University College London

Department of Computer Science

A thesis submitted in partial fulfilment of the requirements for the degree of Master of Science in Emerging Digital Technologies, University College London

Commutative Supersingular Isogeny-based Diffie-Hellman Group Action Through Lattice Reduction Algorithms

Author
Bryan Kumara

Academic Supervisor
Professor Philipp Jovanovic
Department of Computer Science
University College London

Secondary Supervisor
Ms. Maria Corte-Real Santos
Department of Computer Science
University College London

September 11, 2022

This dissertation is submitted as part requirement for the MSc in Emerging Digital Technologies degree at UCL. It is substantially the result of my own work except where explicitly indicated in the text.

Abstract

Quantum computers pose a challenge to public-key cryptography whose security is based on the integer factorization problem and the discrete logarithm problem. To defend against this threat, post-quantum cryptography seeks to find standards and mathematical problems that would be difficult to break even with a quantum computer.

One such approach involves the use of a mathematical structure called an isogeny which is a type of mapping over specific forms of elliptic curves. The Commutative Supersingular Isogeny-based Diffie Hellman (CSIDH) is an isogeny-based post-quantum cryptography approach that provides one of the smallest key sizes available, but comes at a very high computational cost.

Due to the recent development of this standard, there are plenty of open problems such as its exact security against quantum computers and existing libraries only support very limited features. This project uses the most commonly used parameter set of CSIDH available on Go and adds a key functionality that would allow converting a random sample from a modular arithmetic group to an exponent vector that serves as the secret key for CSIDH. This functionality is useful as it serves as the foundation of some digital signature schemes and secret sharing protocols using CSIDH without resorting to Fiat-Shamir aborts. This implementation is of interest as it utilizes lattices, another mathematical structure also relevant to another post-quantum cryptographic approach and examines different bases for lattice reductions needed to compute the exponent vector.

Our results show that this approach is inferior in terms of efficiency when compared to other implementations of CSIDH, a lattice-based digital signature, and current tasks involving elliptic curves. Despite the high computational cost, this feature is of interest in schemes that involve multiple parties as it necessary for some constructions since it is the only approach in CSIDH where the secret starts as an integer instead of an exponent vector needed for modular arithmetic, and would be vital in various implementations as current libraries in Go do not support this functionality.

Acknowledgments

I would like to thank Prof. Philipp Jovanovic and Ms. Maria Corte-Real Santos for their immense guidance and support throughout this project. Introducing me to post-quantum cryptography and helping me explore isogenies has been the academic highlight of my time at UCL and I hope to continue learning it after.

I would also like to thank my Father, Mother, and Brother for their constant encouragement and love. Thanks also to my friends in UCL and outside, whose kind words helped me trudge through the finish line.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

With the development of quantum computing, some widely used cryptographic protocols and primitives may be broken due to the computational advantages quantum computers. While efforts to find suitable replacements are underway, such as a public competition run by the US-based National Institute of Standards and Technology (NIST), potential standards still require further testing and evaluation [44].

Generally, currently used public-key cryptographic protocols, such as the RSA and ElGamal, are based on the hardness of the Integer Factorization or the Discrete Logarithm Problem respectively. Mathematical problems for cryptography such as the aforementioned are considered theoretically secure if it takes more than polynomial time for an adversary to break. However, these two problems are solveable in polynomial time using a quantum computer. A quantum computer could utilize algorithms that involve quantum mechanics such as Shor's algorithm and Grover's algorithm to compute prime factorization and discrete logarithm in polynomial time while provide quadratic speed-up for search algorithms [72]. Thus, the task of post-quantum cryptography is to provide cryptographic tools that would be difficult to break even in the presence of quantum computers.

This thesis project will focus on the practicality of a post-quantum cryptographic standard by implementing a specific feature that is useful in other parts of cryptography which involves multiple parties and examines the suitability of currently used parameters with regards to performance speed of said standard.

1.2 Aims

The aim of this project is to chart the viability of post-quantum cryptography based on commutative supersingular isogenies as a drop-in replacement of some key exchange procedures which currently relies on the Discrete Logarithm Problem. We will examine

a particular parameter set and consider its performance alongside its security against the latest attacks.

This project's particular implementation is of interest since it allows for random sampling from a group and convert it into a group action in Go, which is a necessary feature for some cryptographic schemes that rely on isogenies such as digital signatures and secret sharing schemes. In addition, it approaches the computation of isogenies between elliptic curves through the lens of lattices, which is another vital mathematical structure used in lattice-based post-quantum cryptography and homomorphic encryption. With isogeny-based post-quantum cryptography generally having the shortest key size but being relatively slow compared to other approaches, it would be beneficial to see the impact of using lattice reduction algorithms on speed performance and examine the implication this has on isogeny-based post-quantum cryptography.

After implementing this post-quantum cryptographic standard using Go to complement a current cryptographic library for isogenies, we will collect its CPU performance metrics to assess the viability of lattice-based algorithms for isogeny computations.

1.3 Scope and Approach

Given the wide scope of the field, the project has narrowed it down to consider three different bases for a specific lattice-based algorithm [42]. Other viable bases and solvers of the closest vector problem have been left open for future work to integrate with existing CSIDH libraries. To maintain a feasible implementation of parameters, only the CSIDH-512 parameter set from the CIRCL library by Cloudflare is considered and other post-quantum approaches and CSIDH parameters were not included. For the latter, this is because the lattice reduction algorithm cannot be done for larger parameter sets presently as they require the computation of the structure of $Cl(\mathbb{O})$ (i.e computing $Cl(\mathbb{O}) \cong \mathbb{Z}/N\mathbb{Z}$) which is not presently feasible for larger parameters. All performance tests were carried out on Go as an effort to complement some computational features not covered in the CIRCL package for CSIDH-512 that would be of benefit for vital CSIDH-based cryptographic actions.

1.4 Structure of the Thesis

The rest of the thesis is organized as follows: Chapter 2 provides the necessary background to understand the basics of public-key cryptography, the threat of quantum computers, the approaches of post-quantum cryptography alongside the mathematical structures of isogenies and lattices involved in this project. Chapter 3 provides a comprehensive literature review relevant to the topic, with areas ranging from theoretical considerations to improving implementations. Chapter 4 details the main post-quantum cryptographic

standard used and shows a particular way of computing the secret key using lattice reduction algorithms. Chapter 5 sets out the parameters of the implementation and discusses the results. Lastly, Chapter 6 provides a conclusion to this thesis with suggestions for future paths that can be built from this research.

Chapter 2

Background

2.1 Cryptography

Cryptography, the study of secure communications, can be broadly divided into two categories: symmetric and asymmetric cryptography. In the former, both the sender and receiver share the same secret key. Some commonly used protocols for symmetric cryptography include Advanced Encryption Standard (AES) and Rivest Cipher 4 (RC4). A common issue of symmetric cryptography is that parties need to find a way to securely possess the same secret key. This problem as led to the development of asymmetric cryptography, and this thesis will focus solely on asymmetric cryptography.

Asymmetric cryptography, also known as public-key cryptography, is an approach where two different but mathematically related keys are used. One is a public key that can be shared, and the other is a private key that needs to be kept secret to ensure security of the scheme. The public key is generally used for encryption and the private key is used for decryption. This method was discovered in the 70's, and has since been widely used in cryptosystems such as the Rivest, Shamir, and Adleman (RSA) algorithm [38]. The security of public-key cryptography is based on the complexity of problems conjectured to be 'hard' [52]. Two well-known problems are the integer factorization problem used in RSA and the discrete logarithm problem used for the Diffie-Hellman key-exchange (DH).

Problems are considered hard if they cannot be solved 'efficiently', which generally means in polynomial time in the length for the input [52]. There are no known current proofs that demonstrate hardness, only reductions which roughly shows that systems are secure assuming that certain conjectured problems are indeed hard to solve. The two most vital hard problems in asymmetric cryptography are:

- Integer Factorization Problem: Given a composite number N, find two primes p and q such that $N = p \times q$.
- Discrete Logarithm Problem: Given a cyclic group G, a generator $g \in G$, and an

element $h \in G$, find an $x \in G$ such that $h = g^x$.

Currently, the quickest classical approach to solve the Integer Factorization Problem is the modified versions of the General Number Field Sieve which has a run time of $\mathcal{O}(exp((\sqrt[3]{\frac{64}{9}} + o(1))(\ln n)^{\frac{1}{3}}(\ln \ln n)))^{\frac{2}{3}}$ [53]. Similarly, the best classical solver for the Discrete Logarithm Problem is a modified version of the Special Number Field Sieve algorithm that has an exponential run time of $L_p(\frac{1}{3}, \frac{32}{9}^{\frac{1}{3}})^{1+o(1)}$ where $L_p(e, c) = exp((c \log p)^e (\log \log p)^{1-e})$ [29]. However, quantum computers would be able to solve these problems in polynomial time [72] and thus pose a threat to any cryptographic protocols based on these two problems. To understand the threat of quantum computers, the basics of quantum computing must first be covered.

2.2 Quantum Computing

In classical computing, the basic unit of information is a bit. A bit must strictly be either 1 or 0 but not both or a mix of them. In quantum computing, the basic unit of information is instead a qubit. Qubits have base states of $|1\rangle$ and $|0\rangle$. The key difference is that qubits allow for a coherent superposition of both states. A more in-depth explanation of superposition and quantum computing in general can be found in [62] but mathematically it can be best understood and represented as $|\Phi\rangle = \alpha|0\rangle + \beta|1\rangle$ where α and β are complex numbers. When measuring a qubit (what exactly a measurement entails is still an ongoing issue, and papers such as [14] gives a better overview on the topic), we will always get either 0 or 1 with probability $|\alpha|^2$ and $|\beta|^2$ respectively. This can be expressed as $|\alpha|^2 + |\beta|^2 = 1$.

Though it looks like we can store an infinite amount of data in a qubit, when a qubit is measured it only ever gives 0 or 1 with a certain probability. Measurement collapses superposition, but physical phenomenas such as entanglement are useable for parallel computation not possible on a classical computer. Quantum entanglement can be conceptualized as a phenomenon where the quantum sates of two or more particles can only be described with reference to each other regardless of the distance between them. Changes to one will immediately causes the properties of other particles to change. Utilizing superposition and entanglement is what gives rise to algorithms that can break cryptographic schemes based on the IFP and DLP.

Logic gates are gates that transform an input into an output. Classical gates operate on bits whilst quantum gates operates on qubits. Quantum gates uses superposition and entanglement, and entangled qubits as input remain entangled as output. One such example is a quantum NOT gate, where a qubit with state $|0\rangle$ becomes $|1\rangle$ while $|1\rangle$ becomes $|0\rangle$, and also swapping the probabilities of a qubit in superposition.

Other notable quantum gates include the controlled-NOT (CNOT) gate and the Hadamard gate. CNOT has two input qubits, one is the control qubit and the other is the target qubit. If the control qubit is 0, the target qubit is unchanged. If the control qubit is 1,

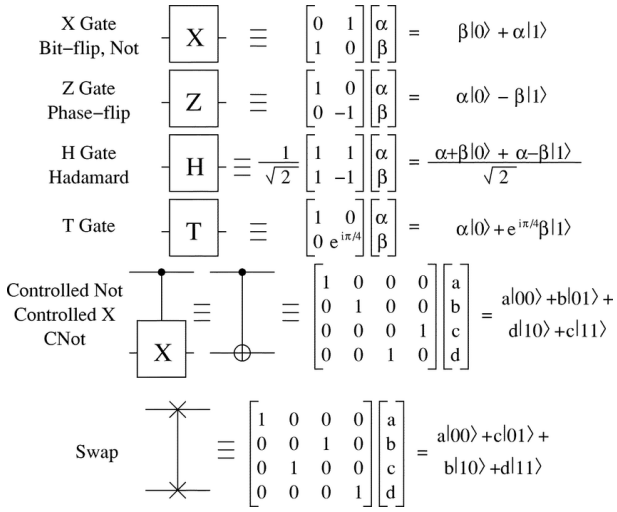

Figure 2.1: Several examples of quantum gates from Figure 1 in [19]

the target qubit is flipped. The Hadamard gate takes a single input qubit, turns $|0\rangle$ to $(|0\rangle+|1\rangle)/\sqrt{2}$, and turns $|1\rangle$ to $(|0\rangle-|1\rangle)/\sqrt{2}$. Conceptually, this gate turns basis state into an equal superposition state. These gates, seen in Figure 2.1, are vital to understanding how Shor's and Grover's algorithm work.

