordered so that

$$|e_1^*|\kappa_1 \leq |e_2^*|\kappa_2 \leq \cdots \leq |e_n^*|\kappa_n$$

that we should alternately assign the primes to B_L and B_R, so that

$$\begin{aligned} B_L &= \{\ell_j \,:\, j \equiv 1 \pmod 2 \text{ and } 1 \leq j \leq n\} \\ B_R &= \{\ell_j \,:\, j \equiv 0 \pmod 2 \text{ and } 1 \leq j \leq n\}. \end{aligned}$$

6.9 Simulation Results

Table 6.1 reports on the mean number of attacks used in our simulations for each vector to reach a certainty level of 1%, 50%, 99%, and 99.9%. In particular, for the unsigned setting we found that the mean number of faults required for the attacker to reach 1% certainty was 15921 for HLKA, 12067 for MCR, and 10584 for Uniform, while reaching 99.9% certainty required a mean of 52092 attacks for HLKA, 39738 for MCR, and 35129 for Uniform. In contrast, this is a drastic improvement over the attacks needed in the real-then-dummy setting (exactly $\sum_j \lceil \log(b_j) + 1 \rceil$; see Section 6.8.1), in which the number of attacks required to reach 99.9% certainty increased by a factor of about 146 for HLKA, 116 for MCR, and 95 for Uniform. Reaching the same certainty levels in the signed setting requires fewer attacks since the key is only learned up to sign and the bound vector entries are typically smaller. Reaching a certainty level of 1% in the signed setting used 3039 faults for HLKA, 3552 for OAYT, and 2484 for Uniform, while a certainty level of 99.9% needed 10734 faults on average for HLKA, 12447 for OAYT, and 8890 for Uniform. In the signed setting, the number of attacks used to learn the key up to sign with 99.9% certainty increased by a factor of 40 for HLKA when using a dynamic decision vector, a factor of 46 for OAYT, and of 30 for Uniform.

Table 6.2 contains the upper bounds on the number of faults required at the four certainty levels 1%, 50%, 99%, 99.9% for the six bound vectors listed here. At each certainty level these upper bounds are on the order of 3 to 5 times as large as the number of faults required in our simulations.

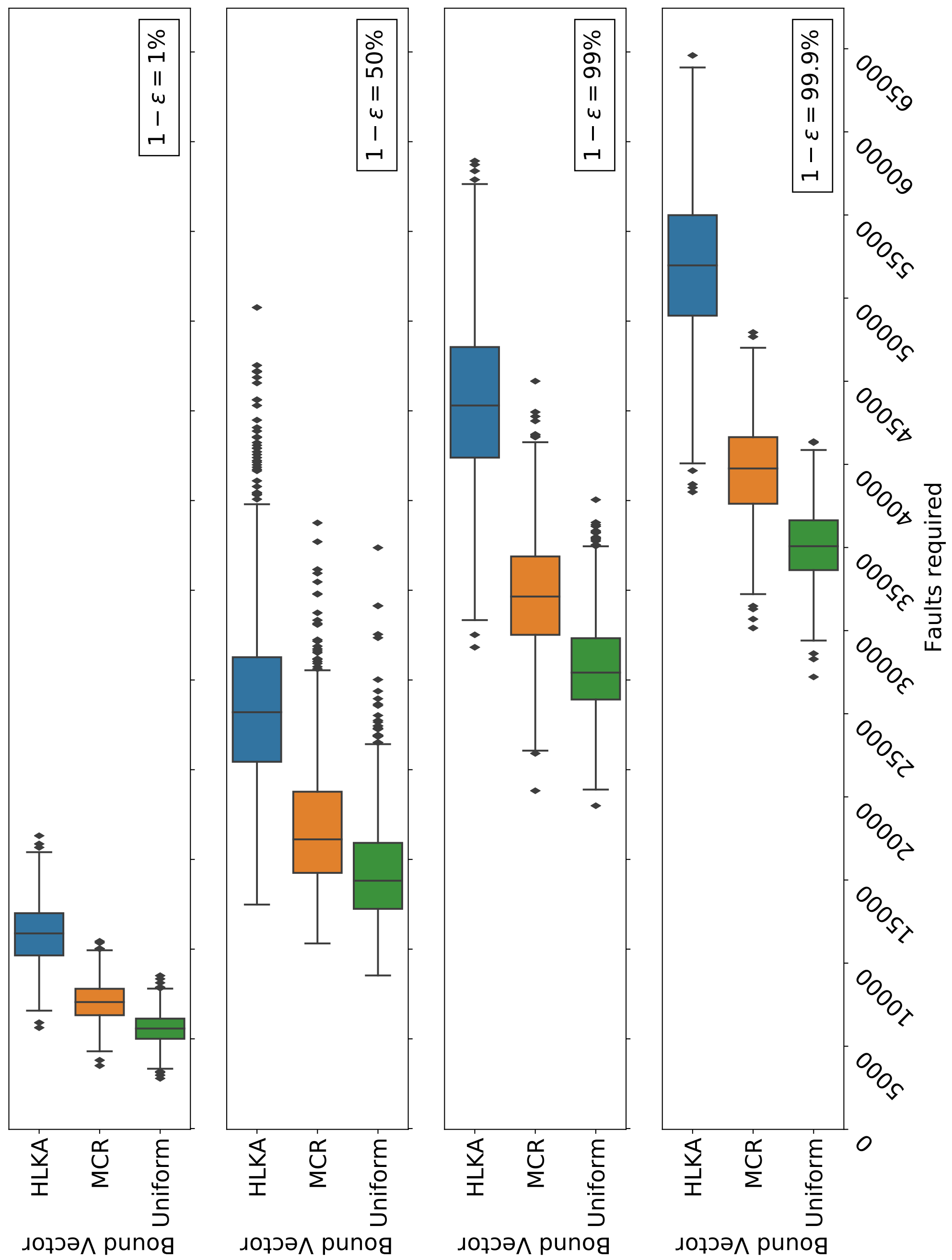


Figure 6.1: Box plots depicting the distribution of the number of faults required to achieve four certainty levels for three bound vectors in the unsigned setting, over 1000 trials.

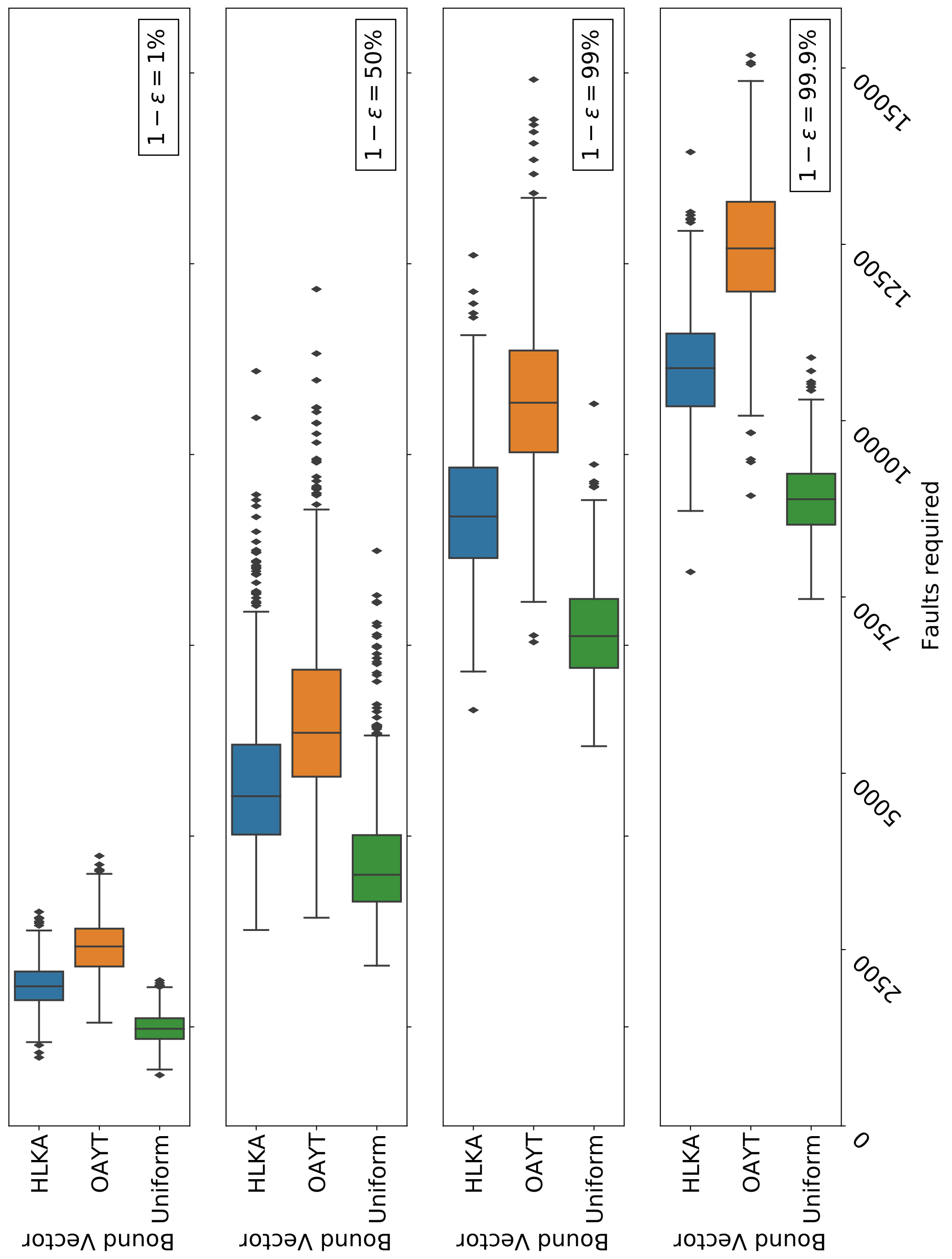


Figure 6.2: Box plots depicting the distribution of the number of faults required to achieve four certainty levels for three bound vectors in the signed setting, over 1000 trials.

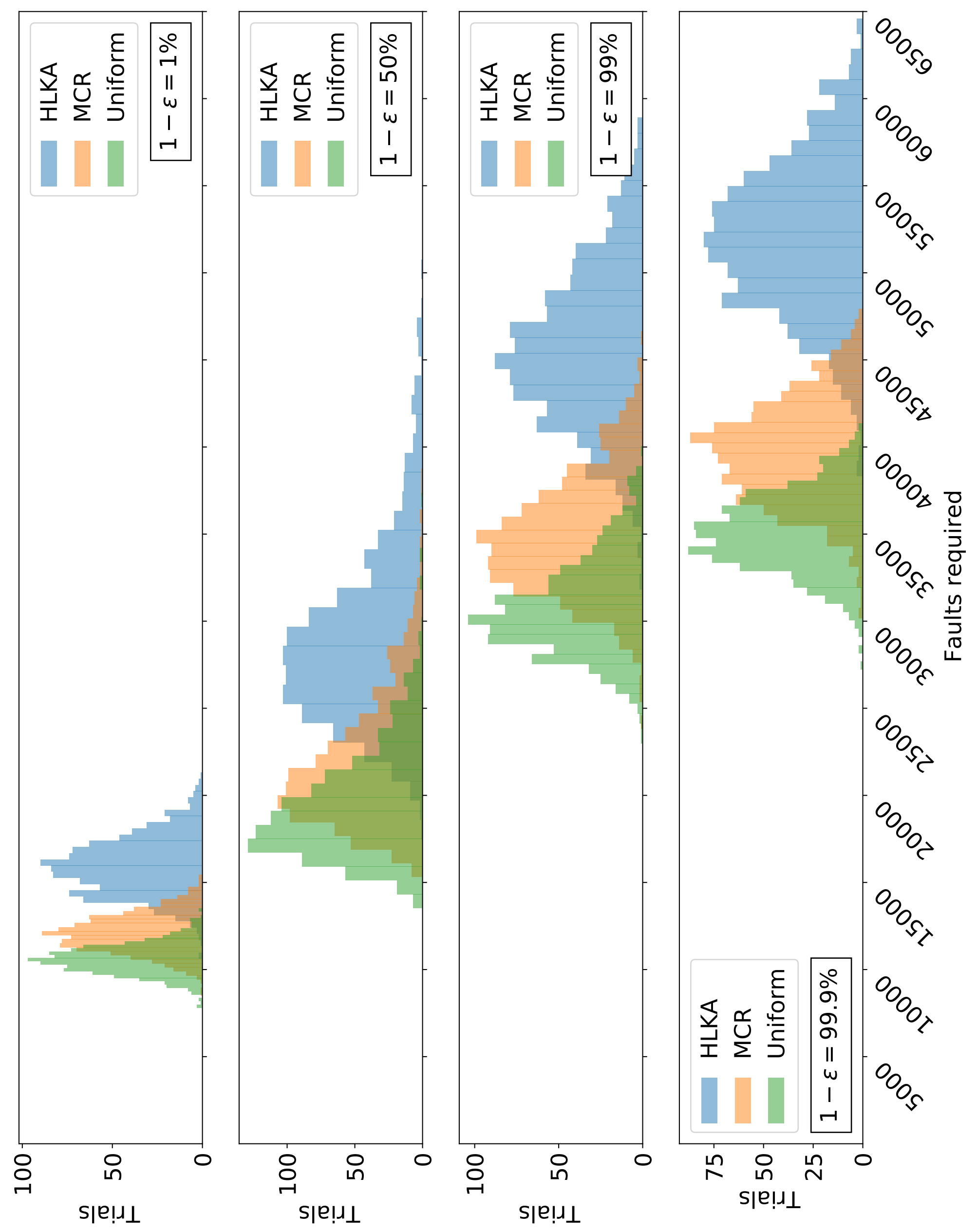


Figure 6.3: Histograms depicting the distribution of the number of faults required to achieve four certainty levels for three bound vectors in the unsigned setting, over 1000 trials.

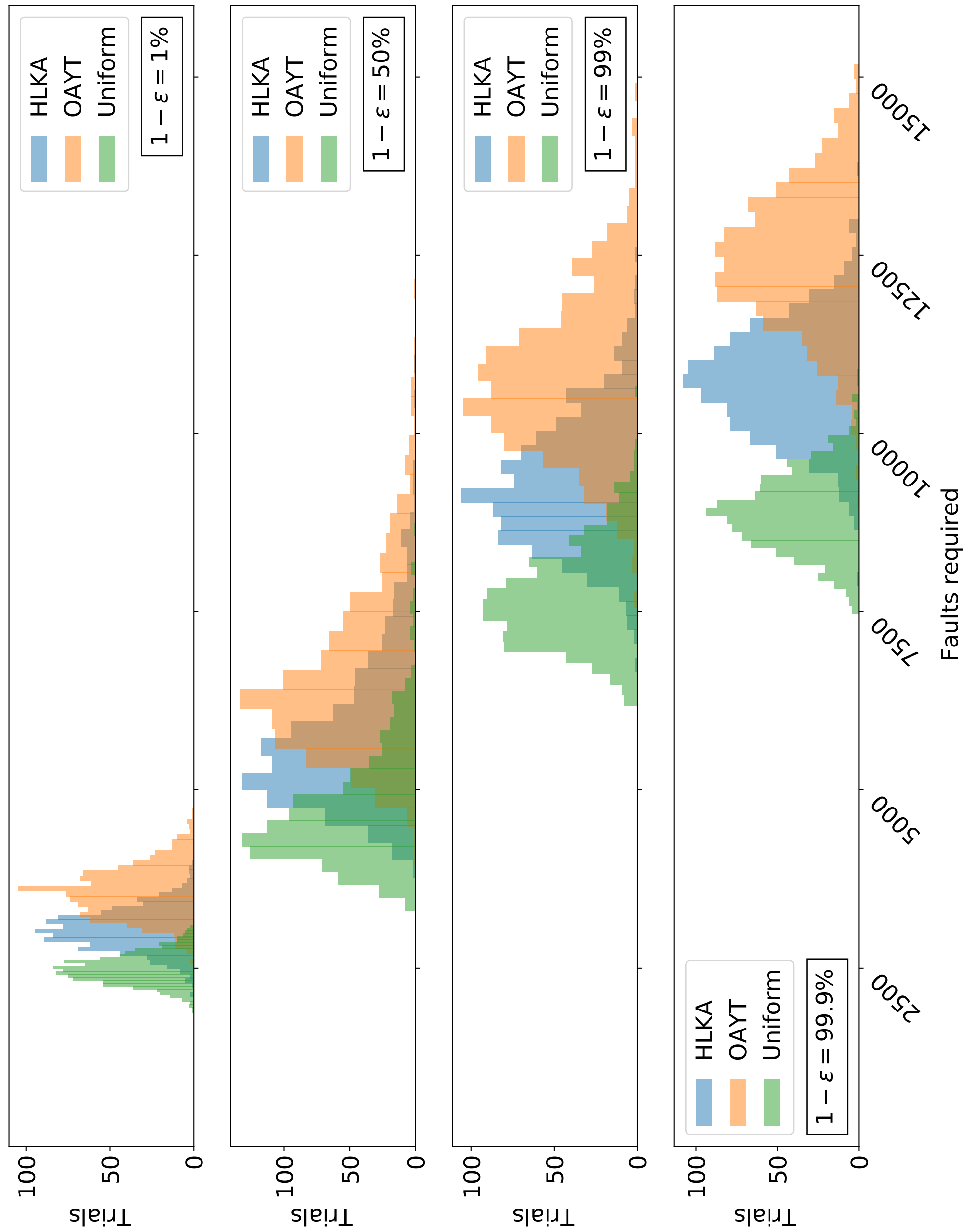


Figure 6.4: Histograms depicting the distribution of the number of faults required to achieve four certainty levels for three bound vectors in the signed setting, over 1000 trials.

