Contents

Chapter 1

Introduction

The word "*cryptography*" derives from the Greek word κρυπτὸς, *kryptos*, meaning hidden. Its origin is usually dated back to 2000 B.C., and associated with the Egyptian practice of hieroglyphics which full meaning was only known to an elite few. Formally, cryptography is the science of protecting information exchanged by two or more parties. In the last decades, it has advanced exponentially by taking advantage of the most developed formalisations and by relying on hard mathematical problems for security. Its algorithms are designed around computational complexity assumptions: it is theoretically possible to break a cryptosystem but it is infeasible in practice due to excessively long running times. These schemes are therefore said computationally secure.

Introduced in the mid 19-th century, modern cryptography is a wider field than classical cryptography since it also offers new additional properties vital to modern communications that can be summed up as follows:

- Authenticity - ensures that the information comes from the source it is claimed to be from;
- Confidentiality - ensures that the information is not made available or disclosed to unauthorised individuals, entities, or processes. This is the only property targeted also by classical cryptography;
- Integrity - ensures the information sent is exactly the same information re-

ceived;

In modern cryptography, symmetric-key and public-key encryption schemes are the best known and world-wide used. The symmetric-key algorithms make use of a single key to both encrypt and decrypt messages. This key is shared by the communicating parties hence the name symmetric. An example is the Advanced Encryption Standard (AES), designed in 1997 to supersede its predecessor DES in terms of both efficiency and security. More on this protocol can be found in [18].
public-key encryption schemes use a pair of keys: the public key serves to encrypt messages, the private one to decrypt them. These special algorithms are also named asymmetric since encryption and decryption keys are different and not shared by the parties.
These schemes are commonly implemented in a 2-parties version in which the users Alice and Bob want to communicate without revealing their messages' content to others. Many protocols have been proposed to this purpose: RSA [19] is the most spread algorithm [29], while Elliptic Curve Cryptography (ECC) [20] provides the most recent and efficient schemes.

In parallel with cryptography's evolution, in 1982 Richard Feynman observed that simulating quantum physics on classical computers seemed an intractable task [2]. He then followed up by conjecturing that physical devices relying on quantum phenomena would have been good candidates for simulating quantum mechanics. Soon after, David Deutsch in [21] formalised the notion of quantum Turing machine, or universal quantum computer: it can simulate any quantum mechanical process with small overhead and independently of the substrate.
The basic idea behind a universal quantum computer is to exploit quantum bits, or qubits for short. As opposed to binary bits, qubits can achieve additional states in between the two binary states. These are named superposition of the digital states. A full treatment about quantum computers is beyond the scope of this thesis but a curious reader can refer to [1, 2, 4].
It is natural wondering about whether quantum computers could solve classical computational problems faster than classical computers. To this question Shor answered in 1994 with a paper [5] in which he presented polynomial-time quantum algorithms

to solve the integer factorisation and discrete logarithm problems for which no efficient classical algorithms exist. The impact on modern public-key cryptography is dramatic: large enough quantum computers will break factorisation-based cryptosystems, such as RSA, as well as cryptosystems based on the discrete logarithm problem, such as elliptic curve cryptosystems.

1.1 Shor's algorithm

Theorised in 1994, Shor's algorithm is one of the most important for post-quantum cryptography.
The idea behind Shor's algorithm [4, 5] is to utilise quantum computing to compare the phases of prime numbers "represented as sinus waves" to factorise big integers. Using number theory, the problem of integer factorisation can be converted into a search for the period of a really long sequence, or rather, the length at which a sequence repeats itself. Then this periodic pattern is run through a quantum computer which works as a computational interferometer creating an interference pattern. This will output the period, which can be processed using a classical computer, finally being able to factorise the initially given number.

1.2 Post-Quantum Cryptography

With the term "*Post-quantum cryptography*" we refer to the cryptography's branch aiming at protecting information from both quantum and classical attacks, as well as to the collection of classical schemes that accomplish this task. Unfortunately, the transition to post-quantum cryptography is not cost-free. The hard problems exploited by post-quantum cryptography, except for hash inversion, have been studied less than integer factorisation and the discrete logarithm problem. Consequently a post-quantum hard problem inevitably conveys a lighter security assurance compared to a classic alternative. The reason is due to the greater potential of future improvements on quantum attacks. Additionally, many of the hard problems that hold promise of resisting attacks from quantum computers require cryptosystems with far greater memory and bandwidth impeding their adoption for low-cost de-

vices.