For Shor's Algorithm, given a composite number N, we do the following:

- Randomly pick $1 < a < N$. Then compute $K = gcd(a, N)$. If $K \neq 1$, then it is a non-trivial factor and we are done.
- If not, use the quantum period-finding subroutine of the following: $f(x) = a^x \bmod N$ and it returns r which is the smallest positive integer that satisfies $a^r \equiv 1 \bmod N$ [72].
- The quantum period-finding subroutine does the following: if r is odd and if $a^{\frac{r}{2}} = -1 \bmod N$, we restart the process.
- Otherwise, we get nontrivial factors of N as $gcd(a^{\frac{r}{2}} + 1, N)$ or $gcd(a^{\frac{r}{2}} - 1, N)$.

This algorithm runs in $\mathcal{O}(\log N)$ which is in polynomial time for the length of the input, and thus solves the Integer Factorization Problem efficiently.

In Grover's Algorithm, given a number N representing the size of a function f's domain and a specific element $w \in N$, we do the following:

- Initialize the system to a uniform superposition over all the states.
- Iterative search by applying quantum gates such as the Hadamard gate and the diffusion operator. The diffusion operator creates an inversion about the mean so that tiny differences in earlier steps can be made large enough to be measurable.

- We measure the resulting quantum state in the computational basis. If we choose our iterative search values carefully, there is a very high likelihood that we have found w.

The algorithm run time is $\mathcal{O}(\sqrt{N})$. However, the caveat here is that we have chosen our iterative search values carefully.

Thus we can see that with these two algorithms, theoretically the IFP and DLP would have their hardness assumptions broken by a quantum adversary. We can factor integers in polynomial time using Shor's algorithm and can modify this algorithm to solve the DLP in polynomial time as well [11]. Grover's algorithm can also utilize brute-force searches for keys of shorter length. While quantum computers currently do not have the scale of qubits to break current implementations of RSA and DH, they could do so in the future. With developments such as IBM presenting a 127-qubit microprocessor in 2021, traditional cryptographic is under an upcoming and realistic threat even if they are still years away [69].

2.3 Post-quantum Cryptography

Given the abilities of quantum computers to break widely-used cryptographic protocols, post-quantum cryptography is a research area that aims to find cryptographic protocols that would be secure even against an adversary with access to a quantum computer. The NIST is conducting a public competition for different post-quantum cryptography protocols. This competition contains submissions that have been scrutinized in multiple rounds, with current winners for public key encryption and digital signatures selected in 2022. Short-listed candidates have also been considered for additional testing in a fourth round and additional calls for new submissions have been made for future consideration.

There are several general approaches to post-quantum cryptography. These can be summarized by the following:

- Lattices: Lattice-based problems such as the Shortest Vector Problem (the problem of finding the shortest non-zero vector in a lattice) are conjectured to be difficult even for quantum computers. Lattices seem to be the leading approach in post-quantum cryptography as the NIST has selected CRYSTALS-KYBER as the winner for public-key encryption, with lattice-based digital signatures CRYSTALS-Dilithium and FALCON as 2 out of the 3 winners of the digital signatures category of the competition. Additionally, they are fast with short-ish key sizes. Lattices also feature in other areas of cryptography where the only known construction are based on lattices such as homomorphic encryption, indistinguishability obfuscation, and functional encryption.

- Multivariate: This approach of asymmetric cryptography uses multivariate polynomials such as $x^3 + 3x + 7$ which are defined over a finite field $\mathbb{F}$. These schemes rely on the hardness of solving systems of multivariate polynomial equations. Rainbow is a multivariate digital signature scheme that was considered as an NIST candidate, but was broken recently [5].

- Hash-based: Hash functions for digital signatures were invented in the 1970's. Despite their drawback of possessing a limit on the number of signatures that can be signed using a corresponding private key, they are conjectured to be resistant to attacks by quantum computers and have been around for some time so are more reliable. In the NIST competition, the hash-based signature scheme SPHINCS+ is the only non-lattice approach to be selected as a winner.

- Isogeny-based: This approach relies on supersingular elliptic curves and isogeny graphs. Whilst currently facing efficiency issues, this approach can preserve the forward secrecy property which insures against keys being compromised in the future which is a property not found in other approaches. With this approach being very recent, there has been limited time to provide different submissions for the NIST competition. The SIKE public-key encryption scheme has made it to the NIST fourth round for further evaluation. However a recent paper [8] indicates that the SIDH assumption which underlies SIKE may be broken by just a classical computer so the future of this SIDH-based scheme is uncertain. However, other isogeny-based schemes such as CSIDH and SQI-Sign are unaffected by this attack.

- Code-based: Code-based cryptography relies on the hardness of decoding error correcting codes (which may have a specific structure such as a Goppa code). Whilst very quick, this approach tends to have the highest key size to maintain security. From the NIST competition, the Classic McEliece, BIKE, and HPQ are code-based public-key encryption schemes that have made it to the fourth round for further examination [44].

Having given an overview of how quantum computers threaten currently used cryptography and the various post-quantum approaches being considered, this project will now examine the mathematics behind isogenies needed to understand a commutative version of SIDH alongside lattices to see how lattice reduction algorithms can be used for this variant of SIDH.

2.4 Elliptic Curves and Isogenies

For a prime number $p > 3$ and an integer $n \leq 2$, we let $\mathbb{F}_{p^n}$ be a finite field with p^n elements. We restrict ourselves to $n \leq 2$ as all supersingular elliptic curves are always

given by an equation defined over $\mathbb{F}_{p^2}$. An elliptic curve E over $\mathbb{F}_{p^2}$ can be written in the Weierstrass form, which is $E : y^2 = x^3 + ax + b$ where $a, b \in \mathbb{F}_{p^2}$ and $4a^3 + 27b^2 \neq 0$. One can visualize an instance of an elliptic curve in Figure 2.2. Points on the elliptic curves over $\mathbb{F}_{p^2}$ are pairs $(x, y) \in \mathbb{F}_{p^2}$ that satisfy the above equation with an extra point ∞. Let $E/\mathbb{F}_{p^2}$ denote that E is defined over $\mathbb{F}_{p^2}$ and $E(\mathbb{F}_{p^2})$ indicate a group of points of a curve E over $\mathbb{F}_{p^2}$. $E(\mathbb{F}_{p^2})$ forms a group because they adhere to the group law, as they are associative, with an identity element and an inverse element. Let a group action $\star$ be defined as $\star : G \times S \to S$ for a finite commutative group G and a set S. We assume that $\star$ is an efficiently computable, free, and transitive group action.

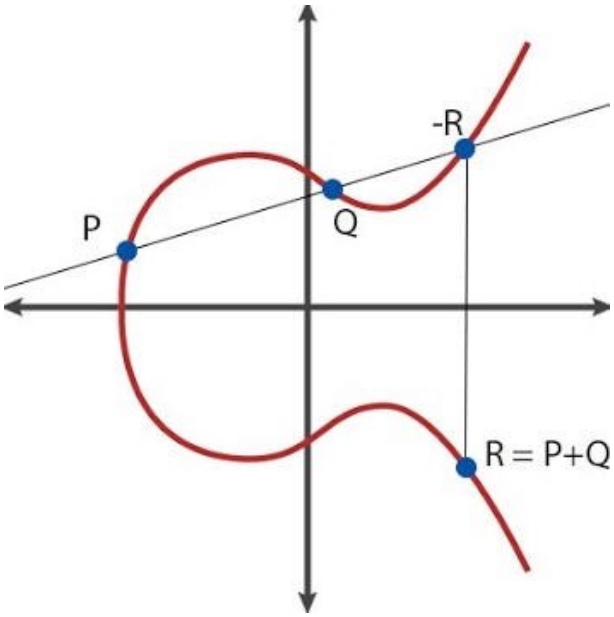

Figure 2.2: An elliptic curve with points labeled from Figure 1 in [60]

An isogeny is between two elliptic curves $E_1/\mathbb{F}_{p^2}$ and $E_2/\mathbb{F}_{p^2}$ is a non-constant rational function that maps points from E_1 to E_2 and is compatible with the group law. Figure 2.3 illustrates what an isogeny could look like. An isogeny maps the neutral element ∞_1 on E_1 to ∞_2 on E_2. For isogenies that we are interested in, the degree l of an isogeny is the number of points on E_1 over $\mathbb{F}_{p^2}$ that maps to ∞_2. The difference between an isogeny and an isomorphism is that the former does not need to by injective though usually have several points that map to ∞_2. We can also think of an isomorphism is just an isogeny of degree 1 since only the identity element of the domain maps to the identity element of the co-domain.

A mathematical theorem states that for every isogeny $\phi : E_1 \to E_2$ there is a dual isogeny $\hat{\phi} : E_2 \to E_1$ which has the same degree l, where $\phi \circ \hat{\phi} = [l]_{E_2}$ and $\hat{\phi} \circ \phi = [l]_{E_1}$ [73]. Two elliptic curves are isomorphic over $\mathbb{F}_{p^2}$ if there is a polynomial mapping over $\mathbb{F}_{p^2}$ that is injective and compatible with the group operation.

CSIDH works with elliptic curves up to isomorphism (thus works with isomorphism

classes) and requires a unique representative for each class. The choice of invariant for isomorphism classes is usually the j-invariant. In the Weierstrass form, the j-invariant is $j = 1728 * 4a^3/(4a^3 + 27b^2)$. Note that the j-invariant uniquely describes isomorphism classes over an algebraic closure of $\mathbb{F}_{p^2}$, and two curves with the same invariant does not mean they must be isomorphic over $\mathbb{F}_{p^2}$. An endomorphism ring of $\mathbb{F}_p$ is denoted by $\mathbb{O} = End(\mathbb{F}_p)$. This is the set of homomorphisms of $\mathbb{F}_p$ unto itself, and an isogeny is a surjective homomorphism thus is contained in the endomorphism ring $\mathbb{O}$. The ideal class group of $\mathbb{O}$, $Cl(\mathbb{O})$, is the quotient group of the fractional invertible ideals in $\mathbb{O}$, and most importantly, the CSIDH-based digital signature scheme CSI-FiSh finds an isomorphism such that $Cl(\mathbb{O}) \cong \mathbb{Z}_N$ [42]. The group action is defined by $Cl(\mathbb{O}) \times \mathcal{E} \to \mathcal{E}$, but with the aforementioned fact one can see group action as a mapping $\mathbb{Z}_N \times \mathcal{E} \to \mathcal{E}$.