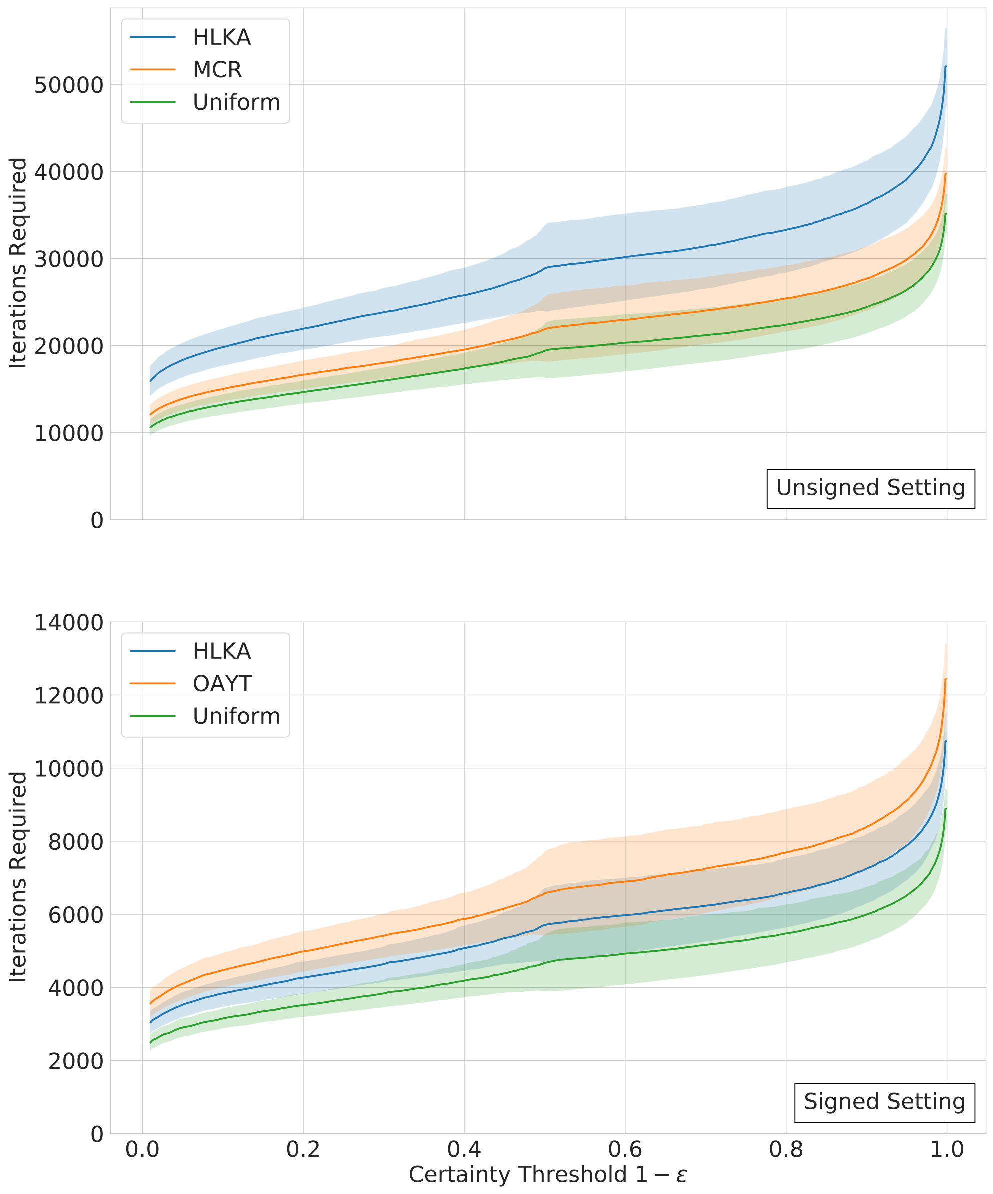


Figure 6.5: Mean number of attacks required for each certainty threshold value for three different bound vectors in both the unsigned (top) and signed (bottom) settings over 1000 trials. The shaded regions indicate standard deviation for each vector.

	Certainty:	1%	50%	99%	99.9%	$\sum\lceil\log(b_j)+1\rceil$	$\sum b_j$
Unsigned Setting	HLKA	15921	28865	45561	52062	356	815
	MCR	12067	21872	34855	39738	342	763
	Uniform	10584	19387	30760	35129	370	740
Signed Setting	HLKA	3039	5708	9272	10734	263	388
	OAYT	3552	6574	10741	12447	266	404
	Uniform	2484	4667	7686	8890	296	370

Table 6.1: Mean number of attacks used to reach specified certainty thresholds for various bound vectors over 1000 randomly generated private keys. For each bound vector $\mathbf{b} = (b_1, \ldots, b_n)$ the sums $\sum_{j=1}^{n}\lceil\log(b_j)+1\rceil$ (a sufficient number of attacks to learn the key with 100% certainty in the real-then-dummy setting) and $\sum_{j=1}^{n} b_j$ (a sufficient number of attacks to learn the key with 100% certainty when the decision vector $\mathbf{x}$ is fixed) are also reported.

	Certainty:	1%	50%	99%	99.9%
Unsigned Setting	HLKA	76058	116125	200531	250326
	MCR	58808	90067	158754	197264
	Uniform	52022	79698	142154	176194
Signed Setting	HLKA	15679	23981	42043	52270
	OAYT	18177	27812	48157	60024
	Uniform	13024	19980	35594	44104

Table 6.2: Upper bounds on the number of required faults to achieve certainty 1%, 50%, 99%, and 99.9% for six bound vectors, obtained using Theorem 6.13.

6.9.1 Performance of Gray Code Method

Here we compare the standard meet-in-the-middle approach to determining the signs of the private key to the Gray code approach of Section 6.8.4. It is difficult to analytically estimate the cost of the naïve method for iterating through the sets T_L and T_R of Section 6.8.4 since the adversary is not obligated to use constant-time algorithms to construct the curves, and thus can use a permutation and strategy which is optimized for the computed key magnitudes (see [39] for a discussion of permutations and strategies). Hence, we were not able to analytically compare the cost of the Gray code method to the naïve method.

Instead, we estimated the cost of the naïve method as follows. Since for a fixed key $\mathbf{e}$ the curves in T_L each require roughly the same amount of work to compute under the naïve method, the cost for computing all curves in T_L is approximately 2^k times the cost of computing a single curve, and the latter cost is that of performing the action $(\mathbf{e}, E) \mapsto \left[\prod_{i=1}^k \mathfrak{l}_i^{e_i}\right] * E$ in non-constant time. A similar statement holds for T_R. To approximate this, we sampled 1000 random keys from the keyspace defined by the HLKA signed setting bound vector from [39]. For each key, we found an optimal non-constant time strategy (*i.e.*, a strategy not employing dummy isogenies) and permutation using the code publicly provided in [39], and estimated the cost of executing the optimal strategy under the optimal permutation using the cost model of [39, Table 1]. Over the 1000 keys, the average cost of computing $(\mathbf{e}, E) \mapsto \left[\prod_{i=1}^k \mathfrak{l}_i^{e_i}\right] * E$ was about 150 000 many $GF(p)$ multiplications, where we have assumed $1\mathbf{M} = 0.8\mathbf{S}$ and ignored additions. The total cost of computing both T_L and T_R for HLKA is then $(2^{37} + 2^{36}) \cdot 1.5 \times 10^5 \approx 2^{54.87}$, assuming on average only half of T_R needs to be computed before a collision is found.

For particular values of $\mathbf{b}$ and $\boldsymbol{\kappa}$, we can estimate the cost of method based on Gray codes as follows. As before, let $k = \lceil \frac{n}{2} \rceil$. First note that since the ordering based on Gray codes is optimal, we can upper bound the expected cost by instead computing the expected cost when the ordering is such that $B_L = \{\ell_1, \dots, \ell_k\}$ and $B_R = \{\ell_{k+1}, \dots, \ell_n\}$ and

$$\frac{b_1(b_1+1)}{2b_1+1}\kappa_1 \leq \frac{b_2(b_2+1)}{2b_2+1}\kappa_2 \leq \cdots \leq \frac{b_k(b_k+1)}{2b_k+1}\kappa_k, \text{ and}$$
$$\frac{b_{k+1}(b_{k+1}+1)}{2b_{k+1}+1}\kappa_{k+1} \leq \frac{b_{k+2}(b_{k+2}+1)}{2b_{k+2}+1}\kappa_{k+2} \leq \cdots \leq \frac{b_n(b_n+1)}{2b_n+1}\kappa_n.$$

Then, noting that in expectation only half of the right table will need to be computed, we have

$$\begin{aligned}
\mathbb{E}[\text{Gray code cost}] &\leq \mathbb{E}_{\mathbf{e}}\left[\sum_{j=1}^{k} 2 \cdot 2^{k-j} \cdot |e_j|\kappa_j + \frac{1}{2}\sum_{j=1}^{n-k} 2 \cdot 2^{(n-k)-j} \cdot |e_{j+k}|\kappa_{j+k}\right] \\
&= \sum_{j=1}^{k} 2 \cdot 2^{k-j} \cdot \mathbb{E}_{\mathbf{e}}[|e_j|]\kappa_j + \frac{1}{2}\sum_{j=1}^{n-k} 2 \cdot 2^{(n-k)-j} \cdot \mathbb{E}_{\mathbf{e}}[|e_{j+k}|]\kappa_{j+k} \\
&= \sum_{j=1}^{k} 2 \cdot 2^{k-j} \cdot \frac{b_j(b_j+1)}{2b_j+1}\kappa_j + \frac{1}{2}\sum_{j=1}^{n-k} 2 \cdot 2^{(n-k)-j} \cdot \frac{b_{j+k}(b_{j+k}+1)}{2b_{j+k}+1}\kappa_{j+k}.
\end{aligned}$$

This cost only accounts for the “transition costs;” that is, the total costs of moving from each element to the next within each table. To account for finding the first curve in the table (which is done by using the naïve technique starting from the base curve E), we would add the expected cost of evaluating the action of $[\prod_{i=1}^{k} \mathfrak{l}_i^{|e_i^*|}]$ and the expected cost of evaluating the action of $[\prod_{i=k+1}^{n} \mathfrak{l}_i^{|e_i^*|}]$. Again, because we cannot analytically determine these costs, we instead use the experimentally-determined mean cost. Substituting the values of the κ_i and b_i using the cost model and bound vector of [39] (noting that these κ_i are *not* the same as those from [39]) we arrive at an upper bound of 1.725×10^4 field-multiplication-equivalent operations per table entry, with an average-case cost of approximately $2^{51.66}$ $GF(p)$ multiplication-equivalent operations for computing T_L and T_R. The bound on the expected cost of the Gray code method is approximately 88% less than the estimated expected cost of the naïve technique using per-key optimized permutations and strategies.

6.10 Conclusions and Future Work

Based on our analysis, randomizing the order of real and dummy isogeny computations in constant-time CSIDH implementations dramatically increases the number of faults to required to achieve a reasonable success probability in a fault attack of the types we have described. We do not expect that reordering isogeny computations will have an appreciable impact on the running time of CSIDH, and thus recommend that future implementations implement this feature for added fault attack resilience.

Chapter 7

Conclusions and Future Work

We have seen a number of analyses of the post-quantum key establishment protocol CSIDH—including quantum cryptanalysis, optimizations, and fault attack analysis—which have yielded a better understanding of the security of CSIDH, and faster, more secure implementations. Moreover, in the course of these analyses we have introduced formalisms that faciliate further development of isogeny-based cryptography.

There are a number of possible directions for future work; we give a few such directions here.

Implementation of New CSIDH Parameter Sets. In light of recent research due to Peikert [62], CSIDH-512 may not provide security at the level of NIST Category 1. Recent analyses [11] suggest that a base field $GF(p)$ with $\log_2 p \approx 2260$ (in an optimistic estimate) or $\log_2 p \approx 5280$ (in a conservative estimate) is required for that security level. Once appropriate primes are found, the techniques of Chapter 5 can be used to find optimized algorithms for implementing the class group action, and optimized bound vectors. One should also consider whether non-multiplication-based strategies are useful for the two-point technique in the setting of larger primes.

Improved Techniques for Optimization of Complex Multiplication Algorithms. The techniques of Chapter 5 do not generally yield globally-optimal (permutation, strategy) pairs to use to evaluate the action of $\mathrm{cl}(\mathcal{O})$. A method to find optimal (or simply better) solutions to this optimization problem would likely yield faster implementations of CSIDH.

Lower Bounds for Estimated Costs of Complex Multiplication. If it is not possible to find globally-optimal (permutation, strategy) pairs, it would be useful to understand how suboptimal our solutions are. In Appendix B we discuss an approach to this problem based on duality, but it has unfortunately been fruitless so far. More careful analysis may lead to duality lower bounds; moreover, it may be possible to derive bounds on the extent to which our alternating algorithm yields suboptimal solutions using the theory of approximation algorithms.

Fault Attack Analysis in More General Fault Models. The fault attacks we consider in Chapter 6 use bit-level faults (and, implicitly, faults that always succeed). It is straightforward to show that our analyses also apply almost unchanged to the weaker, byte-level faults considered in [15], and also that they can be extended (with some modifications) and fault models where faults do no always succeed are likely sufficient to launch a fault attack on CSIDH. This would yield a better understanding of the fault attack resilience of CSIDH under more realistic assumptions.

New Fault Attack Countermeasures. The fault attack countermeasure of [15] perfectly prevents a certain variety of fault attack, at the cost of approximately doubling the running time of CSIDH. Some preliminary results suggest that there exists a countermeasure which prevents the same variety of attack, and which runs in significantly less than twice the running time of CSIDH, at the cost of doubling its communication requirement. At present CSIDH is quite slow, but has extraordinarily small communication cost, and so this tradeoff is likely to be favourable. We would like to develop and implement this countermeasure.

References

[1] Sedat Akleylek, Nina Bindel, Johannes Buchmann, Juliane Krämer, and Giorgia Azzurra Marson. An efficient lattice-based signature scheme with provably secure instantiation. In David Pointcheval, Abderrahmane Nitaj, and Tajjeeddine Rachidi, editors, *Progress in Cryptology – AFRICACRYPT 2016*, pages 44–60, Cham, 2016. Springer International Publishing.

[2] Reza Azarderakhsh, Matthew Campagna, Craig Costello, Luca De Feo, Basil Hess, Amir Jalali, Brian Koziel, Brian LaMacchia, Patrick Longa, Michael Naehrig, Geovandro Pereira, Joost Renes, Vladimir Soukharev, and David Urbanik. Supersingular isogeny key encapsulation. Technical report, 2020. `https://sike.org/files/SIDH-spec.pdf`.

