THE PENNSYLVANIA STATE UNIVERSITY
SCHREYER HONORS COLLEGE

DEPARTMENT OF MATHEMATICS

A Computational Comparison of CSIDH and Constant-Time CSIDH

HARRISON LEMLEY
SPRING 2025

A thesis
submitted in partial fulfillment
of the requirements
for baccalaureate degrees
in Mathematics
with honors in Mathematics

Reviewed and approved* by the following:

Kirsten Eisenträger
Professor of Mathematics
Pentz Professor of Science
Thesis Supervisor

Dmitri Burago
Distinguished Professor of Mathematics
Honors Adviser

*Signatures are on file in the Schreyer Honors College.

Abstract

In recent years, focus has been placed on the development of key exchange protocols that could serve as post-quantum successors to existing cryptographic infrastructure. Some of these potential successors are based on the action of the class group of an order of an imaginary quadratic field on the set of $\mathbb{F}_p$-rational elliptic curves defined over $\mathbb{F}_p$. These protocols are thought to have the potential to become a suitable alternative to lattice-based protocols currently being standardized by NIST, the National Institute of Standards and Technology, which is holding a competition calling for the submission of new post-quantum cryptosystem candidates. In this thesis, I examine a promising post-quantum key exchange protocol, CSIDH, and a constant-time variant of the cryptosystem, CSIDH-CT, which has been developed to defend against a range of side-channel attacks to which CSIDH may be vulnerable. In order to compare the performance of CSIDH and CSIDH-CT, I present a programmatic implementation of each, along with the data resulting from testing them, reaching conclusions about their comparative performance in terms of computation time.

Table of Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank the following faculty members for their contribution to my study of mathematics, and for guiding me through my undergraduate career. They have all served to influence my path in some way throughout my four years at the Pennsylvania State University.

Steven Hair, for being a resourceful advisor throughout my eight semesters at Penn State, and especially for putting up with me as a freshman.

Spencer Dang, for talking with me about mathematics after class when I took MATH 251. I was new to the field and didn't know what kind of research I wanted to pursue, and Spencer helped me understand what kind of math there was and which interested me most.

Leonid Beryland, for inviting me to his research group on partial differential equations, machine learning, and epidemiology. Though I did not stay for long, as I realized that I did not want to perform research in mathematical biology, this was my first encounter with a research group in which graduate students participated. This was a meaningful experience for me because it gave me an idea of what research as a Ph.D. student might look like.

Jack Huizenga, for offering to let me take MATH 538: Commutative Algebra when he taught it. I didn't end up having time for it, but I appreciate his confidence in me, and he was very communicative with me about taking the course.

Ae Ja Yee, for offering to let me take MATH 536: Abstract Algebra as a sophomore. I ended up waiting another year to take the course, but Professor Yee's confidence in me and her willingness to let me enter her class at such an early stage in my academic career was kind and encouraging.

Linda Westrick, for allowing me to take MATH 558: Foundations of Mathematical Logic I in my junior year, which solidified my interest in logic and introduced me to graduate-level coursework.

Tim Reluga, for teaching me many things, which I will briefly summarize here. Firstly, for teaching me the importance of writing proofs that are both accurate and clear in all ways, including in handwriting, such as writing my u's and v's in a clearly distinguishable way. Secondly, for pushing me to go beyond what I was used to and work harder than I was comfortable with. Finally, for continuing to suggest I try to generate Mandelbrot and Julia sets in Python as a part of my project, eventually leading me to do so, starting my interest in mathematical programming and indirectly leading to my current hobbies and research and career interests.

Kris Jenssen, for teaching an interesting class in ODEs, and for writing a letter of recommendation for me for REU programs.

Jan Reimann, for having interesting discussions in MATH 457: Introduction to Mathematical Logic, both in class and over email, and for allowing me to take MATH 597: Reverse Mathematics, and providing the class in a very interesting and engaging format.

Naser Talebizadeh Sardari, for suggesting a research project I could work on that might even-

tually lead to my thesis, and for being understanding when I chose not to take the project.

Luen-Chau Li, for showing me that it's okay to fail at something and come back to it, and for teaching me that developing mathematical maturity and learning deeply matters more than grades or speed.

Jim Kowalski, for many things, from being a resourceful undergraduate advisor, to holding an independent study in number theory with me, to offering insightful life advice when I needed it.

Diane Henderson, for holding a fruitful independent study with me in numerical PDEs and MATLAB during the first semester of my junior year. This project, like that which I had completed as a part of class with Dr. Reluga, influenced the development of my interest in mathematical programming, and further exposed me to the pride and joy which can result from succeeding in writing a well-functioning program. I would like to thank Dr. Henderson not only for helping me learn so much about mathematical programming, but also for the life lesson she taught me during our last meeting together at the end of the Fall 2023 semester.

Next, I would like to give an extra special thanks to a few key people who have been responsible for influencing my path in a way incomparable to any other. Without any one of these individuals, I would not have been able to discover my interest in algebraic number theory, cryptography, or mathematical programming, and I would have been lacking essential guidance that made the process of completing my undergraduate years much easier and more rewarding.

Dmitri Burago, for being very helpful as my honors advisor, and for consistently being a positive influence on me throughout my studies. Dima has always answered my emails quickly, and has responded to personal, professional, and mathematical questions alike with precision and detail. Our meetings have always been fruitful, and I have always been confident that he would see the merit in my perspective and give me useful advice. I am glad to have been Dr. Burago's honors advisee.

Sean Hallgren, for the many things he has done to help me develop my interest and academic career in cryptography. It was through Dr. Hallgren's class on cryptography, CMPSC 497, that I learned how to think like a cryptographer, and through my honors project for the class that I first learned about the NIST standards and Kyber. This project also advanced my knowledge of and interest in mathematical programming, like the projects I had pursued before with Dr. Reluga and Dr. Henderson. I also had conversations with Dr. Hallgren about graduate programs, and he advised me both on how to write my personal statement and how to choose a Ph.D. program that was a good fit for my interests. Finally, he wrote a letter of recommendation for me for Ph.D. programs, which was essential to gaining admission to a top program.

Svetlana Katok, who was possibly the most important influence to me since the beginning of my career at Penn State. I met Dr. Katok on my first day of my undergraduate studies, in MATH 311W: Discrete Mathematics, which was one of my first university math courses. The lecture in our class on that first day was about propositional logic, just the basics that were required for any meaningful mathematics to be done. I remember walking up to the front of the class carrying a stack of books from the library in philosophy, including books on logic, to have a discussion about the concepts we had discussed in class. She referred me to Jan Reimann to study logic in more depth, which she would go on to do a few times in the future during our discussions of logic. Near the end of the semester, the students chose topics for group projects. I chose group theory, because I found groups the most interesting out of the content that had been taught. As a result, Dr. Katok suggested I take her class MATH 436: Abstract Algebra the following semester, citing her reason for suggesting it as being the fact that I enjoyed group theory. I didn't know what

abstract algebra was at the time, or how it was related to groups, but I took the course, and I'm very glad I did. It ended up being one of my favorite classes at Penn State, leading me to take three graduate-level courses in algebra and algebraic number theory and helping me decide my research topic for this thesis. Dr. Katok has also helped me in other ways, most notably offering letters of recommendation for REU programs and for the Schreyer Honors College, to which I was admitted and under which I am currently writing this thesis. It is thanks to her that I was able to discover my passion for abstract algebra, and also that I was able to enter the SHC. For this I extend my deepest gratitude, and wish Dr. Katok great success for the remainder of her career.

Wen-Ching "Winnie" Li, who has had an influence on my education similar to that of Dr. Katok, and has in some ways been even more impactful. Before the first semester of my junior year, I was signed up to take MATH 536: Abstract Algebra, the first graduate course in algebra, with John Lesieutre. However, the instructor was switched to Winnie, and so I took the class with her. Throughout the course, she was very encouraging to me, reassuring me when I was uncertain of my progress. At the end, she wrote a letter of recommendation for me for REU programs. A year later, during the Fall 2024 semester, I took MATH 567: Number Theory I with her, which is likely my favorite out of all the courses I have taken at Penn State so far. Within it I learned much of the knowledge required for me to complete this thesis, including the concepts of number fields, orders, the ideal class group, and much more. At the end of the course, when I was beginning to apply to Ph.D. programs, Winnie kindly wrote a letter of recommendation for me again, and she reviewed my personal statement a few times, giving careful advice on how I can improve it. In the end, her advice was very helpful to me, and I received offers from a few high-quality Ph.D. programs. I am very grateful for all that she has done for me, and I hope that her career continues to be as successful as it has been.

Finally, last but certainly not least, Kirsten Eisenträger, for everything she has done for me so far as my honors thesis advisor. When I first began to work with Kirsten, I knew almost nothing about cryptography, and I couldn't even recall the general formula for an elliptic curve. However, by the end of our first semester together, I had thoroughly learned the basics of the theory behind elliptic curves, and I had written my own implementation of elliptic curves over prime fields, which I was using in cryptography demonstrations. Throughout the next semester, we mostly focused on understanding CSIDH and related post-quantum protocols involving elliptic curves, the knowledge of which I used to develop the CSIDH implementations referenced in this thesis. She spent many weeks helping me understand the concepts and begin to write what would become sections of this thesis, as well as helping me to read papers when I found them challenging. Ultimately, this thesis would not have been written without the continuous support of Kirsten, which is why I have saved her paragraph for the last of this section.