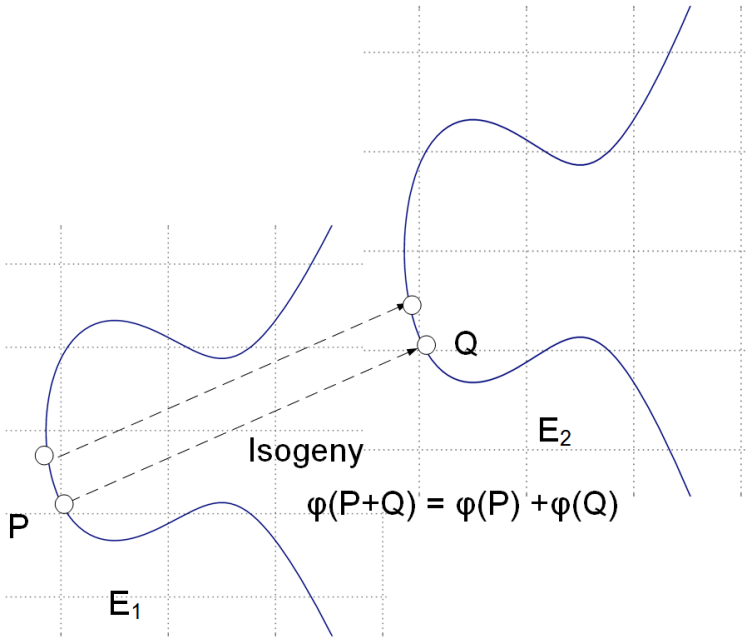

Figure 2.3: An isogeny between elliptic curves E_1 and E_2 from Figure 1 in [63]

The crucial point from isogenies is that we can form an undirected graph where nodes are the isomorphic classes of elliptic curves over $\mathbb{F}_{p^2}$ and an edge that connects two node occurs if there is an l-isogeny from one curve to in the class to another in the other class. The graph formed by this is called the l-isogeny graph over $\mathbb{F}_{p^2}$ and will be the main setting where isogeny-based cryptography takes place. This graph is undirected due to the existence of a dual isogeny for each isogeny, so from knowing a directed edge from one node to another must be the case that there is an edge directed the opposite way. In this graph type, each node has zero, one, two, or $l+1$ edges and this number is dependent on the structure of the set of points of order l on the elliptic curve. The set of all curves over $\mathbb{F}_{p^2}$ splits into disjoint components that consists of l-isogenous curves to one another and thus contain their own sub-cycles. We will only consider curves that have the same number of points. If we consider supersingular elliptic curves over $\mathbb{F}_p$ and isomorphisms defined over

$\mathbb{F}_p$, we get a radically different graph. In either case, isogeny graphs fall under a category of graphs called expander graphs. Expander graphs are graphs with only a relatively few amount of edges, but are well connected from one node to another. The choice we made on finite fields ($\mathbb{F}_{p^2}$ or $\mathbb{F}_p$) leads to two different foundations in isogeny-based cryptography. The former leads to Supersingular Isogeny-based Diffie Hellman (SIDH) while the latter leads to Commutative Supersingular Isogeny-based Diffie Hellman (CSIDH). We can see the radically different visualization of CSIDH and SIDH in Figure 2.4 where the former is more structured than the latter. This structure, commutativity, is a source of desirable properties as well as possible attack points for CSIDH discussed in later sections. We will exclusively consider the latter as both approaches lead to different security and usages. As mentioned in the previous section, though SIDH is currently used to build the NIST submission SIKE, it has recently has been dealt a critical blow by a key recovery attack using only a classical computer [8] and another which prevents mitigating actions to the key recovery attack [68] so future prospects of SIDH so far look pessimistic. However, this attack does not apply to CSIDH.

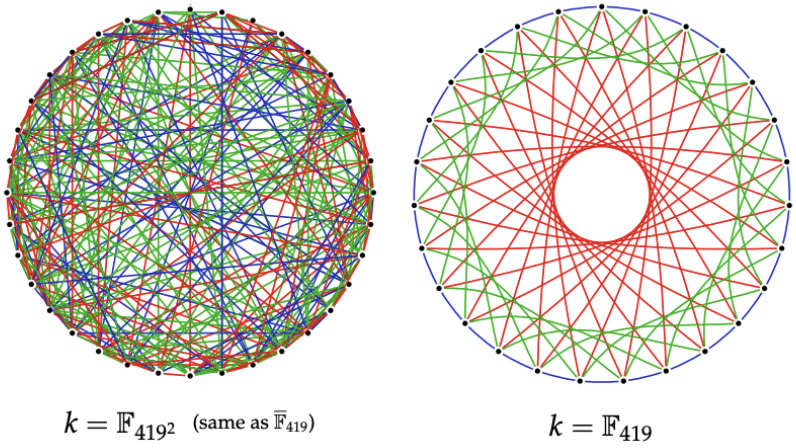

Figure 2.4: An example of SIDH on the right and an example of CSIDH graph on the left [75]

To better visualize CSIDH, consider the following isogeny graph in Figure 2.5: The nodes are isomorphism classes over $\mathbb{F}_p$ where $p = 419$. The edges can be divided into 3 different isogenies (blue for 3, red for 5, and green for 7-isogenies). Supersingular curves have $p+1$ points so $p+1 = 420 = 4*3*5*7$ thus $\mathbb{F}_p$-rational points of orders 3,5,7 exist on each of the curves.

Having covered the necessary information for isogenies, we will now need to cover lattices to see how they are used to compute isogenies in CSIDH.

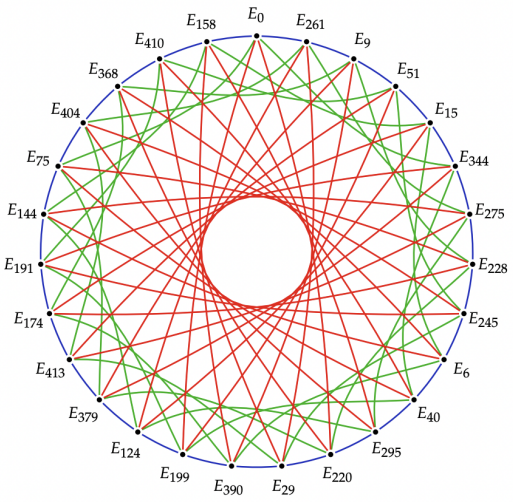

Figure 2.5: An image of CSIDH where $p = 419$. Note the different colouring of edges to represent different isogenies [75]

2.5 Lattices

A vector space V over a field K is an algebraic system that consists of a non-empty set V alongside a binary operation $+$ for vector addition and $\cdot$ which acts as a scalar multiplication such that $(a, v) \in K \times V \to a \cdot v \in V$. Several axioms must be satisfied to define a vector space. These include the existence of a zero vector, associativity and commutativity of the vector addition, distributivity of scalar multiplication, and the existence of an identity element for scalar multiplication.

Vectors can be geometrically interpreted as arrows that start at some origin point and ends at a point in a vector space. Scalar multiplication of vectors in this case is stretching the arrows in specific direction depending if the scalar is positive or negative. Geometrically, the addition of two vectors is done by the parallelogram law. Alternatively, vectors can be algebraically interpreted. These can be represented by an ordered list $(x_1, ..., x_n)$ for a vector space with n dimensions. Accordingly, vector multiplication for some scalar a is $a(x_1, ..., x_n) = (ax_1, ..., ax_n)$ and vector addition for vectors $(x_1, ..., x_n)$ and $(y_1, ..., y_n)$ is defined by $(x_1, ..., x_n) + (y_1, ..., y_n) = (x_1 + y_1, ..., x_n + y_n)$.

A linear combination of two or more vectors is the vector obtained through vector addition and scalar multiplication defined previously using multiple vectors. Given a set of vectors in a vector space V over the field K, let $v_1, ...v_n$ be vectors in V. This set of vectors form a basis over V if both of the following conditions are satisfied:

1. Linear Independence: For scalars $c_1, ..., c_n$ where $c_1v_1 + ... + c_nv_n = 0$ then necessarily $c_1 = ... = c_n = 0$.

2. Spanning: For every vector $v \in V$, we can find scalars $c_1, ..., c_n$ such that $v = c_1v_1 + ... + c_nv_n$

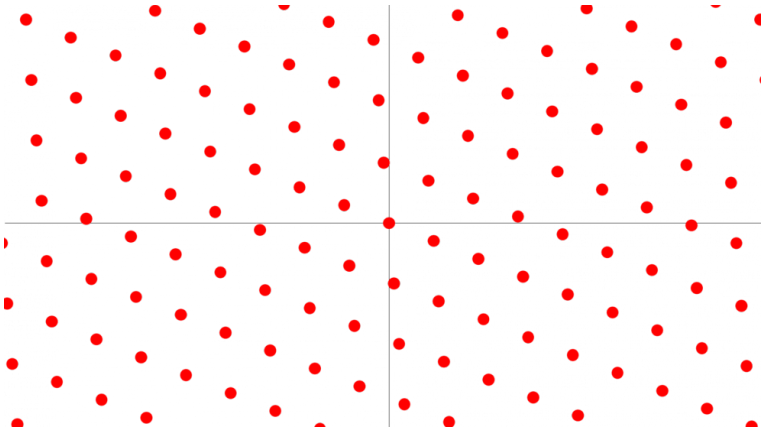

Figure 2.6: An example of a lattice [66].

A vector space can have several bases, but all the bases have the same number of elements in them, which defines the dimension of the vector space. In the following, we will see that the choices of bases has applications in cryptography.

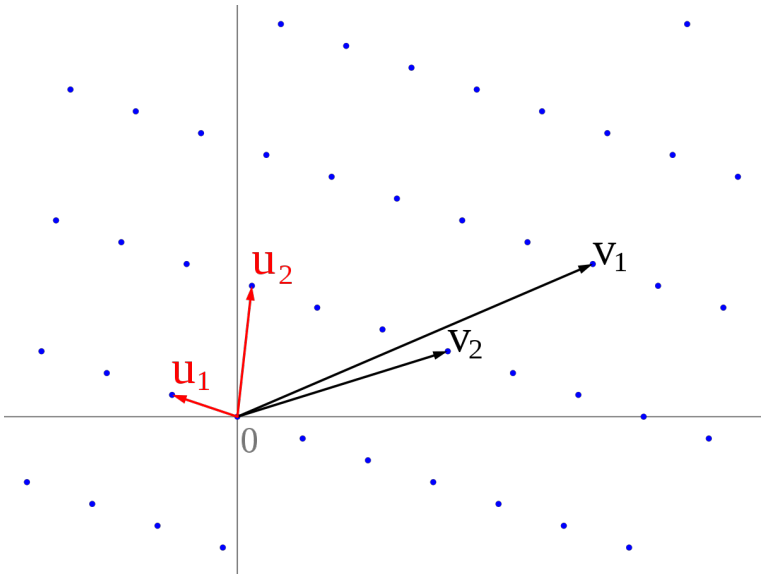

Figure 2.7: A lattice with 'good' basis (u_1, u_2) and 'bad' basis (v_1, v_2) [22].

A lattice $\mathcal{L}$ of $\mathbb{R}^n$ is a discrete subgroup of $\mathbb{R}^n$. It can be visualized as an infinite set of points in space such that addition and subtraction of points produces another point in this space. Points in a lattice are separated by some minimum distance, and every point in this space can be represented by a vector. A basis of a lattice $\mathcal{L}$ is an ordered set $\mathbb{B} = (b_1, b_2, ..., b_n)$ such that $\mathcal{L} = (\mathbb{B}) = \mathbb{B} \times \mathbb{Z}^n = \{\Sigma_{i=1}^n c_i b_1 : c_i \in \mathbb{Z}\}$. Essentially, it is a linearly independent collection of points such that every element in a lattice $\mathcal{L}$ can be represented by a linear combination of these points. We can see that lattices are just vector spaces where only integer scalars are allowed which gives us a discrete set of points.

For applications in isogeny-based cryptography, lattice-based cryptography and other areas such as homomorphic encryption, we are concerned with 'good' and 'bad' bases.