[3] Reza Azarderakhsh, David Jao, and Christopher Leonardi. Post-quantum static-static key agreement using multiple protocol instances. In Carlisle Adams and Jan Camenisch, editors, *Selected Areas in Cryptography – SAC 2017*, pages 45–63, Cham, 2018. Springer International Publishing.

[4] Daniel Bernstein, Luca De Feo, Antonin Leroux, and Benjamin Smith. Faster computation of isogenies of large prime degree. *IACR Cryptol. ePrint Arch.*, 2020:341, 2020.

[5] Daniel J. Bernstein, Tanja Lange, Chloe Martindale, and Lorenz Panny. Quantum circuits for the csidh: Optimizing quantum evaluation of isogenies. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019*, pages 409–441, Cham, 2019. Springer International Publishing.

[6] Ward Beullens, Thorsten Kleinjung, and Frederik Vercauteren. CSI-FiSh: Efficient isogeny based signatures through class group computations. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology – ASIACRYPT 2019*, pages 227–247, Cham, 2019. Springer International Publishing.

[7] Jean-François Biasse and Fang Song. Efficient quantum algorithms for computing class groups and solving the principal ideal problem in arbitrary degree number fields. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '16, page 893–902, USA, 2016. Society for Industrial and Applied Mathematics.

[8] Jean-François Biasse, Annamaria Iezzi, and Michael J. Jacobson. A note on the security of CSIDH. In Debrup Chakraborty and Tetsu Iwata, editors, *Progress in Cryptology – INDOCRYPT 2018*, pages 153–168, Cham, 2018. Springer International Publishing.

[9] Jean-François Biasse, David Jao, and Anirudh Sankar. A quantum algorithm for computing isogenies between supersingular elliptic curves. In Willi Meier and Debdeep Mukhopadhyay, editors, *Progress in Cryptology – INDOCRYPT 2014*, pages 428–442, Cham, 2014. Springer International Publishing.

[10] Avrim Blum, Adam Kalai, and Hal Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. *J. ACM*, 50(4):506–519, July 2003.

[11] Xavier Bonnetain and André Schrottenloher. Quantum security analysis of CSIDH. Cryptology ePrint Archive, Report 2018/537, 2018. `https://eprint.iacr.org/2018/537`.

[12] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[13] Ran Canetti and Hugo Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In Birgit Pfitzmann, editor, *Advances in Cryptology — EUROCRYPT 2001*, pages 453–474, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[14] Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. CSIDH: An efficient post-quantum commutative group action. Cryptology ePrint Archive, Report 2018/383, 2018. `https://eprint.iacr.org/2018/383`.

[15] 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.

[16] Andrew Childs, David Jao, and Vladimir Soukharev. Constructing elliptic curve isogenies in quantum subexponential time. *Journal of Mathematical Cryptology*, 8:1–29, 2014.

[17] Andrew Childs, David Jao, and Vladimir Soukharev. Constructing elliptic curve isogenies in quantum subexponential time. *Journal of Mathematical Cryptology*, 8:1–29, 2014.

[18] Yuan Chow and Henry Teicher. *Probability theory: independence, interchangeability, martingales*. Springer, New York, 1997.

[19] Henri Cohen and Hendrik 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.

[20] Henri Cohen and Hendrik W. Lenstra Jr. Heuristics on class groups of number fields. *Number theory, Noordwijkerhout 1983 (Noordwijkerhout, 1983), Lecture Notes in Math.*, 1068:33–62, 1984.

[21] Jean-Marc Couveignes. Hard homogeneous spaces. Cryptology ePrint Archive, Report 2006/291, 2006. `https://eprint.iacr.org/2006/291`.

[22] David Cox. *Primes of the form $x^2 + ny^2$: Fermat, class field theory, and complex multiplication*. Wiley, New York, 1989.

[23] Luca De Feo, David Jao, and Jérôme Plût. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. *Journal of Mathematical Cryptology*, 8:209–247, 2014.

[24] Christina Delfs and Steven D. Galbraith. Computing isogenies between supersingular elliptic curves over $\mathbb{F}_p$. *Des. Codes Cryptography*, 78(2):425–440, February 2016.

[25] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, November 1976.

[26] Samuel Dobson, Steven D. Galbraith, Jason LeGrow, Yan Bo Ti, and Lukas Zobernig. An adaptive attack on 2-SIDH. Cryptology ePrint Archive, Report 2019/890, 2019. `https://eprint.iacr.org/2019/890`.

[27] Léo Ducas, Alain Durmus, Tancrède Lepoint, and Vadim Lyubashevsky. Lattice signatures and bimodal gaussians. In Ran Canetti and Juan A. Garay, editors, *Advances*

in Cryptology – CRYPTO 2013, pages 40–56, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[28] Léo Ducas, Vadim Lyubashevsky, and Thomas Prest. Efficient identity-based encryption over NTRU lattices. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology – ASIACRYPT 2014*, volume 8874 of *Lecture Notes in Computer Science*, pages 22–41. Springer, 2014.

[29] Taher Elgamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[30] J. Mark Ettinger, Peter Høyer, and Emanuel Knill. The quantum query complexity of the hidden subgroup problem is polynomial. *Information Processing Letters*, 91:43–48, July 2004.

[31] Steven D. Galbraith, Christophe Petit, Barak Shani, and Yan Bo Ti. On the security of supersingular isogeny cryptosystems. In *Advances in Cryptology – ASIACRYPT 2016*, pages 63–91. Springer, 2016.

[32] Alexandre Gélin and Benjamin Wesolowski. Loop-abort faults on supersingular isogeny cryptosystems. In Tanja Lange and Tsuyoshi Takagi, editors, *8th International Conference on Post-Quantum Cryptography*, pages 93–106, Cham, 2017. Springer International Publishing.

[33] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, STOC '08, page 197–206, New York, NY, USA, 2008. Association for Computing Machinery.

[34] Bertrand Guenin, Jochen Könemann, and Levent Tunçel. *A gentle introduction to optimization*. Cambridge University Press, Cambridge, 2014.

[35] Uffe Haagerup. The best constants in the khintchine inequality. *Studia Mathematica*, 70(3):231–283, 1981.

[36] Robin Hartshorne. *Algebraic geometry*. Springer Science+Business Media, Inc, New York, 2010.

[37] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

[38] Aaron Hutchinson and Koray Karabina. Constructing canonical strategies for parallel implementation of isogeny based cryptography. In *Progress in Cryptology – INDOCRYPT 2018*, pages 169–189. Springer, 12 2018.

[39] Aaron Hutchinson, Jason LeGrow, Brian Koziel, and Reza Azarderakhsh. Further optimizations of CSIDH: A systematic approach to efficient strategies, permutations, and bound vectors. *Proceedings of Applied Cryptograhy and Network Security 2020 (ACNS 2020)*, 2020. (Forthcoming).

[40] David Jao, Jason T. LeGrow, Christopher Leonardi, and Luis Ruiz-Lopez. A subexponential-time, polynomial quantum space algorithm for inverting the CM group action. *Journal of Mathematical Cryptology*, 14, 2020.

[41] David Jao, Stephen D. Miller, and Ramarathnam Venkatesan. Expander graphs based on GRH with an application to elliptic curve cryptography. *J. Number Theory*, 129(6):1491–1504, 2009.

[42] Samuel Jaques and John Schanck. Quantum cryptanalysis in the RAM model: Claw-finding attacks on SIKE. In *Advances in Cryptology – CRYPTO 2019*, pages 32–61, 08 2019.

[43] Selçuk Kayacan. A note on the static-static key agreement protocol from supersingular isogenies. Cryptology ePrint Archive, Report 2019/815, 2019. Withdrawn.

[44] Aleksandr Khintchine. Über dyadische brüche. *Mathematische Zeitschrift*, 18(1):109–116, 1923.

[45] Daniel Kirkwood, Bradley C. Lackey, John McVey, Mark Motley, Jerome A. Solinas, and David Tuller. Failure is not an option: Standardization issues for post-quantum key agreement on the security of supersingular isogeny cryptosystems. Workshop on Cybersecurity in a Post-Quantum World (2015).

[46] Harold W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955.

[47] Greg Kuperberg. A subexponential-time quantum algorithm for the dihedral hidden subgroup problem. *SIAM Journal on Computing*, 35(1):170–188, 2005.

[48] Greg Kuperberg. Another subexponential-time quantum algorithm for the dihedral hidden subgroup problem. In *8th Conference on the Theory of Quantum Computation, Communication and Cryptography*, pages 20–34, 2013.

[49] Péter Kutas, Chloe Martindale, Lorenz Panny, Christophe Petit, and Katherine E. Stange. Weak instances of sidh variants under improved torsion-point attacks. Cryptology ePrint Archive, Report 2020/633, 2020. `https://eprint.iacr.org/2020/633`.

[50] Jason LeGrow. Post-quantum security of authenticated key establishment protocols. Master's thesis, University of Waterloo, 2016. `http://hdl.handle.net/10012/10386`.

[51] Leibo Liu, Bo Wang, and Shaojun Wei. *Reconfigurable Cryptographic Processor*. Springer Singapore, 2018.

[52] Chris Lomont. The hidden subgroup problem - review and open problems. ePrint arXiv:quant-ph/0411037, 2004.

[53] Józef Marcinkiewicz and Antoni Zygmund. Sur les fonctions indépendantes. *Fundamenta Mathematicae*, 29(1):60–90, 1937.

[54] Garth P. Mccormick. Computability of global solutions to factorable nonconvex programs: Part I – convex underestimating problems. *Mathematical Programming*, 10(1):147–175, December 1976.

[55] Alfred J. Menezes, Tatsuaki Okamoto, and Scott A. Vanstone. Reducing elliptic curve logarithms in a finite field. *Information Theory, IEEE Transactions on*, 39:1639 – 1646, 10 1993.

[56] Michael Meyer, Fabio Campos, and Steffen Reith. On lions and elligators: An efficient constant-time implementation of CSIDH. Cryptology ePrint Archive, Report 2018/1198, 2018. `https://eprint.iacr.org/2018/1198`.

[57] Rick Miranda. *Algebraic curves and Riemann surfaces*. American Mathematical Society, Providence, R.I, 1995.

[58] Michael Nielsen and Isaac Chuang. *Quantum computation and quantum information*. Cambridge University Press, Cambridge New York, 2010.

[59] 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.

[60] Dan Page and Frederik Vercauteren. A fault attack on pairing-based cryptography. *IEEE Transactions on Computers*, 55(9):1075–1080, 2006.

[61] Raymond E. A. C. Paley and Antoni Zygmund. On some series of functions, (3). *Mathematical Proceedings of the Cambridge Philosophical Society*, 28(2):190–205, 1932.

[62] Chris Peikert. He gives C-sieves on the CSIDH. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020*, pages 463–492, Cham, 2020. Springer International Publishing.

[63] Christophe Petit. Faster algorithms for isogeny problems using torsion point images. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017*, pages 330–353, Cham, 2017. Springer International Publishing.

[64] John E. Prussing. The principal minor test for semidefinite matrices. *Journal of Guidance, Control, and Dynamics*, 9(1):121–122, 1986.

[65] Oded Regev. A subexponential time algorithm for the dihedral hidden subgroup problem with polynomial space. *eprint arXiv:quant-ph/0406151*, June 2004.

[66] Alexander Rostovtsev and Anton Stolbunov. Public-key cryptosystem based on isogenies, April 2006.

[67] Arthur Schmidt. Quantum algorithm for solving the discrete logarithm problem in the class group of an imaginary quadratic field and security comparison of current cryptosystems at the beginning of quantum computer age. In Günter Müller, editor, *Emerging Trends in Information and Communication Security*, pages 481–493, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[68] Peter W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, 1994.

[69] Joseph Silverman. *The Arithmetic of Elliptic Curves.* Springer-Verlag, New York, 2009.

[70] Andrew Sutherland. Structure computation and discrete logarithms in finite abelian p-groups. *Mathematics of Computation*, 80, 09 2008.

[71] Seiichiro Tani. An improved claw finding algorithm using quantum walk. In Luděk Kučera and Antonín Kučera, editors, *Mathematical Foundations of Computer Science 2007*, pages 536–547, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[72] John Tate. Endomorphisms of abelian varieties over finite fields. *Inventiones mathematicae*, 2(2):134–144, Apr 1966.

[73] Edlyn Teske. The Pohlig–Hellman method generalized for group structure computation. *Journal of Symbolic Computation*, 27(6):521 – 534, 1999.

[74] Yan Bo Ti. Fault attack on supersingular isogeny cryptosystems. In Tanja Lange and Tsuyoshi Takagi, editors, *8th International Conference on Post-Quantum Cryptography*, pages 107–122, Cham, 2017. Springer International Publishing.

[75] Jacques Vélu. Isogénies entre courbes elliptiques. *C. R. Acad. Sci. Paris Sér. A-B*, 273:A238–A241, 1971.

[76] William C. Waterhouse. Abelian varieties over finite fields. *Annales scientifiques de l'École Normale Supérieure*, Ser. 4, 2(4):521–560, 1969.

APPENDICES

Appendix A

Optimized CSIDH Parameters

In this appendix we give the optimized CSIDH parameter sets we found using the techniques of Chapter 5. We give the parameter sets used in MCRim, and in CCCDRS-3. Since the CCCDRS-3 SIMBA strategies are all multiplication-based, we give only the SIMBA substrategy sizes, rather than depicting the SIMBA strategies themselves.

A.1 MCRim

A.1.1 Bound Vector

$$\begin{aligned}\mathbf{b}_{\text{MCRim}} = [&8, 17, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18,\\ &18, 18, 17, 16, 15, 13, 13, 13, 13, 13, 12, 12, 11, 11, 11, 10, 11, 10,\\ &10, 10, 9, 9, 8, 8, 8, 8, 7, 7, 7, 7, 7, 6, 7, 7, 7, 7, 6, 7, 7, 6, 6, 6, 6, 6, 6,\\ &6, 5, 5, 5, 6, 5, 5, 5, 6, 4]\end{aligned}$$

A.1.2 Permutations

These permutations should be intepreted in the following way:

$(\sigma_{i,j})_k = t \iff$ The kth prime in SIMBA substrategy j of the ith SIMBA strategy is ℓ_t.