Lastly, there are some special mentions to make. First, I would like to thank Fabio Campos for responding to my email about his paper, an analysis of which is the main content of this thesis. He corrected the code for the dummy isogenies which I was using, which made the data featured in this thesis possible. Next, I would like to acknowledge my friends at PSU, including the Math Club, for being good companions to me while I was here. Finally, I extend a special thanks to my family for all that they have done in helping me succeed throughout my undergraduate journey.

Chapter 1

Introduction

1.1 History of Public Key Cryptography

The security of cryptographic primitives relies on the assumption that certain mathematical problems are difficult to solve. Specifically, that they cannot be solved in polynomial time. Many prevalent cryptosystems today, such as RSA and Elliptic Curve Cryptography, base their security on either integer factorization or the discrete-log problem. However, these problems are not suitable for the post-quantum age, as there are quantum algorithms allowing for these problems to be solved in polynomial time [3].

Elliptic Curve Cryptography (ECC), one of the leading forms of public-key cryptography for security applications today, is an example of a cryptosystem relying on the hardness of computing discrete logarithms. The main scheme underlying ECC is Diffie-Hellman, which involves computing a shared secret point on an elliptic curve. It is particularly versatile because it allows for keys to be reused multiple times without compromising security, and it allows for a non-interactive key exchange. These traits are greatly responsible for its recent popularity, and it is desired that they be present in a successor.

As mentioned, the classically hard problem underlying ECC, the discrete log problem, is not considered hard for quantum computers [3]. There are polynomial-time quantum algorithms which, if implemented using a large-scale quantum computer, would pose a significant threat to existing cryptographic protocols, including ECC. While large-scale quantum computers currently do not exist, it is still necessary to start working on a post-quantum successor today. It will take decades to update existing infrastructure with new standards, and there's no time to waste in doing so, as current messages are already at stake. Any messages encrypted using current standards are susceptible to "harvest and decrypt attacks," in which an attacker stores current communications for the future, where they will presumably have access to an efficient quantum computer [4]. The need to quickly develop post-quantum cryptography has led the National Institute of Standards and Technology (NIST) to hold a competition for candidates, which was first announced in 2016 [4].

One submission to the NIST competition was based on isogeny-based cryptography, which relies on the computation of isogenies between elliptic curves over finite fields [1]. The first isogeny-based cryptosystem was proposed in 1997 by Stolbonov [5] [6], but it was not publicly known until 2006 [7]. The scheme uses the free and transitive action of the class group $\mathrm{Cl}(\mathcal{O})$ for an imaginary quadratic order $\mathcal{O}$ on the $\mathbb{F}_{p^m}$-isomorphism classes of ordinary elliptic curves over $\mathbb{F}_{p^m}$ with endomorphism ring isomorphic to $\mathcal{O}$. The scheme turned out to be too inefficient for practical use. [8].

Another scheme was proposed by Luca De Feo, David Jao, and Jérôme Plût as a potentially quantum-resistant public-key cryptosystems based on the difficulty of finding isogenies between supersingular elliptic curves [9]. The scheme was an analogue of the Diffie-Hellman key exchange, using isogenies of supersingular elliptic curves. It was called Supersingular Isogeny Diffie-Hellman (SIDH), and a protocol based on SIDH was submitted to the NIST competition. In 2022, a theoretical and practical break made it possible to break instances of cryptographic size in an hour, and the scheme was withdrawn from the competition. This created the need for an alternative cryptographic scheme based on isogenies that avoids these problems [10].

1.2 CSIDH

In 2018, the core idea of Couveignes' scheme was reimagined by Castryck *et al.* to use supersingular instead of ordinary curves, promising to overcome the efficiency issues in the original scheme [1]. Given a prime p and a supersingular curve E over $\mathbb{F}_p$, the $\mathbb{F}_p$-rational endomorphism ring of E over $\mathbb{F}_p$ satisfies

$$\mathrm{End}_p \cong \mathcal{O},$$

where $\mathcal{O}$ is an order in the quadratic field $\mathbb{Q}(\sqrt{-p})$. This fact is used to construct the group action underlying the protocol that ended up being called CSIDH: Commutative Supersingular Isogeny Diffie-Hellman. CSIDH can be more efficiently computed than Couveignes' original scheme, and thus allows for a faster key exchange, being a more promising post-quantum alternative to ECC.

Since the introduction of CSIDH, much work has been done in continuing its development. Later in 2018, a paper published by Michael Meyer, Fabio Campos, and Steffen Reith [2] offers an analysis of side-channel attacks that could be performed against an implementation of CSIDH. The paper presents a "constant-time" implementation of CSIDH, the computation time of which is claimed to be independent of the private key. This implementation is presented as a potential mitigation to the potential side-channel attacks mentioned in the paper.

1.3 Overview of Thesis

The goal of this thesis is to better understand how the performance of the "constant-time" variant of CSIDH proposed in [2] compares to standard CSIDH, presented in [1]. To do so, I have written implementations of both CSIDH and constant-time CSIDH, referred to as CSIDH-CT, in Python using the SageMath library, and I have measured the performance of each implementation for various values of the parameters.

This thesis is organized as follows. Chapter 2 is an overview of the mathematical background required to understand CSIDH, including sections on elliptic curves and the ideal class group. Chapter 3 gives an overview of the CSIDH protocol, including its security and the hard problem underlying it. The protocol and its parameters are defined, and the CSIDH group action is presented in the form of pseudo-code. The same is then done for CSIDH-CT. Chapter 4 explains the methodology of the tests performed on these implementations, including a link to the repository containing the code itself. The code used to compute the group action for each implementation is compared. Finally, an explanation of the tests is given, the resulting data is presented, and a section containing extra tables and graphs resulting from the computations is provided for further analysis.

Chapter 2

Mathematical Background

2.1 Elliptic Curves

Most material in this section can be understood by referring to chapters 1-6 of Washington's book on elliptic curves [11]. More information on the representation of $\text{End}_p(E)$ and its relationship to $\mathbb{Q}(\sqrt{-p})$ can be found in the original paper on CSIDH written by Castryck *et al.* [1]. Another resource is Joseph Silverman's book on elliptic curves [12].

2.1.1 Basic theory

Let E be a curve over a field K with $\text{char}(K) = p \neq 2, 3$. E is said to be an elliptic curve over K when it can be represented by an equation of the form

$$y^2 = x^3 + Ax + B$$

with coefficients in K, where the discriminant $\Delta = -16(4A^3 + 27B^2) \neq 0$. This equation is then said to be the simplified Weierstrass form of E. When E can be written in the form

$$y^2 = x^3 + Ax^2 + x$$

it is called a Montgomery curve, and this equation is referred to as the Montgomery form of the curve.

The set of rational points of E over K, denoted $E(K)$, forms a commutative group under the operation of point addition. For two points P and Q on E, the point $P+Q$ is defined geometrically as follows: the line passing through P and Q intersects E at a third point R. If $P = Q$, the line is taken to be the one tangent to P on the curve. The reflection of R across the x-axis is defined to be the sum of P and Q. This operation is well-defined when E is considered in the projective plane, where the point at infinity, denoted O, is the identity element of the group.

For a positive integer $n \neq p$, the n-torsion of the curve E is defined as the set of points

$$E[n] := \{P \in E(\overline{K}) : nP = O\},$$

defining nP as the sum of P with itself n times, with $0P = O$. It is known that, when $p \nmid n$,

$$E[n] \cong \mathbb{Z}/n\mathbb{Z} \oplus \mathbb{Z}/n\mathbb{Z}, \text{ [11]}$$

and when $p > 0$ and $p \mid n$ such that $n = p^r n'$, then

$$E[n] \cong \mathbb{Z}/n'\mathbb{Z} \oplus \mathbb{Z}/n'\mathbb{Z} \text{ or } \mathbb{Z}/n\mathbb{Z} \oplus \mathbb{Z}/n'\mathbb{Z}.$$

A curve E is called ordinary if $E[p] \cong \mathbb{Z}/p\mathbb{Z}$, and supersingular if $E[p] \cong 0$, denoting the group of one element. An equivalent definition of supersingularity relevant for defining the CSIDH group action is stated in terms of the properties of the Frobenius endomorphism, which will be discussed in the next section.

An isogeny between two elliptic curves E_1 and E_2 is a morphism of algebraic varieties

$$\phi : E_1 \to E_2$$

that is also a group homomorphism. Remarkably, isogenies preserve both the algebraic and geometric structure of an elliptic curve, making them useful in the development of cryptographic primitives.