The former generally refers to lattice basis that is relatively short and orthogonal, while the latter refers to relatively long and non-orthogonal basis. Orthogonality is a measure of perpendicularity, and two vectors are orthogonal if their dot product, the sum of each vector's entry multiplied with the corresponding entry of the other vector, is zero (for vectors $a = (a_1, ..., a_n)$ and $b = (b_1, ...b_n)$, $a \cdot b = \Sigma_{i=1}^{n}(a_i + b_i) = 0$). One measure for how orthogonal vectors is called the orthogonality defect. In this, one compares the products of the lengths of the basis with the volume of the parallelepiped that they define. Perfectly orthogonal basis vectors will make the lengths of the basis equal to the volume of the parallelepiped. The application of lattices for CSIDH involves finding a 'good' basis in a lattice given a 'bad' basis and interestingly, this problem for larger dimensions is conjectured to be hard which is used to build lattice-based cryptography.

Chapter 3

Related Work

The primary aim of this chapter is to give readers more depth on the area of CSIDH and lattice reductions. The related works of this chapter will be significant when discussing the implications of this project's results.

3.1 Research Area Overview

Isogeny based schemes were first proposed by Jean-Marc Couveignes in 1997[12], however this was never publicly circulated [42]. The same idea was later rediscovered independently in 2006 by Rostovtsev and Soltubunov to create the CRS scheme [42]. The SIDH scheme, based on the one proposed in [35] was invented in 2011 and is used to crease the SIKE scheme. Though the SIDH assumption has since taken quite a large slice of the isogeny research area, there are other isogeny schemes that have been proposed. These include OSDIH, CSIDH, and SQI-Sign. The SIDH research area has branched out into developing signature schemes such as SHealS, multiparty computation features such as commitment schemes, cryptanalysis works such as fault attacks, and more towards the mathematics of SIDH generally. With the latest attack on SIDH that seems to break the scheme, there are also some research endeavours examining if it is possible make alterations to save SIDH by masking most information surrounding the degrees of hidden isogenies [61].

3.2 Recent Contribution

CSIDH is quite a recently developed protocol. This was first proposed in December 2018 on [75] with this paper being the most seminal contribution to the area. Since then, there are several other notable works in various areas. These range from theoretical explorations that examine the security of CSIDH to implementations that seek to improve performance and cryptographic schemes which carry out certain tasks.

When examining the security of CSIDH against a quantum adversary, the main work

revolves around Kuperberg's algorithm. [7] first argued that the CSIDH parameters originally proposed were too optimistic by showing an attack that used 2^{35} qubits using Kuperberg's algorithm, but this is a contested claim. [65] worked on improving the performance of quantum cryptanalysis by focusing on the collimation sieve algorithm. This algorithm is used to detect the most significant bits which is a necessary part of Kuperberg's attack. On the other hand, [31] further examined the trade-offs between classical and quantum circuit size against CSIDH to better find the precise needed cost to attack CSIDH from a classical and quantum perspective. [18] examined possible speedups to quantum algorithm attacks used by previous papers, but noted that some previous costs assumptions such as oracle query cost might have been too optimistic, leading to more questions about the exact security of CSIDH. [28] has also proposed new parameters in response to the attack by [7] assuming that it is indeed as crippling as described.

However, using a quantum algorithm is not the only research done to attack CSIDH. There are attacks that focus on classical computers to flaws in implementation to try and break CSIDH. [30] proposed an algorithm to compute an isogeny between curves that ran in exponential time with polynomial memory requirements on both classical and quantum computers. Further work by [55] shows that fault attacks on CSIDH are possible using a maximum likelihood approach. [70] noted that CSIDH key exchange allows a very special form of shared key control via quadratic twists which allowed for a possible point of attack. [23] proposed an alternative attack on CSIDH called safe-error attacks, which used targeted bit flips during group actions that might lead to full recovery using voltage and clock glitching demonstrating that full key recovery is possible in practice if CSIDH is implemented incorrectly.

Interestingly, [74] notes that CSIDH security can be reduced to a class of computing endomorphism rings of oriented curves and suggest that all isogeny based schemes may have their security predicated on computing endomorphism ring of orientable curves. It is still unclear how the recent attacks on SIDH as outlined in [8] and [57] would impact the security implications laid out by [74].

One of the key applications of CSIDH has been in the digital signatures and public key encryption area. The CSI-FiSh digital signature was the first to use lattice reduction algorithms to create digital signatures, and will feature more in later chapters [75]. This digital signature scheme allows the use of lattice reduction algorithm due to computations on the class group such that $Cl(\mathbb{O}) = \mathbb{Z}/N\mathbb{Z}$. Without this, the alternative is to use Fiat-Shamir with aborts. The SeaSign signature scheme for isogenies uses CSIDH with the Fiat-Shamir aborts to create signatures that was less than 1 kilo-byte in size, but notes that signing and verification actions are quite costly [48]. [40] provided an improvement to SeaSign by allowing the prover to not answer a limited number of parallel executions to lower the rate of rejection. The Calamari and Falafl ring signature schemes are also based on CSIDH, but the authors develop them such that it works using lattices as well

through the MLWE assumption which is a novel union of the two security assumptions [45]. Lastly, CSI-IBS is a CSIDH digital signature scheme that used the Fiat-Shamir transform to create the first CSIDH-based identity based signature scheme [17]. Other public key encryption schemes have also been proposed, notably SiGamal and C-SiGamal which is similar to the ElGamal encryption schemes [41] and a one-round authenticated group key exchange has also been developed [50]. [1] proposed a CSIDH based password authenticated key exchange, and so far seems to be the only CSIDH based protocol in the field.

CSIDH has seen quite a rapid development in the multiparty computation space which includes threshold cryptography and oblivious transfers. [49] first studied threshold schemes from CSIDH by developing a threshold version of CSI-FiSh, while CSI-RAShi is a distributed key generation algorithm that was built with components of CSI-FiSh which allowed for desirable properties such as zero-knowledge proofs and an isogeny version of the Pedersen commit to prevent parties from lying about their computation [43]. CSI-RAShi needs CSI-FiSh as it requires an isomorphism up to $\mathbb{Z}_N$ to be able to do Shamir's Secret Sharing. In the Sashimi protocol [13], the authors build upon the CSI-FiSh signature scheme to create a threshold cryptography scheme while [37] has also proposed secure multiparty computational schemes against adaptive adversaries using receiver oblivious sample-ability property. This leads to research in the oblivious transfers space, where [54] first proposed a compact and efficient CSIDH based oblivious transfer scheme with universal composability security, and this was complemented in [15] by proposing a CSIDH and SIDH based scheme with semi-commutative masking. A follow-up work by [54] also added collusion resistant in CSIDH group signatures to the list of desirable properties that can be implemented in CSIDH. Lastly, [34] provided the first CSIDH scheme that allowed for anonymity and accountability while [64] developed DeCSIDH which allows for secure and verifiable delegation of isogeny computations in the CSIDH setting.

With regards to implementations and improvements in practice, [3] conducted an implementation of the Signal protocol using CSIDH in place of the ECDH assumption. The benchmark results indicates that it takes roughly 8 seconds to initialize a session in CSIDH-512 and over 40 in CSIDH-1024. Another implementation by [51] demonstrated a constant time CSIDH on embedded devices. [55] proposed a framework to improve upon previous speedup techniques using linear, dynamic, and convex programming techniques that would be applicable to any CSIDH security level. [20] has fine-tuned some efficient parameters by noting that using CSIDH on the surface of Montgomery curves such that $p \equiv 7 \mod$ was most likely to be efficient. Their proceeding work [21] built upon their previous results by examining two-torsion points which gave a 6.4% performance increase compared to the original CSIDH. Similar to [20] which worked on the choices of elliptic curves, [58] has provided speedup techniques for CSIDH by restructuring elliptic curve point manipulations that yielded a speed-up factor of 1.33. Another paper,[9] introduced

projective radial isogenies to save time on costly inversions and improved on the evaluation strategy of CSIDH to save point samplings. These improvements claimed to be far more efficient in calculating finite field multiplications for CSIDH.

There are also proposals for constant time evaluations of CSIDH. CTIDH aims at implementing a CSIDH group action in constant time [24], but the key space is not compatible with previous key spaces. Their results has reduced have claimed to reduce the computation time of CSIDH-512 in half compared to the one proposed in [75]. The paper [2] examines Velu's formula used to construct and evaluate isogenies. This paper presents a concrete computational analysis of a proposed speedup technique by implementing it on Python 3 with results ranging from 5-26 percent faster depending on the CSIDH-parameters. Some research has also drawn upon optimizing strategies in other schemes. [10] discussed how to transfer strategies from SIDH to CSIDH for improved performance, and [47] used optimization techniques found in other schemes called Elligator and SIMBA to improve on performance speed as well. Another approach transferable from a different field include [26] aimed to speed CSIDH computations by maximizing throughput and minimizing latency using AVX-512 vector extensions techniques.

Additionally, research into improving CSIDH performance has expanded beyond evaluating group actions. [24] notes that key validation for CSIDH can be costly, and developed two new alternatives that improves on speed and memory.

On the other hand, lattice reduction has a much older research history than CSIDH. The first lattice reduction algorithm was proposed in 1773 called the Lagrange/Gauss reduction for 2D lattices [67]. In 1982, the Lenstra-Lenstra-Lovasz reduction was proposed by [4], and the notable Block Korkine-Zolotarev (BKZ) was proposed in 1987 [67]. It was only in 2020 that the first rigorous dynamic analysis of BKZ was introduced. This analysis guaranteed the quality of the current lattice basis during execution and provides the current best and simplest bounds for output quality and run time [56].

Lattice reduction algorithms have also been used in multiple-input multiple-out [27] communication systems. The linear detectors that take in signal vectors might be defective, and lattice reduction algorithms have been applied to build a more powerful detector for this purpose [27]. A recent paper by [36] has also shown that in a truncated version of a lattice reduction algorithm has been applied to discrete pruning for lattice enumeration, which is a method developed to solve the SVP. This method has been noted to outperform the extreme pruning technique for lattices.

3.3 Open Problems

The main open problem surrounding CSIDH is related to its security, specifically with quantum cryptanalysis. The exact cryptanalysis approaches will be discussed in the following chapter, but for now the key issue rests on the quantum oracle used to determine

CSIDH group action. From [28], these include finding out how oracle errors affect the quantum algorithm, what overheads are incurred from wielding a large number of qubits, seeing if it is possible to decrease the cost of a quantum oracle query, what exactly is the quantum oracle query cost, and also to optimize performance for the quantum algorithms involved. This question is quite a vital one. Examining the quantum cryptanalysis implications of CSIDH will determine if the scheme could be feasibly broken like SIDH, and if not, what the parameters are to meet desirable levels of security.

Chapter 4

CSIDH Using Lattice Reduction Algorithms

Having covered the mathematical foundations for CSIDH and lattice reductions, this chapter will explain the CSIDH protocol, how we can use lattice reduction algorithms for CSIDH, and the security of CSIDH.

4.1 CSIDH Overview

In the discrete logarithm Diffie-Hellman key exchange, we work over a finite group $(G, *)$ where $g \in G$ and $g^x = g * ... * g$ x-times. The current protocol generally works as follows:

- A picks $a \in \mathbb{Z}$ and sends g^a to B.
- B picks $b \in \mathbb{Z}$ and sends g^b to A.
- A computes $(g^b)^a$ while B computes $(g^a)^b$.
- Their shared key is $k = (g^b)^a = (g^a)^b$.

This protocol works because of commutativity, $(g^b)^a = (g^a)^b$, but the protocol is predicated on the security of the DLP which we know is vulnurable to quantum computers. The idea of isogeny-based cryptography is to replace the exponent function $f : \mathbb{Z} \times G \to G$ with a group action on a group G, a set S, and a function $i : G \times S \to S$. In these approaches, the challenge is to find a path from one node to another in a graph that is built on isogenous elliptic curves. The procedure of CSIDH that is used for CSI-FiSh is as follows [42]:

- Pick out a set of small prime numbers $l_1, ..., l_n$ such that $p = 4 \cdot l_1 \cdot ... \cdot l_n - 1$ is also prime. This results in a curve that has points of order l_i over $\mathbb{F}_p$. Efficient computations use Velu's formula found in [28].

