Implementing a Key Exchange Protocol using Isogenies on Elliptic Curves in Python

Thomas Schwartzenburg

Spring 2022

Project Advisor: Dr. Aaron Hutchinson

Abstract

The study of Elliptic Curves continues to reveal more secure and efficient methods to encrypt information. For example, in De Feo, Jao, and Plût's paper *Towards Quantum-Resistant Cryptosystems From Supersingular Elliptic Curve Isogenies*, they provide an abstract key exchange protocol using isogonies on supersingular elliptic curves. The main idea of this research is to present an accurate manifestation of the abstract protocol constructed by De Feo et al., which will be realized using Python. This protocol is founded in the Diffie-Hellman algorithm using isogenies on Elliptic Curves as its primary operation. Background on Elliptic Curve Cryptography will also be provided in order to fully understand the presented encryption process.

Contents

1 Introduction

Encrypted Mesopotamian tablets containing recipes for pottery glaze have been dated as far back as 1500 BC [12]. More recently, in 2020, over two billion people purchased a good or a service online [3], which requires encrypting their personal and billing information. Regardless of how many centuries have passed, the heart of cryptography has stayed the same — studying how to hide information in plain sight. With the 21st century's growing dependence on computer-to-computer communication, the importance of efficient and robust cryptographic schemes is at an all-time high.

Mathematicians and computer scientists have been able to deliver satisfactory cryptosystems in the pre-quantum computing era. However, research shows that quantum computing would make many of the most common cryptographic schemes, such as RSA encryption and basic Elliptic Curve Cryptography, obsolete. In order to prepare for the forthcoming quantum computer, much research on quantum-resistant cryptography has been conducted. One such paper is De Feo, Jao, and Plût's *Towards Quantum-Resistant Cryptosystems From Supersingular Elliptic Curve Isogenies* [6]. In that paper an abstract key-exchange protocol is provided that uses isogonies on supersingular elliptic curves. Encryptions manufactured by this protocol would be unfeasible to decrypt, even in a post-quantum age, which is a feat most cryptographic schemes commonly used today could not claim.

In this paper, the key exchange protocol proposed by De Feo et al. will be implemented using Python. In Section 2 all necessary mathematical background will be covered, and Section 3 will provide needed cryptographic background. Section 4 will discuss the intricacies of the quantum-resistant protocol. In addition to this, Section 5 details the structure used in order to implement the protocol in Python.

2 Mathematical Background

In this section, some necessary mathematical background required for understanding De Feo et al.'s abstract protocol will be presented.

2.1 Basic Definitions and Notations

This section contains the definitions and notations of terms that will be used throughout the paper, broken down into three subsections: field definitions, group theory definitions, and miscellaneous.

2.1.1 Finite Field Definitions

Most operations within this protocol will be done over a finite field. This makes knowing and understanding the definitions pertaining to finite fields very valuable.

Definition 1. A **finite field** is a field with finitely many elements. For a prime p and positive integer k, let $\mathbb{F}_{p^k}$ denote a field with p^k many elements.

In this paper, we take

- $\mathbb{F}_p := \mathbb{Z}/p\mathbb{Z}$
- $\mathbb{F}_{p^2} := \mathbb{F}_p[i]/\langle i^2+1\rangle$, the quotient of the polynomial ring $\mathbb{F}_p[i]$ over $\mathbb{F}_p$ by the ideal generated by i^2+1. The prime p in $\mathbb{F}_{p^2}$ must satisfy $p \equiv 3 \mod 4$.

Notation 1. The algebraic closure of a field $\mathbb{F}$ is notated as $\overline{\mathbb{F}}$.

2.1.2 Group Theory Definitions

Most cryptographic schemes rely on unique properties of groups. Some basic definitions related to group theory that will be used throughout this paper are listed below.

Definition 2. A **subgroup** is a group that is contained within another group, where both groups have the same operation.

Definition 3. Let $(A, \star)$ and $(B, *)$ be groups. A function $\phi : A \to B$ is a **homomorphism** if for all $x, y \in A$ we have $\phi(x \star y) = \phi(x) * \phi(y)$.

Definition 4. A let $\phi : G \to G'$ be homomorphism. The map ϕ is an **isomorphism** if there exists an inverse homomorphism $\phi^{-1} : G' \to G$ such that for any $x \in G$, $\phi^{-1}(\phi(x)) = x$ and for any $y \in G'$, $\phi(\phi^{-1}(y)) = y$.

Definition 5. An **automorphism** is an isomorphism from a group to itself.

Definition 6. Let $(G, +)$ be an abelian group and let $x \in G$. Then the **subgroup generated by x** is the subgroup $\langle x\rangle = \{nx \in G : n \in \mathbb{Z}\}$ of G. We say that x **generates** the subgroup $\langle x\rangle$.

Definition 7. Let $\phi : G \to G'$ be a homomorphism of groups. The subgroup

$$\phi^{-1}(e') = \{x \in G : \phi(x) = e'\}$$

is the **kernel of** ϕ, denoted as $\ker \phi$.

2.1.3 Miscellaneous Definitions

Notation 2. The set of prime numbers is denoted as $\mathbb{P}$.

Definition 8. For $a, b \in \mathbb{F}_p$, the **discrete logarithm** of a to the base b is any integer k such that $b^k = a$. The notation $\log_b(a)$ is the integer k that has the smallest absolute value which satisfies $b^k = a$. While computing b^k is simple, if one is given a, b, and p, finding k is very difficult. This is called the **Discrete Logarithm Problem**.

2.2 Elliptic Curves

The first recorded mention of elliptic curves was in the second or third century AD by the Alexandrian mathematician Diophantus in his text *Arithmetica*. These curves were later popularized by Leonardo Fibonacci in the eleventh century. Various other well known mathematicians have researched elliptic curves, such as Claude Bachet, Leonhard Euler, and Pierre de Fermat. However, it was not until the late twentieth century when it was discovered that elliptic curves could be used cryptographically that they began to be researched heavily. The cryptographic importance of elliptic curves will be discussed further in Section 3.3. For more on the history of elliptic curves, [1] provides plenty of information on the subject.

As a note, the source material for much of this section and Section 3.3 came from Hoffestein, Pipher, and Silverman's *An Introduction to Mathematical Cryptography* [7]. This text is an excellent resource for learning more on elliptic curves and their applications, as well as various other topics in the field of cryptography.

2.2.1 Elliptic Curve Definition

This subsection contains the definition of an elliptic curve and some basic examples.

Definition 9. An **elliptic curve** (in short Weierstrass form[1]) over the field $\mathbb{F}$ is the collection of all points $(x, y) \in \overline{\mathbb{F}} \times \overline{\mathbb{F}}$ that satisfies the equation

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

in addition to the point at infinity $\mathcal{O}$, where $A, B \in \mathbb{F}$ such that

$$4A^3 + 27B^2 \neq 0$$

holds.

[1]There are other forms of elliptic curves, such as the Jacobian curve or the Montgomery curve, all of which are equivalent. For the scope of this paper, we will only concern ourselves with short Weierstrass curves.

The value $4A^3+27B^2$ is the discriminant of an elliptic curve. Ensuring that the discriminant is nonzero means that all three roots of the cubic that defines the curve are distinct. A curve with a zero discriminant has singular points, which hinders the addition law of elliptic curves. This law will be further discussed in Section 2.2.2.

For this paper, elliptic curves will be over the field $\mathbb{F}_{p^2}$. The subset of points of E which have coordinates in $\mathbb{F}_{p^2}$ will be denoted as

$$E(\mathbb{F}_{p^2}) = \{(x, y) \in \mathbb{F}_{p^2} \times \mathbb{F}_{p^2} : y^2 = x^3 + Ax + B\} \cup \{\mathcal{O}\}.$$

A few examples of elliptic curves over $\mathbb{R}$ are give in Figure 1. In Figures 2 and 3 the elliptic curve $y^2 = x^3 - 3x - 5$ is displayed, the only difference being that one is over $\mathbb{R}$ while the other is over $\mathbb{F}_{17}$. It becomes much more difficult to graph elliptic curves over the finite field, so visual examples of curves over $\mathbb{F}_{p^2}$ are omitted.