Note that the primes are indexed from 0.

$$
\begin{aligned}
\sigma_{1,1} &= [72, 46, 45, 51, 42, 66, 62, 63, 65, 22, 67, 20, 28, 37, 64, 19, 23, 17, 38, 12, 6]\\
\sigma_{1,2} &= [71, 44, 52, 50, 43, 56, 60, 16, 61, 21, 57, 33, 58, 34, 35, 59, 15, 24, 9, 40, 41,\\
&\qquad 13, 30, 18]\\
\sigma_{1,3} &= [73, 47, 48, 49, 69, 68, 53, 25, 54, 26, 55, 29, 31, 14, 70, 27, 32, 36, 11, 39, 8,\\
&\qquad 7, 10, 4, 5, 3, 2, 0, 1]\\
\sigma_{2,1} &= [72, 46, 45, 51, 42, 66, 62, 63, 65, 22, 67, 20, 28, 37, 64, 19, 23, 17, 38, 12, 6]\\
\sigma_{2,2} &= [71, 44, 52, 50, 43, 56, 60, 16, 61, 21, 57, 33, 58, 34, 35, 59, 15, 24, 9, 40, 41,\\
&\qquad 13, 30, 18]\\
\sigma_{2,3} &= [73, 47, 48, 49, 69, 68, 53, 25, 54, 26, 55, 29, 31, 14, 70, 27, 32, 36, 11, 39, 8,\\
&\qquad 7, 10, 4, 5, 3, 2, 0, 1]\\
\sigma_{3,1} &= [72, 46, 45, 51, 42, 66, 62, 63, 65, 22, 67, 20, 28, 37, 64, 19, 23, 17, 38, 12, 6]\\
\sigma_{3,2} &= [71, 44, 52, 50, 43, 56, 60, 16, 61, 21, 57, 33, 58, 34, 35, 59, 15, 24, 9, 40, 41,\\
&\qquad 13, 30, 18]\\
\sigma_{3,3} &= [73, 47, 48, 49, 69, 68, 53, 25, 54, 26, 55, 29, 31, 14, 70, 27, 32, 36, 11, 39, 8,\\
&\qquad 7, 10, 4, 5, 3, 2, 0, 1]\\
\sigma_{4,1} &= [72, 46, 45, 51, 42, 66, 62, 63, 65, 22, 67, 20, 28, 37, 64, 19, 23, 17, 38, 12, 6]\\
\sigma_{4,2} &= [71, 44, 52, 50, 43, 56, 60, 16, 61, 21, 57, 33, 58, 34, 35, 59, 15, 24, 9, 40, 41,\\
&\qquad 13, 30, 18]\\
\sigma_{4,3} &= [73, 47, 48, 49, 69, 68, 53, 25, 54, 26, 55, 29, 31, 14, 70, 27, 32, 36, 11, 39, 8,\\
&\qquad 7, 10, 4, 5, 3, 2, 0, 1]\\
\sigma_{5,1} &= [72, 52, 51, 49, 46, 54, 68, 66, 65, 36, 58, 20, 21, 33, 67, 18, 37, 13, 41, 11]\\
\sigma_{5,2} &= [71, 50, 47, 44, 0, 61, 62, 63, 29, 64, 31, 60, 14, 17, 34, 35, 55, 38, 39, 40]\\
\sigma_{5,3} &= [70, 42, 43, 48, 45, 16, 57, 69, 19, 56, 28, 32, 53, 15, 24, 22, 23, 12, 59, 25, 26,\\
&\qquad 27, 10, 30, 9, 6, 8, 5, 7, 4, 3, 2, 1]\\
\sigma_{6,1} &= [72, 50, 43, 44, 53, 63, 49, 16, 20, 58, 25, 36, 39, 27, 29, 9, 35, 19, 10, 17]\\
\sigma_{6,2} &= [65, 51, 41, 52, 40, 57, 56, 42, 28, 30, 62, 23, 33, 34, 13, 59, 15, 26, 8, 37, 31,\\
&\qquad 14, 18]\\
\sigma_{6,3} &= [64, 45, 46, 47, 48, 60, 61, 54, 24, 21, 11, 55, 22, 32, 12, 38, 7, 6, 4, 5, 3,\\
&\qquad 2, 0, 1]\\
\sigma_{7,1} &= [56, 47, 48, 43, 36, 50, 45, 20, 35, 46, 18, 33, 31, 39, 29, 23, 30, 11, 32, 17]
\end{aligned}
$$

$$\sigma_{7,2} = [55, 40, 38, 41, 44, 26, 53, 28, 42, 15, 19, 27, 8, 34, 10, 7, 37, 16, 24, 21, 25, 9, 14, 6, 22, 13, 12, 5, 4, 3, 2, 0, 1]$$

$$\sigma_{8,1} = [44, 29, 40, 41, 42, 39, 27, 35, 23, 22, 37, 28, 32, 33, 14, 38, 16, 21, 19, 30, 20, 17, 18, 36, 24, 25, 26, 8, 9, 4, 31, 13, 15, 7, 34, 12, 6, 11, 10, 5, 3, 2, 0, 1]$$

$$\sigma_{9,1} = [39, 29, 28, 36, 38, 37, 14, 33, 15, 21, 34, 22, 19, 30, 32, 16, 26, 18, 13, 20, 9, 17, 35, 23, 24, 25, 8, 27, 12, 7, 6, 31, 11, 10, 4, 5, 3, 2, 0, 1]$$

$$\sigma_{10,1} = [35, 29, 28, 33, 34, 27, 16, 26, 32, 21, 19, 31, 18, 22, 24, 13, 14, 6, 30, 15, 8, 23, 9, 4, 25, 11, 12, 7, 20, 17, 10, 5, 3, 2, 1]$$

$$\sigma_{11,1} = [29, 28, 27, 26, 24, 9, 2, 25, 14, 15, 7, 31, 20, 22, 32, 16, 17, 8, 19, 13, 4, 3, 30, 23, 21, 12, 18, 11, 10, 6, 5, 1]$$

$$\sigma_{12,1} = [30, 27, 28, 23, 26, 16, 21, 24, 14, 19, 18, 13, 25, 15, 20, 17, 10, 9, 4, 5, 22, 8, 11, 12, 7, 6, 3, 2, 1]$$

$$\sigma_{13,1} = [28, 25, 21, 26, 15, 7, 23, 12, 13, 9, 2, 24, 14, 17, 22, 16, 18, 19, 8, 4, 5, 20, 10, 11, 6, 3, 1]$$

$$\sigma_{14,1} = [26, 24, 21, 25, 23, 12, 15, 4, 2, 22, 9, 13, 14, 8, 16, 7, 10, 5, 20, 19, 18, 17, 11, 6, 3, 1]$$

$$\sigma_{15,1} = [24, 22, 23, 20, 16, 21, 15, 18, 19, 13, 12, 11, 14, 7, 9, 5, 17, 10, 8, 6, 4, 3, 2, 1]$$

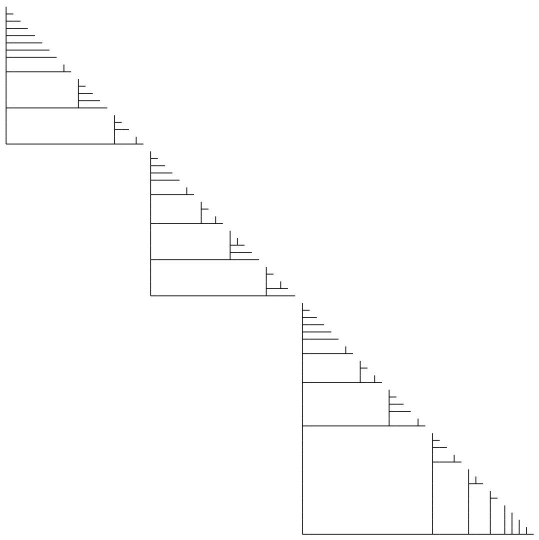
$$\sigma_{16,1} = [23, 16, 19, 22, 14, 20, 9, 15, 18, 11, 21, 12, 13, 7, 4, 2, 17, 10, 8, 6, 5, 3, 1]$$

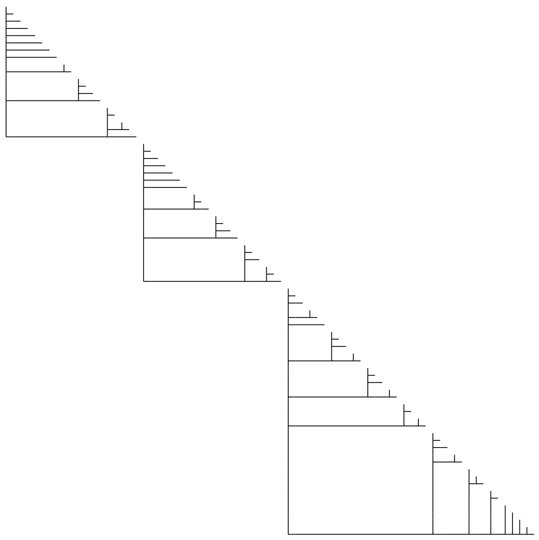
$$\sigma_{17,1} = [21, 20, 19, 15, 18, 14, 17, 16, 9, 11, 12, 8, 4, 5, 13, 10, 7, 6, 3, 2, 1]$$

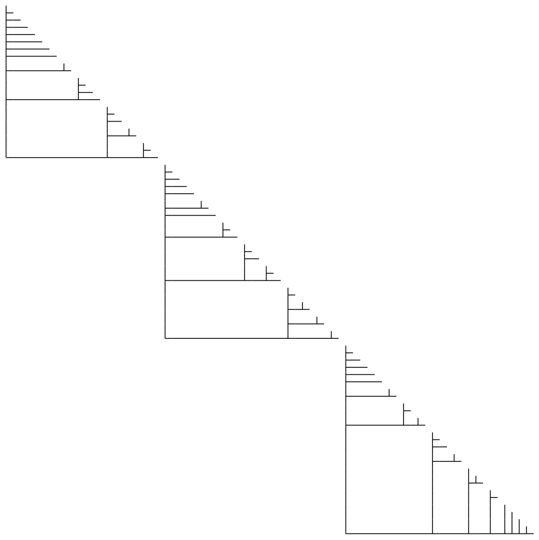
$$\sigma_{18,1} = [21, 18, 20, 16, 19, 13, 12, 8, 4, 17, 9, 7, 14, 11, 15, 10, 6, 5, 2, 3]$$

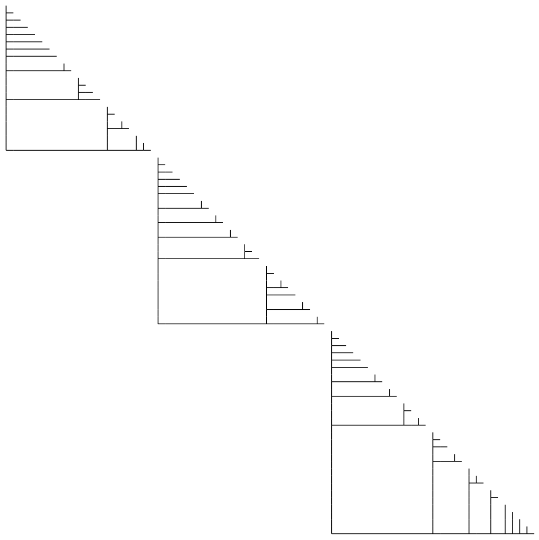
$$\sigma_{19,1} = [18, 13, 12, 15, 7, 4, 14, 9, 8, 2, 3, 17, 10, 11, 6, 5]$$

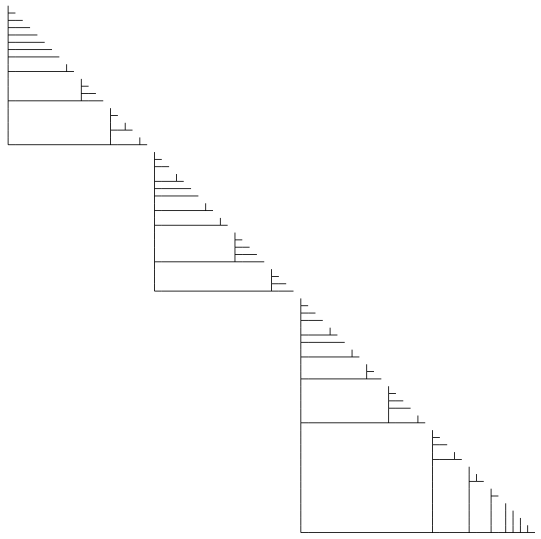
A.1.3 SIMBA Strategies

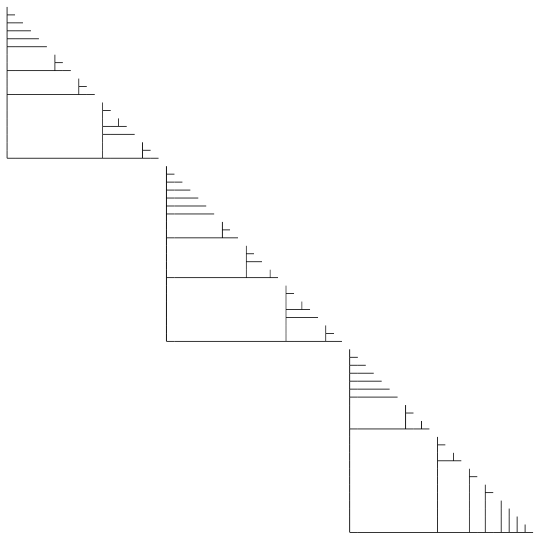

(a) The SIMBA strategy used in round 1.

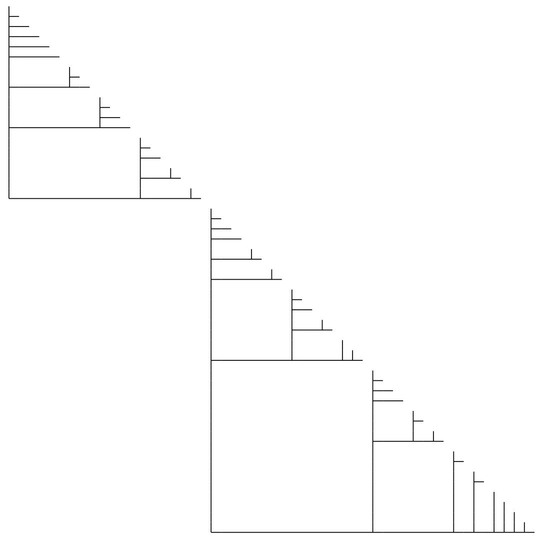

(b) The SIMBA strategy used in round 2.

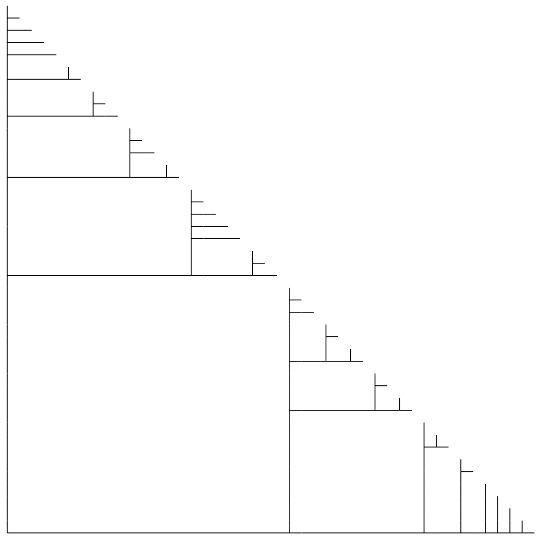

(c) The SIMBA strategy used in round 3.

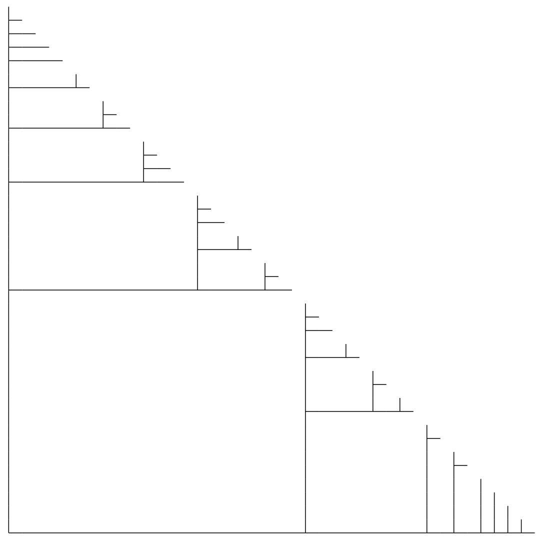

(d) The SIMBA strategy used in round 4.

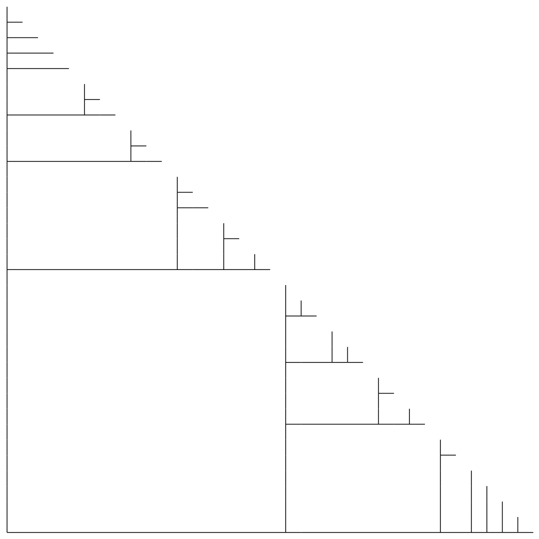

(e) The SIMBA strategy used in round 5.