- Pick another system parameter m. The original proposal picks m and r such that $(2m+1)^r \approx \sqrt{p}$ to store roughly $(\log p)/2$ bits. A current implementation of CSIDH-512 uses $m = 5$. This gives us public key sizes of 64, 128, and 224 bytes for CSIDH-512, CSIDH-1024, and CSIDH 1792 respectively.

- Alice generates a secret key, which is the exponent vector $(a_1, a_2, ..., a_r)$ where $a_i \in [-m, m]$ that tells how many times the l_i isogeny is computed. We can think of it as the sequence of group actions on elliptic curves.

- Alice computes her public key which is the elliptic curve that is reached after a_i isogenies of degree l_i for $i \leq i \leq r$ are computed. Alice's public key is the coefficient A of the resulting curve.

- Bob does the same with his exponent vector $(b_1, b_2, ..., b_r)$ to get his secret key and similarly computes his public key A'.

- The shared key is the curve reached when $(a_1 + b_1, a_2 + b_2, ..., a_r + b_r)$ is reached. Going back to the analogy in previous steps, this is computed by group actions on each entry of the exponent vector since $[a][b]E = [a + b]E$. This computation can be done by applying each party's sequence of isogeny to the other party's public key curve.

The last step outlined above applies only to the CSI-FiSh case, however CSIDH in general works as a nice replacement of traditional Diffie-Hellman key exchange as cycles are commutative, and several computing algorithms have been claimed to calculate isogeny evaluations efficiently.

4.2 Lattice Problems and Reductions

As seen in previous chapters, there are hard problems based on lattices that are conjectured to be safe against quantum adversaries. Two notable ones relevant to this project are:

- Shortest Vector Problem (SVP): In SVP, we are given a basis of a vector space V and a norm N is for a lattice $\mathcal{L}$. The task is to find the shortest non-zero vector as measured by N in $\mathcal{L}$.

- Closest Vector Problem (CVP): In CVP, we are given a basis of a vector space V, a metric M, and a vector $v \in \mathcal{L}$ for a lattice $\mathcal{L}$. The task is to find a vector closes to v as measured by M.

One can see that the CVP is just a generalization of SVP. Given an oracle for CVP, one can solve for SVP with a reduction outlined in [59]. These problems are easy to solve if the algorithm is given a good basis. And if we are given a 'bad' basis, lattice reduction

algorithms aims to use a given basis for a lattice to output a 'good' new basis. The task of lattice reduction with the aim of finding a basis with the smallest possible defect is NP-hard, but algorithms in polynomial time to exists to find a basis with a defect below some constant c that depends on the number of basis vectors and dimension of the lattice. In practice, this constraint will do for our CSIDH purposes. The orthogonality

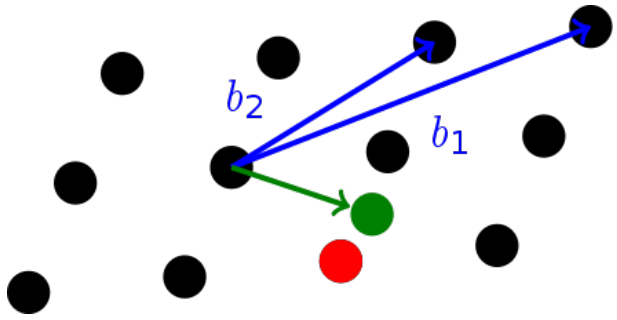

Figure 4.1: CVP: Given basis $b1$ and $b2$, find the nearest vector to the green vector [71].

defect mentioned previously can be defined for a given basis B as $\delta(B) = \frac{\Pi_{i=1}^{n}||b_i||}{\sqrt{det(B^T B)}}$ and $\delta(B) \geq 1$ with equality if and only if the basis is orthogonal. For this project, a classic lattice reduction algorithm, the Lenstra-Lenstra-Lovasz (LLL) algorithm, which can output an almost reduced lattice basis in polynomial time, will be foundational for implementing group action. The LLL has a run time of $\mathcal{O}(d^5 n \log^3 C)$ where C is the length of the longest basis under the Euclidean norm, and versions of the LLL has been used for cryptanalysis such as factorizing polynomials and simultaneous rational approximations until new results in the late 1990's demonstrated the hardness of lattice problems.

Algorithm 1 Lenstra-Lenstra-Lovasz Algorithm

Input: Basis $B = (b_1, ..., b_n) \in \mathbb{Z}^n$ for a lattice $\mathcal{L}$.
Output: A $\delta - LLL$ reduced basis for $\mathcal{L}(B)$.

Start: Compute $(b'_1, ..., b'_n)$
Reduction Step:
for $i = 2, ..., n$ **do**
  for $j = i - 1$ **do**
    $b_i \leftarrow b_i - c_{i,j} b_j$ where $c_{i,j} = \lceil \langle b_i, b'_j \rangle / \langle b'_j, b'_j \rangle \rfloor$ and $\lceil . \rfloor$ returns the nearest integer
  end for
end for
Swap Step:
if $\exists i$ such that $\delta ||b'_i||^2 > ||\mu_{i+1,i} b'_i + b'_{i+1}||^2$ **then**
  $b_i \leftrightarrow b_{i+1}$
  Compute $(b'_1, ..., b'_n)$
end if
return B

In the LLL algorithm, we are given a basis $B = (b_1, ..., b_n)$ and we can define its Gram-Schmidt orthogonal basis as $B' = (b'_1, ..., b'_n)$ with the Gram-Schmidt coefficients $\mu_{i,j} = \frac{\langle b_i, b_j * \rangle}{\langle b'_j, b'_j \rangle}$ for $1 \leq j < i \leq n$. The basis B is LLL-reduced if there is a parameter

$\delta \in (0.25, 1]$ if the following conditions hold:

1. Size-reduced: $|\mu_{i,j}| \leq 0.5$ meaning that this guarantees length reduction of the ordered basis.

2. Lovasz Condition: For $k = 2, ..., n : \delta||b'_{k-1}||^2 \leq ||b'_k||^2 + \mu^2_{k,k-1}||b'_{k-1}||^2$.

Estimating the value of δ, we can see how well the basis is reduced. The greater δ is, the stronger the reductions. Note that though $\delta = 1$ is well defined for the LLL algorithm, the current complexity holds for $\delta \in (0.25, 1)$. The LLL algorithm works by first ensuring that the basis becomes size-reduced. After this, the only way for LLL algorithm to fail is if the second condition is violated. To prevent this, the algorithm swaps b_i with b_{i+1}. For multiple values that violate the second condition, a parallel version of the LLL algorithm can be constructed. However, after a swap, the basis may be not be size-reduced anymore so it must repeat the first component.

Upon finishing, the basis is LLL-reduced as it is size reduced and no component needs to be swapped. This algorithm is proven to terminate in polynomial time and more details can be found in [59].

The LLL algorithm is used in another lattice reduction algorithm: Babai's Nearest Plane algorithm which solves an approximated version of the CVP with an approximation factor of $2(2/\sqrt{3})^n$ where n is the dimension of a lattice. This algorithm is of particular relevance as it is used to calculate isogeny group actions through vectors and lattices.

In Babai's Nearest Plane, the LLL reduction is applied to reduce the input lattice. Afterwards, it searches for a combination of integers of the basis vectors that gives us a closest vector to the target vector. The proof of correctness and polynomial complexity can be found in [67].

Algorithm 2 Babai's Nearest Plane

Input: Basis $B = (b_1, ..., b_n) \in \mathbb{Z}^n$, a vector $t \in \mathbb{Z}^n$, and a distance metric M for a lattice $\mathcal{L}$.

Output: A vector $x \in \mathcal{L}(B)$ such that $||x - t|| \leq 2^{\frac{n}{2}}$ where $||.||$ is the distance metric M.

Run the LLL algorithm on B where $\delta = \frac{3}{4}$.
$b \leftarrow t$
for $j = n...1$ **do**
  $b \leftarrow b - c_j b_j$ where $c_j = \lceil \langle b, b'_j \rangle / \langle b'_j, b'_j \rangle \rfloor$ and $\lceil . \rfloor$ returns the nearest integer
end for
return $x = t - b$

4.3 Applying Lattice Reduction Algorithms to CSIDH

Having outlined lattice reduction algorithms, we now tie it back to CSDIH and show how to compute group actions using lattices. Our work is based on [42].

For a group action $[\ ] : \mathbb{Z}_N \times \mathcal{E} \to \mathcal{E}$, we need to find an equivalent representation of the ideal class of $\mathbf{g}^a$ given an $a \in \mathbb{Z}_N$. Note that $\mathbf{g} = \Pi_{i=1}^n \mathbf{l}_i^{g^i}$. Thus, we get an exponent vector $\mathbf{e} = (e_1, ..., e_n)$ from a secret $a \in \mathbb{Z}_N$ which tells us which isogenies to compute and in which direction (determined by $-$ and $+$). This vector needs to be reduced modulo the relation lattice $\mathcal{L} := \{\mathbf{z} = (z_1, ..., z_n) \in \mathbb{Z}^n : \Pi_{i=1}^n \mathbf{l}_i^{z_i} = (1)\}$. This lattice has a rank of n and volume $N = \#Cl(\mathcal{O})$ as the kernel of the surjective group homomorphism that maps $\mathbb{Z}^n \to Cl(\mathcal{O}) : \mathbf{z} = (z_1, ..., z_n) \to \Pi_{i=1}^n \mathbf{l}_i^{z_i} = (1)$.

The main complexity of a CSIDH group action in this is determined by the l_1-norm of the exponent vector, and so we want to solve the CVP in $\mathcal{L}$ with the l_1-norm and a target vector $\mathbf{e}$. Any vector $\mathbf{z} \in \mathcal{L}$ that is close to $\mathbf{e}$ will give a vector $\mathbf{e} - \mathbf{z}$ such that $\|\mathbf{e} - \mathbf{z}\|_1$ is small and efficiently computable. A first approximation for this would be to use either Babai's rounding or nearest plane algorithm.

Babai Rounding is a computationally simpler method than Babai's Nearest Plane. After applying LLL, we approximate each coefficient that constitutes an element of a vector in the basis by rounding it to the closest integer. However, this method comes at an exponential constant of the correct solution after applying LLL [59].

Algorithm 3 Babai Rounding

Input: Basis $B = (b_1, ..., b_n) \in \mathbb{Z}^n$, a vector $t \in \mathbb{Z}^n$, and a distance metric M for a lattice $\mathcal{L}$.

Output: A vector $x \in \mathcal{L}(B)$ such that $||x - t|| \leq (1 + 2n(9/2)^{\frac{n}{2}})||x - v||$ where $||.||$ is the distance metric M and $\forall v \in \mathcal{L}$.

Run the LLL algorithm on B where $\delta = \frac{3}{4}$.
Set $t = \Sigma_{i=1}^n l_i b_i$
for $i = 1...n$ **do**
    $l_i' = \lfloor l_i \rceil b_i$
end for
$x = \Sigma_{i=1}^n l_i'$
return x

Given a set of basis $B := (b_1, ..., b_n)$, let $B' := (b_1', ..., b_n')$ be the corresponding vectors after the Gram-Schmidt process. We let $\mathcal{P}(B) = \{\sigma_{i=1}^n \alpha_i \mathbf{b} | \alpha_i \in [-1/2, 1/2]\}$. Babai rounding returns a lattice vector in $\mathbf{e} + \mathcal{P}(B)$ and Babai's nearest plane returns a lattice vector in $\mathbf{e} + \mathcal{P}(B')$. So, $\mathbf{e} - \mathbf{z}$ is in either $\mathcal{P}(B)$ or $\mathcal{P}(B')$ depending on which algorithm we choose. So given B and its corresponding Gram-Schmidt basis B' it is easy to bound $\|\mathbf{e} - \mathbf{z}\|_1$. This also shows that a good basis will give better results. In this project, we will only use Babai's nearest plane algorithm as it is superior to Babai rounding given its precision and lower complexity.