2.1.2 Endomorphisms of an elliptic curve

An endomorphism of an elliptic curve E is an isogeny between the curve and itself, i.e. a map of the form

$$\phi : E \to E.$$

For the remainder of this paper, let E be an elliptic curve defined over the finite field $\mathbb{F}_p$ with $p \geq 5$ being a prime. The ring of endomorphisms of E rational over $\mathbb{F}_p$, also referred to as its $\mathbb{F}_p$-rational endomorphisms, is denoted $\text{End}_p(E)$.

A special endomorphism defined for every elliptic curve is derived from the map defined by

$$\begin{aligned} \pi : \overline{\mathbb{F}_p} &\to \overline{\mathbb{F}_p} \\ x &\mapsto x^p, \end{aligned}$$

where $\overline{\mathbb{F}_p}$ denotes the algebraic closure of $\mathbb{F}_p$. The map is referred to as the Frobenius map for $\mathbb{F}_p$. π acts on the points of $E(\overline{\mathbb{F}_p})$ through the Frobenius map on E, which is defined by

$$\pi(x, y) = (x^p, y^p),\ \pi(O) = O\,.$$

A very important fact about the Frobenius endomorphism π is that it satisfies the characteristic equation

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

in $\text{End}_p(E)$, where $t \in \mathbb{Z}$ is the trace of Frobenius [11, p.92].

An important fact about the Frobenius trace is its involvement in an equivalent condition for the supersingularity of an elliptic curve. E is supersingular if and only if $t = 0$, such that

$$\pi^2 + p = 0.$$

This means that $\pi = \pm\sqrt{-p}$. Furthermore, $\text{End}_p(E)$ is an order $\mathcal{O}$ in the quadratic field $k = \mathcal{O} \otimes_{\mathbb{Z}} \mathbb{Q} \cong \mathbb{Q}\left(\sqrt{t^2 - 4p}\right)$, which means that in the case of supersingularity, it holds that

$$\text{End}_p(E) \cong \mathbb{Q}\left(\sqrt{-4p}\right) \cong \mathbb{Q}\left(\sqrt{-p}\right) \cong \mathbb{Q}(\pm\pi).$$

This is therefore another equivalent condition for the supersingularity of E.

2.2 The Ideal Class Group

The class group of a quadratic number field acts on its orders in a way that is used in CSIDH. The material from this section can be found in Castryck's original paper [1], or in the paper on constant-time CSIDH by Meyer *et al.* [2]. Any necessary background on algebraic number theory is covered in Marcus' book Number Fields [13].

2.2.1 Fractional ideals and the class group

Let K be a number field, i.e. a finite-dimensional field extension of the field of rational numbers $\mathbb{Q}$, and let $\mathcal{O}_K$ denote its ring of integers, i.e. the maximal order of K. Let $\mathcal{O}$ be an order of K, i.e. a submodule of $\mathcal{O}_K$ that is also a subring and contains a $\mathbb{Q}$-basis of K.

A fractional ideal $\mathfrak{a}$ of $\mathcal{O}$ is a finitely-generated $\mathcal{O}$-submodule of the form αI, where I is an $\mathcal{O}$-ideal and $\alpha \in K^\times$. A fractional ideal $\mathfrak{a}$ is invertible if some $\mathfrak{b} = \mathfrak{a}^{-1}$ exists such that $\mathfrak{ab} = \mathcal{O}$. All principal ideals $\alpha\mathcal{O}$ are clearly invertible, as $\alpha \in K^\times$ ensures this.

The set of all invertible fractional ideals of $\mathcal{O}$ forms a commutative group $I(\mathcal{O})$ under ideal multiplication. Furthermore, it has a subgroup $P(\mathcal{O})$, which is defined as the set of all principal fractional ideals of $\mathcal{O}$ and is normal, since $I(\mathcal{O})$ is commutative. The class group of $\mathcal{O}$ is then defined as

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

Every class $[\mathfrak{a}] \in \mathrm{cl}(K)$ can be represented by an integer.

2.2.2 The class group action

An action of a group G on a set X is a map assigning to each $g \in G$ and $x \in X$ an element $g \cdot x \in X$, satisfying two conditions: that forall $x \in X$, $e \cdot x = x$, where e is the identity of G; and that $(gh) \cdot x = g \cdot (h \cdot x)$ for all $g, h \in G$ and all $x \in X$. A group action is said to be transitive if for all $x, y \in X$ there exists some $g \in G$ such that $g \cdot x = y$. Finally, the action is said to be free if only the identity fixes any point, i.e. $g \cdot x = x$ requires that $g = e$ for any $x \in X$.

In the case of an order $\mathcal{O}$ of an imaginary quadratic field, it is known that the class group $\mathrm{cl}(\mathcal{O})$ acts freely and transitively on the set

$$\mathcal{E}\ell\ell_p(\mathcal{O}, \pi)$$

defined as the set of isomorphism classes of elliptic curves E over $\mathbb{F}_p$ with $\mathrm{End}(E) \cong \mathcal{O}$ and the Frobenius map of E being equal to π. This will be explained in more detail later in this section.

Regarding such curves, an important result holds: for any curve $E/\mathbb{F}_p$ and a finite $\mathbb{F}_p$-rational subgroup $G \leq E/\mathbb{F}_p$, there is an elliptic curve $E'/\mathbb{F}_p$ and an isogeny $\phi : E \to E'$ which is separable, meaning ϕ can be expressed in terms of rational functions over $\mathbb{F}_p$, such that $\ker(\phi) = G$. This result guarantees that any subgroup of an elliptic curve E defines a unique isogeny to another curve in the same class, which is necessary to ensure that the CSIDH protocol, which will be discussed in detail later, is well-defined [2].

Now, recall that, for a supersingular curve E, it holds that

$$\pi^2 = -p,$$

where π is the Frobenius endomorphism. Given that $K = \mathbb{Q}(\sqrt{-p}) \cong \mathbb{Q}(\pi)$, it holds that $\mathcal{O}$ always contains π, and thus $\mathbb{Z}[\pi]$ as well, the order generated by the Frobenius endomorphism.

Any invertible ideal $\mathfrak{a}$ of $\mathcal{O}$ splits into a product of $\mathcal{O}$-ideals of the form $(\pi\mathcal{O})^r\mathfrak{a}$, such that one can define $E/\mathfrak{a}$, along with the isogeny $\phi_\mathfrak{a} : E \to E/\mathfrak{a}$. This is done by first constructing $\phi_\mathfrak{a}$ as an isogeny with E as its domain and $\bigcap_{\alpha\in\mathfrak{a}} \ker(\alpha)$ as its kernel, defining $E/\mathfrak{a}$ as the image. This is possible due to the separability of $\phi_\mathfrak{a}$ [1]. This, along with the previous results, leads to the conclusion that the group action on the set of elliptic $\mathbb{F}_p$-rational elliptic curves with coordinates

contained in $\mathcal{O}$ and endomorphisms generated by π is well defined. Specifically, we have the group action

$$\begin{aligned} \mathrm{cl}\left(\mathcal{O}\right) \times \mathcal{E}\ell\ell_p\left(\mathcal{O}, \pi\right) &\to \mathcal{E}\ell\ell\left(\mathcal{O}, \pi\right) \\ \left([\mathfrak{a}], E\right) &\mapsto E/\mathfrak{a}\,. \end{aligned}$$

When discussing the action of an ideal class $[\mathfrak{a}]$ on a curve E, producing $E/\mathfrak{a}$, it will be denoted $\mathfrak{a} * E = [\mathfrak{a}] = E/\mathfrak{a}$.

Chapter 3

Fault Injection Attacks and Constant-Time CSIDH

3.1 The CSIDH protocol

In this section, the logic of the CSIDH key exchange protocol will be introduced, and pseudo-code will be provided. The information in this section can be found in the paper by Castryck *et al.* from 2018 [1].

3.1.1 Introduction and security overview

The underlying math behind CSIDH is explained in the previous chapter, where the action of the ideal class group of an imaginary quadratic order generated by $\pm\pi$ on the set of $\mathbb{F}_p$-rational elliptic curves is shown to be well-defined. This group action, while it is relatively simple to compute, is quite computationally difficult to reverse, making it ideal for a cryptographic setting.

For a field k and a prime $\ell \nmid \text{char}(k)$, the k-rational ℓ-isogeny graph $G_{k,\ell}$ is defined as having all the elliptic curves defined over k as its vertices, and having a directed edge (E_1, E_2) for each k-rational ℓ-isogeny from E_1 to E_2. The supersingular isogeny path problem, stated simply, is to find a walk through the graph $G_{k,\ell}$ for a given ℓ. This is the hard problem underlying the CSIDH group action [1].

When E_1 and E_2 are supersingular, the isogeny graph is an expander graph, meaning random walks mix quickly [1]. However, since only $\mathbb{F}_p$-rational isogenies are considered, only a smaller subset of the graph is used, and its properties are not fully understood. It might not be fully connected, as demonstrated by an example of such a graph, which is given in Figure 3.1.

There is no known quantum algorithm with a speed advantage over classical algorithms, and while the best possible classical algorithms run in subexponential but superpolynomial time. Thus, it is thought that the ability to solve the problem–and thus to break a cryptographic protocol like CSIDH–will not be improved in the near future, and that such systems will remain secure against classical and quantum attacks [1].