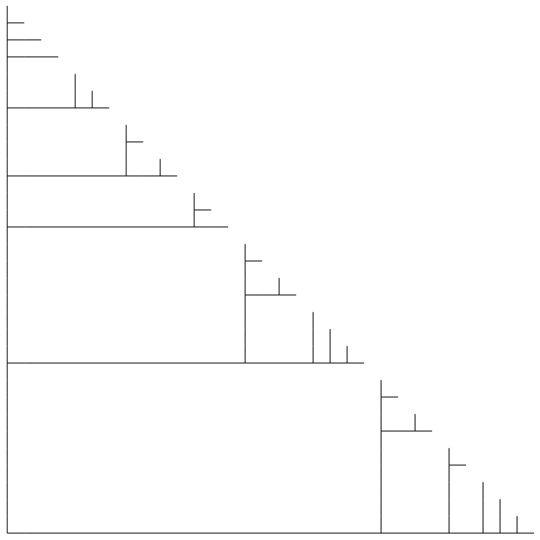

(f) The SIMBA strategy used in round 6.

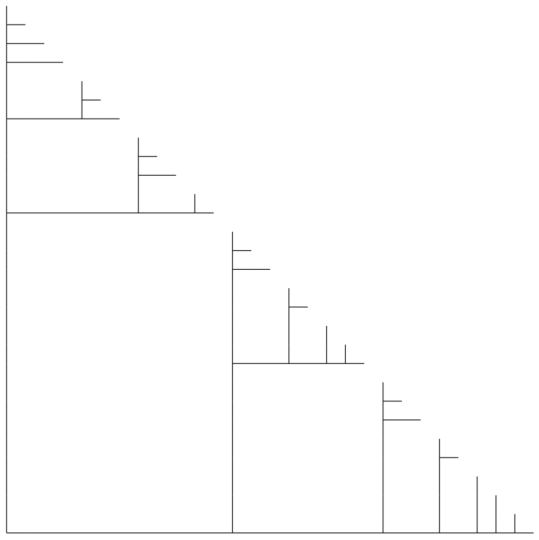

(g) The SIMBA strategy used in round 7.

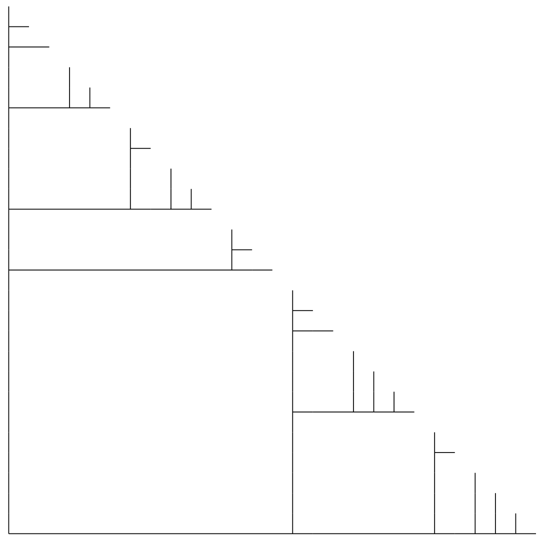

(h) The SIMBA strategy used in round 8.

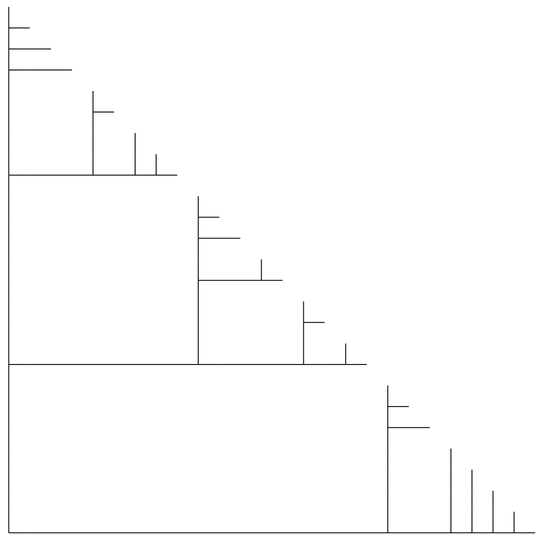

(i) The SIMBA strategy used in round 9.

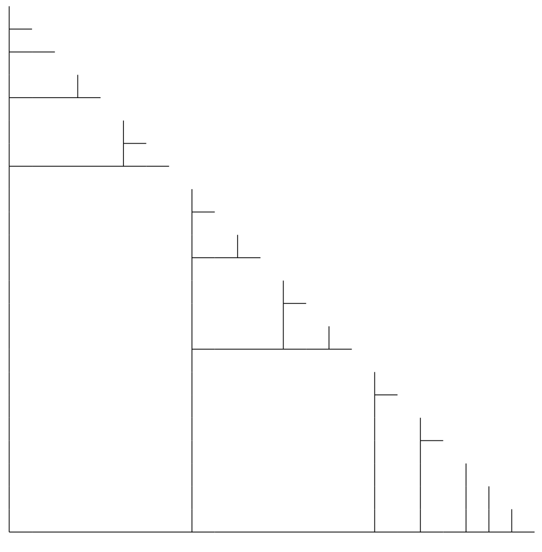

(j) The SIMBA strategy used in round 10.

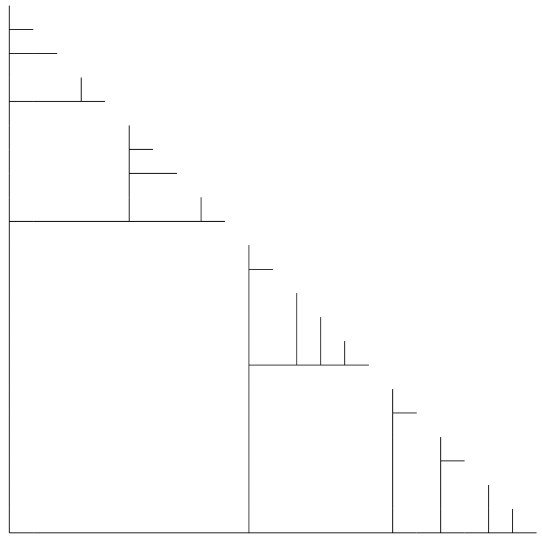

(k) The SIMBA strategy used in round 11.

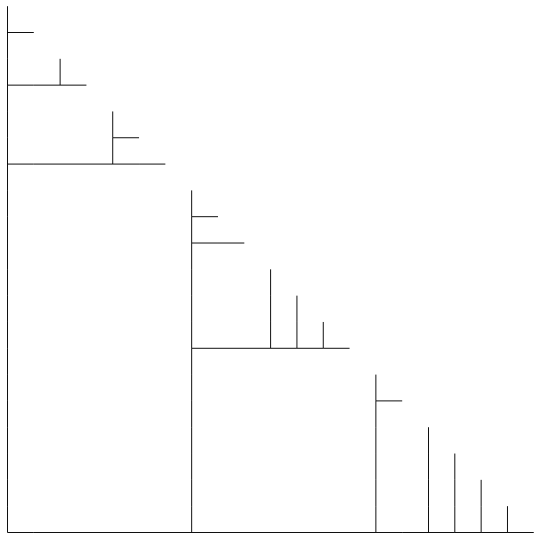

(l) The SIMBA strategy used in round 12.

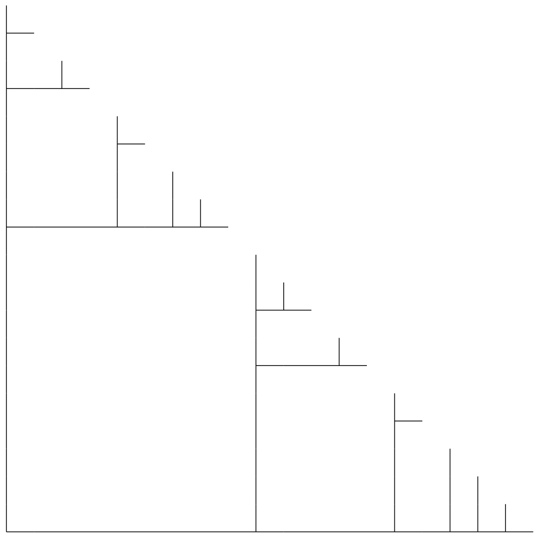

(m) The SIMBA strategy used in round 13.


(n) The SIMBA strategy used in round 14.


(o) The SIMBA strategy used in round 15.


(p) The SIMBA strategy used in round 16.


(q) The SIMBA strategy used in round 17.


(r) The SIMBA strategy used in round 18.

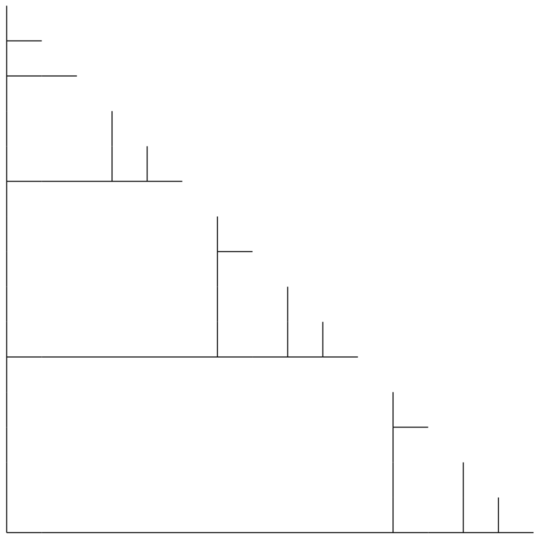

(s) The SIMBA strategy used in round 19.

Figure A.1: The SIMBA strategies used in MCRim.

A.2 CCCDRS-3

A.2.1 Bound Vector

$$\begin{aligned}\mathbf{b}_{\text{CCCDRS-3}} = [&3,6,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,6,\\ &6,7,6,6,6,6,6,5,6,5,6,5,5,5,4,4,4,4,4,4,4,4,4,4,4,4,4,4,\\ &4,4,4,4,4,3,3,3,3,3,3,3,3,3,3,3,2]\end{aligned}$$

A.2.2 Permutations

These permutations should be intepreted in the following way:

$(\sigma_{i,j})_k = t \iff$ The k^{th} prime in SIMBA substrategy j of the i^{th} SIMBA strategy is ℓ_t.

Note that the primes are indexed from 0.

$$\begin{aligned}
\sigma_{1,1} &= [70,2,5,16,10,19,17,18,44,43,42,41,40,39,59,56,57,58]\\
\sigma_{1,2} &= [73,3,7,14,11,24,25,26,45,34,35,30,31,37,64,63,62,61]\\
\sigma_{1,3} &= [72,1,4,6,13,12,22,29,23,46,47,48,49,50,51,67,68,53,54]\\
\sigma_{1,4} &= [71,0,9,8,15,20,21,28,27,33,36,32,38,52,69,60,55,65,66]\\
\sigma_{2,1} &= [70,2,5,16,10,19,17,18,44,43,42,41,40,39,59,56,57,58]\\
\sigma_{2,2} &= [73,3,7,14,11,24,25,26,45,34,35,30,31,37,64,63,62,61]\\
\sigma_{2,3} &= [72,1,4,6,13,12,22,29,23,46,47,48,49,50,51,67,68,53,54]\\
\sigma_{2,4} &= [71,0,9,8,15,20,21,28,27,33,36,32,38,52,69,60,55,65,66]\\
\sigma_{3,1} &= [71,1,6,14,15,22,19,27,36,43,38,41,40,39,62,61,60,59]\\
\sigma_{3,2} &= [72,3,8,11,16,23,24,25,26,44,32,30,31,50,58,65,64,63]\\
\sigma_{3,3} &= [69,4,5,7,12,18,20,29,37,45,46,47,48,49,68,67,66,53]\\
\sigma_{3,4} &= [70,0,2,9,10,13,17,21,28,35,34,33,42,52,51,55,57,56,54]\\
\sigma_{4,1} &= [60,2,6,10,15,18,17,29,48,33,51,52,50,47,53]\\
\sigma_{4,2} &= [61,4,7,14,20,21,22,23,38,37,36,35,34,55,56]\\
\sigma_{4,3} &= [58,8,5,11,13,26,25,27,39,40,41,42,43,44,57]\\
\sigma_{4,4} &= [59,1,3,9,16,12,19,24,28,32,30,45,31,46,49,54]
\end{aligned}$$

$$\sigma_{5,1} = [60, 2, 3, 9, 5, 15, 16, 11, 10, 19, 22, 23, 21, 28, 27, 37, 36, 35, 34, 33, 30]$$
$$\sigma_{5,2} = [42, 1, 4, 8, 7, 6, 14, 13, 12, 18, 29, 20, 24, 26, 17, 25, 38, 39, 40, 41, 32, 31]$$
$$\sigma_{6,1} = [37, 2, 3, 9, 5, 6, 15, 16, 10, 17, 20, 26, 25, 24, 23, 22, 34, 31]$$
$$\sigma_{6,2} = [39, 1, 4, 8, 7, 14, 13, 12, 11, 28, 29, 27, 18, 19, 21, 35, 32, 30]$$
$$\sigma_{7,1} = [31, 3, 4, 5, 6, 15, 12, 10, 11, 18, 19, 23, 21, 20]$$
$$\sigma_{7,2} = [27, 2, 9, 8, 7, 14, 16, 13, 24, 25, 26, 17, 22, 30]$$

A.2.3 SIMBA Substrategy Sizes

SIMBA Strategy	SIMBA Substrategy	Size
1	1	18
	2	18
	3	19
	4	19
2	1	18
	2	18
	3	19
	4	19
3	1	18
	2	18
	3	18
	4	19
4	1	15
	2	15
	3	15
	4	16
5	1	21
	2	22
6	1	18
	2	18
7	1	14
	2	14

Table A.1: SIMBA substrategy sizes for each round of CCCDRS-3.

Appendix B

Failed Attempts at Duality Lower Bounds for CSIDH

For any mathematical program in continuous variables, we may construct *dual* programs which bound the optimal value of the initial program (the so-called *primal* problem). When we cannot solve the primal program to optimality, dual programs give us an way to bound the extent to which our non-optimal solution is non-optimal. We attempted to use duality to derive lower bounds on the estimated running time of class group action evaluation algorithms of the form we consider; however, we were unable to derive non-trivial lower bounds in this way. In this chapter we discuss the two approaches we considered.

B.1 Formulating the Problem

In order to derive dual programs, we must first formulate the problem of finding an optimal (strategy, permutation) pair as a mathematical program. We have already seen (*q.v.* Section 5.5.2) how to formulate the problem of finding the optimal permutation for a given strategy as a linear program; in this section we proceed by first constructing a mixed-integer linear program which encodes the problem of finding an optimal strategy (encoded as a pair of strategy matrices as described in Section 5.5.1) for a fixed permutation, and then we augment it to construct a mixed-integer bilinear program in which neither the strategy nor the permutation is fixed.

B.1.1 A Mixed-Integer Linear Program for the Optimal Strategy

When the permutation is fixed, at the highest level we wish to formulate the problem

$$\begin{array}{rl} \text{Minimize} & C_M(S,\sigma) \\ \text{Subject to} & S \text{ is a strategy} \end{array} \tag{B.1}$$

We deal with the objective function in the same way as in Section 5.5.2; we substitute strategy matrices H, V and a permutation matrix Σ, and rewrite

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

which is clearly linear in the variables H, V.