Since the lattice is fixed for a given class group, a large effort can be spent in reducing the lattice bases during precomputation. Some examples of reductions include the BKZ-40 and HKZ. For each reduced bases, we run Babai nearest plane to get the exponent vector. It is possible to lower the l_1-norm costs further using the Doulgerakis, Laarhoven, and de

Algorithm 4 DLW Algorithm

Input: A list $\mathcal{S} \subset \mathcal{L}$ of short vectors, a target vector $e \in \mathbb{Z}^n$, number of iterations M
Output: Approximate closest lattice vector z to e

```
z ← 0
for i = 1, ..., M − 1 do
    Randomize e with random small lattice vector to obtain e'
    for s ∈ S do
        if ||e' − s||_1 < ||e'|| then
            e' ← (e' − s) and restart the loop for s ∈ S
        end if
    end for
    if ||e'||_1 < ||e − z|| then
        z ← (e − e')
    end if
end for
return z
```

Weger (DLW) algorithm. DLW takes a list $\mathcal{S}$ of short vectors in $\mathcal{L}$ and tries to construct a vector that is closer than the current vector $\mathbf{z}$ by creating $\mathbf{z} \pm \mathbf{s}$ for all $\mathbf{s} \in \mathcal{S}$. The procedure is repeated on small shifts of the target vector.

Having discussed both the theories of isogenies and lattices alongside how they intertwine to perform group action on CSIDH, the following chapter examines an implementation of the CSI-FiSh lattice reduction method for computing group action but optimizing performance using the DLW algorithm is left as future work.

4.4 Security of CSIDH

The very properties that sets CSIDH apart from SIDH and gives it the advantage of allowing non-interactive key exchange, namely commutativity, is also a point of possible exploitation. This commutative structure allows quantum attacks that asymptotically take sub-exponential time. Thus, CSIDH key size must grow super-linearly to reach certain levels of post-quantum security. However, this is still an open problem. There is no clear-cut method to assign weights on these asymptotics. For instance, we do not know at what key size does the subexponential quantum attacks perform better than exponential-time non-quantum attacks. It is a particularly challenging problem as there are no quantum computers of necessary scale to actually determine the concrete complexity of these attacks and as demonstrated in the case of SIDH, an ingenious solution on a classical computer may be enough to break the security of some schemes.

The central security problem of CSIDH is the key recovery problem. Given two supersingular elliptic curves E and E' defined over $\mathbb{F}_p$ with the same $\mathbb{F}_p$-rational endomorphism ring $\mathcal{O}$ (which one can think of having morphisms unto itself and forms a ring), an adversary must find an ideal α such that $[\alpha]E = E'$.

For a non-quantum adversary, the brute-force key search has an exponential complexity of $2^{\log \sqrt{p}-\epsilon}$ for an ϵ that is small when compared to $\log \sqrt{p}$. Meanwhile, meet-in-the-middle attacks have an optimal time complexity of $O(\sqrt[4]{p})$. However, when considering quantum attacks, the cost of evaluating a group action and a series of isogenies in superposition leads to a bottleneck that quantum attacker incurs this cost multiple times.

According to latest results [6], CSIDH group actions can be carried out in B non-linear bit operations with a failure probability of at most ϵ. This approach also knows when they have failed. The paper suggest that a reversible computation of CSDIH group action with a failure of at most ϵ is possible using at most $2B$ Toffoli quantum gates. The development applies to Kuperberg's algorithm that uses $exp((\log(N)^{1/2+o(1)})$ queries and $exp((\log(N)^{1/2+o(1)})$ operations on $exp((\log(N)^{1/2+o(1)})$ qubits to solve the hidden-subgroup problem that is used to break CSIDH. While the exact number of queries for various subgroup algorithms and exact query cost are an open problem, this is at present the most viable attack against CSIDH.

In an earlier paper by [7], the claim was made that a specific parameter set of CSIDH, CSIDH-512, was broken by 2^{71} quantum gates where each query uses 2^{37} quantum gates. The quantum abelian hidden shift algorithm can be used to recover the secret key in a CSIDH key-exchange without asympotitc cost. However, there are many tradeoffs in quantum hdiden shift algorithms and this is still an ongoing research area. [7] claims that under this development, CSDIH5-512 would be at least 1000 times easier to break quantumly than AES-128.

Another add-on the Kuperberg's algorithm is the collimation sieve. This uses exponentially less quantum memory and offers more parameter tradeoffs. It has been generalized in [65] to work for arbitrary finite cyclic groups and was run on CSIDH-512. The claimed result was that a key recovery of CSIDH-512 in 2^{16} quantum evaluations using 2^{40} bits of quantumly accessible memory. Like the other studies, however, there is an assumption that the number of qubits for the oracle is negligible to support the stated results. [18] indicates that though there are ways to reduce the number of qubits for an oracle, extrapolating from memory consumption of CSIDH to the number of qubits needed is not justified and argues that fault-tolerance quantum computers are quite difficult as we add more qubits. From a quantum security perspective, we must consider Grover's algorithm that is applied via the claw finding method detailed in [31] as it is applicable to CSIDH. It leads to $O(\sqrt[6]{p})$ calls to a quantum oracle to compute our group action. The overview of this attack is to split the search space for collisions into a classical $O(\sqrt[6]{p})$ target and a $O(\sqrt[3]{p})$ search segment to apply the quantum search algorithm. Therefore, the choice of p must be large enough to imply quantum security.

In short, while there are doubts about the security of CSIDH and its most common parameter set of CSIDH-512, it is still an ongoing research area if these attacks are as successful as they claim to be and latest research such as [18] indicates that the compu-

tational cost of evaluating group actions for an adversary may be more prohibitive than initially assumed.

Chapter 5

Implementation and Results

In this section, we will outline the CSIDH parameters used to implement random sampling and converting it to an exponent vector alongside any associated libraries. We will also provide our findings with regards to performance speed of this approach for group action and discuss its implications.

5.1 CSIDH-512

The most commonly examined and implemented parameter set of CSIDH is CSIDH-512. This parameter set is conjectured to achieve NIST Level 1 quantum security, which implies that there is no key-recovery attack that runs faster than an exhaustive key search of AES-128 (an instance of AES where the key length is 128-bits) [44]. While security of CSIDH-512 is an open problem as discussed in previous chapters, being the most implemented parameter set means that we already have some libraries to work with. The public key size of CSIDH-512 is 64 bytes with a private key size of 32 bytes which again reinforces the attractiveness of relatively short key size for this post-quantum standard notwithstanding the run time. For comparison, at NIST Level 1 quantum security, the code-based McEliece using Goppa codes require 1 megabyte for its public key and 11.5 kilobyte for the public key, and the lattice-based NTRU-Encrypt has a 766 byte public key with private key of size 842 byte.

In CSIDH-512, we let $n = 74$ thus giving us primes $l_1 = 2, ..., l_n = 587$ and $N = 4 \times l_1 \times ... \times l_n - 1$. We also let $B = [-5, 5]$. It is assumed that by the value of B, we are able to cover a set 2^{256} which is a bit less than half of the total size of the class group. CSIDH-512 has been used to implement CSIDH-based schemes such as CSI-FiSh, CSI-RAShi, Lossy CSI-FiSh, DeCSIDH which allow for multiparty computation, signatures, and delegated computing.

5.2 Libraries Utilized

The Cloudflare Interoparable Reusable Cryptographic Library (CIRCL) is a library that contains cryptographic primitives for the language Go. The aim of this library is for experimental deployment in the post-quantum and elliptic curve cryptographic space. This library implements some key exchange and digital signature schemes alongside lattice-based cryptography such as Kyber. CIRCL also implements CSIDH-512 and SIKE with various parameters. Unsurprisingly, we will only utilize the CSIDH-512 aspects of this library and this is the main library that our implementation depends on. Further documentation on the CIRCL package can be found here (LINK).

Other helpful libraries include the crypto/rand library for random sampling of involved in selecting the secret element with more documentation here (https://pkg.go.dev/crypto/rand). The math/big library was used to represent the large numbers calculations involved for CSIDH-512 and its documentation can be found here (https://pkg.go.dev/math/big). Lastly, the strconv library needed to interpret the three bases involved in this implementation and its documentation can be found here (https://pkg.go.dev/strconv).

5.3 Implementation

A drawback of using CIRCL is that most of its underlying CSIDH functions are not exportable. CSIDH-512 is only usable in CIRCL to derive a shared secret, generate public and private keys, and to validate them. The generation of a secret comes from directly generating an exponent vector through random sampling from B instead of the composite N. Though this is a nice implementation, it does not allow us to randomly sample an integer from $\mathbb{Z}_N$ and then use this as the basis of our exponent vector. This feature is quite desirable, especially if one were to implement specific distributed tasks that requires sampling an element from $\mathbb{Z}_N$ like in CSI-FiSh and CSI-RAShi. Thus as outlined in previous chapters, we have implemented a lattice reduction algorithm whilst creating a copy of CIRCL to export some hidden functions.

In this project, we will examine three precomputed bases BKZ-40, BKZ-50, and HKZ. These three bases correspond to that involved in the LLL, BKZ, and HKZ reduction algorithms respectively. Each of these bases will go through Babai's Nearest Plane and results in our exponent vector. This implementation used the v0.35.2 of Go and key pieces of the code can be found in the Appendix section.

5.4 Results

The code execution has been carried out on the MacBook Air using M1 Chip and the Big Sur v11.6 macOS. The results have been taken from an average of three lattice reduction

times, and are shown below. In addition, we have also included using the l_2-norm as an alternative distance metric. The l_1 norm for a vector $v = (a_1, ..., a_n)$ is defined as $l_1(v) = |a_1| + ... + |a_n|$ while the l_2 norm for a vector $v = (a_1, ..., a_n)$ is defined as $l_2(v) = \sqrt{(a_1^2 + ... + a_n^n)}$.

Basis	l_1-Norm	l_2-Norm
BKZ-40	35.03	30.20
BKZ-50	33.23	30.21
HKZ	29.28	29.12

Table 5.1: Group Action through lattice reduction, measured in seconds.

From the results, we can see that there is a noticeable difference in each lattice and norm used. The HKZ basis with the l_2-norm runs the quickest, and the BKZ-40 basis with the l_1-norm ran the slowest. In addition, changing from l_1 to l_2 provided a noticeably faster computation time for BKZ-40 and BKZ-50 with minor speedup for HKZ.

5.5 Discussion

Our results indicate that the group action using lattice reduction algorithms is quite a slow process across the board but is necessary for some algorithms that involve CSIDH. Though there are noticeable differences in performance through the use of different bases and norms, they are not substantial enough to make this procedure of CSIDH to be viable as a drop-in replacement for ECDH and does not compare well against other post-quantum cryptographic standards. This finding would not be particularly surprising, especially given the results of multiple CSIDH papers mentioned in Chapter 3 such as the Falafl signature scheme. The results are also broadly in line with computation found in CSI-FiSh [42] where there are noticeable differences, such as HKZ under the l_2 norm being the quickest, but none to suggest a radically efficient computation.