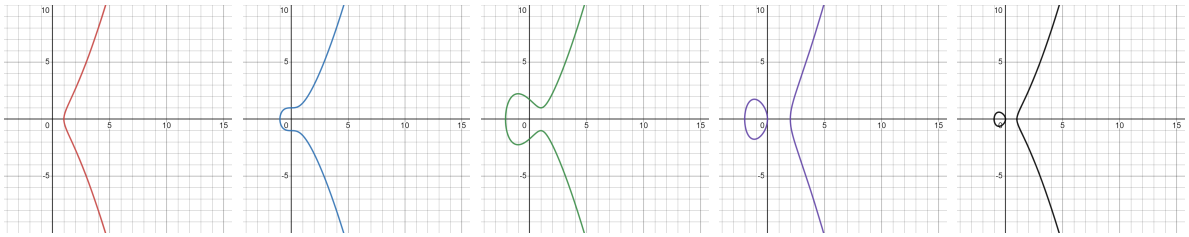

Figure 1: Various elliptic curves over $\mathbb{R}$. [2]

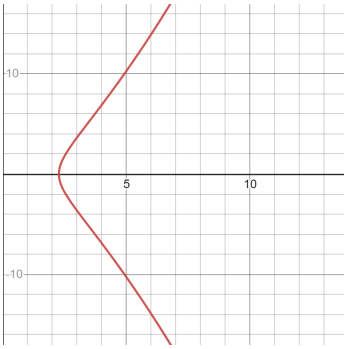

Figure 2: $y^2 = x^3 - 3x - 5$ over $\mathbb{R}$. [2]

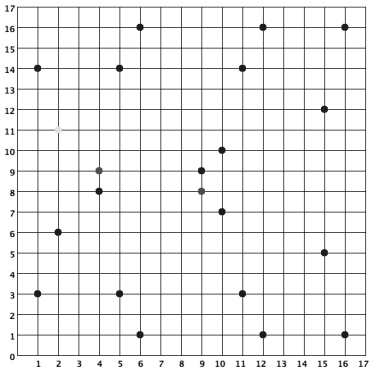

Figure 3: $y^2 = x^3 - 3x - 5$ over $\mathbb{F}_{17}$ [11]

2.2.2 Addition Law for Elliptic Curves

One of the characteristics that make elliptic curves so useful is their ability to function as an abelian group. This is made possible by the addition law of points on an elliptic curve, which is described in Theorem 1.

Theorem 1 (Elliptic Curve Addition Law)**.** *Let E be an elliptic curve. Then for each $P, Q, R \in E$, we have:*

(a)	$P + \mathcal{O} = P$	$[Identity]$
(b)	$P + (-P) = \mathcal{O}$	$[Inverse]$
(c)	$(P + Q) + R = P + (Q + R)$	$[Associative]$
(d)	$P + Q = Q + P$	$[Commutative]$

The heart behind the addition law over $\mathbb{F}_{p^2}$ is the same, but with a few minor clarifications. These are detailed in Theorem 2.

Theorem 2. *Let E be an elliptic curve over $\mathbb{F}_{p^2}$ and $P, Q \in E(\mathbb{F}_{p^2})$. Then*

(a) Adding P and Q results in a point in $E(\mathbb{F}_{p^2})$, denoted as $P + Q$.

(b) The addition law (Algorithm 1) on $E(\mathbb{F}_{p^2})$ makes $E(\mathbb{F}_{p^2})$ a finite group, fulfilling the properties in Theorem 1.

There are a series of formulas that allow us to add and subtract points on an elliptic curve. While derivations of these formulas will be left out, they are not very complex in nature, mostly relying on basic calculus. These formulas work over $\mathbb{R}$ and $\mathbb{F}_{p^2}$ (as well as other fields). Algorithm 1 shows how to add any two points on an elliptic curve.

Algorithm 1: Elliptic Curve Addition

Input: $P, Q \in E(\mathbb{F}_{p^2})$, *where E is an elliptic curve over $\mathbb{F}_{p^2}$.*
Output: $P + Q \in E(\mathbb{F}_{p^2})$

if $P = \mathcal{O}$ **then**
  Return Q
if $Q = \mathcal{O}$ **then**
  Return P
$P = (x_1, y_1)$ and $Q = (x_2, y_2)$
if $x_1 = x_2$ **and** $y_1 = -y_2$ **then**
  Return $\mathcal{O}$

$$\lambda = \begin{cases} \dfrac{y_1 - y_2}{x_1 - x_2} & \text{if} \quad P \neq Q \\ \dfrac{3x_1^2 + A}{2y_1} & \text{if} \quad P = Q \end{cases}$$

$x_3 = \lambda^2 - x_1 - x_2$
$y_3 = \lambda(x_1 - x_3) - y_1$
Return (x_3, y_3)

Example 1. Consider the curve $E : y^2 = x^3 + 6x + 7$ over the field $\mathbb{F}_{11}$. Points $P = (2, 7)$ and $Q = (9, 8)$ are both in $E(\mathbb{F}_{11})$. The arithmetic involved in performing the calculation is omitted, but following along with Algorithm 1 will result in $P + Q = (9, 3)$, which is in also in $E(\mathbb{F}_{11})$, as expected.

When dealing with an elliptic curve E, it is common to see the notation nP, where $n \in \mathbb{Z}$ and $P \in E$, which describes

$$nP = \begin{cases} \mathcal{O}, & \text{if } n = 0 \\ P + P + P + \cdots + P(\text{n times}), & \text{if } n > 0 \\ -(P + P + P + \cdots + P)(\text{-n times}), & \text{if } n < 0 \end{cases}$$

Algorithm 1 would certainly be able to accomplish this task by computing $R = 2P$ and then iteratively adding P to R. However, Algorithm 2, which is commonly referred to as the Double-and-Add algorithm, accomplishes the computation of nP much more efficiently by leveraging binary numbers.

Algorithm 2: Double-and-Add

Input: $a \in \mathbb{Z}$; $P \in G$.
Output: $aP \in G$

Let $a_0, \ldots, a_n$ be the bits of a so that $a = \sum_{i=0}^{n} a_i 2^i$
$Q \leftarrow id_G$
for $i = 0$ *to* n **do**
    $Q \leftarrow 2Q$
    if $a_{n-i} = 1$ **then**
        $Q \leftarrow Q + P$
Return Q

2.2.3 j-Invariants of Elliptic Curves

An invariant is a value of a certain mathematical structure that remains unchanged regardless of the transformations performed on the structure [14]. A common invariant used for elliptic curves is the j-invariant, a value that is preserved through isomorphisms. The j-invariant and its properties will be further explored in the following definitions and theorems.

As a note, much of the content from this section as well as Section 2.2.5 and Section 2.3 comes from lectures within Sutherland's course on elliptic curves at the Massachusetts Institute of Technology [10].

Definition 10. The j**-invariant** of an elliptic curve $E : y^2 = x^3 + Ax + B$ over the field $\mathbb{F}$ is

$$j(E) = j(A, B) = 1728\frac{4A^3}{4A^3 + 27B^2}$$

such that $j(E) \in \mathbb{F}$.

It is important to note that $j(E)$ will never be undefined, since $4A^3+27B^2$ is the discriminant of an elliptic curve, which, by definition, cannot be zero.

The j-invariant of an elliptic curve is closely related to isomorphisms between elliptic curves, which will be discussed in Theorem 5 once some more background information has been acquired. However, for now it is important to know that j-invariants are very valuable in De Feo, Jao, and Plût's key exchange protocol.

2.2.4 n-Torsion Subgroups

A specific kind of subgroup related to elliptic curves is the n-torsion subgroup. The structure of these groups provide a lot of insight into structure of the elliptic curve they are on. For this paper, n-torsion subgroups are notable because they are necessary for the construction of isogenies. In this section the definition of an n-torsion subgroup will provided, as well as a few of their properties.