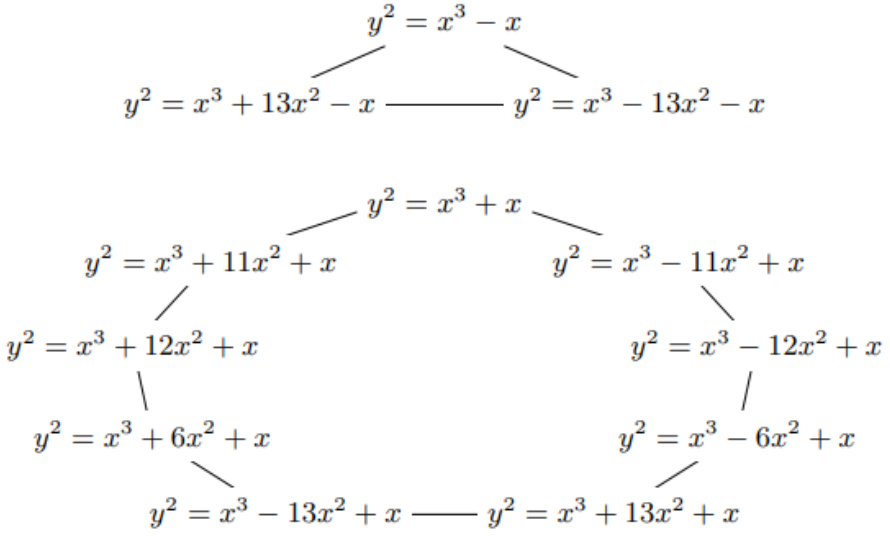

Figure 3.1: An example of the graph $G_{k,\ell}$ for $k = \mathbb{F}_{83}$ and $\ell = 3$. Taken from [1].

3.1.2 CSIDH key exchange

The CSIDH group action is applied in practice in the form of a non-interactive key exchange between two parties, which will be referred to as Alice and Bob. It will be assumed that both Alice and Bob are able to compute the CSIDH group action on an arbitrary curve. Specifically, given an ideal $\mathcal{A}$ and a curve E, each party is able to compute the action of an ideal class $[\mathcal{A}]$ on the curve E, producing the curve $E_A = E/\mathcal{A} = [\mathcal{A}]E$, for an ideal $\mathcal{A}$.

When the protocol is established, a security parameter n is selected as the number of small primes l_i to generate, which are selected such that the quantity $p = 4l_1 \cdots l_n - 1$ is prime. To do this, the first n odd primes are selected to be $l_1, \ldots, l_n$, and if p is not prime, then l_n is defined as the $(n+1)$th odd prime and then, if necessary, the $(n+2)$th, etc., until p is prime. Since $p \equiv 3 \mod 4$, the curve $E_0 : y^2 = x^3 + x$ defined over $\mathbb{F}_p$ is supersingular, so it has $\mathbb{F}_p$-rational endomorphism ring $\mathcal{O} = \mathbb{Z}[\pi]$, where π is defined to be the Frobenius map.

Alice and Bob each have a private key $(e_1, \ldots, e_n)$, where each e_i is an integer selected randomly from a range $\{-m, \ldots, m\}$ for a chosen integer m. In the standard implementation of CSIDH outlined in [1], $m = 5$ is taken. The integers e_i represent the ideal class $[\mathcal{A}] = [L_1^{e_1}] \cdots [L_n^{e_n}] = [L_1^{e_1} \cdots L_n^{e_n}] \in \mathrm{cl}(\mathcal{O})$, where $L_i = (l_i, \pi - 1)$. Each party has a public key equal to the Montgomery coefficient $A \in \mathbb{F}_p$ and $B \in \mathbb{F}_p$ of the curves $E_A = [\mathcal{A}]E_0 : y^2 = x^3 + Ax^2 + x$ and $E_B = [\mathcal{B}]E_0 : y^2 = x^3 + Bx^2 + x$ respectively, obtained by applying the action of $[\mathcal{A}]$ or $[\mathcal{B}]$ to the curve E_0, where $\mathcal{A}$ and $\mathcal{B}$ are the ideals resulting from each party applying their respective private exponents e_i to the ideals L_i and computing their product.

Alice and Bob compute their respective key pairs $([\mathcal{A}], A)$ and $([\mathcal{B}], B)$. They then send one another their public keys A and B. Alice verifies that the curve $E_B : y^2 = x^3 + Bx^2 + x$ is a valid curve in $\mathrm{Ell}_p(\mathcal{O}, \pi)$ and then applies the action of her secret key $[\mathcal{A}]$ to the curve to compute the shared secret $[\mathcal{A}]E_B = [\mathcal{A}][\mathcal{B}]E_0 = E_{AB}$. Bob performs a similar computation using Alice's public and his private key. Since $[\mathcal{A}][\mathcal{B}]E_0 = [\mathcal{B}][\mathcal{A}]E_0$, Alice and Bob compute the same shared secret, and can then use it for symmetric encryption. The key exchange is thus complete.

3.1.3 Outline of CSIDH implementation

An implementation of CSIDH involves applying the action of the ideal classes corresponding to the exponents in the private key in succession, resulting in the computation of the private curve of A or B respectively. This process is outlined in the following pseudo-code, which is given in [1]. Note that, in the figure, the integer m is referred to as as B, and the Montgomery coefficient is denoted a instead of A. This is consistent with the notation used in [1].

Algorithm 1: Evaluating the class group action.

Input : $a \in \mathbb{F}_p$ such that $E_a : y^2 = x^3 + ax^2 + x$ is supersingular, and a list of integers $(e_1, ..., e_n)$ with $e_i \in \{-B, ..., B\}$ for all $i \leq n$.

Output: $a' \in \mathbb{F}_p$, the curve parameter of the resulting curve $E_{a'}$.

```
1  while some e_i ≠ 0 do
2      Sample a random x ∈ F_p.
3      Set s ← +1 if x^3 + ax^2 + x is a square in F_p, else s ← −1.
4      Let S = {i | sign(e_i) = s}.
5      if S = ∅ then
6          Go to line 2.
7      P = (x : 1), k ← ∏_{i∈S} ℓ_i, P ← [(p + 1)/k]P.
8      foreach i ∈ S do
9          K ← [k/ℓ_i]P.
10         if K ≠ ∞ then
11             Compute a degree-ℓ_i isogeny φ : E_a → E_{a'} with ker(φ) = ⟨K⟩.
12             a ← a', P ← φ(P), k ← k/ℓ_i, e_i ← e_i − s.
```

Figure 3.2: Pseudo-code for an implementation of the CSIDH key exchange, from [1]

3.2 Fault Injection Attacks

3.2.1 Introduction

A fault injection attack is a method by which an attacker intentionally introduces errors into a hardware system in order to manipulate an "honest" party into revealing information about (otherwise secure) encryption [2]. In a paper by Michael Meyer, Favio Campos, and Steffen Reith, potential avenues by which fault injection attacks could be performed against an implementation of CSIDH and ways of mitigating them are discussed. The authors present recommendations for how to mitigate the potential attacks, including modifications to the key generation process and the group action.

The paper discusses three main leakage scenarios, all of which are considered in the variant of CSIDH presented by Meyer *et al*.

The first scenario is a timing leakage, i.e. a situation in which private information about the private key could be leaked by analyzing how much time it takes for the group action to be computed. In this case, an adversary could eliminate many possible keys with a great degree of certainty, challenging the security of CSIDH. The authors use as an example of timing leakage the fact that the key $(5, 5, 5, \ldots, 5)$ results in a longer group action computation than the key $(0, 0, 0, \ldots, 0)$, which requires no computations at all. If the key were one of these two tuples, an attacker would be able to determine which one it is by analyzing the time taken to compute the group action. While this is a stark example, it makes the point that the time taken to compute the group action depends on the key used, making a means of preventing such an attack desirable. In order to prevent the timing leakage scenario, the running time for the CSIDH group action should be independent of the private key entirely [2].

Next, the second scenario is one in which the attacker can gather information about the private key through power analysis. In this scenario, it is assumed that an adversary is capable of measuring the power consumption of the algorithm, and through this determine the memory blocks which represent point multiplication and isogeny computation. As a result of this, they would be able to separate the operations from one another. This allows them to determine which private key elements are of the same sign, as they are applied together. Such information could greatly decrease the size of the key space which would have to be considered, making it significantly easier to determine the shared secret by brute force. [2].

Finally, the third scenario is that of a cache timing attack. This is a type of side-channel attack in which an attack gains information about a system by measuring the amount of time taken by the algorithm to access data in the cache. The authors do not go into much detail about this kind of attack, and they state that their implementation already includes existing measures against such attacks [2].

3.2.2 Mitigations of the Attacks

Meyer *et al.* then go on to suggest various changes to CSIDH in order to mitigate the side channel attacks mentioned in the paper. The main mitigations mentioned are changing the scheme so that all integers in the exponent vector are positive, and implementing "dummy isogenies" to ensure the time taken to compute the group action is independent of the private key.