As a comparison, the CIRCL implementation of generating the exponent vector takes on an average of three runs $46.25\mu s$ where μs refers to a microsecond (10^{-6} of a second). Clearly, this is a much more desirable run-time. However, speed is not the only factor here. CIRCL implements CSIDH exponent vector by random sampling for each dimension of the vector, so it $46.25\mu s$ is effectively the result of 74 random sampling. The structure of the CSIDH private key is also rather different in CIRCL. Instead of a random vector length 74, CIRCL leaves half of the vector as a buffer using fpRngGen and the other half as the exponent vector. It is clear that this approach of vector generation is substantially quicker, but this implementation does not allow for the key feature of using an integer as the secret key and it is not clear how to incorporate the different structure into other applications of CSIDH.

A particular case of where this project's implemented feature is desirable is when im-

plementing the distributed key generation algorithm CSI-RAShi [43]. Indeed, this implementation was the original aim of the thesis using only the CIRCL CSIDH implementation. However, using a randomly generated exponent vector directly as the secret meant that it was impossible to perform calculations such as $sd \mod N$ where s is the secret and d is some element from a group $\mathbb{Z}_N$ by nature of using a vector to perform modular arithmetic. Indeed, it was this lack of feature that prompted a change of thesis direction to implement random sampling of an element and turning it into the exponent vector.

However, if this project's implementation was used to implement CSI-RAShi, the run time would be far from practical. CSI-RAShi required that each party participating in the threshold scheme to sample λ number of secrets where λ is the security parameter (CSI-RAShi proposed it to be 128). In these multiple secrets, each party would have to convert them into a exponent vector for CSIDH group action m times and use the secret integer for modular arithmetic too. This cost can quickly add up, and assuming the lattice reduction speed of 30 seconds per secret, it will take one party more than 64 minutes to perform one part of the CSI-RAShi algorithm. Other CSIDH-based digital signatures also do not fare well in terms of efficiency. The SeaSign digital signature on average takes more than 9 minutes to sign, and 3 minutes to verify a signature [48]. By contrast, a current ECDH distributed key generation scheme implemented in [46] takes 0.3 seconds to run a key generation scheme for 20 parties.

The results can also be compared to other post-quantum standards to get a better sense of how CSIDH lattice reduction stacks up. A natural consideration would be SIDH and lattice-based cryptography due to the mathematical structures involved. In SIDH-sign, an SIDH based signature scheme proposed by [32], signing takes a median of 5139 millisecond (10^{-3} seconds) to sign for an SIDH parameter that was (previously) conjectured to give NIST Level 1 security. For the lattice-based digital signature CRYSTALS-Dilithium with an NIST Level 3 security proposal [39], a recent implementation was able to sign 15832 messages per second and verify 10524 signatures per second. It is quite clear that the CSIDH lattice reduction algorithm is far off the superior run times of these post-quantum signature schemes.

Having examined the results if implemented for CSI-RAShi and comparing it to the implementation done by CIRCL and lattice-based cryptography, we should also note that this has only been an implementation using CSIDH-512. As discussed in previous chapters, the security of CSIDH is an open problem. While the exact standards are still under question, it would be important to consider higher levels of security against an adversary even if just conjectural. In [28], a new parameter of CSIDH-4096 is proposed to meet the NIST Level 1 security, CSIDH-6144 for Level 2, and CSIDH-8192 for Level 3 assuming the attacks by [7] held up. Though it is uncertain how lattice reduction performs exactly for higher parameters of CSIDH, it is not unreasonable to guess that speed could go well into several minutes for the most basic task in CSIDH given our results.

Seeing the impractical results, we should consider if there are solutions to improve it. As discussed in Chapter 3, there are multiple proposals that aim to quicken CSIDH computation. However, these proposals seem to focus on improving the group action itself, instead of finding more efficient ways to convert an integer into an exponent vector. For instance, in [58], the improvement techniques revolved around elliptic curves arithmetic, and the same focus applies for [26], [9], [51], [33], It seems that the functionality of sampling secrets from a group of integers may not be the focus of current research so there is quite a limited scope to suggest possible improvements. A constant time proposal for CSIDH, CTIDH, has been evaluated in [25]. However, again the focus is also on strategies for evaluating group action, and this parameter set is not compatible with other implementations of CSIDH. The only follow-up work to CSI-FiSh that the author is aware of is in [16]. However, this work largely examines the theoretical security of the digital signature scheme. The only suggestion for improved lattice performance was noting that optimizations may be specific to CSI-FiSh such as halving the number of CVP problems for key generation.

Based on our results, we can see that lattice reduction algorithms for CSIDH allows a very important feature, but puts it out of practical implementation. Compared to other implementations of CSIDH-512, it is vastly slower and its out-shined by other post-quantum cryptographic standars with much faster run times. Even with current proposals to improve CSIDH speed, a lot of work still needs to be done to make CSIDH a viable contender against other post-quantum standards and current DH implementations.

Chapter 6

Conclusion

In this chapter, we will summarize the research project and provide directions for future works based on our findings.

6.1 Summary

Post-quantum cryptography is a complex field that contains vastly different approaches. In this project, we have examined only one technique for one particular standard that relies on one particular mathematical structure. Though the threat of viable quantum computers may be some years away, ensuring the security of post-quantum cryptography schemes alongside examining its viability is a task that needs plenty of years to do.

In this research project, we have explored the underlying mathematics of current public-key cryptography. By giving an overview of quantum computing, it is clear how they can break the hardness assumption of current public-key cryptography. Having seen the current proposals for post-quantum cryptography, this paper explores the mathematics behind isogenies and lattices.

With sufficient mathematical background, the project then discusses related work for CSIDH and lattice reduction, noting that current research in this area spans from theoretical to implementation in applications such as digital signatures and oblivious transfers. For lattice reduction, we see that despite being a fairly old mathematical technique, still finds application in linear detectors and solving certain lattice enumerations.

Afterwards, an in-depth examination of CSIDH is laid out. Having seen how CSIDH operates, we examine the relevant lattice reduction algorithms for this project and tie the two together by explaining how to use lattice reduction algorithms to convert a secret integer to the CSIDH secret key. The implementation is done on Go, and we discuss the results.

We conclude that lattice reduction algorithms are a very slow approach to implement this CSIDH functionality and is vastly inferior with the CIRCL CSIDH approach,

CRYSTALS-Dilithium, and current ECDH run time. However, random sampling of a secret integer is a key component in several distributed cryptography tasks. With this approach being the only one to implement the feature, may be a bitter pill to accept for attempts to implement some CSIDH distributed tasks though this pill need only be swallowed once as it is only needed for key generation which needs to be done only once for each party.

6.2 Future Directions

The results of this project leads to several natural extensions and future directions. In the former, improving the performance of lattice reduction is a most pressing issue. One could consider different more sets of bases and norms alongside a combinations of lattice reduction algorithms. Several variants of lattice reductions have an approximation parameter and some have a limit on the number of iterations to perform at a much better speed. Based on related works, there are studies that apply truncated versions of lattice reductions that seem to perform better compared to other techniques such as extreme pruning. The insights of these methods may be applicable to improving lattice reduction in our scenario.

Another future work that is worth exploring is finding if there are other methods to convert a secret integer into the CSIDH exponent vector. As far as the author is aware, CSI-FiSh is the only scheme to implement this conversion and other papers have instead focused on efficient group action or assumed that there is some procedure to work with both the secret as an integer and as an exponent vector needed in [43]. Discovering alternative methods with faster run time would go a long way to making certain CSIDH-based schemes more attractive.

Bibliography

[1] M. Abdalla. "password-authenticated key exchange from group actions.". *Cryptology EPrint Archive*, 16 June 2022.

[2] e. a. Adj, Gora. "karatsuba-based square-root vélu's formulas applied to two isogeny-based protocols.". *Cryptology EPrint Archive*, 5 Sept. 2021.

[3] M. Alvila. "a performance evaluation of post-quantum cryptography in the signal protocol.". *DIVA*, 27 June 2019,.

[4] H. W. L. Arjen Klaas Lenstra and L. Lov´asz. "factoring polynomials with rational coefficients". *Mathematische Annalen*, 1982.

[5] W. Beullens. "breaking rainbow takes a weekend on a laptop.". *Cryptology EPrint Archive*, 21 June 2022.

[6] X. Bonnetain and A. Schrottenloher. "quantum security analysis of csidh.". *SpringerLink*, 1 Jan. 2022.

[7] X. Bonnetain and A. Schrottenloher. "quantum security analysis of csidh and ordinary isogeny-based schemes.". *Semantic Scholar*, 17 Nov. 2020.

[8] W. Castryck and T. Decru. "an efficient key recovery attack on sidh (preliminary version).". *Cryptology EPrint Archive*, 5 Aug. 2022.

[9] J.-J. Chi-Domínguez and K. Reijnders. "fully projective radical isogenies in constant-time.". *Cryptology EPrint Archive*, 2 Dec. 2021.

[10] J.-J. Chi-Domínguez and F. Rodríguez-Henríquez. "optimal strategies for csidh.". *Cryptology EPrint Archive*, 5 Aug. 2020.

[11] A. Childs. "lecture 2: The hsp and shor's algorithm for discrete log". *University of Waterloo*, 2008.

[12] J.-M. Couveignes. "hard homogenous spaces". *Cryptology ePrint Archive*, 24 Aug 2006.

[13] D. Cozzo and N. P. Smart. "sashimi: Cutting up csi-fish secret keys to produce an actively secure distributed signing protocol.". *SpringerLink*, 10 Apr. 2020.

[14] L. A. de Castro et al. "an introduction to quantum measurements with a historical motivation". *ArXiv.org*, 11 Aug. 2019.

[15] C. D. de Sain Guilhem et al. "semi-commutative masking: A framework for isogeny-based protocols, with an application to fully secure two-round isogeny-based ot.". *Cryptology EPrint Archive*, 19 Sept. 2020.

[16] A. E. K. et al. "lossy csi-fish: Efficient signature scheme with tight reduction to decisional csidh-512 ". *Cryptology ePrint Archive*, 11 June 2020.

[17] C. P. et al. "csiibs: A post-quantum identity-based signature scheme based on isogenies.". *Journal of Information Security and Applications*, 12 Aug. 2020.

[18] D. B. et al. "quantum circuits for the csidh: Optimizing quantum evaluation of isogenies.". *Cryptology EPrint Archive*, 5 Mar. 2019.

[19] D. C. et al. "toward a scalable, silicon-based quantum computing". Dec. 2003.

[20] D. H. et al. "on the performance analysis for csidh-based cryptosystems.". *MDPI*, 2 Oct. 2020.

[21] D. H. et al. "optimized csidh implementation using a 2-torsion point.". *Cryptology EPrint Archive*, 28 Apr. 2020.

[22] D. J. et al. "two quantum ising algorithms for the shortest vector problem". *Researchgate*, June 2020.

[23] F. C. et al. "safe-error attacks on sike and csidh.". *Cryptology EPrint Archive*, 22 Nov. 2021.

[24] G. B. et al. "efficient supersingularity testing over $\mathcal{O}_p$ and csidh key validation.". *Cryptology EPrint Archive*, 26 July 2022.

[25] G. B. et al. "ctidh: Faster constant-time csidh.". *Cryptology EPrint Archive*, 26 May 2021.

[26] H. C. et al. "batching csidh group actions using avx-512.". *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 11 Aug. 2021.

[27] H. K. et al. "low-complexity lattice reduction algorithm for mimo detectors with tree searching". *EURASIP Journal on Wireless Communications and Networking*, 18 Jan. 2017,.

[28] J. C.-S. et al. “the squale of csidh: Sublinear vélu quantum-resistant isogeny action with low exponents”. *SpringerLink*, 31 Aug. 2021.

[29] J. F. et al. “a kilobit hidden snfs discrete logarithm computation”. *Cryptology ePrint Archive*, 5 Nov 2016.

[30] J.-F. B. et al. “a trade-off between classical and quantum circuit size for an attack against csidh.”. *De Gruyter*, 17 Nov. 2020.