Definition 11. The n**-torsion subgroup** of an elliptic curve E over the field $\mathbb{F}$ is the set $\{P \in E(\overline{\mathbb{F}}) : nP = \mathcal{O}\}$, where n is a positive integer. An n-torsion subgroup of an elliptic curve E is notated as $E[n]$.

As stated above, the structure of torsion subgroups can provide details on the structure of elliptic curves. The underlying structure of n-torsion subgroups is shown in Theorem 3.

Theorem 3. *Let E be an elliptic curve over $\mathbb{F}$ of characteristic p. If $\ell \neq p$ is a prime number, then*

$$E[\ell^e] \cong \mathbb{Z}/\ell^e\mathbb{Z} \oplus \mathbb{Z}/\ell^e\mathbb{Z}.$$

Torsion subgroups can sometimes be difficult to visualize, so Example 2 is provided.

Example 2. For the elliptic curve $E : y^2 = x^3 - 11x + 14$ over $\mathbb{F}_{71^2}$,

$$E[3] = \{\mathcal{O}, (12, 30), (12, 41), (43{+}13i, 18{+}27i), (43{+}13i, 53{+}44i), (43{+}58i, 53{+}27i), (43 + 58i, 53 + 27i), (44, 29i), (44, 42i)\}.$$

The reader can validate that for any $P \in E[3]$, the value of $3P$ is $\mathcal{O}$. One can also see that $|E[3]| = 9$, which is consistent with Theorem 3.

2.2.5 Supersingular Elliptic Curves

Supersingular curves are used in cryptography for various reasons. One reason is that the number of points on a supersingular curve over $\mathbb{F}_p$ or $\mathbb{F}_{p^2}$ is always known. The definition of a supersingular elliptic curve and some of their properties are provided in this section.

Definition 12. Let E be an elliptic curve over the field $\mathbb{F}$ that has positive characteristic p. The curve E is said to be **supersingular** when

$$E[p] = \{\mathcal{O}\}.$$

When a curve is not supersingular, it is called **ordinary**.

Theorem 4. *Let E be an elliptic curve over $\mathbb{F}_p$. If E is supersingular, then $|E(\mathbb{F}_p)| = p+1$. If E is over $\mathbb{F}_{p^2}$, then $|E(\mathbb{F}_{p^2})| = (p+1)^2$.*

Theorem 4 details one of the unique properties that supersingular curves have, and why they are often used in cryptography. When discussing De Feo, Jao, and Plût's key exchange in Section 4, this theorem will be used to determine the number of points on the protocol's elliptic curve.

2.3 Isogenies

The foundation of De Feo, Jao, and Plût's key exchange protocol is the isogeny. Much of the following information on isogenies were found in [10].

2.3.1 Definition and Examples

While there are various ways to define an isogeny, Definition 13 will be how it is defined in this paper.

Definition 13. An **isogeny** $\phi : E \to E'$ of elliptic curves defined over $\mathbb{F}$ is a non-constant rational map that sends $\mathcal{O}$ on E to $\mathcal{O}$ on E'.

An alternate definition of an isogeny that provides more insight into their group structure is provided below. However, for this paper Definition 13 will primarily be used.

Definition 14. An **isogeny** $\phi : E \to E'$ of elliptic curves defined over $\mathbb{F}$ is a surjective morphism of curves that induces a group homomorphism $E(\overline{\mathbb{F}}) \to E'(\overline{\mathbb{F}})$.

In order to have a clearer understanding of isogenies, there are four examples provided. Examples 3, 4, and 5 are slightly trivial, while Example 6 is more consistent with isogenies one would see in use, and, thus, is more involved.

Example 3. Let E be an elliptic curve, and $\phi : E \to E$ be an isogeny defined as $\phi(x, y) = (x, y)$. The isogeny ϕ would be an isogeny from one curve to itself, which is often referred to as the **identity isogeny**. While not noteworthy, it does satisfy Definition 13.

Example 4. Let E, E' be elliptic curves and $\phi : E \to E'$ be an isogeny defined as $\phi(x, y) = \mathcal{O}$. Arguably, this isogeny is less notable than the one in Example 3. This isogeny is often called the **zero isogeny** because it maps all points from E to the identity element of E'.

Example 5. Let m be a positive integer and E be an elliptic curve. Let $\phi : E \to E$ be an isogeny such that $\phi(Q) = mQ$ for all $Q \in E(\mathbb{F})$. The isogeny ϕ maps every point on E to a multiple of itself on E. When $m = 0$ this isogeny is the zero isogeny, and, likewise, when $m = 1$ this isogeny is the identity isogeny.

Example 6. Let $E : y^2 = x^3 + 60x + 14$ and $E' : y^2 = x^3 + 41x + 40$ over the field $\mathbb{F}_{71^2}$. The isogeny $\phi : E \to E'$ is

$$\phi(x, y) = \left(\frac{x^2 - 46x + 18}{x - 46}, \frac{(x - 46)^2 - 18}{(x - 46)^2} y \right).$$

The point $P = (70, 33)$ lies on E, and $Q = \phi(P) = (53, 12)$, which lies on E'.

Definition 15. Given an isogeny $\phi : E \to E'$, we call E and E' **isogenous**. If $P \in E(\mathbb{F})$ and $Q = \phi(P)$, we say that Q is the **image** of P under ϕ.

Isogenies play a vital role in determining whether two elliptic curves are isomorphic. This is detailed in the following definition.

Definition 16. Elliptic curves E and E' defined over a field $\mathbb{F}$ are **isomorphic** if there exist isogenies $\phi_1 : E \to E'$ and $\phi_2 : E' \to E$ whose composition is the identity. The isogenies ϕ_1 and ϕ_2 function as **isomorphisms**.

An elliptic curve's j-invariant, as mentioned in Section 2.2.3, plays an important role in isomorphisms between curves, as detailed in the following theorem.

Theorem 5. *Let E and E' be elliptic curves over $\mathbb{F}$. Then E and E' are isomorphic over $\overline{\mathbb{F}}$ if and only if $j(E) = j(E')$.*

An important property of supersingular (or ordinary) elliptic curves is that they are invariant over isogenies. Theorem 6 provides more insight into this invariance.

Theorem 6. *Let $\phi : E \to E'$ be an isogeny of elliptic curves. Then E is supersingular if and only if E' is supersingular (and E is ordinary if and only if E' is ordinary).*

The previous theorem is invaluable to the key exchange protocol that will be implemented later in this paper.

2.3.2 Kernels of Isogenies

Every isogeny requires a subgroup of elliptic points in order to be created. Often times the subgroup used is the kernel of an isogeny.

Theorem 7. *Every nonzero isogeny $\phi : E \to E'$ can be rewritten as*

$$\phi(x, y) = \left(\frac{u(x)}{v(x)}, \frac{s(x)}{t(x)} y \right),$$

where u and v as well as s and t are pairs of relatively prime polynomials.

Theorem 7 does not contain any restrictions on the denominators (the functions $v(x)$ and $t(x)$) that would prevent them from being zero, implying that having a zero-denominator is acceptable. In the case of isogenies of elliptic curves, having zero denominators is permitted and actually produces the kernel of isogenies, as shown in the following definition.

Definition 17. Let ϕ be an isogeny written in the form shown in Theorem 7. The kernel of ϕ is defined as

$$\ker(\phi) = \{(x_0, y_0) \in E(\overline{\mathbb{F}}) : v(x_0) = 0\} \cup \{\mathcal{O}\}.$$

As with any kernel, the kernel of the isogeny $\phi : E \to E'$ is the collection of the points $P \in E$, such that when P is passed through ϕ, the outcome would be $\mathcal{O}$ on E'. The way this is described in mathematical notation is $\phi(P) = \mathcal{O}$.

Let E be an elliptic curve. It would be advantageous that if one were given a finite subgroup G of $E(\mathbb{F})$ that there would be an isogeny ϕ from E to another elliptic curve E', that had G as its kernel. Luckily, this is the case and is detailed in Theorem 8.