In order to prevent keys in which the sign distribution is unbalanced, in which case it would be possible to shrink the key space through time leakage and power analysis as discussed in the previous section, it is advised by the authors that purely non-negative keys be used. This would involve changing the range from which exponents are selected from $[-m, m]$ to $[0, 2m]$, e.g. that $[-5, 5]$ should become $[0, 10]$ in standard CSIDH. They warn that this would increase the runtime of CSIDH significantly, but that it would protect against these kinds of attacks. This version of the cryptosystem is technically equivalent to the original CSIDH, as the original keys with exponents $e_i \in [-5, 5]$ can be shifted by $+5$ to equal an equivalent key in the new system, and vice versa. When discussing constant-time CSIDH, m will refer to the upper bound of the range $[0, m]$ instead of that of $[-m, m]$.

Next, the authors describe the use of dummy isogenies, which perform the same computations as normal isogenies, but do not actually affect the curve parameters. The computation of a dummy isogeny involves performing the standard isogeny computation, but instead of keeping the resulting curve, discarding it. More detail on this process is given in the pseudo-code presented later in Figure 3.3.

In order to ensure that the running time is not dependent on the private key, a constant number of isogenies of each degree l_i should be computed. However, only e_i isogenies of degree l_i should be applied to the curve for each small prime l_i used to generate p, so $m - e_i$ dummy isogenies must be computed, where $m = 10$ is standard. This results in a variant of CSIDH that may be referred to as "constant-time," as the time taken to compute the group action is not dependent on the private key, since the same number of isogenies of each degree is always computed. Note that the implementation is not truly constant-time, as there is randomness in how the group action is applied, but this is not relevant to the attacks considered by Meyer *et al.*

The pseudo-code for the CSIDH group action including positive signs and dummy isogenies, again provided by Meyer *et al.*, is provided below, in order to elucidate the process by which this mitigation would be implemented. The notation in this figure is similar to that of Figure 3.2.

Algorithm 2: Constant-time evaluation of the class group action in CSIDH-512.

Input : $a \in \mathbb{F}_p$ such that $E_a : y^2 = x^3 + ax^2 + x$ is supersingular, and a list of integers $(e_1, ..., e_n)$ with $e_i \in \{0, 1, .., 10\}$ for all $i \leq n$.

Output: $a' \in \mathbb{F}_p$, the curve parameter of the resulting curve $E_{a'}$.

```
Initialize k = 4, e = (e_1, ..., e_n) and f = (f_1, ..., f_n), where f_i = 10 − e_i.
while some e_i ≠ 0 or f_i ≠ 0 do
    Sample random values x ∈ F_p until we have some x where x^3 + ax^2 + x is
     a square in F_p.
    Set P = (x : 1), P ← [k]P, S = {i | e_i ≠ 0 or f_i ≠ 0}.
    foreach i ∈ S do
        Let m = ∏_{j∈S, j>i} ℓ_i.
        Set K ← [m]P.
        if K ≠ ∞ then
            if e_i ≠ 0 then
                Compute a degree-ℓ_i isogeny φ : E_a → E_{a'} with ker(φ) = ⟨K⟩.
                a ← a', P ← φ(P), e_i ← e_i − 1.
            else
                Compute a degree-ℓ_i dummy isogeny:
                a ← a, P ← [ℓ_i]P, f_i ← f_i − 1.
            if e_i = 0 and f_i = 0 then
                Set k ← k · ℓ_i.
```

Figure 3.3: Pseudo-code for an implementation of the "constant-time" CSIDH group action, from [2]

Chapter 4

Comparing Implementations of Standard and Constant-Time CSIDH

4.1 Implementation

To compare the efficiency of standard CSIDH with the constant-time variant proposed by Meyer *et al.*, I have written an implementation of each, with the goal of running tests to compare their time consumption. Included in the testing and comparison is the generation of exponential approximations for the time complexity of each implementation, measured against the number n of small primes l_i used to compute the prime p defining the base field $\mathbb{F}_p$ over which the group action is computed.

4.1.1 Notes on Implementation

The implementations of CSIDH and constant-time CSIDH, which will be referred to as CSIDH-CT, were written in Python using the SageMath package. Both programs share the same methods for parameter generation, and make use of methods to convert to and from the Weierstrass form of the curve, i.e. converting from the standard Weierstrass form

$$y^2 = x^3 + ax + b$$

to the equation used for Montgomery curves, where the public key is the coefficient A, given by an equation of the form

$$y^2 = x^3 + Ax^2 + x.$$

Minor changes to the key generation method are made between CSIDH and CSIDH-CT, in order to account for the values of e_i being entirely positive in CSIDH-CT.

The most major changes between the two implementations is in the method used to compute the group action. In both implementations, a while True loop is used to apply each isogeny individually. The main difference is that, in CSIDH-CT, the computation of dummy isogenies is included, and the computations of real and dummy isogenies are performed within the same loop to defend against power analysis attacks. More details on the implementation of each protocol are offered in the following subsection.

All code referenced in this section is available in the GitHub repository corresponding to this thesis: `https://github.com/quaternyan/csidh-analysis`, which I maintain. Whenever I refer to a file or directory, it is referring to the contents of this repository.

4.1.2 Implementations of CSIDH and CSIDH-CT

My implementation of standard CSIDH is labeled 'csidh.py', and is based on pseudo-code listed in 3.2. The isogenies are applied in a loop, with positive and negative exponents being applied to the curve in separate iterations until no isogenies remain to be applied.

The implementation is contained in the 'CSIDH' class. Its "__init__" method takes a single input, the value of n, and defines the parameters of the curve, including the array of small primes 'l_primes', the prime 'p', the object 'F' representing $\mathbb{F}_p$, and the key tuples for Alice and Bob respectively. These keys are generated by the 'gen_key' method, which takes in the parameter defining the exponent range, m, which is hard-coded as 5.

The code for the group action itself very closely follows the pseudo-code presented in Figure 3.2. The exponents are stored in the list 'e_list', and a 'while True' loop computes the isogenies as depicted in the pseudo-code.

The first major difference with CSIDH-CT is that all exponents are positive, as shown in Figure 3.3. This changes the structure of the code leading up to the loop applying isogenies, making the set S in Figure 3.2 containing indices corresponding to exponents with the same sign unnecessary. Furthermore, since dummy isogenies must be applied in the same loop in order to mitigate the potential power analysis attacks mentioned in Section 3.2.1. This requires, in addition to the list of exponents 'e_list', a list of "dummy exponents" to decrease when dummy isogenies are applied, which is the function of the 'f_list'. This is done so that $e_i + f_i = m = 10$ for each index i in the list.

The loop used to used to apply the isogenies corresponding to each exponent in CSIDH is as follows. For definitions of the dummy points, see the pseudo-code in Figure 3.2.

```
# Apply isogeny corresponding to each l_prime power to compute E_B
for i in S:
    assert k % l_primes[i] == 0
    R = (k // l_primes[i]) * Q
    if R.is_zero():
        continue
    phi = E.isogeny(R)
    E = phi.codomain()
    Q = phi(Q)
    assert k % l_primes[i] == 0
    k = k // l_primes[i]
    e_list[i] -= s
    if s == -1:
        E = E.quadratic_twist()
    return self.convert_from_Weierstrass(E)
```

For comparison, the following code is used for the isogeny loop in 'csidh_ct.py' for CSIDH-CT.

```
# Apply real and/or dummy isogenies in the same loop
if not K.is_zero():
    if e_list[i] != 0:
        phi = E.isogeny(K)
        E = phi.codomain()
        P = phi(P)
        Discard = l_primes[i]*P
        e_list[i] -= 1
    else:
        phi = E.isogeny(K)
        Discard = phi.codomain()
        Discard = phi(P)
        P = l_primes[i]*P
        f_list[i] -= 1
    if e_list[i] == 0 and f_list[i] == 0:
        k = k * l_primes[i]
```

```
# Convert back from Weierstrass form
return self.convert_from_Weierstrass(E)
```

4.2 Tests and Results

4.2.1 Overview of testing methodology

The goal of testing my implementations of CSIDH and CSIDH-CT is to compare their performance in terms of time efficiency. It is known that the CSIDH-CT group action takes longer to compute due to the presence of dummy isogenies, and I am interested in knowing how much longer it takes for my implementation.

The main test performed is 'performance_comparison.py', which is in the 'Tests' directory. It tests the running time of the CSIDH group action for the 'csidh.py' and 'csidh_ct.py' programs for various values of n, labeled 'N' in the program, which is the number of primes used to compute p in the definition of the protocol. It does so by going through a list of n-values, labeled 'N_values', testing both CSIDH and CSIDH-CT for each value a number of times, then averaging the results together for each protocol and each value of N to ensure accuracy. The result of the test is two sets of values, containing the average running time for CSIDH and CSIDH-CT respectively for each value of N tested. This data is then used to generate an exponential approximation of the running time for each group action wrt the value of N, which is finally plotted along with the data itself.