[31] J.-F. B. et al. “a note on the security of csidh.”. *SpringerLink*, 5 Dec. 2018.

[32] J.-J. C.-D. et al. “sidh-sign: an efficient sidh pok-based signature”. *Cryptology ePrint Archive*, 25 April 2022.

[33] J. L. et al. “further optimizations of csidh: A systematic approach to efficient strategies, permutations, and bound vectors”. *Cryptology EPrint Archive*, 4 Aug. 2020.

[34] K.-M. C. et al. “group signatures and accountable ring signatures from isogeny-based assumptions.”. *Cryptology EPrint Archive*, 17 Apr. 2022.

[35] L. D. F. et al. “towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies”. *Cryptology ePrint Archive*, 2011.

[36] L. L. et al. “lattice enumeration with discrete pruning: Improvement, cost estimation and optimal parameters.”. *Cryptology EPrint Archive*, 17 Aug. 2022.

[37] N. A. et al. “two-round adaptively secure mpc from isogenies, lpn, or cdh.”. *SpringerLink*, 1 Dec. 2021.

[38] R. R. et al. “a method for obtaining digital signatures and public-key.”. 1977.

[39] S. R. et al. “implementing crystals-dilithium signature scheme on fpgas”. *Cryptology ePrint Archive*, 1 February 2021.

[40] T. D. et al. “faster seasign signatures through improved rejection sampling”. *Cryptology EPrint Archive*, 25 Jan. 2019.

[41] T. M. et al. “sigamal: A supersingular isogeny-based pke and its application to a prf.”. *Cryptology EPrint Archive*, 6 Nov. 2020.

[42] W. B. et al. “csi-fish: Efficient isogeny based signatures through class group computations.”. *Cryptology EPrint Archive*, 20 May 2019.

[43] W. B. et al. “csi-rashi: Distributed key generation for csidh.”. *Cryptology EPrint Archive*, 23 Oct. 2020,.

[44] W. B. et al. “getting ready for post-quantum cryptography: Exploring challenges associated with adopting and using post-quantum cryptographic algorithms.”. *CSRC*, 28 Apr. 2021.

[45] W. B. et al. “calamari and falafl: Logarithmic (linkable) ring signatures from isogenies and lattices.”. *Cryptology EPrint Archive*, 3 June 2020.

[46] Y. L. et al. “fast secure multiparty ecdsa with practical distributed key generation and applications to cryptocurrency custody”. *Cryptology ePrint Archive*, 18 October 2018.

[47] M. M. et al 2019. “on lions and elligators: An efficient constant-time implementation of csidh.”. *Cryptology EPrint Archive*, 12 Feb. 2019.

[48] L. D. Feo and S. D. Galbraith. “seasign: Compact isogeny signatures from class group actions.”. *Cryptology EPrint Archive*, 20 May 2019.

[49] L. D. Feo and M. Meyer. “threshold schemes from isogeny assumptions.”. *Springer-Link*, 29 Apr. 2020.

[50] A. Fujioka. “one-round authenticated group key exchange from isogenies.”. *Springer-Link*, 26 Sept. 2019.

[51] A. Jalali. “towards optimized and constant-time csidh on embedded devices.”. *Cryptology EPrint Archive*, 20 Mar. 2019.

[52] J. Katz and Y. Lindell. “introduction to modern cryptography”. *CRC Press*, 2021.

[53] T. Kleinjung. “on polynomial selection for the general number field sieve”. *Mathematics of Computation*, October 2006.

[54] Y.-F. Lai and S. Dobson. “collusion resistant revocable ring signatures and group signatures from hard homogeneous spaces.”. *Cryptology EPrint Archive*, 20 Nov. 2021.

[55] J. LeGrow and A. Hutchinson. “an analysis of fault attacks on csidh.”. *Cryptology EPrint Archive*, 24 Sept. 2020.

[56] J. Li and P. Q. Nguyen. “a complete analysis of the bkz lattice reduction algorithm.”. *Cryptology EPrint Archive*, 5 Mar. 2022.

[57] L. Maino and C. Martindale. “an attack on sidh with arbitrary starting curve”. *Cryptology EPrint Archive*, 5 Aug. 2022.

[58] M. Meyer and S. Reith. “ “a faster way to the csidh.””. *Cryptology EPrint Archive*, 11 Nov. 2018.

[59] D. Micciancio. “the lll algorithm”. *UCSD*, 2012.

[60] P. Miller. “learning fast elliptic-curve cryptography.”. 6 Apr. 2020.

[61] T. Moriya. “masked-degree sidh”. *Cryptology ePrint Archive*, 9 August 2022.

[62] M. A. Nielsen and I. L. Chuang. “quantum computation and quantum information”. *Cambridge University Press*, 2021.

[63] B. B. OBE. “supersingular isogeny diffie-hellman for key generation.”. *ASecuritySite*, 22 Apr. 2020.

[64] R. Pedersen. “decsidh: Delegating isogeny computations in the csidh setting.”. *Cryptology EPrint Archive*, 4 Jan. 2022.

[65] C. Peikert. “he gives c-sieves on the csidh”. *Cryptology EPrint Archive*, 24 Feb. 2020.

[66] A. Pellet-Mary. “co6gc: Finding short vectors in lattices.”. *COSIC*, 10 June 2020.

[67] O. Regev. “lecture 3 cvp algorithm”. *NYU*, 2004.

[68] D. Robert. “evaluating isogenies in polylogarithmic time.”. *Cryptology EPrint Archive*, 21 Aug. 2022.

[69] J. Russell. “hpcwire quantum survey: First up – ibm and zapata – on algorithms, error mitigation, more.”. *HPCwire*, 18 Aug. 2022.

[70] A. Sanso. “a note on key control in csidh.”. *Cryptology EPrint Archive*, 1 Aug. 2022.

[71] S. Schmittner. “file:cvp.svg.””. *Wikimedia Commons*, October 2015.

[72] P. Shor. “algorithms for quantum computation: Discrete logarithms and factoring.”. *IEEE Xplore*, 20 Nov. 1994.

[73] A. Sutherland. “lecture 5 elliptic curves”. *MIT*, 2015.

[74] B. Wesolowski. “orientations and the supersingular endomorphism ring problem.”. *Cryptology EPrint Archive*, 21 Mar. 2021.

[75] C. M. L. P. Wouter Castryck, Tanja Lange and J. Renes. “csidh: An efficient post-quantum commutative group action.”. *Cryptology EPrint Archive*, 30 Apr. 2018.

Appendix A

Title of first appendix

A.1 List of Codes

The full code can be found in (https://github.com/brykumara/CSI-RAShi). Here are some key components from the implementation on Go:

```
import (
        c "crypto/rand"
        "fmt"
        "math/big"
        "strconv"

        "github.com/brykumara/circlclone/csidh"
)

var PrivateKeySize = 37
var ExponentVectorLength = 74
var rng = c.Reader

const (
    prec = 100000
    Prime float64 = 37 * 1407181 *
        51593604295295867744293584889 *
        3159941450468199585300827874558783220490 9

)

func main() {
        start := time.Now()
```

```
        Secret := SampleSecret(Prime)
        Vec := Secret2Vec(Secret)
        elapsed := time.Since(start)
        fmt.Println("Time: ", elapsed)
        fmt.Println(Vec)
}

func SampleSecret(Prime float64) float64 {
        var prime = (int64)(Prime)
        secret, err := c.Int(rng, big.NewInt(prime))
        if err != nil {
                panic(err)
        }
        Secrets := secret.Int64()
        var Secret float64 = (float64)(Secrets)
        return Secret
}

func Secret2Vec(secret float64) []float64 {
        Target := make([]float64, csidh.PrimeCount)
        for i := 1; i < csidh.PrimeCount; i++ {
                Target[i] = 0
        }
        Target[0] = secret //
        C := make([]float64, csidh.PrimeCount)
        C[0] = secret
        B := make([]float64, csidh.PrimeCount*csidh.PrimeCount)
        // Babai Nearest Plane
        for i := (csidh.PrimeCount - 1); i >= 0; i-- {
                for j := 0; j < len(B); j++ {
                        B[j] = B[j] + (float64)(i*74)
                }
                TargetxB := Innerproduct(Target, B)
                for j := 0; j < len(B); j++ {
                        B[j] = 0
                } //
                HKZIPS, _ := strconv.ParseFloat(HKZIPStrings[i],
                64)
                ip1 := new(big.Float).SetPrec(prec).
                Quo(big.NewFloat(TargetxB), big.NewFloat(HKZIPS))
```

```
                if ip1.Sign() < 0 {
                        delta := -0.5
                        ip1.Add(ip1, new(big.Float)
                        .SetFloat64(delta))
                }
                if ip1.Sign() < 0 {
                        delta := 0.5
                        ip1.Add(ip1, new(big.Float)
                        .SetFloat64(delta))
                }
                ip1int, _ := ip1.Int(nil)
                ip1floor := new(big.Float).SetInt(ip1int)
                remainder := new(big.Float).SetPrec(prec).
                Sub(ip1, ip1floor)
                if remainder.Cmp(big.NewFloat(0.5)) > 0 {
                        ip1floor = ip1floor.Add(ip1floor,
                        new(big.Float).
                        SetFloat64(1))
                }
                r, _ := ip1floor.Float64()
                A := make([]float64,
                csidh.PrimeCount*csidh.PrimeCount)
                for j := 0; j < len(A); j++ {
                        A[j] = A[j] + (float64)(i*74)
                }
                for j := 0; j < len(C); j++ {
                        C[j] = C[j] - A[j]*r
                }

        }
        Vec := C
        Reduce(Vec, 2, 10000)
        return Vec
}

func Reduce32(vec []float64, pool_vectors float64) {
        var norm = l_1NormforOneVec(vec)
        var counter = 0
        for {
```

```
                var change = 0
                for i := 0; i < (int)(pool_vectors); i++ {
                        for j := 0; j < len(pool); j++ {
                                pool[j] = pool[j] + (int32)(i*K)
                        }
                        poolconv := make([]float64, len(pool))
                        for n := 0; n < len(poolconv); n++ {
                                poolconv[n] = (float64)(pool[n])
                        }
                        var plus_norm =
                        l_1NormSumforTwoVec(vec, poolconv)
                        if plus_norm < norm {
                                norm = plus_norm
                                counter += 1
                                AddVec(vec, poolconv)
                                change = 1
                        }
                        var minus_norm =
                        l_1NormDiffforTwoVec(vec, poolconv)
                        if minus_norm < norm {
                                norm = minus_norm
                                counter += 1
                                SubVec(vec, poolconv)
                                change = 1
                        }
                }
                if change == 0 {
                        return
                }
        }
}
}

func Reduce(vec []float64, pool_vectors, trials float64) {
        var VEC = make([]float64, K)
        for i := 0; i < K; i++ {
                VEC[i] = vec[i]
        }
```

```
        var Best = make([]float64, K)
        Reduce32(VEC, pool_vectors)

        Best = VEC
        var best_len = l_1NormforOneVec(VEC)

        for j := 1; j < (int)(trials); j++ {
                VEC = Best
                for n := 0; n < 2; n++ {
                        index, _ := c.Int(rng,
                        big.NewInt((int64)(rand_max)))
                        var num = index.Int64() % 10000
                        newpool := make([]float64, len(pool))
                        for k := 0; k < len(pool); k++ {
                                newpool[k] = (float64)(pool[k])
                                newpool[k] = newpool[k]
                                + (float64)(74*num)
                        }
                        VEC = AddVec(VEC, newpool)
                }
                Reduce32(VEC, pool_vectors)
                var norm = l_1NormforOneVec(VEC)
                if norm < best_len {
                        best_len = norm
                        Best = VEC
                }
        }

        for i := 0; i < K; i++ {
                vec[i] = Best[i]
        }
}
```