Theorem 8. *Let E be an elliptic curve over $\mathbb{F}$ and let G be a finite subgroup of $E(\overline{\mathbb{F}})$. There exists an elliptic curve E' and an isogeny $\phi : E \to E'$ with $\ker \phi = G$.*

The curve E' in Theorem 8 is unique up to isomorphism. We often use the notation E/G to describe the unique curve from Theorem 8.

How to construct ϕ and E' will be discussed in Section 2.3.3, where there will be certain specifications put on the subgroup G.

The following theorem expresses what it means for an elliptic curve and an isogeny to be unique up to isomorphism.

Theorem 9. *If $\phi_1 : E \to E$ and $\phi_2 : E \to E'$ are isogenies with $\ker(\phi_1) = \ker(\phi_2)$, then there exists an isomorphism $\alpha : E \to E'$ such that $\alpha \circ \phi_1 = \phi_2$.*

2.3.3 Vélu's Formulas

The construction of an isogeny from an elliptic curve and subgroup is a vital part of the key exchange protocol that is to be implemented in this paper. The following theorems provide not only a rule, but also a procedure on how to create these isogenies.

If the subgroup G that is going to be used to create an isogeny is of order two, then Theorem 10 is to be used. If G is of odd order, then Theorem 11 is to be used.

Let E be an elliptic curve given by the Weierstrass equations $y^2 = x^3 + Ax + B$ over some field $\mathbb{F}$, and let $p(x) = x^3 + Ax + B$ which factors into $p(x) = (x - r_1)(x - r_2)(x - r_3)$, where r_1, r_2, and r_3 are distinct roots of $p(x)$ in $\overline{\mathbb{F}}$. There exists a unique relationship between points of order two and the roots $p(x)$. The 2-torsion subgroup of E is

$$E[2] = \{\mathcal{O}, (r_1, 0), (r_2, 0), (r_3, 0)\}.$$

The roots of $p(x)$ are the only points in $E[2]$ (besides $\mathcal{O}$) because if a point is being doubled and the sum is $\mathcal{O}$, then the case must be that the y-coordinate is 0. The only points in $p(x)$ that will have a y-coordinate of 0 are the roots. For further clarification on the addition of the same point with $y = 0$, one can reference Algorithm 1, lines 5 through 7. A point $P \in E[2]$ generates the finite subgroup of $\{P, \mathcal{O}\}$. Having access to these order two points provides a simple way to find roots of the the cubic that defines an elliptic curve and will prove to be very useful in Theorem 10.

Theorem 10 (Vélu)**.** *Let $E : y^2 = x^3 + Ax + B$ be an elliptic curve over $\mathbb{F}$ and let $r \in \overline{\mathbb{F}}$ be a root of $x^3 + Ax + B$. Define $t := 3r^2 + A$ and $w := rt$. Then*

$$\phi(x, y) := \left(\frac{x^2 - rx + t}{x - r}, \frac{(x - r)^2 - t}{(x - r)^2} y \right)$$

is an isogney from E to $E' : y^2 = x^3 + A'x + B'$, where $A' := A - 5t$ and $B' := B - 7w$. The kernel of ϕ is the group of order 2 generated by $(r, 0)$.

The theorem for odd order points differs from the previous theorem because it does not require a root of the cubic function that defines the elliptic curve.

Theorem 11 (Vélu)**.** *Let $E : y^2 = x^3 + Ax + B$ be an elliptic curve over $\mathbb{F}$ and let G be a finite subgroup of $E(\overline{\mathbb{F}})$ of odd order. For each nonzero $Q = (x_Q, y_Q)$ in G define*

$$t_Q := 3x_Q^2 + A, \qquad u_Q := 2y_Q^2, \qquad w_Q := u_Q + t_Q x_Q$$

and let

$$t := \sum_{Q \in G_{\neq 0}} t_Q, \qquad w := \sum_{Q \in G_{\neq 0}} w_Q, \qquad r(x) := x + \sum_{Q \in G_{\neq 0}} \left(\frac{t_Q}{x - x_Q} + \frac{u_Q}{(x - x_Q)^2} \right).$$

Also let $r'(x)$ be the derivative of $r(x)$. Then

$$\phi(x, y) := (r(x), r'(x)y)$$

is an isogeny from E to $E' : y^2 = x^3 + A'x + B'$, where $A' := A - 5t$ and $B' := B - 7w$ with $\ker(\phi) = G$.

Vélu's Formulas seemingly only handle subgroups of order 2 and of odd order, but they can be used to construct isogenies with subgroups of any order. The method of doing this is discussed in Section 4.2.

This concludes the Mathematical Background section. The reader now has all the mathematical tools necessary to understand and perform De Feo, Jao, and Plût's key exchange protocol.

3 Cryptographic Background

This section discusses the topics that are required in order to have a basic understanding of how De Feo, Jao, and Plût's key exchange works cryptographically.

3.1 Trapdoor Functions

Trapdoor functions are widely used in modern cryptography. A function is considered to be a trapdoor if it is simple to compute but difficult to find the inverse of, without special information. The best trapdoor functions are those which finding the inverse is essentially impossible.

Functions that rely on the hardness of the discrete logarithm problem are not technically trapdoor functions since there is no special information to undo the trapdoor. These functions are particularly desirable in cryptographic fields. However, within the scope of this paper, these functions will be considered as trapdoor functions. The following subsections contain examples of trapdoor functions as well as a brief explanation as to why they are trapdoor functions.

3.1.1 Mixing Paint

While mixing paint is not typically considered to be a function, it serves well as an introductory example. Combining two paints to make a new color is a fairly easy task. However, if one is asked to separate a paint into the two colors it is composed of, the request feels near impossible.

3.1.2 Products of Primes

The process of multiplying two numbers is simple for a person to do. It is even more simple for a computer to do. However, the product of two primes (specifically two large primes) is difficult to break down into its prime factorization.

3.1.3 Addition of Elliptic Points

Similar to the previous trapdoor function, adding elliptic points is an easy task to accomplish. However, for an elliptic curve E, point $P \in E$, and integer a, if given P and aP, it is generally difficult to find a.

3.2 Diffie-Hellman Key Exchange Algorithm

The Diffie-Hellman key exchange algorithm was devised in 1978 by Merkle in his paper *Secure Communications Over Insecure Channels* [9]. The algorithm was heavily inspired by Whitfield Diffie and Martin Hellman's revolutionary idea of having a private key and a corresponding public key, as described in their paper *New Directions in Cryptography* [4].

The purpose of this algorithm is to have two entities arrive at a shared secret private key that is produced over an insecure channel. Once the private key is created, the two parties can then use the key to encrypt and decrypt information between themselves. The foundation of the Diffie-Hellman algorithm as well as an abstract performance of the algorithm is presented below.

3.2.1 Prerequisites for the Diffie-Hellman Algorithm

In order to perform the Diffie-Hellman Key Exchange Algorithm the following criteria must be met:

- There are two parties, Alice and Bob, who wish to have a shared secret key.
- Alice and Bob (as well as any one else) have access to some public information.
- Alice and Bob each have their own personal secret key.

3.2.2 Performing the Diffie-Hellman Algorithm

The Diffie-Hellman Key Exchange algorithm is performed as follows:

1. Alice will take her personal secret key and perform a trapdoor operation on the public information. Bob will do the same, except with his secret key. We will call Alice's value A, and Bob's value B.
2. Alice will send A to Bob, and Bob will send B to Alice. Both A and B will be visible to anyone who may be watching since this is done over an insecure channel, but since the operation performed is a trapdoor operation, their personal secret keys are safe.

3. Alice will take her personal secret key and perform a trapdoor operation on B. Similarly, Bob will do the same with A.

4. Now Alice and Bob have the same values, meaning that they have a shared secret key. They can now decrypt and encrypt messages between each other. The Diffie-Hellman algorithm is completed.