All that remains is to introduce constraints that enforce that H, V are a valid pair of strategy matrices. By Lemma 5.2, H and V are a pair of strategy matrices if and only if there is a flow on T_n' which uses only the edges given by H and V, for which the flow along each edge into the sink is at most 1, and for which the total flow into the sink is at least n. We can encode such a flow as a pair of matrices $R, U \in \mathbb{R}^{(n-1)\times(n-1)}$ (the "right" flow along the horizontal edges, and the "up" flow along the vertical edges, respectively) plus an additional vector $\mathbf{t} \in \mathbb{R}^n$ (the flow into the sink). In this formulation, the flow constraints (not including the total flow constraint) become

$$\begin{aligned} R_{i,j} + U_{i,j} - R_{i,j-1} - U_{i+1,j} &= 0 \quad \forall\, 2 \le i,j \le n-2 \\ R_{i,1} + U_{i,1} - U_{i+1,1} &= 0 \quad \forall\, 1 \le i \le n-2 \\ R_{n-1,j} + U_{n-1,j} - R_{n-1,j-1} &= 0 \quad \forall\, 2 \le j \le n-1 \\ U_{1,1} - t_1 &= 0 \\ R_{i-1,i-1} + U_{i,i} - t_i &= 0 \quad \forall\, 2 \le i \le n-1 \\ R_{n-1,n-1} - t_n &= 0 \\ \mathbf{t} &\le \mathbb{1} \\ R, U, \mathbf{t} &\ge 0 \end{aligned}$$

To enforce that the flow only uses the edges specified by H and V, we require $R_{i,j} = 0$ if $H_{i,j} = 0$ and $U_{i,j} = 0$ if $V_{i,j} = 0$. The most straightforward approach to enforcing this is to use quadratic equality constraints; however, such constraints do not generally lead to easily-solved optimization problems. Instead, we note that for any flow on a Steiner arborescence for $(0,0)$ and L, we will necessarily have $R_{i,j}, U_{i,j} \leq n$ for all i and j; thus we can instead enforce that

$$R \leq nH \text{ and } U \leq nV;$$

since H and V are binary matrices, this forces $R_{i,j} = 0$ when $H_{i,j} = 0$ and $U_{i,j} = 0$ when $V_{i,j} = 0$ (that is, the flow only uses the strategy edges), and restricts $H_{i,j}, U_{i,j} \leq n$ otherwise—as noted above, this condition will always be satisfied by any flow on an appropriate Steiner arborescence, and thus does not constrain the flow through strategy edges (that is, any flow that uses only the strategy edges remains feasible, as desired).

The complete problem formulation is then

$$\begin{array}{lrl}
\text{Minimize} & \multicolumn{2}{c}{\langle \mathbb{1}\boldsymbol{\mu}^T \Sigma^T T_L^T, H\rangle_F + \langle T_R \Sigma \boldsymbol{\iota} \mathbb{1}^T, V\rangle_F} \\
\text{Subject to} & R - nH \leq 0 & \\
& U - nV \leq 0 & \\
& R_{i,j} + U_{i,j} - R_{i,j-1} - U_{i+1,j} = 0 & \forall\, 2 \leq i, j \leq n-2 \\
& R_{i,1} + U_{i,1} - U_{i+1,1} = 0 & \forall\, 1 \leq i \leq n-2 \\
& R_{n-1,j} + U_{n-1,j} - R_{n-1,j-1} = 0 & \forall\, 2 \leq j \leq n-1 \\
& U_{1,1} - t_1 = 0 & \\
& R_{i-1,i-1} + U_{i,i} - t_i = 0 & \forall\, 2 \leq i \leq n-1 \\
& R_{n-1,n-1} - t_n = 0 & \\
& \mathbb{1}^T \mathbf{t} = n & \\
& \mathbf{t} \leq \mathbb{1} & \\
& H, V \in \{0,1\}^{(n-1)\times(n-1)} & \\
& R, U, \mathbf{t} \geq 0 &
\end{array} \tag{OSP-$MILP$}$$

B.1.2 A Mixed-Integer Bilinear Program for the Optimal Permutation and Strategy

In order to model we problem of finding an optimal strategy and permutation and strategy, we simply modify (OSP-$MILP$) by allowing Σ to be a variable, and adding the necessary constraints to ensure that it encodes a permutation matrix. The results of Sections 5.5.2 and B.1.1 immediately yield the following MIBLP formulation of the problem:

$$
\begin{array}{lrl}
\text{Maximize} & \multicolumn{2}{c}{\langle \mathbb{1}\boldsymbol{\mu}^T \Sigma^T T_L^T, H\rangle_F + \langle T_R \Sigma \boldsymbol{\iota} \mathbb{1}^T, V\rangle_F} \\
\text{Subject to} & \Sigma\mathbb{1} = \mathbb{1} & \\
& \mathbb{1}^T\Sigma = \mathbb{1}^T & \\
& R - nH \leq 0 & \\
& U - nV \leq 0 & \\
& R_{i,j} + U_{i,j} - R_{i,j-1} - U_{i+1,j} = 0 & \forall\, 2 \leq i,j \leq n-2 \\
& R_{i,1} + U_{i,1} - U_{i+1,1} = 0 & \forall\, 1 \leq i \leq n-2 \\
& R_{n-1,j} + U_{n-1,j} - R_{n-1,j-1} = 0 & \forall\, 2 \leq j \leq n-1 \\
& U_{1,1} - t_1 = 0 & \\
& R_{i-1,i-1} + U_{i,i} - t_i = 0 & \forall\, 2 \leq i \leq n-1 \\
& R_{n-1,n-1} - t_n = 0 & \\
& \mathbb{1}^T\mathbf{t} = n & \\
& \mathbf{t} \leq \mathbb{1} & \\
& \Sigma \in \{0,1\}^{n\times n} & \\
& H, V \in \{0,1\}^{(n-1)\times(n-1)} & \\
& R, U, \mathbf{t} \geq 0 &
\end{array}
\tag{CSIDH-MIBLP}
$$

B.1.3 A Compact Reformulation

Before computing dual programs, we simplify and reformulate the program (CSIDH-MIBLP) as a more standard mixed-integer bilinear program.

The Variables. We vectorize and bundle the variables as

$$
\begin{aligned}
\mathbf{x} &= \operatorname{vec}(\Sigma) \in \mathbb{R}^{n^2} \\
\mathbf{y} &= \left[\operatorname{vec}(H^T)^T, \operatorname{vec}(V)^T\right]^T \in \mathbb{R}^{2(n-1)^2} \\
\mathbf{z} &= \left[\operatorname{vec}(R^T)^T, \operatorname{vec}(U)^T, \mathbf{t}^T\right]^T \in \mathbb{R}^{2(n-1)^2+n}
\end{aligned}
$$

We let $\Pi_{H^T}, \Pi_V, \Pi_{R^T}, \Pi_U$ and $\Pi_{\mathbf{t}}$ be the projectors onto the corresponding subvectors of $\mathbf{y}$ and $\mathbf{z}$; these projectors allow us to more easily express the objective function and constraints of the program (CSIDH-MIBLP).

The Objective Function. Applying the identity $\text{vec}(ABC) = (C^T \otimes A)\text{vec}(B)$, we have

$$\langle T_L^T H^T \mathbb{1}\boldsymbol{\mu}^T + T_R^T V \mathbb{1}\boldsymbol{\iota}^T, \Sigma\rangle_F = \mathbf{x}^T P \mathbf{y}$$

where

$$P = (\boldsymbol{\mu}\mathbb{1}^T \otimes T_L^T)\Pi_{H^T} + (\boldsymbol{\iota}\mathbb{1}^T \otimes T_R^T)\Pi_V$$

The Constraints. We consider the following types of linear constraints that appear in (CSIDH-MIBLP):

1. Permutation Constraints

$$\begin{matrix} \mathbb{1}^T\Sigma = \mathbb{1}^T \\ \Sigma\mathbb{1} = \mathbb{1} \end{matrix} \iff \begin{bmatrix} I_n \otimes \mathbb{1}^T \\ \mathbb{1}^T \otimes I_n \end{bmatrix} \mathbf{x} = \begin{bmatrix} \mathbb{1} \\ \mathbb{1} \end{bmatrix}$$

2. Strategy Constraints

$$\begin{matrix} R - nH \leq 0 \\ U - nV \leq 0 \end{matrix} \iff \begin{matrix} \Pi_{R^T}\mathbf{z} - n\Pi_{H^T}\mathbf{y} \leq \mathbb{0} \\ \Pi_U\mathbf{z} - n\Pi_V\mathbf{y} \leq \mathbb{0} \end{matrix}$$

3. Flow Constraints

   (a) The flow constraints for the vertices not on the x- or y-axes are

$$R_{i,j} + U_{i,j} - R_{i,j-1} - U_{i+1,j} = 0 \;\; \forall\, 2 \leq i, j \leq n-2$$

   This is readily seen to be equivalent to

$$\Big(\big(\mathbf{e}_i^T \otimes (\mathbf{e}_j - \mathbf{e}_{j-1})^T\big)\Pi_{R^T} + \big(\mathbf{e}_j^T \otimes (\mathbf{e}_i - \mathbf{e}_{i+1})^T\big)\Pi_U\Big)\mathbf{z} = 0 \;\; \forall\, 2 \leq i, j \leq n-2$$

   where $\{\mathbf{e}_i\}_{i=1}^{n-1}$ are the standard basis vectors in $\mathbb{R}^{n-1}$.

   (b) The flow constraints for vertices on the axes are

$$\begin{aligned} R_{i,1} + U_{i,1} - U_{i+1,1} &= 0 \;\; \forall\, 1 \leq i \leq n-2 \\ R_{n-1,j} + U_{n-1,j} - R_{n-1,j-1} &= 0 \;\; \forall\, 2 \leq j \leq n-1 \end{aligned}$$

which are equivalent to

$$\Big((\mathbf{e}_i^T \otimes \mathbf{e}_1^T)\Pi_{R^T} + \big(\mathbf{e}_1^T \otimes (\mathbf{e}_i - \mathbf{e}_{i+1})^T\big)\Pi_U\Big)\mathbf{z} = 0 \;\; \forall\, 1 \le i \le n-2$$
$$\Big(\big(\mathbf{e}_{n-1}^T \otimes (\mathbf{e}_j - \mathbf{e}_{j-1})^T\big)\Pi_{R^T} + \big(\mathbf{e}_j^T \otimes \mathbf{e}_{n-1}^T\big)\Pi_U\Big)\mathbf{z} = 0 \;\; \forall\, 2 \le j \le n-1$$

(c) The flow constraints at the terminal are

$$\begin{aligned} U_{1,1} - t_1 &= 0 \\ R_{i-1,i-1} + U_{i,i} - t_i &= 0 \;\; \forall\, 2 \le i \le n-1 \\ R_{n-1,n-1} - t_n &= 0 \end{aligned}$$

which are equivalent to

$$\begin{aligned} \big((\mathbf{e}_1^T \otimes \mathbf{e}_1^T)\Pi_U - \hat{\mathbf{e}}_1^T\Pi_{\mathbf{t}}\big)\mathbf{z} &= 0 \\ \big((\mathbf{e}_{i-1}^T \otimes \mathbf{e}_{i-1}^T)\Pi_{R^T} + (\mathbf{e}_i^T \otimes \mathbf{e}_i^T)\Pi_U - \hat{\mathbf{e}}_i^T\Pi_{\mathbf{t}}\big)\mathbf{z} &= 0 \;\; \forall\, 2 \le i \le n-2 \\ \big((\mathbf{e}_{n-1}^T \otimes \mathbf{e}_{n-1}^T)\Pi_{R^T} - \hat{\mathbf{e}}_n^T\Pi_{\mathbf{t}}\big)\mathbf{z} &= 0 \end{aligned}$$

where $\{\hat{\mathbf{e}}_i\}_{i=1}^n$ are the standard basis vectors in $\mathbb{R}^n$.

(d) The total flow into the terminal must be n; this can be rewritten as

$$\mathbb{1}^T\mathbf{t} = n \iff \mathbb{1}^T\Pi_{\mathbf{t}}\mathbf{z} = n$$

4. Bounds and Binary Variables

We have $\mathbf{x} \in \{0,1\}^{n^2}$, $\mathbf{y} \in \{0,1\}^{2(n-1)^2}$, $\mathbf{z} \ge \mathbb{0}$, and $\Pi_{\mathbf{t}}\mathbf{z} \le \mathbb{1}$.

The Primal Problem. Having rewritten the objective function and constraints, we can rewrite the primal problem as

$$\begin{array}{rr} \text{Minimize} & \mathbf{x}^T P\mathbf{y} \\ \text{Subject to} & A_{\mathbf{x}}\mathbf{x} - \mathbf{b}_{\mathbf{x}} = \mathbb{0} \\ & A_{\mathbf{z}}\mathbf{z} - \mathbf{b}_{\mathbf{z}} = \mathbb{0} \\ & C_{\mathbf{y}}\mathbf{y} + C_{\mathbf{z}}\mathbf{z} \le \mathbb{0} \\ & \mathbf{x} \in \{0,1\}^{n^2} \\ & \mathbf{y} \in \{0,1\}^{2(n-1)^2} \\ & \mathbf{z} \ge \mathbb{0} \\ & \Pi_{\mathbf{t}}\mathbf{z} \le \mathbb{1} \end{array} \tag{CSIDH-S-MIBLP}$$

where

$$P = (\boldsymbol{\mu}\mathbb{1}^T \otimes T_L^T)\Pi_{H^T} + (\boldsymbol{\iota}\mathbb{1}^T \otimes T_R^T)\Pi_V,$$

$$A_{\mathbf{x}} = \begin{bmatrix} I_n \otimes \mathbb{1}^T \\ \mathbb{1}^T \otimes I_n \end{bmatrix}, \ \mathbf{b}_{\mathbf{x}} = \mathbb{1}_{2n},$$

$$A_{\mathbf{z}} = \begin{bmatrix} \Big[\Big(\big(\mathbf{e}_i^T \otimes (\mathbf{e}_j - \mathbf{e}_{j-1})^T\big)\Pi_{R^T} + \big(\mathbf{e}_j^T \otimes (\mathbf{e}_i - \mathbf{e}_{i+1})^T\big)\Pi_U\Big)\Big]_{i,j=2}^{n-2} \\ \Big[\Big(\big(\mathbf{e}_i^T \otimes \mathbf{e}_1^T\big)\Pi_{R^T} + \big(\mathbf{e}_1^T \otimes (\mathbf{e}_i - \mathbf{e}_{i+1})^T\big)\Pi_U\Big)\Big]_{i=1}^{n-2} \\ \Big[\Big(\big(\mathbf{e}_{n-1}^T \otimes (\mathbf{e}_j - \mathbf{e}_{j-1})^T\big)\Pi_{R^T} + \big(\mathbf{e}_j^T \otimes \mathbf{e}_{n-1}^T\big)\Pi_U\Big)\Big]_{j=2}^{n-1} \\ \mathbb{1}^T \end{bmatrix}, \ \mathbf{b}_{\mathbf{z}} = \begin{bmatrix} 0_{(n-3)^3+2(n-3)} \\ n \end{bmatrix}$$

$$C_{\mathbf{y}} = \begin{bmatrix} -n\Pi_{H^T} \\ -n\Pi_V \end{bmatrix}, \text{ and } C_{\mathbf{z}} = \begin{bmatrix} \Pi_{R^T} \\ \Pi_U \end{bmatrix}$$

B.1.4 Relaxation and Lower Bounds

The problem (CSIDH-S-MIBLP) is still difficult to solve because of its integer variables and non-convex objective. We relax away the difficult parts; in particular, we replace the constraints $\mathbf{x} \in \{0,1\}^{n^2}$, $\mathbf{y} \in \{0,1\}^{2(n-1)^2}$ by

$$\mathbf{x} \ge \mathbb{0} \qquad \mathbf{x} \le \mathbb{1} \qquad \mathbf{y} \ge \mathbb{0} \qquad \mathbf{y} \le \mathbb{1}$$

and introduce new variables $W = \mathbf{x}\mathbf{y}^T$ to make the objective linear: $\mathbf{x}^T P\mathbf{y} = \langle P, W\rangle$. Of course, the constraint $W = \mathbf{x}\mathbf{y}^T$ is non-linear, which is not desirable. We relax this constraint using McCormick envelopes [54] to obtain

$$W \geq 0 \qquad W - \mathbf{x}\mathbb{1}_{2(n-1)^2}^T - \mathbb{1}_{n^2}\mathbf{y}^T \geq -J \qquad W - \mathbb{1}_{n^2}\mathbf{y}^T \leq 0 \qquad W - \mathbf{x}\mathbb{1}_{2(n-1)^2}^T \leq 0$$

which yields the full relaxed program

$$\begin{array}{rr} \text{Minimize} & \langle P, W\rangle_F \\ \text{Subject to} & A_{\mathbf{x}}\mathbf{x} - \mathbf{b}_{\mathbf{x}} = \mathbb{0} \\ & A_{\mathbf{z}}\mathbf{z} - \mathbf{b}_{\mathbf{z}} = \mathbb{0} \\ & C_{\mathbf{y}}\mathbf{y} + C_{\mathbf{z}}\mathbf{z} \leq \mathbb{0} \\ & \mathbf{x} \geq \mathbb{0} \\ & \mathbf{x} \leq \mathbb{1} \\ & \mathbf{y} \geq \mathbb{0} \\ & \mathbf{y} \leq \mathbb{1} \\ & \mathbf{z} \geq \mathbb{0} \\ & \Pi_{\mathbf{t}}\mathbf{z} \leq \mathbb{1} \\ & W \geq 0 \\ & W - \mathbf{x}\mathbb{1}_{2(n-1)^2}^T - \mathbb{1}_{n^2}\mathbf{y}^T \geq -J \\ & W - \mathbb{1}_{n^2}\mathbf{y}^T \leq 0 \\ & W - \mathbf{x}\mathbb{1}_{2(n-1)^2}^T \leq 0 \end{array} \tag{CSIDH-LPR}$$