The program allows for the values 'ivl', 'itr', 'N_start', and 'N_stop' to be set by the user, which correspond to the interval between values of n tested, the number of iterations averaged together for each, the first value of N to test, and the final value of N to test, respectively. The default number of iterations is set to 30 so that the Central Limit Theorem applies, meaning that, for a given value of N, the resulting set of running times for either of the protocols corresponding to that value can be expected to follow a normal distribution. This allows for a rigorous comparison of the running times with results that are very unlikely to be biased by randomness.

4.2.2 Comparison of performance

The test program was run with 'N_start=5', 'N_stop=100', 'ivl=5' and 'itr=30', resulting in the generation of data for values of N equal to all multiples of 5 between 5 and 100. This is comparable to the size of the primes used in current implementations of CSIDH, which range from $n = 74$ to $n = 101$. The data from this test can be seen below.

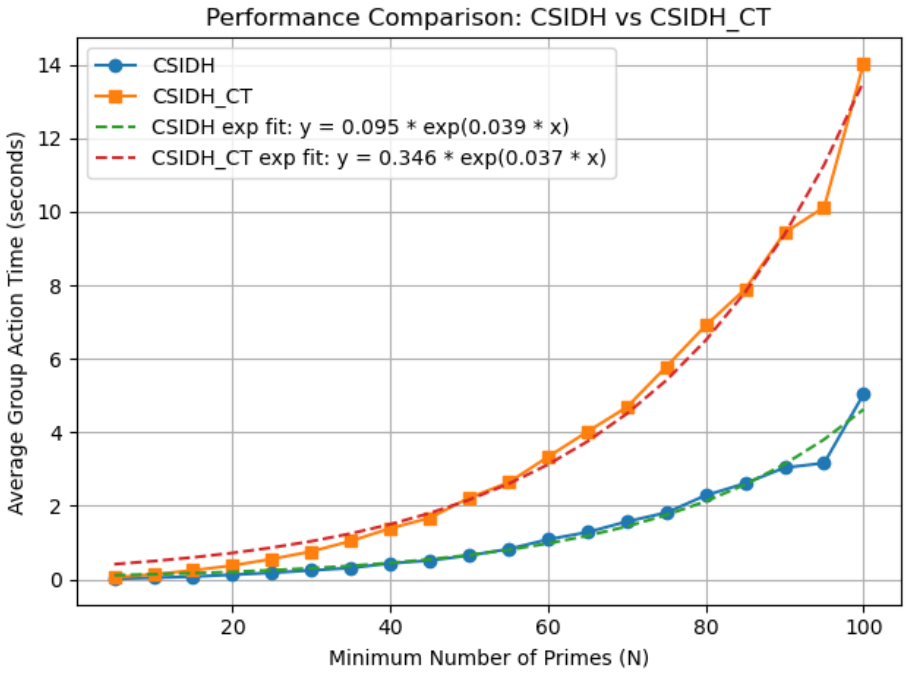

Figure 4.1: Results of the performance test on CSIDH and CSIDH-CT for $N = 5$ to $N = 100$, averaged over 30 iterations of each.

As reflected by the plot, the exponential growth of CSIDH and CSIDH-CT are about the same. However, CSIDH-CT, labeled 'CSIDH_CT' in the plot, takes about $\frac{0.346}{0.095} \approx 3.6$ times longer to compute the group action than CSIDH does. This is consistent with the fact that making all private key exponents positive and performing dummy isogenies significantly increases the time taken to compute the group action [2].

It can also be seen in the data that there is an anomaly at $N = 95$, where both the CSIDH and CSIDH-CT running times fall significantly below the exponential model of their growth. This may indicate that the properties of the class group corresponding to $N = 95$ make it significantly easier to compute, and the fact that a proportional gap is present in the results for both CSIDH and CSIDH-CT indicates that it is not likely a result of using dummy isogenies or positive exponents. It may be fruitful to investigate this anomaly in greater detail in future work, in order to determine which class groups offer easier computation.

In addition to this plot, data was collected for all values of N considered to correspond to the levels of security proposed by NIST. These values are $N = 74$, $N = 85$, and $N = 101$, approximately corresponding to NIST security level 1, level 2, and level 3 respectively [1]. Each of these values was computed with 50 iterations averaged together, resulting in the following table.

NIST Level	N (min # of primes)	CSIDH Avg Time (s)	CSIDH-CT Avg Time (s)
Level 1	74	1.88773	5.63833
Level 2	85	2.61493	7.89714
Level 3	101	10.99087	32.68741

Table 4.1: Performance Comparison of CSIDH and CSIDH_CT at NIST Levels 1, 2, and 3

As can be seen from the table, CSIDH-CT once again takes over 3 times as long to perform the group action. This is consistent with the data displayed in the plot resulting from the main test.

4.3 Additional Data

In order to generate the plot displayed in Figure 4.1, a large amount of data for CSIDH and CSIDH-CT was computed. The data is included in the following tables, which represent the results of testing performed on my implementations of the two protocols.

N	CSIDH Avg Time (s)	CSIDH-CT Avg Time (s)
5	0.01918	0.05184
10	0.04755	0.14231
15	0.07760	0.24653
20	0.12868	0.38055
25	0.18109	0.55686
30	0.24779	0.76210
35	0.31278	1.04565
40	0.43853	1.39165
45	0.51520	1.67299
50	0.65324	2.21943
55	0.82678	2.63936
60	1.08940	3.33496
65	1.28569	4.02046
70	1.57513	4.68912
75	1.81827	5.78858
80	2.28715	6.91097
85	2.61493	7.89714
90	3.04169	9.43396
95	3.16423	10.12073
100	5.04018	14.02915

Table 4.2: Combined Average Times for CSIDH and CSIDH-CT from $n = 5$ to $n = 100$, with intervals of 5, each value representing an average over 30 iterations with a given value of n.

To further illuminate the process of producing a single average time from 30 iterations, and to show how much variance exists for a given value of n for CSIDH and CSIDH-CT respectively, the following tables contain data for a few single values of n. Specifically, for $n = 20$, $n = 40$, $n = 60$, $n = 80$, and $n = 100$, a range which represents the bulk of the data collected. Note that the individual times are stored in more significant figures than the average times, such that the averages have high accuracy.

Iteration	CSIDH Time (s)	CSIDH-CT Time (s)
1	0.134120345115661620	0.441130876541137700
2	0.115742206573486330	0.373944878578186040
3	0.120429515838623050	0.424205780029296900
4	0.105661273002624510	0.370450615882873540
5	0.189961910247802730	0.375123500823974600
6	0.134049296379089360	0.366574287414550800
7	0.122961759567260740	0.361855626106262200
8	0.102064371109008790	0.371255755424499510
9	0.103600621223449710	0.359922289848327640
10	0.125799417495727540	0.373545765876770000
11	0.113211989402771000	0.364358901977539060
12	0.105194568634033200	0.364394664764404300
13	0.120184421539306640	0.364479422569274900
14	0.200868129730224600	0.365383744239807130
15	0.118139028549194340	0.366848707199096700
16	0.109076261520385740	0.362341523170471200
17	0.116776347160339360	0.364195108413696300
18	0.098758816719055180	0.373784780502319340
19	0.197428464889526370	0.374407529830932600
20	0.128149271011352540	0.352147817611694340
21	0.120160579681396480	0.362357378005981450
22	0.143779754638671880	0.423596620559692400
23	0.196123123168945300	0.362093448638916000
24	0.124801278114318850	0.417828559875488280
25	0.138739705085754400	0.373302459716796900
26	0.114483594894409180	0.423043608665466300
27	0.116398692131042480	0.364571094512939450
28	0.121268033981323240	0.421978354454040500
29	0.120241761207580570	0.369580030441284200
30	0.102315306663513180	0.427866458892822270

Table 4.3: Individual iteration times for $n = 20$

Iteration	CSIDH Time (s)	CSIDH-CT Time (s)
1	0.413218498229980470	1.294372200965881300
2	0.444556474685668950	1.319734692573547400
3	0.397571206092834500	1.426985502243042000
4	0.420539975166320800	1.439398288726806600
5	0.501547098159790000	1.586625814437866200
6	0.401395916938781740	1.604522943496704000
7	0.478752970695495600	1.498658657073974600
8	0.436891317367553700	1.548980116844177200
9	0.397192001342773440	1.412002921104431200
10	0.394228696823120100	1.357637286186218300
11	0.395265579223632800	1.410754442214965800
12	0.371790885925292970	1.424215912818908700
13	0.433698177337646500	1.395554065704345700
14	0.471700668334960940	1.318419575691223100
15	0.413240075111389160	1.306871771812439000
16	0.533102869987487800	1.375724434852600000
17	0.397401452064514160	1.400605916976928700
18	0.547206521034240700	1.398145675659179700
19	0.454171419143676760	1.388642430305481000
20	0.511899948120117200	1.300016403198242200
21	0.419301986694335940	1.302551507949829000
22	0.425954103469848630	1.394007563591003400
23	0.386811017990112300	1.388738512992858900
24	0.374574780464172360	1.380706310272216800
25	0.502562880516052200	1.395047426223754900
26	0.381205439567565900	1.296916246414184600
27	0.478703975677490230	1.297587156295776400
28	0.459436178207397460	1.303907155990600600
29	0.505125403404235800	1.390746235847473100
30	0.406909346580505370	1.391284942626953100