Two parties mixing paint is often used as an easy visualization of the Diffie-Hellman key exchange algorithm, as shown in Figure 4, but mixing paint has very little value when coming to encrypting and decrypting information. The following subsections will expand on the mathematical manifestations of the Diffie-Hellman protocol.

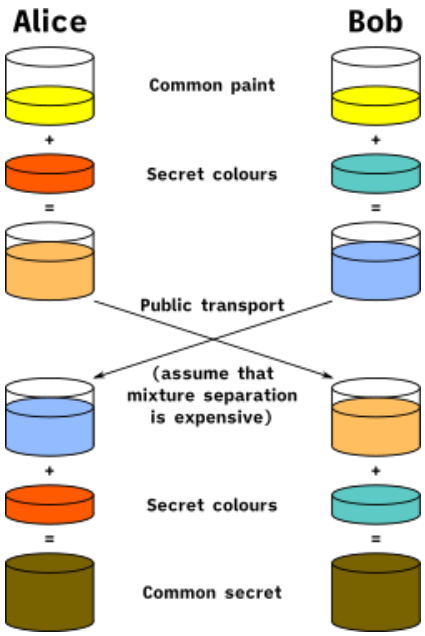

Figure 4: The Diffie-Hellman protocol shown using paint mixing as the trapdoor operation [13].

Another common example of the Diffie-Hellman algorithm is using integers instead of paint and modular arithmetic instead of paint-mixing. This particular Diffie-Hellman scheme is the backbone for RSA encryption. The algorithm is described below in Algorithm 3.

Algorithm 3: RSA Diffie-Hellman

Public Input: $p \in \mathbb{P}$, $g \in \mathbb{F}_p$

$\mathcal{A}$		$\mathcal{B}$
Private Input: $a \in \mathbb{N}$		**Private Input:** $b \in \mathbb{N}$
$A = g^a \mod p$		$B = g^b \mod p$
	$A \longrightarrow$	
	$\longleftarrow B$	
$A_B = B^a \mod p$		$B_A = A^b \mod p$
Return A_B		**Return** B_A

The shared secret key is $A_B = B^a = g^{ba} = g^{ab} = A^b = B_A$

3.3 Elliptic Curve Cryptography

Elliptic curve cryptography is most easily applied via the Diffie-Hellman algorithm. The public information that Alice, Bob, and any eavesdroppers have access to is the elliptic curve $E(\mathbb{F}_p)$, and the point $P \in E(\mathbb{F}_p)$. Alice and Bob's secret integers are $n_A, n_B \in \mathbb{F}_p$, respectively.

Next, Bob and Alice compute a new point using their secret integer. The process is as follows: $Q_A = n_A P$ and $Q_B = n_B P$. Then they will exchange points, so that Alice has Q_B and Bob has Q_A. Following the exchange, they will each compute another point as follows: $K_A = n_A Q_B$ and $K_B = n_B Q_A$.

Alice and Bob now share a secret key value that they can use to encrypt and decrypt information between themselves. The equivalence between points is shown below:

$$K_A = n_A Q_B = n_A n_B P = n_B n_A P = n_B Q_A = K_B$$

Table 1 serves as a summary of the Diffie-Hellman algorithm using elliptic curves.

Public parameter creation	
A trusted party chooses and publishes a (large) prime p, an elliptic curve E over $\mathbb{F}_p$, and a point P in $E(\mathbb{F}_p)$.	
Private computations	
Alice	**Bob**
Chooses a secret integer n_A. Computes the point $Q_A = n_A P$.	Chooses a secret integer n_B. Computes the point $Q_B = n_B P$.
Public exchange of values	
Alice sends Q_A to Bob $\longrightarrow$ Q_A	
Q_B $\longleftarrow$ Bob sends Q_B to Alice	
Further private computations	
Alice	**Bob**
Computes the point $n_A Q_B$.	Computes the point $n_B Q_A$.
The shared secret value is $n_A Q_B = n_A(n_B P) = n_B(n_A P) = n_B Q_A$.	

Table 1: Elliptic curve cryptography via Diffie-Hellman. [7]

4 De Feo, Jao, and Plût's Key Exchange

This section provides an overview of De Feo, Jao, and Plût's key exchange, as well as the strategies involved in producing it. This section also provides a brief explanation as to why this protocol is considered to be quantum-resistant.

4.1 Key Exchange Overview

De Feo, Jao, and Plût's key exchange is an extension of both the Diffie-Hellman algorithm and elliptic curve cryptography. The public information consists of the following:

- A prime number p of the form $2^a \cdot 3^b - 1$, with $a, b \in \mathbb{N}$
- A supersingular elliptic curve E_0 over the field $\mathbb{F}_{p^2}$.
- Linearly independent 2^a-order points P_A, Q_A
- Linearly independent 3^b-order points P_B, Q_B

The private information for Alice is $0 \leq m_A, n_A < 2^a$ such that m_A and n_A are not both divisible by 2. Bob's secret values are $0 \leq m_B, n_B < 3^b$ such that m_B and n_B are not both divisible by 3.

Once all the public information is set, the first round of calculations can be performed. Alice and Bob will construct their own isogenies, notated as $\phi_A : E_0 \to E_A$ and $\phi_B : E_0 \to E_B$. Recall, by Velu's formulas, that isogenies can be completely defined by their kernels. The subgroups for Alice and Bob's isogenies are $\ker(\phi_A) = \langle [m_A]P_A + [n_A]Q_A \rangle$ and $\ker(\phi_B) =$

$\langle [m_B]P_B + [n_B]Q_B \rangle$, respectively. Their isogenies will output the new curves E_A and E_B. Along with creating a new curve, Alice will compute the points $\phi_A(P_B)$ and $\phi_A(Q_B)$, while Bob computes $\phi_B(P_A)$ and $\phi_B(Q_A)$.

Now Alice and Bob exchange information over an insecure channel. Alice sends E_A, $\phi_A(P_B)$ and $\phi_A(Q_B)$ to Bob. Bob sends E_B, $\phi_B(P_A)$ and $\phi_B(Q_A)$ to Alice.

The second round of calculations will now be performed. Alice, who just received $\phi_B(P_A)$ and $\phi_B(Q_A)$, will construct the isogeny $\phi'_A : E_B \to E_{AB}$ using $\ker(\phi'_A) = \langle [m_A]\phi_B(P_A) + [n_A]\phi_B(Q_A) \rangle$. The isogeny ϕ'_A will produce the elliptic curve E_{AB}. Similarly, Bob will construct the isogeny $\phi'_B : E_A \to E_{BA}$ using $\ker(\phi'_B) = \langle [m_B]\phi_A(P_B) + [n_B]\phi_A(Q_B) \rangle$, which will produce the elliptic curve E_{BA}.

Lastly, Alice and Bob will compute the j-invariant of E_{AB} and E_{BA}, respectively. This value will be the same for both parties and will function as their shared private key. The equivalence of their j-invariants is due to the fact that

$$\ker(\phi_B \circ \phi_A) = \ker(\phi_A \circ \phi_B) = \langle m_A P_A + n_A Q_A, m_B P_B + n_B Q_B \rangle of E_0,$$

which by Theorem 9 we know that there exists an isomorphism between E_{BA} and E_{AB}. The existence of an isogeny isomorphism implies that the curves are isomorphic, meaning, by Theorem 5, they should have the same j-invariant.

Algorithm 4 is an algorithmic portrayal of the above description, while Figure 5 provides a visual for the key exchange protocol.