From here we simply compute the dual. The Lagrangian is

$$\begin{aligned} &\mathcal{L}(\mathbf{x}, \mathbf{y}, \mathbf{z}, W; \boldsymbol{\lambda}_{\mathbf{x}}, \boldsymbol{\lambda}_{\mathbf{z}}, \boldsymbol{\nu}, \boldsymbol{\beta}_{\mathbf{x}}, \boldsymbol{\beta}_{\mathbf{y}}, \boldsymbol{\beta}_{\mathbf{z}}, \boldsymbol{\gamma}_{\mathbf{x}}, \boldsymbol{\gamma}_{\mathbf{y}}, \boldsymbol{\gamma}_{\mathbf{z}}, \Theta_1, \Theta_2, \Theta_3, \Theta_4) \\ &\quad = \langle P, W\rangle_F + \boldsymbol{\lambda}_{\mathbf{x}}^T(A_{\mathbf{x}}\mathbf{x} - \mathbf{b}_{\mathbf{x}}) + \boldsymbol{\lambda}_{\mathbf{z}}^T(A_{\mathbf{z}}\mathbf{z} - \mathbf{b}_{\mathbf{z}}) + \boldsymbol{\nu}^T(C_{\mathbf{y}}\mathbf{y} + C_{\mathbf{z}}\mathbf{z}) \\ &\qquad -\boldsymbol{\beta}_{\mathbf{x}}^T\mathbf{x} - \boldsymbol{\beta}_{\mathbf{y}}^T\mathbf{y} - \boldsymbol{\beta}_{\mathbf{z}}^T\mathbf{z} + \boldsymbol{\gamma}_{\mathbf{x}}^T(\mathbf{x} - \mathbb{1}) + \boldsymbol{\gamma}_{\mathbf{y}}^T(\mathbf{y} - \mathbb{1}) + \boldsymbol{\gamma}_{\mathbf{z}}^T(\Pi_{\mathbf{t}}\mathbf{z} - \mathbb{1}) \\ &\qquad -\langle \Theta_1, W\rangle_F - \langle \Theta_2, W - \mathbf{x}\mathbb{1}_{2(n-1)^2}^T - \mathbb{1}_{n^2}\mathbf{y}^T + J\rangle_F + \langle \Theta_3, W - \mathbb{1}_{n^2}\mathbf{y}^T\rangle_F \\ &\qquad +\langle \Theta_4, W - \mathbf{x}\mathbb{1}_{2(n-1)^2}^T\rangle_F \\ &\quad = \langle W, P - \Theta_1 - \Theta_2 + \Theta_3 + \Theta_4\rangle_F + \mathbf{x}^T(A_{\mathbf{x}}^T\boldsymbol{\lambda}_{\mathbf{x}} - \boldsymbol{\beta}_{\mathbf{x}} + \boldsymbol{\gamma}_{\mathbf{x}} + (\Theta_2 - \Theta_4)\mathbb{1}) \\ &\qquad \mathbf{y}^T(C_{\mathbf{y}}^T\boldsymbol{\nu} - \boldsymbol{\beta}_{\mathbf{y}} + \boldsymbol{\gamma}_{\mathbf{y}} + (\Theta_2 - \Theta_3)^T\mathbb{1}) + \mathbf{z}^T(A_{\mathbf{z}}^T\boldsymbol{\lambda}_{\mathbf{z}} + C_{\mathbf{z}}^T\boldsymbol{\nu} - \boldsymbol{\beta}_{\mathbf{z}} + \Pi_{\mathbf{t}}^T\boldsymbol{\gamma}_{\mathbf{z}}) \\ &\qquad -\langle J, \Theta_2\rangle_F - \mathbf{b}_{\mathbf{x}}^T\boldsymbol{\lambda}_{\mathbf{x}} - \mathbf{b}_{\mathbf{z}}^T\boldsymbol{\lambda}_{\mathbf{z}} - \mathbb{1}^T\boldsymbol{\gamma}_{\mathbf{x}} - \mathbb{1}^T\boldsymbol{\gamma}_{\mathbf{y}} - \mathbb{1}^T\boldsymbol{\gamma}_{\mathbf{z}} \end{aligned} \tag{B.2}$$

which yields the dual objective function

$$\begin{aligned}
&g(\boldsymbol{\lambda}_{\mathbf{x}}, \boldsymbol{\lambda}_{\mathbf{z}}, \boldsymbol{\nu}, \boldsymbol{\beta}_{\mathbf{x}}, \boldsymbol{\beta}_{\mathbf{y}}, \boldsymbol{\beta}_{\mathbf{z}}, \boldsymbol{\gamma}_{\mathbf{x}}, \boldsymbol{\gamma}_{\mathbf{y}}, \boldsymbol{\gamma}_{\mathbf{z}}, \Theta_1, \Theta_2, \Theta_3, \Theta_4) \\
&\quad = \inf_{\mathbf{x},\mathbf{y},\mathbf{z},W} \mathcal{L}(\mathbf{x}, \mathbf{y}, \mathbf{z}, W; \boldsymbol{\lambda}_{\mathbf{x}}\, \boldsymbol{\lambda}_{\mathbf{z}}, \boldsymbol{\nu}, \boldsymbol{\beta}_{\mathbf{x}}, \boldsymbol{\beta}_{\mathbf{y}}, \boldsymbol{\beta}_{\mathbf{z}}, \boldsymbol{\gamma}_{\mathbf{x}}, \boldsymbol{\gamma}_{\mathbf{y}}, \boldsymbol{\gamma}_{\mathbf{z}}, \Theta_1, \Theta_2, \Theta_3, \Theta_4) \\
&\quad = \begin{cases} \begin{array}{c} -\langle J, \Theta_2\rangle_F - \mathbf{b}_{\mathbf{x}}^T \boldsymbol{\lambda}_{\mathbf{x}} - \mathbf{b}_{\mathbf{z}}^T \boldsymbol{\lambda}_{\mathbf{z}} \\ -\mathbb{1}^T \boldsymbol{\gamma}_{\mathbf{x}} - \mathbb{1}^T \boldsymbol{\gamma}_{\mathbf{y}} - \mathbb{1}^T \boldsymbol{\gamma}_{\mathbf{z}} \end{array} & \text{if} \quad \begin{array}{r} A_{\mathbf{x}}^T \boldsymbol{\lambda}_{\mathbf{x}} - \boldsymbol{\beta}_{\mathbf{x}} + \boldsymbol{\gamma}_{\mathbf{x}} + (\Theta_2 - \Theta_4)\mathbb{1} = \mathbb{0} \\ C_{\mathbf{y}}^T \boldsymbol{\nu} - \boldsymbol{\beta}_{\mathbf{y}} + \boldsymbol{\gamma}_{\mathbf{y}} + (\Theta_2 - \Theta_3)^T \mathbb{1} = \mathbb{0} \\ A_{\mathbf{z}}^T \boldsymbol{\lambda}_{\mathbf{z}} + C_{\mathbf{z}}^T \boldsymbol{\nu} - \boldsymbol{\beta}_{\mathbf{z}} + \Pi_{\mathbf{t}}^T \boldsymbol{\gamma}_{\mathbf{z}} = \mathbb{0} \\ \Theta_1 + \Theta_2 - \Theta_3 - \Theta_4 = P \end{array} \\ -\infty & \text{otherwise} \end{cases}
\end{aligned}$$

and hence the “naïve” Lagrangian dual program

$$\begin{aligned}
\text{Maximize} \quad & -\langle J, \Theta_2\rangle_F - \mathbf{b}_{\mathbf{x}}^T \boldsymbol{\lambda}_{\mathbf{x}} - \mathbf{b}_{\mathbf{z}}^T \boldsymbol{\lambda}_{\mathbf{z}} - \mathbb{1}^T \boldsymbol{\gamma}_{\mathbf{x}} - \mathbb{1}^T \boldsymbol{\gamma}_{\mathbf{y}} - \mathbb{1}^T \boldsymbol{\gamma}_{\mathbf{z}} \\
\text{Subject to} \quad & A_{\mathbf{x}}^T \boldsymbol{\lambda}_{\mathbf{x}} - \boldsymbol{\beta}_{\mathbf{x}} + \boldsymbol{\gamma}_{\mathbf{x}} + (\Theta_2 - \Theta_4)\mathbb{1} = \mathbb{0} \\
& C_{\mathbf{y}}^T \boldsymbol{\nu} - \boldsymbol{\beta}_{\mathbf{y}} + \boldsymbol{\gamma}_{\mathbf{y}} + (\Theta_2 - \Theta_3)^T \mathbb{1} = \mathbb{0} \\
& A_{\mathbf{z}}^T \boldsymbol{\lambda}_{\mathbf{z}} + C_{\mathbf{z}}^T \boldsymbol{\nu} - \boldsymbol{\beta}_{\mathbf{z}} + \Pi_{\mathbf{t}}^T \boldsymbol{\gamma}_{\mathbf{z}} = \mathbb{0} \\
& \Theta_1 + \Theta_2 - \Theta_3 - \Theta_4 = P \\
& \boldsymbol{\nu}, \boldsymbol{\beta}_{\mathbf{x}}, \boldsymbol{\beta}_{\mathbf{y}}, \boldsymbol{\beta}_{\mathbf{z}}, \boldsymbol{\gamma}_{\mathbf{x}}, \boldsymbol{\gamma}_{\mathbf{y}}, \boldsymbol{\gamma}_{\mathbf{z}} \geq \mathbb{0} \\
& \Theta_1, \Theta_2, \Theta_3, \Theta_4 \geq 0
\end{aligned} \tag{CSIDH-NLD}$$

Noting that $\boldsymbol{\beta}_{\mathbf{x}}, \boldsymbol{\beta}_{\mathbf{y}}, \boldsymbol{\beta}_{\mathbf{z}}$, and Θ_1 do not appear in the objective function, the program can be simplified to (after reindexing the Θ variables):

$$\begin{aligned}
\text{Maximize} \quad & -\langle J, \Theta_1\rangle_F - \mathbf{b}_{\mathbf{x}}^T \boldsymbol{\lambda}_{\mathbf{x}} - \mathbf{b}_{\mathbf{z}}^T \boldsymbol{\lambda}_{\mathbf{z}} - \mathbb{1}^T \boldsymbol{\gamma}_{\mathbf{x}} - \mathbb{1}^T \boldsymbol{\gamma}_{\mathbf{y}} - \mathbb{1}^T \boldsymbol{\gamma}_{\mathbf{z}} \\
\text{Subject to} \quad & A_{\mathbf{x}}^T \boldsymbol{\lambda}_{\mathbf{x}} + \boldsymbol{\gamma}_{\mathbf{x}} + (\Theta_1 - \Theta_3)\mathbb{1} \geq \mathbb{0} \\
& C_{\mathbf{y}}^T \boldsymbol{\nu} + \boldsymbol{\gamma}_{\mathbf{y}} + (\Theta_1 - \Theta_2)^T \mathbb{1} \geq \mathbb{0} \\
& A_{\mathbf{z}}^T \boldsymbol{\lambda}_{\mathbf{z}} + C_{\mathbf{z}}^T \boldsymbol{\nu} + \Pi_{\mathbf{t}}^T \boldsymbol{\gamma}_{\mathbf{z}} \geq \mathbb{0} \\
& \Theta_1 - \Theta_2 - \Theta_3 \leq P \\
& \boldsymbol{\nu}, \boldsymbol{\gamma}_{\mathbf{x}}, \boldsymbol{\gamma}_{\mathbf{y}}, \boldsymbol{\gamma}_{\mathbf{z}} \geq \mathbb{0} \\
& \Theta_1, \Theta_2, \Theta_3 \geq 0
\end{aligned} \tag{CSIDH-LD}$$

Unfortunately, in numerical experiments for CSIDH parameter sets of very small order, (CSIDH-LD) has optimal value 0, making it not useful as a method to obtain lower bounds.

B.2 A Quadratic Programming Approach

In Section B.1.2 we wrote the (strategy, permutation) problem as a mixed-integer bilinear program in order to obtain duality lower bounds. Notably, we can also write the problem as a *quadratic program* and attempt to get duality lower bounds from there; we detail the formulation here. As before, we will bundle variables together; in particular, we define

$$\mathbf{x} = \left[\mathrm{vec}(\Sigma)^T, \mathrm{vec}(H^T)^T, \mathrm{vec}(V)^T, \mathrm{vec}(R^T)^T, \mathrm{vec}(U)^T, \mathbf{y}^T\right]^T \in \mathbb{R}^{n^2+4(n-1)^2+n}$$

We let $\Pi_\Sigma, \Pi_{H^T}, \Pi_V, \Pi_{R^T}, \Pi_U$ and $\Pi_{\mathbf{y}}$ be the projectors from $\mathbb{R}^{n^2+4(n-1)^2+n}$ onto the corresponding subvectors of $\mathbf{x}$; these projectors allow us to more easily express the objective function and constraints of CSIDH-MIBLP

The Objective Function. Applying the identity $\mathrm{vec}(ABC) = (C^T \otimes A)\mathrm{vec}(B)$, we have

$$\langle T_L^T H^T \mathbb{1}\boldsymbol{\mu}^T + T_R^T V \mathbb{1}\boldsymbol{\iota}^T, \Sigma\rangle_F = \frac{1}{2}\mathbf{x}^T P \mathbf{x}$$

where

$$P = \left(\Pi_V^T(\mathbb{1}\boldsymbol{\iota}^T \otimes T_R) + \Pi_{H^T}^T(\mathbb{1}\boldsymbol{\mu}^T \otimes T_L)\right)\Pi_\Sigma + \Pi_\Sigma^T\left((\boldsymbol{\mu}\mathbb{1}^T \otimes T_L^T)\Pi_{H^T} + (\boldsymbol{\iota}\mathbb{1}^T \otimes T_R^T)\Pi_V\right)$$

The Constraints. We consider the following types of linear constraints that appear in (CSIDH-MIBLP):

1. Permutation Constraints

$$\begin{matrix}\mathbb{1}^T\Sigma = \mathbb{1}^T \\ \Sigma\mathbb{1} = \mathbb{1}\end{matrix} \iff \begin{bmatrix}(I_n \otimes \mathbb{1}^T)\Pi_\Sigma \\ (\mathbb{1}^T \otimes I_n)\Pi_\Sigma\end{bmatrix}\mathbf{x} = \begin{bmatrix}\mathbb{1} \\ \mathbb{1}\end{bmatrix}$$

2. Strategy Constraints

$$\begin{matrix}R - nH \leq 0 \\ U - nV \leq 0\end{matrix} \iff \begin{matrix}(\Pi_{R^T} - n\Pi_{H^T})\mathbf{x} \leq \mathbb{0} \\ (\Pi_U - n\Pi_V)\mathbf{x} \leq \mathbb{0}\end{matrix}$$

3. Flow Constraints

(a) The flow constraints for the vertices not on x or y axes are

$$R_{i,j} + U_{i,j} - R_{i,j-1} - U_{i+1,j} = 0 \ \ \forall\, 2 \leq i, j \leq n-2$$

This is readily seen to be equivalent to

$$\Big(\big(\mathbf{e}_i^T \otimes (\mathbf{e}_j - \mathbf{e}_{j-1})^T\big)\Pi_{R^T} + \big(\mathbf{e}_j^T \otimes (\mathbf{e}_i - \mathbf{e}_{i+1})^T\big)\Pi_U\Big)\mathbf{x} = 0 \ \ \forall\, 2 \leq i, j \leq n-2$$

where $\{\mathbf{e}_i\}_{i=1}^{n-1}$ are the standard basis vectors in $\mathbb{R}^{n-1}$.