Table 4.4: Individual iteration times for $n = 40$

Iteration	CSIDH Time (s)	CSIDH-CT Time (s)
1	1.102036237716674800	3.326190471649170000
2	1.012094378471374500	3.347107887268066400
3	0.975169062614440900	3.350012898445129400
4	1.226460456848144500	3.329605340957641600
5	1.218529343605041500	3.328133344650268600
6	0.993739962577819800	3.329089999198913600
7	1.111643671989441000	3.343518376350403000
8	1.080459594726562500	3.325138568878174000
9	1.196164846420288000	3.312317967414856000
10	1.043657779693603500	3.334645509719848600
11	0.979645252227783200	3.339659094810486000
12	1.074415087699890100	3.339296936988830600
13	1.204210162162780800	3.345960855484009000
14	1.226502537727356000	3.331100702285766600
15	0.937474489212036100	3.337764143943786600
16	1.035789370536804200	3.330418109893799000
17	1.237427711486816400	3.345390319824218800
18	1.199827790260315000	3.330029129981994600
19	0.970144510269165000	3.326116681098938000
20	1.001343369483947800	3.317159056663513000
21	1.088916063308715800	3.331795096397400000
22	1.009963870048523000	3.339965820312500000
23	1.039611458778381300	3.339375495910644500
24	1.045464158058166500	3.338224530220031700
25	1.280127406120300300	3.319593310356140000
26	1.166763544082641600	3.351613879203796400
27	0.943779587745666500	3.338709950447082500
28	1.076067209243774400	3.334973692893982000
29	1.063079953193664600	3.348597526550293000
30	1.141636967658996600	3.337410569190979000

Table 4.5: Individual iteration times for $n = 60$

Iteration	CSIDH Time (s)	CSIDH-CT Time (s)
1	2.387087583541870000	7.046096682548523000
2	2.076478123664856000	7.026163339614868000
3	2.026831030845642000	6.732890129089355500
4	2.432551264762878400	7.033457756042480500
5	2.436481833457946800	7.013224244117737000
6	2.320936799049377400	7.049929022789001500
7	2.202043890953064000	6.707639098167419000
8	2.485967397689819300	6.737827181816101000
9	2.495973348617553700	6.701441287994385000
10	2.396632671356201000	7.045970320701599000
11	1.937731266021728500	7.027782201766968000
12	2.195520520210266000	7.049672007560730000
13	2.477446079254150400	6.715247869491577000
14	2.392145752906799300	6.728513598442078000
15	2.241334915161133000	7.052767157554626500
16	1.980015158653259300	7.032373309135437000
17	2.392193436622619600	7.027636051177978500
18	2.567704319953918500	6.815943837165832500
19	2.178929209709167500	6.711654067039490000
20	2.057440280914306600	7.024302721023560000
21	2.457152128219604500	7.028144955635071000
22	2.366315960884094200	6.730992317199707000
23	2.293629288673401000	6.708661317825317000
24	2.134127616882324000	7.023148894309998000
25	2.074648857116699000	7.034745931625366000
26	2.468082547187805000	7.030208468437195000
27	2.452095866203308000	6.695259571075439500
28	2.091017842292785600	6.709697365760803000
29	2.163532972335815400	7.056498765945435000
30	2.432315707206726000	7.031317710876465000

Table 4.6: Individual iteration times for $n = 80$

Iteration	CSIDH Time (s)	CSIDH-CT Time (s)
1	5.078094124794006000	13.450518012046814000
2	5.043377399444580000	13.484892249107360000
3	5.284787535667419000	13.515259385108948000
4	4.975950360298157000	13.468243241310120000
5	5.109407663345337000	13.442924022674560000
6	4.991971254348755000	13.416416168212890000
7	5.187807798385620000	13.491970658302307000
8	5.045115590095520000	13.485083580017090000
9	4.990854859352112000	13.475697040557861000
10	4.947780489921570000	13.425153017044067000
11	5.144143223762512000	13.562234163284302000
12	4.954221129417419000	13.493811249732971000
13	5.032445788383484000	13.429169654846191000
14	5.286386966705322000	13.460463285446167000
15	4.023810982704163000	13.502463698387146000
16	5.232809782028198000	13.423568606376648000
17	4.695528864860535000	13.445865273475647000
18	4.835403323173523000	13.465720057487488000
19	5.024602651596069000	13.449262022972107000
20	5.182280182838440000	12.748421669006348000
21	5.363870024681091000	14.226853251457214000
22	5.068773269653320000	15.333666086196900000
23	5.074607014656067000	14.604440689086914000
24	5.064419269561768000	15.313353300094604000
25	5.256307125091553000	15.889300107955933000
26	5.517276048660278000	16.106909394264220000
27	4.938627123832703000	14.741169452667236000
28	4.991465210914612000	14.732825040817260000
29	4.950374007225037000	15.685684561729431000
30	4.912905335426331000	15.603052616119385000

Table 4.7: Individual iteration times for $n = 100$

The following graphs are histogram plots of the data for each of these four values of n, including the mean standard deviation corresponding to each, along with a bell curve approximation of the resulting normal distribution. The goal, like with the previous tables, is to offer insight into the distribution of times for both CSIDH and CSIDH-CT, and to compare them.

In these graphs, the vertical axis, "density," represents the normalized proportion of the data that falls within a given range or is expected to be at a certain value. This is necessary so that the area under the histogram and the bell curve are equal to 1, such that the data can directly be compared to a continuous probability distribution.

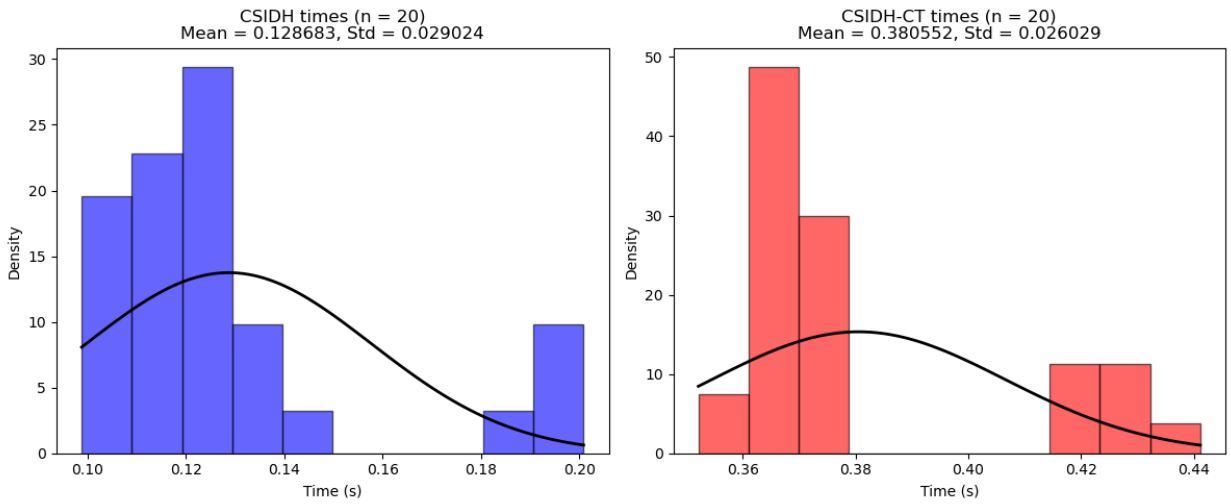

Figure 4.2: Comparison of distributions for $n = 20$.

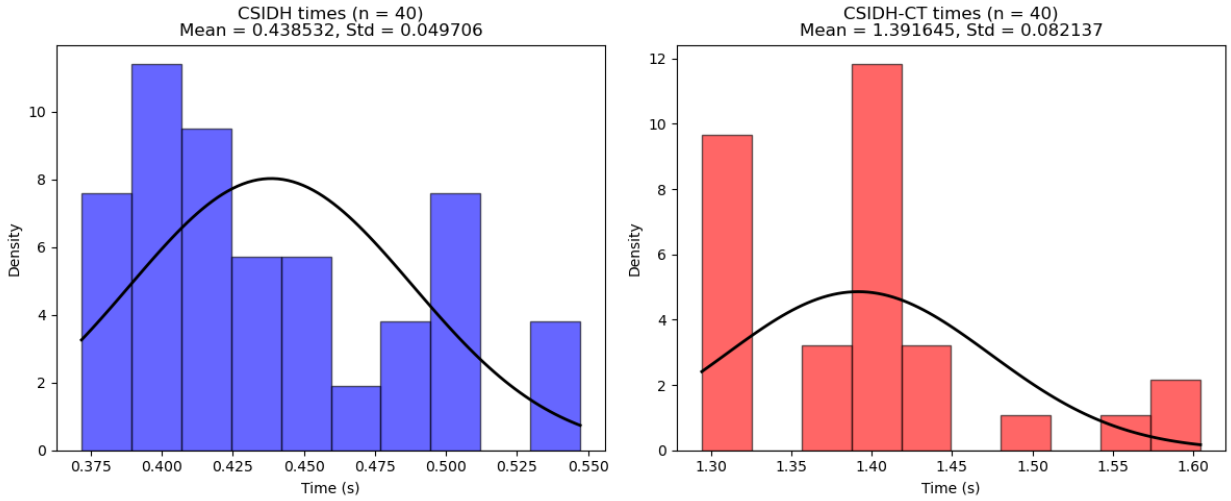

Figure 4.3: Comparison of distributions for $n = 40$.