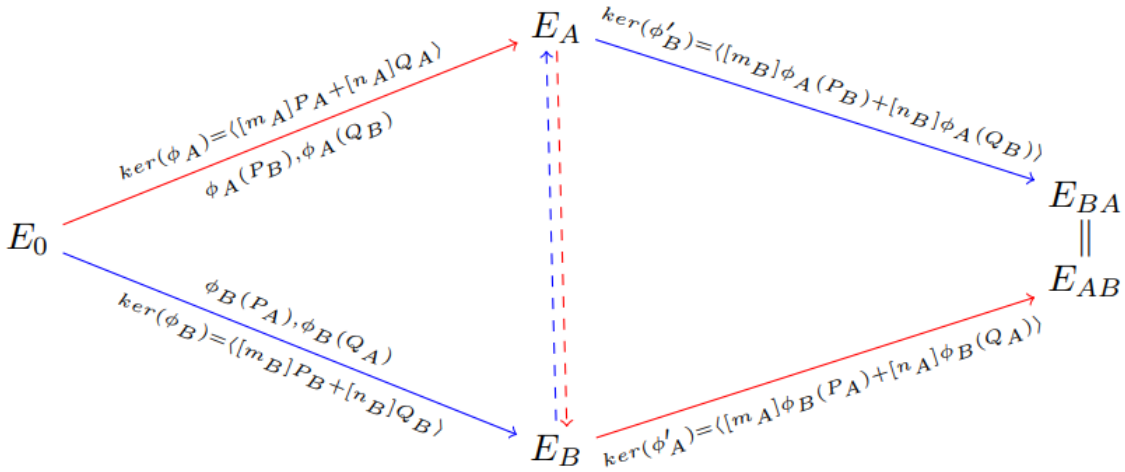

Figure 5: De Feo, Jao, and Plût's key exchange protocol using isogenies. [6]

Algorithm 4: De Feo, Jao, and Plût's Key Exchange Protocol

Public Input: $p = 2^a \cdot 3^b - 1 \in \mathbb{P}$, E_0, $\{P_A, Q_A\}$, $\{P_B, Q_B\}$

$\mathcal{A}$		$\mathcal{B}$
Private Input: $m_A, n_A \in \mathbb{F}_p$		**Private Input:** $m_B, n_B \in \mathbb{F}_p$
$R_A := [m_A]P_A + [n_A]Q_A$		$R_B := [m_B]P_B + [n_B]Q_B$
$\phi_A : E_0 \to E_A$ with $\ker(\phi_A) = \langle R_A \rangle$		$\phi_B : E_0 \to E_B$ with $\ker(\phi_B) = \langle R_B \rangle$
	$E_A, \phi_A(P_B), \phi_A(Q_B)$ $\longrightarrow$	
	$E_B, \phi_B(P_A), \phi_B(Q_A)$ $\longleftarrow$	
$R'_A := [m_A]\phi_B(P_A) + [n_A]\phi_B(Q_A)$		$R'_B := [m_B]\phi_A(P_B) + [n_B]\phi_A(Q_B)$
$E_{AB} := E_B/\langle R'_A \rangle$		$E_{BA} := E_A/\langle R'_B \rangle$
Return $j(E_{AB})$		**Return** $j(E_{BA})$

4.2 Strategies

Figure 5 is slightly deceptive because it appears as though ϕ_A (or ϕ_B) is the only isogeny used in order to get from curve E_0 to curve E_A. However, in order to be as computationally efficient as possible, ϕ_A is actually computed as a composition of a many degree two isogenies using Theorem 10. So the path from E_0 to E_A instead of looking like $E_0 \to E_A$, would look more like

$$E_0 \to E_1 \to E_2 \to \cdots \to E_{a-1} \to E_a = E_A.$$

The unique way of navigating through smaller isogenies is called a *strategy*.

It is best to construct an isogeny ϕ on the a curve E using the subgroup $\langle R \rangle$, where the point R is a sum of two specific points on E that are multiples of the user's personal secret key on E of order ℓ^e, where ℓ is a prime number and $e \in \mathbb{N}$. In Alice's case $\ell = 2$ and $e = a$, where for Bob $\ell = 3$ and $e = b$. Before creating a strategy, it is helpful to have a tree-structure to map out the isogenies and the points that would be computed, as shown in Figure 6. In order to make this structure let $E_0 = E$, $R_0 = R$, and, for $0 \leq i < e$, and let

$$E_{i+1} = E_i/\langle \ell^{e-i-1} R_i \rangle, \quad \phi_i : E_i \to E_{i+1} \text{ with } \ker(\phi_i) = \langle \ell^{e-i-1} R_i \rangle, \quad R_{i+1} = \phi_i(R_i).$$

Then $E/\langle R \rangle = E_e$ and $\phi = \phi_{e-1} \circ \cdots \circ \phi_0$.

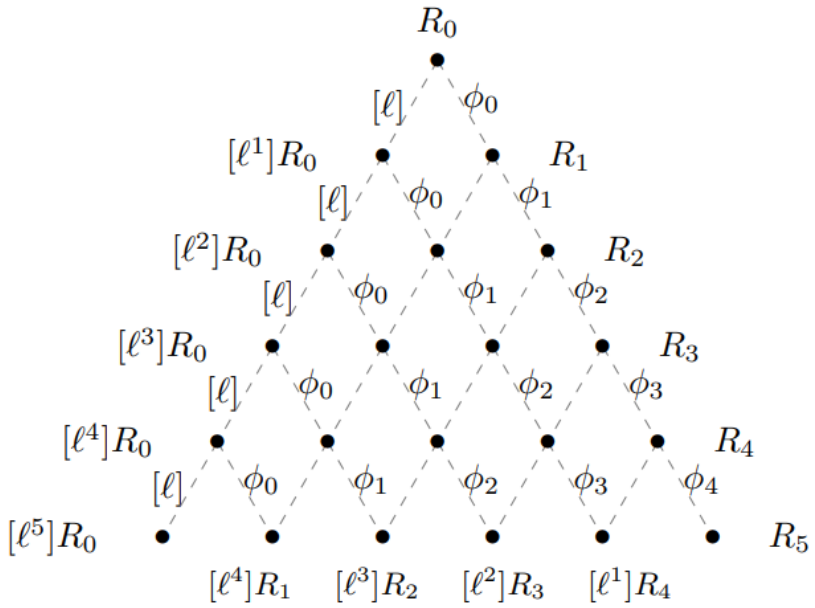

Figure 6: An example of a map of isogenies used to construct strategies. source of this figure is [6]

Using Vélu's formulas lead to the construction of E_{i+1} and the isogeny ϕ_i, as long as the ℓ-order subgroup $\langle \ell^{e-i-1} R_i \rangle$ of E_i is known.

In relation to Figure 6, a strategy is the collection of edges used in order to reach all of the bottom-most nodes. Some strategies are more efficient than others depending on the situation, but this is outside of the scope of this paper. The strategy used for this implementation is the *multiplication-based* strategy. This strategy is formed by computing one isogeny on the right-most diagonal, and then computing its multiples on the left diagonals until a leaf is reached. Then the next isogeny on the right-most diagonal will be computed, followed by computing its multiples on the left diagonals until a leaf is reached. This process will be repeated until all leaves have been computed. A diagram for a multiplication-based strategy is shown in Figure 7.

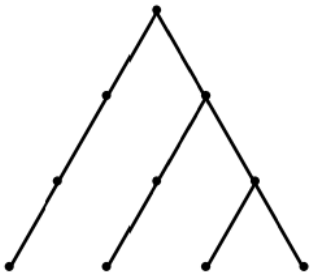

Figure 7: An example of a multiplication-based strategy. The source of this figure is [6]

4.3 Quantum-Resistant Properties

Quantum resistance of a key exchange protocol is a deep and vast area of study in its own right, so much of what makes (or does not make) a protocol resistant to quantum computers falls outside of the scope of this paper.

The foundation of traditional computers is binary. Currently, computers can only process a bit as a one or as a zero. In most instances this is no problem because they can process an astounding amount of bits each second. The creation of the quantum computer, however, is centered around the qubit, which is capable of superposition and entanglement [8]. This means that that the quantum computer would be capable of not just processing a one or a zero, but also both in a single moment, significantly enhancing their processing speed.

Current cryptographic schemes such as RSA and ECC rely on the Discrete Logarithm Problem in order to be safe from attackers. However, the greatly improved processing speed would allow quantum computers to brute-force these encryptions. De Feo, Jao, and Plût's key exchange does not rely on the Discrete Logarithm Problem, but rather the isogeny-based encryption is complex enough that a brute-force attempt by a quantum computer is still not feasible to accomplish.