(b) The flow constraints for vertices on the axes are

$$\begin{aligned} R_{i,1} + U_{i,1} - U_{i+1,1} &= 0 \ \ \forall\, 1 \leq i \leq n-2 \\ R_{n-1,j} + U_{n-1,j} - R_{n-1,j-1} &= 0 \ \ \forall\, 2 \leq j \leq n-1 \end{aligned}$$

which are equivalent to

$$\begin{aligned} \Big(\big(\mathbf{e}_i^T \otimes \mathbf{e}_1^T\big)\Pi_{R^T} + \big(\mathbf{e}_1^T \otimes (\mathbf{e}_i - \mathbf{e}_{i+1})^T\big)\Pi_U\Big)\mathbf{x} &= 0 \ \ \forall\, 1 \leq i \leq n-2 \\ \Big(\big(\mathbf{e}_{n-1}^T \otimes (\mathbf{e}_j - \mathbf{e}_{j-1})^T\big)\Pi_{R^T} + \big(\mathbf{e}_j^T \otimes \mathbf{e}_{n-1}^T\big)\Pi_U\Big)\mathbf{x} &= 0 \ \ \forall\, 2 \leq j \leq n-1 \end{aligned}$$

(c) The flow constraints at the terminal are

$$\begin{aligned} U_{1,1} - y_1 &= 0 \\ R_{i-1,i-1} + U_{i,i} - y_i &= 0 \ \ \forall\, 2 \leq i \leq n-1 \\ R_{n-1,n-1} - y_n &= 0 \end{aligned}$$

which are equivalent to

$$\begin{aligned} \big((\mathbf{e}_1^T \otimes \mathbf{e}_1^T)\Pi_U - \hat{\mathbf{e}}_1^T \Pi_{\mathbf{y}}\big)\mathbf{x} &= 0 \\ \big((\mathbf{e}_{i-1}^T \otimes \mathbf{e}_{i-1}^T)\Pi_{R^T} + (\mathbf{e}_i^T \otimes \mathbf{e}_i^T)\Pi_U - \hat{\mathbf{e}}_i^T \Pi_{\mathbf{y}}\big)\mathbf{x} &= 0 \ \ \forall\ 2 \leq i \leq n-2 \\ \big((\mathbf{e}_{n-1}^T \otimes \mathbf{e}_{n-1}^T)\Pi_{R^T} - \hat{\mathbf{e}}_n^T \Pi_{\mathbf{y}}\big)\mathbf{x} &= 0 \end{aligned}$$

where $\{\hat{\mathbf{e}}_i\}_{i=1}^{n}$ are the standard basis vectors in $\mathbb{R}^n$.

(d) The total flow into the terminal must be n; this can be rewritten as

$$\mathbb{1}^T \mathbf{y} = n \iff \mathbb{1}^T \Pi_{\mathbf{y}} \mathbf{x} = n$$

4. Bounds

   We have

$$\mathbf{x} \geq \mathbb{0} \text{ and } \begin{bmatrix} \Pi_\Sigma \\ \Pi_{H^T} \\ \Pi_V \\ \Pi_{\mathbf{y}} \end{bmatrix} \mathbf{x} \leq \mathbb{1}$$

The Primal Problem. Having rewritten the objective function and constraints, we can rewrite the primal problem as

$$\begin{array}{rl} \text{Minimize} & \frac{1}{2}\mathbf{x}^T P \mathbf{x} \\ \text{Subject to} & A_{\text{eq}}\mathbf{x} - \mathbf{b}_{\text{eq}} = \mathbb{0} \\ & A\mathbf{x} - \mathbf{b} \leq \mathbb{0} \end{array} \tag{CSIDH-MIQP}$$

where

$$P = \big((\Pi_V^T(\mathbb{1}\boldsymbol{\iota}^T \otimes T_R) + \Pi_{H^T}^T(\mathbb{1}\boldsymbol{\mu}^T \otimes T_L)\big)\Pi_\Sigma + \Pi_\Sigma^T\big((\boldsymbol{\mu}\mathbb{1}^T \otimes T_L^T)\Pi_{H^T} + (\boldsymbol{\iota}\mathbb{1}^T \otimes T_R^T)\Pi_V\big)$$

$$A_{\text{eq}} = \begin{bmatrix} (I_n \otimes \mathbb{1}^T)\Pi_\Sigma \\ (\mathbb{1}^T \otimes I_n)\Pi_\Sigma \\ \Big[\Big(\big(\mathbf{e}_i^T \otimes (\mathbf{e}_j - \mathbf{e}_{j-1})^T\big)\Pi_{R^T} + \big(\mathbf{e}_j^T \otimes (\mathbf{e}_i - \mathbf{e}_{i+1})^T\big)\Pi_U\Big)\mathbf{x}\Big]_{i,j=2}^{n-2} \\ \Big[\Big(\big(\mathbf{e}_i^T \otimes \mathbf{e}_1^T\big)\Pi_{R^T} + \big(\mathbf{e}_1^T \otimes (\mathbf{e}_i - \mathbf{e}_{i+1})^T\big)\Pi_U\Big)\mathbf{x}\Big]_{i=1}^{n-2} \\ \Big[\Big(\big(\mathbf{e}_{n-1}^T \otimes (\mathbf{e}_j - \mathbf{e}_{j-1})^T\big)\Pi_{R^T} + \big(\mathbf{e}_j^T \otimes \mathbf{e}_{n-1}^T\big)\Pi_U\Big)\mathbf{x}\Big]_{j=2}^{n-1} \end{bmatrix}$$

$$A = \begin{bmatrix} \Pi_{R^T} - n\Pi_{H^T} \\ \Pi_U - n\Pi_V \\ -I_{n^2+4(n-1)^2+n} \\ \Pi_\Sigma \\ \Pi_{H^T} \\ \Pi_V \\ \Pi_{\mathbf{y}} \end{bmatrix}$$

$$\mathbf{b}_{\text{eq}} = \begin{bmatrix} \mathbb{1}_{2n} \\ \mathbb{0}_{(n-3)^2+3(n-2)} \\ n \end{bmatrix}$$

$$\mathbf{b} = \begin{bmatrix} \mathbb{0}_{2(n-1)^2} \\ \mathbb{0}_{n^2+4(n-1)^2+n} \\ \mathbb{1}_{n^2+2(n-1)^2+n} \end{bmatrix}$$

Dual Programs. To begin, we have the Lagrangian

$$\mathcal{L}(\mathbf{x}; \boldsymbol{\lambda}, \boldsymbol{\mu}) = \frac{1}{2}\mathbf{x}^T P \mathbf{x} + (A_{\text{eq}}^T \boldsymbol{\lambda} + A^T \boldsymbol{\mu})^T \mathbf{x} - \mathbf{b}_{\text{eq}}^T \boldsymbol{\lambda} - \mathbf{b}^T \boldsymbol{\mu}$$

We first derive the Lagrangian dual program. The Lagrangian dual function is

$$\begin{aligned} g(\boldsymbol{\lambda}, \boldsymbol{\mu}) &= \inf_{\mathbf{x} \in \mathbb{R}^{n^2+4(n-1)^2+n}} \mathcal{L}(\mathbf{x}; \boldsymbol{\lambda}, \boldsymbol{\mu}) \\ &= \begin{cases} -\infty & \text{if } P \not\succeq 0 \text{ or } (A_{\text{eq}}^T \boldsymbol{\lambda} + A^T \boldsymbol{\mu}) \notin \mathcal{R}(P) \\ -\frac{1}{2}(A_{\text{eq}}^T \boldsymbol{\lambda} + A^T \boldsymbol{\mu})^T P^+ (A_{\text{eq}}^T \boldsymbol{\lambda} + A^T \boldsymbol{\mu}) - \mathbf{b}_{\text{eq}}^T \boldsymbol{\lambda} - \mathbf{b}^T \boldsymbol{\mu} & \text{otherwise} \end{cases} \end{aligned}$$

Unfortunately, we have the following result:

Claim B.1. For any generalized measure $M = ((\ell_i)_{i=1}^n, f_H, f_V)$, $P \not\succeq 0$.

Proof. Note that

1. $P = P^T$,
2. $\mathbf{e}_1^T P \mathbf{e}_1 = 0$, and
3. $\mathbf{e}_1^T P \mathbf{e}_{n^2+1} = \mathbf{e}_{n^2+1}^T P \mathbf{e}_1 = \boldsymbol{\mu}_1 > 0$,

so that $P \not\succeq 0$ by Sylvester's Criterion (for positive semidefiniteness) [64]. □

Having failed to obtain a lower bound using the Lagrangian dual of the exact primal problem, we relax the primal and try again.

Consider the following semidefinite programming relaxation of (CSIDH-MIQP)

$$\begin{aligned} \text{Minimize} \quad & \tfrac{1}{2}\langle P, X \rangle_F \\ \text{Subject to} \quad & \tfrac{1}{2}\langle Q, X \rangle_F + \mathbf{q}^T \mathbf{x} \leq 0 \\ & A_{\text{eq}} \mathbf{x} - \mathbf{b}_{\text{eq}} = \mathbb{0} \\ & A\mathbf{x} - \mathbf{b} \leq \mathbb{0} \\ & \begin{bmatrix} 1 & \mathbf{x}^T \\ \mathbf{x} & X \end{bmatrix} \succeq 0 \end{aligned} \tag{CSIDH-SDPR}$$

We see that whenever $\mathbf{x}$ is feasble for (CSIDH-MIQP), the pair $(\mathbf{x}, X = \mathbf{x}\mathbf{x}^T)$ is feasible for (CSIDH-SDPR), and has the same objective value; thus (CSIDH-SDPR) really is a relaxation of (CSIDH-MIQP). Any lower bound for (CSIDH-SDPR) thus yields a lower bound for (CSIDH-MIQP).

The Lagrangian for this problem is

$$\mathcal{L}(\mathbf{x}, X; \boldsymbol{\nu}, \boldsymbol{\mu}, \boldsymbol{\lambda}_0, \boldsymbol{\lambda}, \Lambda) = \frac{1}{2}\langle P - \Lambda, X\rangle_F + \mathbf{x}^T(A_{\text{eq}}^T\boldsymbol{\nu} + A^T\boldsymbol{\mu} - 2\boldsymbol{\lambda}) - \mathbf{b}_{\text{eq}}^T\boldsymbol{\nu} - \mathbf{b}^T\boldsymbol{\mu} - \boldsymbol{\lambda}_0$$

As before, we compute the Lagrangian dual function

$$\begin{aligned} g(\boldsymbol{\nu}, \boldsymbol{\mu}, \boldsymbol{\lambda}_0, \boldsymbol{\lambda}, \Lambda) &= \inf_{\substack{X \in \mathbb{R}^{(n^2+4(n-1)^2+n)^2} \\ \mathbf{x} \in \mathbb{R}^{n^2+4(n-1)^2+n}}} \mathcal{L}(\mathbf{x}, X; \boldsymbol{\nu}, \boldsymbol{\mu}, \boldsymbol{\lambda}_0, \boldsymbol{\lambda}, \Lambda) \\ &= \begin{cases} -\mathbf{b}_{\text{eq}}^T\boldsymbol{\nu} - \mathbf{b}^T\boldsymbol{\mu} - \boldsymbol{\lambda}_0 & \text{if } \Lambda = P \text{ and } A_{\text{eq}}^T\boldsymbol{\nu} + A^T\boldsymbol{\mu} - 2\boldsymbol{\lambda} = \mathbb{0} \\ -\infty & \text{otherwise} \end{cases} \end{aligned}$$

and thus construct the naïve Lagrangian dual program

$$\begin{array}{rl} \text{Maximize} & -\mathbf{b}_{\text{eq}}^T\boldsymbol{\nu} - \mathbf{b}^T\boldsymbol{\mu} - \boldsymbol{\lambda}_0 \\ \text{Subject to} & A_{\text{eq}}^T\boldsymbol{\nu} + A^T\boldsymbol{\mu} + -2\boldsymbol{\lambda} = \mathbb{0} \\ & \boldsymbol{\mu} \geq \mathbb{0} \\ & \begin{bmatrix} \boldsymbol{\lambda}_0 & \boldsymbol{\lambda}^T \\ \boldsymbol{\lambda} & P \end{bmatrix} \succeq 0 \end{array} \tag{SDPR-NLD}$$

Unfortunately, Claim B.1 immediately implies that (SDPR-NLD) is infeasible, since again we require $P \succeq 0$. To overcome this problem, we slightly modify the naïve semidefinite relaxation (CSIDH-SDPR) by introducing the constraint $X \geq 0$; we note that this new program is still a relaxation of (CSIDH-MIQP) since the pair $(\mathbf{x}, X = \mathbf{x}\mathbf{x}^T)$ remains feasible whenever $\mathbf{x}$ is feasible for (CSIDH-MIQP) (recall that $\mathbf{x} \geq \mathbf{0}$ is a subsystem of $A\mathbf{x} - \mathbf{b} \leq \mathbf{0}$). The new Lagrangian dual is

$$\begin{array}{rl} \text{Maximize} & -\mathbf{b}_{\text{eq}}^T\boldsymbol{\nu} - \mathbf{b}^T\boldsymbol{\mu} - \boldsymbol{\lambda}_0 \\ \text{Subject to} & A_{\text{eq}}^T\boldsymbol{\nu} + A^T\boldsymbol{\mu} - 2\boldsymbol{\lambda} = \mathbb{0} \\ & \Lambda \leq P \\ & \boldsymbol{\mu} \geq \mathbb{0} \\ & \begin{bmatrix} \boldsymbol{\lambda}_0 & \boldsymbol{\lambda}^T \\ \boldsymbol{\lambda} & \Lambda \end{bmatrix} \succeq 0 \end{array} \tag{SDPR-NLD'}$$

The program (SDPR-NLD') is clearly feasible (take all variables to be 0). We can simplify the program considerably by noting that $\operatorname{diag}(P) = \mathbb{0}$ so that the only solution to $\Lambda \leq P, \Lambda \succeq 0$ is $\Lambda = 0$ (by Sylvester's criterion). This then requires $\boldsymbol{\lambda} = \mathbb{0}$, so that the program becomes

$$\begin{array}{rl} \text{Maximize} & -\mathbf{b}_{\text{eq}}^T \boldsymbol{\nu} - \mathbf{b}^T \boldsymbol{\mu} - \boldsymbol{\lambda}_0 \\ \text{Subject to} & A_{\text{eq}}^T \boldsymbol{\nu} + A^T \boldsymbol{\mu} = \mathbb{0} \\ & \boldsymbol{\mu} \geq \mathbb{0} \\ & \boldsymbol{\lambda}_0 \geq 0 \end{array} \tag{S-SDPR-NLD}$$

Unfortunately, we have the following result.

Claim B.2. The optimal value of (S-SDPR-NLD) is 0.

Proof. The dual of (S-SDPR-NLD) is

$$\begin{array}{rl} \text{Maximize} & \mathbb{0}^T \hat{\mathbf{x}} \\ \text{Subject to} & A_{\text{eq}} \hat{\mathbf{x}} = -\mathbf{b}_{\text{eq}} \\ & A\hat{\mathbf{x}} \geq -\mathbf{b} \end{array} \tag{S-SDPR-NLDD}$$

Since (S-SDPR-NLD) is feasible (take $\boldsymbol{\nu} = \mathbb{0}, \boldsymbol{\mu} = \mathbb{0}$, and $\boldsymbol{\lambda}_0 = 0$), by the strong duality theorem for linear programs, (S-SDPR-NLDD) is also feasible. But any feasible point for (S-SDPR-NLDD) has objective value 0; thus by strong duality (S-SDPR-NLD) has optimal value 0. □