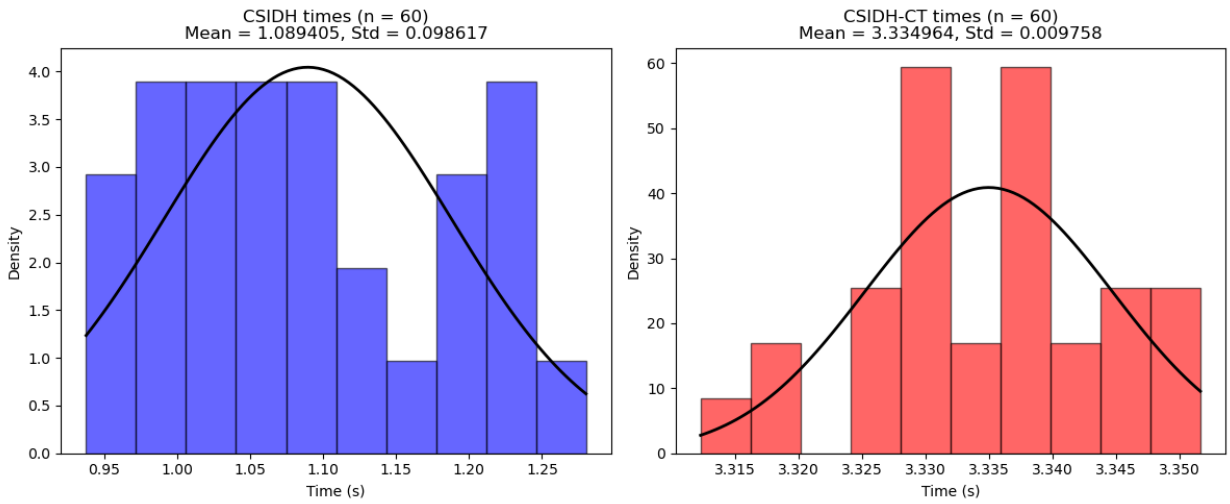

Figure 4.4: Comparison of distributions for $n = 60$.

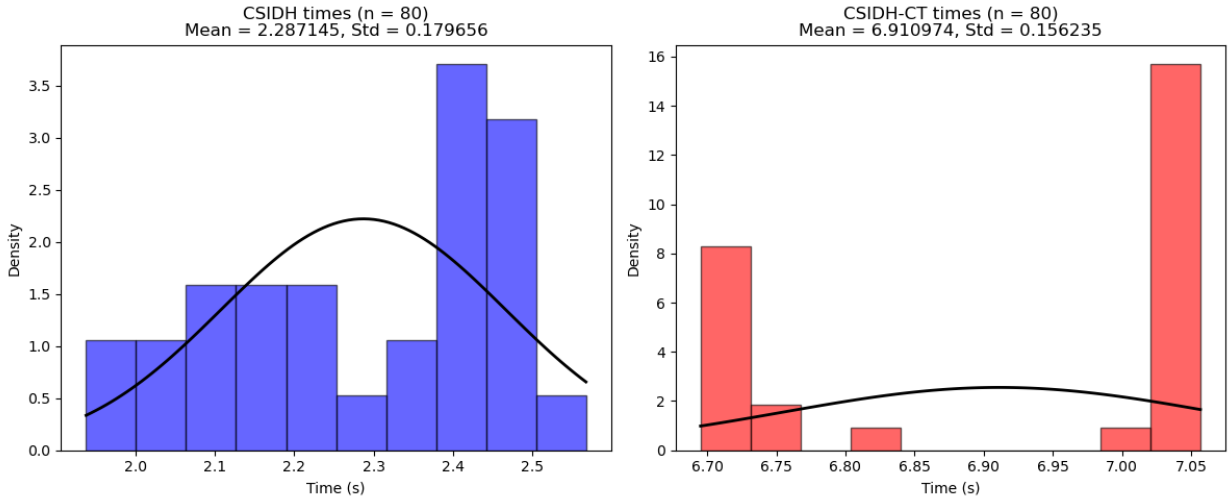

Figure 4.5: Comparison of distributions for $n = 80$.

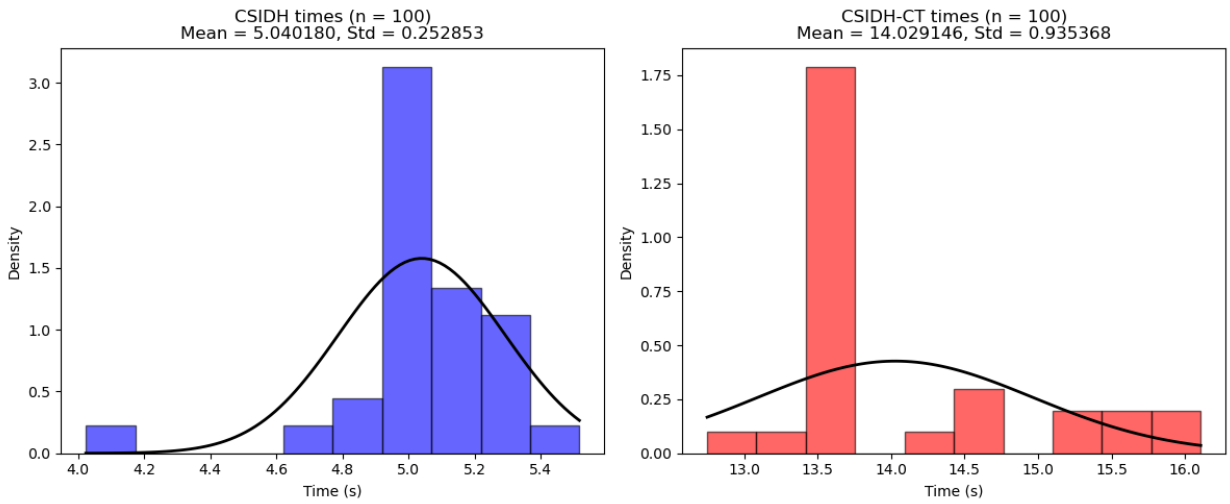

Figure 4.6: Comparison of distributions for $n = 100$.

Bibliography

[1] Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. CSIDH: An efficient post-quantum commutative group action. In *Advances in Cryptology – ASIACRYPT 2018*, volume 11274 of *Lecture Notes in Computer Science*, pages 395–427. Springer, 2018.

[2] Michael Meyer, Fabio Campos, and Steffen Reith. On lions and elligators: An efficient constant-time implementation of CSIDH. Technical Report 2018/1198, Cryptology ePrint Archive, 2018.

[3] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, October 1997.

[4] National Institute of Standards and Technology. What is post-quantum cryptography?, 2024. Accessed: 2025-03-28.

[5] Anton Stolbunov. Public-key encryption based on cycles of isogenous elliptic curves. Master's thesis, Saint-Petersburg State Polytechnical University, 2004. In Russian.

[6] Anton Stolbunov. Constructing public-key cryptographic schemes based on class group action on a set of isogenous elliptic curves. *Advances in Mathematics of Communications*, 4(2):215–235, 2010.

[7] Jean-Marc Couveignes. Hard homogeneous spaces. Technical Report 2006/291, Cryptology ePrint Archive, 2006. `https://eprint.iacr.org/2006/291.pdf`.

[8] Luciano Maino, Marzio Mula, and Federico Pintore. A review of mathematical and computational aspects of CSIDH algorithms. *Journal of Algebra and Its Applications*, 23(7):2530002, 2024.

[9] David Jao and Luca De Feo. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In *PQCrypto*, volume 7071 of *Lecture Notes in Computer Science*, pages 19–34. Springer, 2011.

[10] Wouter Castryck and Thomas Decru. An efficient key recovery attack on SIDH. *Cryptology ePrint Archive*, 2022:975, 2022.

[11] Lawrence C. Washington. *Elliptic Curves: Number Theory and Cryptography*. CRC Press LLC, 2nd edition, 2008. ProQuest Ebook Central, `https://ebookcentral.proquest.com/lib/pensu/detail.action?docID=359948`.

[12] Joseph H. Silverman. *The Arithmetic of Elliptic Curves*, volume 106 of *Graduate Texts in Mathematics*. Springer, 2nd edition, 2009.

[13] Daniel A. Marcus. *Number Fields*. Universitext. Springer, 1977.