5 Implementation

Now that all required background has been covered, the process of implementing De Feo, Jao, and Plût's key exchange protocol can be described. First, the parameter set used is discussed, followed by the way the protocol was realized in Python.

5.1 Parameter Set

The term *parameter set* is used to describe what values are used for the public information in a key exchange protocol. In the De Feo et al.'s protocol, the following is required: a prime number, a supersingular elliptic curve, and two sets of points. Supersingular Isogeny Key Encapsulation (SIKE) is a key encapsulation that performs the same protocol as this paper [5]. SIKE is a candidate for standardization by National Institute of Standards and Technology (NIST). SIKE provides their fully-realized and reliable parameter set for their isogeny-based key exchange protocol in their specification document on the website [5], which is shown Appendix A.

Due to limited resources and the fact that the implementation of this protocol will not be used for actual encryption of information, the parameter set used for this paper is significantly smaller. Table 2 contains the parameters used in this implementation.

Parameter Set for Implementation	
$p = 2^3 \cdot 3^2 - 1 = 71$	
$E_0 : y^2 = x^3 + (60 + 0i)x + (14 + 0i)$ over $\mathbb{F}_{p^2}$	
$P_A = (18 + 37i, 57 + 32i)$	$Q_A = (41 + 63i, 17 + 13i)$
$P_B = (41 + 7i, 60 + 12i)$	$Q_B = (58 + 6i, 50 + 56i)$

Table 2: Public parameters used for this paper's implementation.

Recall that certain points on an elliptic curve serve as torsion subgroup generators for that elliptic curve. For this implementation, we have $E_0[2^3] = \langle P_A, Q_A \rangle$ and $E_0[3^2] = \langle P_B, Q_B \rangle$. To be clear, these points P_A, Q_A, P_B, and Q_B are the points in Table 2.

5.2 Code

This section is specific to how the key exchange protocol was implemented in Python. While various other languages could also be used to implement this protocol, Python was chosen. This language was picked because of some packages that aided in the computer networking aspect of this project as well as Python's relaxed syntax which aided in the more dynamic coding style that was sometimes required to compensate for mid-project changes.

5.2.1 Class Structure

Five classes, two helper files, and one global variable were used in order to implement this protocol. Descriptions of the components will be provided below. If a class has methods, they will be briefly summarized (excluding their getter and setter methods, as well as the toString method).

Global Variable: Field

The global variable **Field** is an integer that serves as the field that all computations will be made over. In the case of this implementation, the field is 71.

Class: Fraction

The **Fraction** class is not used too frequently, yet still fundamental to the implementation of this protocol. This class is initialized with two integers, the first serving as the numerator and the second serving as the denominator. This class contains no methods, but is a vital part of dividing two instances of the ComplexNumber class.

Class: ComplexNumber

ComplexNumber is a class that simulates the properties of any complex number. The class constructor's parameters are two integers, one to represent the real part of a complex

number, and the other representing the imaginary part. Providing a *None* type is also valid, as long as both the real and imaginary values are both *None*. The reasoning for this will be elaborated in the EllipticCurve class.

The class methods are as follows:

- ADD: Adds two instances of ComplexNumber.
- MULTIPLY: Multiplies two instances of ComplexNumber.
- SUBTRACT: Subtracts two instances of ComplexNumber.
- SIMPLIFYFRAC: Takes a fraction as a parameter, and simplifies the fraction according to the Field value. An integer is returned.
- DIVIDE: Divides two instances of ComplexNumber. This requires making two instances of the Fraction class, one fraction that contains the real part and one that contains the imaginary part. Then the method SIMPLIFYFRAC is called. An instance of ComplexNumber that is the quotient of the original two ComplexNumbers is returned.
- EQUAL: Compares two ComplexNumbers to see if they are equivalent. Returns a boolean.
- POWER: Raises a ComplexNumber to the power of a positive integer vale.

Class: Point

The **Point** class that acts as a pair of complex numbers. It takes two ComplexNumber instances as constructor parameters, one serving as the x-coordinate, one serving as the y-coordinate. The only class method is *equal*, which returns a boolean depending on if two points are equivalent or not.

Class: EllipticCurve

EllipticCurve is a class that simulates the properties of an elliptic curve. It is much more computationally heavy than the previous classes. The class is initialized by providing two ComplexNumbers, one that represents the coefficient A, and one the coefficient B in the equation $y^2 = x^3 + Ax + B$. The class also has a class variable of *INF*, which is $\mathcal{O}$, and it programatically composed of a Point containing two ComplexNumbers that have a *None* type as their real and imaginary parts. The class functions are listed below.

- ELLIPTICFUNCTION: Given a ComplexNumber as x, this function computes the corresponding y^2 value.
- ADD: This function takes two Point instances and returns the sum of them, according to the Elliptic Curve Addition Law detailed in Algorithm 1.

- GENERATEPOINTS: Returns a list of all Points on the elliptic curve over the given field.
- DBL_ADD: Given a Point p and an integer n, this function computes nP according to Algorithm 2.
- JINVARIANT: This function returns the j-invariant of the EllipticCurve instance. The formula can be found in Definition 10.

Class: Isogeny

The **Isogeny** class emulates isogenies and their specific functions. The constructor for an Isogeny instance requires an EllipticCurve (domain) and list of Points (which serves as the kernel) as parameters. Additional class variables are the codomain of the isogeny, which is another EllipticCurve, and integer that represents the order of the kernel. The class functions are as follows:

- VELU: This function takes a Point as a parameter and uses the appropriate Vélu formula depending on the order of the subgroup. This function returns the image of the Point under the Isogeny instance.
- CALCISOGENOUSCURVE: This is a private function that calculates an isogenous curve, which results in the codomain of the Isogeny instance.

Helper File: parameterSet.py

The file **parameterSet.py** contains all the public parameters detailed in Table 2. This file is imported in every subsequent file so that the parameters are available for use.

Helper File: keyExchange.py

The helper file **keyExchange.py** contain three functions that aid in the actual key exchange process. They are as follows:

- CREATESUBGROUP: This function takes an EllipticCurve and a Point P as parameters and returns a list of Points. The list of Points contains all multiples of P, up to $\mathcal{O}$.
- TWOSTRATEGY: This function implements a multiplication-based strategy for a 2^a-order point. The parameters are a Point that is of 2^a-order, an optional list of Points that need to be evaluated under isogenies, and an EllipticCurve. The function iteratively computes a isogenies by calling the VELU function, ultimately returning the final Isogeny and a list of Points that have been evaluated under the Isogenies with the strategy, if a list was provided as a parameter of the function.
- ODDSTRATEGY: This function operates in the exact same way as *twoStrategy*, except it requires a Point of odd order instead of order two.

A class diagram has been provided in Figure 8 to show the relationships between the classes that made this key exchange protocol possible. The helper files and class methods have been omitted.

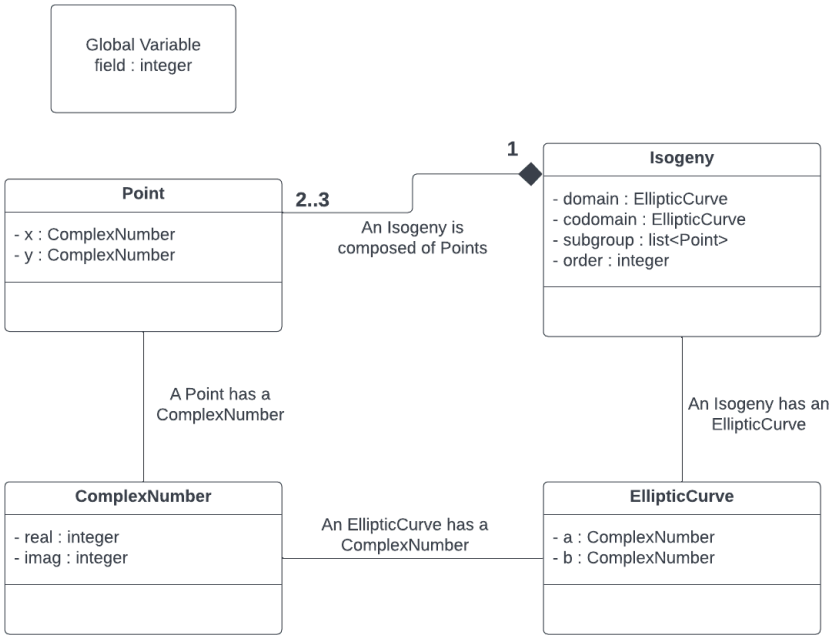

Figure 8: A UML class diagram showing the relationships between classes.

5.2.2 Computer Networking

In order to have this key exchange protocol be as authentic as possible, a client-server relationship was established between two computers in order to simulate communication over an insecure channel.

Two packages, **Socket** and **Pickle**[2], were used in order to program the client-server aspect of this project. Socket was responsible for enabling the two devices to communicate. The client used Socket to connect to the server using the server's IP address as well as its port number. The server used Socket to listen and accept any devices that tried to connect to it. Of course, there is much more to sockets within the realm of computer networking, but anything more than just described is outside the scope of this paper.

The package Pickle was used in order to bundle the Isogeny class as bytes and transfer them over the channel created between the two devices. In other words, Pickle enabled Alice and Bob to exchange their isogenies. Pickle[3] was chosen over other formats such as Socket's *read*

[2]The documentation for Socket and Pickle can be found at https://docs.python.org/3/library/socket.html and https://docs.python.org/3/library/pickle.html, respectively.

[3]A special thank-you to Dr. Andrey Timofeyev who introduced me to Pickle and who also assisted in some computer networking roadblocks that occurred during this stage of the project.

and *write* functions, and functions within the *Json* package. This is because the previously mentioned options required a manual approach to breaking down a class into bytes, which proved to be a considerable challenge. Pickle was by far the more user-friendly option.

Alice served as the client and Bob as the server in their respective files *alice.py* and *bob.py*. Functions were composed in these files that performed computations and the transmission of information. All that was required from the users was to set up Bob first (i.e. run the *bob.py* file), and then set up Alice. Alice's file required Bob's IP address and his port number, which was hard-coded to 12000. Once both programs were running, the users were prompted for their personal secret keys, and their shared secret key was outputted.

5.2.3 Github

This implementation of De Feo, Jao, and Plût's key exchange protocol is available in the public GitHub repository

https://github.com/tsschwartz99/KeyExchangeProtocol,

along with other resources that may prove valuable to running this protocol on one's own device.

6 Conclusions and Future Directions

This section finishes the paper with conclusions that can be drawn as well as future directions this project could take, if desired.

6.1 Conclusions

Cryptography is a vast and deep area of study. Since cryptography deals with keeping information protected, it makes sense that research within this field can become complex very quickly. Despite its complicated nature, this paper has shown that usable cryptography can be accomplished at a much smaller, more reasonable scale using basic programming skills and resources that are widely accessible.

6.2 Future Directions

There are various directions this project could take in the future. The first one would be to optimize the code. As it stands in this project's implementation, generating points on the

elliptic curve is much more computationally demanding than needed. This process could be optimized. Other areas that could be changed in order to optimize the code is the Point class. Currently the Point class uses affine points, but if it were to be made to simulate projective points, this would significantly optimize the code.

Another direction this project could be taken is expanding this protocol to key exchange between three or more parties. This would involve having transmissions over more than just one channel, but the overall structure of the Diffie-Hellman would hold.

A Sike Parameters

The following table contains a functional parameter set for De Feo et al.'s key exchange protocol that could be trusted to encrypt information. This parameter set and others are available on the SIKE website [5].

<table>
<tr><th colspan="2">SIKE p434 Parameters</th></tr>
<tr><td colspan="2">$p = 2^{216} \cdot 3^{137} - 1 = 2443942366134522155190914501145749361908578024376159651132580733620522123933197672597021667182861844589871902669288493934231473356 7$</td></tr>
<tr><td colspan="2">$E_0 : y^2 = x^3 + 6x^2 + x$ $(Montgomery\ Curve)$</td></tr>
<tr><td>$Q_A = (8633131302536015373065425580178973814526244742660764898957635611033517358603093513483897324469034427019598357249425684820405193836 + 1640555213321637736080614728970921962714590288563692816952785470842808462670732196555713644986698688787353020078064569199240185333i, 202764527522206655482021894035981707508349824271307608508132541605503053106599296631233047788142875315007561462048052519637832560373 + 10045306525245350298803819046509877432229480969314772374869175643233206473596453796721101344057683381390923103776706006170651177942i)$</td><td>$Q_B = (1310601591064720145842681149328896592331170290232117979498430679133589826965190167080961911611999795268369761752937950733928898362 2 + 0i, 0 + 1020977593851596250177174150608158042524358870860639246205446239965187139379037251890843542449502134699517363323537399150 4988757970i)$</td></tr>
<tr><td>$P_A = (2634539327592482918121599540115765431217195093350648632832477775508933673747596362667240890051240463853167541162279343167040310088 + 18590308952679468489364793668589003541299106140709579196186461020066893645141198854487503147226318730158493210982567772716162869840i, 184999207277477218275046105494896512286232657693868315586315775566430857668579154653063760554361531088335435592271711 4976263189216 + 10983718925653566249610333622918370357192097441961014913751641775508865561311331364566791542776619041356373750734992554370506677551i)$</td><td>$P_B = (5822289030790821842647204127346110197186614331916510409554480418611145246532692679762948023941031185216358707524703325193156147113 + 0i, 4631002038627486062145710538859886699092897850004224163519174820337269208909673679867855016325656365561668068341925816094377133115 + 0i)$</td></tr>
</table>

Table 3: The parameter set that SIKE uses for their isogeny-based protocol [5].

References

[1] Minal Wankhede Barsagade and Suchitra A. Meshram. Overview of History of Elliptic Curves and its Use in Cryptography. *International Journal of Scientific and Engineering Research*, 5(4), April 2014.

[2] Desmos Graphing Calculator. https://www.desmos.com/calculator, 2022.

[3] D. Coppola. Topic: E-commerce Worldwide. https://www.statista.com/topics/871/online-shopping/#dosierKeyfigures. Accessed: 2021-09-17.

[4] W. Diffie and M. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[5] David Jao et. al. SIKE – Supersingular Isogeny Key Encapsulation. Available at https://sike.org/#nist-submission (2021/06/09).

[6] Luca De Feo, David Jao, and Jérôme Plût. Towards Quantum-Resistant Cryptosystems from Supersingular Elliptic Curve Isogenies. *Journal of Mathematical Cryptology*, 8(3):209–247, 2014.

[7] J. Hoffstein, J. Pipher, and J.H. Silverman. *An Introduction to Mathematical Cryptography*. Undergraduate Texts in Mathematics. Springer New York, 2014.

[8] Avesta Hojjati. The Threat of Quantum Computers and the Solutions that Can Protect Us Today. Available at https://www.helpnetsecurity.com/2019/03/13/threat-quantum-computers.

[9] Ralph C. Merkle. Secure Communications over Insecure Channels. *Commun. ACM*, 21(4):294–299, apr 1978.

[10] Andrew Sutherland. Lecture notes in elliptic curves, Spring 2021.

[11] Jun Tao, Jun Ma, Melissa Keranen, Jean Mayo, and Ching-Kuang Shene. Ecvisual: A visualization tool for elliptic curve based ciphers. *SIGCSE'12 - Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, 02 2012.

[12] James J. Tattersall. *Elementary Number Theory in Nine Chapters.* Cambridge University Press, 1999.

[13] A.J. Han Vinck. Introduction to Public Key Cryptography. https://www.uni-due.de/imperia/md/images/dc/crypto_chapter_5_public_key.pdf. Accessed: 3-28-2022.

[14] Eric W. Weisstein. Invariant. https://mathworld.wolfram.com/Invariant.html.