1.2.1 NIST's Standardisation Competition

Over the last decade there has been an intense research effort to find hard mathematical problems that would be at the same time quantum resistant and could be used to build new efficient cryptosystems.
The above mentioned effort was also triggered by "*The Post-Quantum-Cryptography Standardization Challenge*" which is a competition started by the National Institute of Standards and Technology (NIST) in April 2016 accepting proposals for post-quantum protocols. It entered in the first evaluation stage in November 2017 when NIST stopped accepting new algorithms for consideration. On 30 January 2019, the project went into the second evaluation stage, said *Round 2*, with NIST announcing 26 out of 69 original submissions as final competitors. This round may take up to 18 months before completion, after which there may be a third round and only then official standard algorithms will be chosen.
The final algorithms will be rated into five different levels, summarised in the table below:

Quantum Level	Security	Reference protocol
I	Comparable to or greater than that of a block cipher with a 128-bit key against an exhaustive key search	AES128
II	Comparable to or greater than that of a 256-bit hash function against a collision search	SHA256
III	Comparable to or greater than that of a block cipher with a 192-bit key against an exhaustive key search	AES192
IV	Comparable to or greater than that of a 384-bit hash function against a collision search	SHA384
V	Comparable to or greater than that of a block cipher with a 256-bit key against an exhaustive key search	AES256

As a final note, most published post-quantum schemes can be divided into the following families [1]:

- Hash-based cryptography (e.g. Merkle's hash-tree signature system [30]);
- Multivariate cryptography (e.g. HFE signature scheme [31]);
- Lattice-based cryptography (e.g. NTRU encryption scheme [34]);
- Code-based cryptography (e.g. McEliece encryption scheme [33], Niederreiter encryption scheme [32]).

A new family, called isogeny-based cryptography, recently originated and it is the central point of this thesis.

1.3 This Thesis

In this thesis we want to present a C implementation of a post-quantum protocol introduced in [9] by Vanessa Vitse. The proposed algorithm relies on the supersin-

gular isogeny problem, which is supposed to be quantum resistant, for its security. By implementing this protocol first we reiterate that classical computers are central in post-quantum cryptography or, viceversa, quantum computers are not compulsory when implementing post-quantum algorithms. Most importantly we adapted an open source library to produce the first public implementation, as far as our knowledge goes, of the above mentioned algorithm. This thesis is organised as follows.

In chapter 2 we will introduce elliptic curve cryptography providing a brief but exhaustive overview on elliptic curves and isogenies, among other topics. Whenever possible we will present examples. Chapter 3 is the *theoretical core* of this thesis since it focuses on two cryptographic protocols, see Sections 3.1, 3.2, which can be used in symbiosis to create a third scheme, see Section 3.3. We also supply a thorough security analysis of the latter before jumping to Chapter 4. This chapter is our *practical core*: there will be a full commentary on the implementation of the SIDH-based OT protocol from Section 3.3.

Chapter 2

Elliptic Curve Cryptography

Elliptic Curve Cryptography (ECC) is an alternative to RSA-based public-key schemes based on the mathematics of elliptic curves. Cryptographic algorithms within ECC require elliptic curves over finite fields and provide the same security offered by other public-key schemes while requiring much smaller keys. ECC main applications fall in the area of encryption and key-exchange, but it can also be used in digital signatures, pseudo-random number generators and many others.
ECC security lies on the Elliptic Curve Discrete Logarithm Problem (ECDLP) which will be discussed in the following section after a brief introduction on elliptic curves to familiarise with them.

2.1 Elliptic Curves

An elliptic curve [22] is a cubic curve defined over a field $\mathbb{K}$ and having a specified point $\mathcal{O}$. Whenever the field $\mathbb{K}$ has characteristic $char(\mathbb{K}) \neq \{2, 3\}$, an elliptic curve can be defined as the locus of points satisfying an affine equation which comes with a smoothness condition:

$$\textit{Weierstrass equation:} \begin{cases} y^2 = x^3 + ax + b \\ 4a^3 \neq 27b^2 \end{cases} \tag{2.1}$$

together with the point at infinity $\mathcal{O}$ as expressed in [23].
Points of an elliptic curve E form a group $(E, +)$ [24, 25]. A rigorous definition of

the addition + can be found in [23]:

Definition 2.1.1. Group Law Let P, Q be two points on the elliptic curve E, let L be the line through P and Q (if $P = Q$, let L be the tangent line to E at P), and let R be the third point of intersection of L with E. Let now L' be the line through R and $\mathcal{O}$. Then L' intersects the curve E at R, $\mathcal{O}$, and a third point. We denote that third point by $P + Q$.

We now briefly list some of the most important properties for an elliptic curve from a cryptographic point of view:

- The point addition allows to sum two different points $P = (x_P, y_P)$ and $Q = (x_Q, y_Q)$ of the curve obtaining a third point $R = (x_R, y_R) = P + Q$ computed as follows:

$$\begin{cases} m = \dfrac{y_P - y_Q}{x_P - x_Q} \\ x_R = m^2 - (x_P + x_Q) \\ y_R = m(x_P - x_R) - y_P \end{cases}$$

- The above formula cannot be used for the Point Doubling, i.e. for summing P, Q when $P = Q$. In that case the line's slope coefficient m changes to $m = \dfrac{3x^2 + a}{2y}$;

- The points of a curve E form an *abelian group* [25] meaning that the group law is commutative, hence $R = P + Q = Q + P$ for every $P, Q \in E$;

- The operation of adding a point P to itself k times is written as $[k]P$, or simply kP, and sometimes called scalar multiplication (between the scalar k and the point P);

- Given a point P of an elliptic curve, its order is the minimum number k such that $[k]P = \mathcal{O}$. All the points $\{G, [2]G, [3]G, \ldots, [k-1]G, [k]G\}$ a subgroup denoted as $\langle G \rangle$ and having cardinality k;

- An elliptic curve, defined over a finite field $\mathbb{F}_p$ with p prime greater than 3, having $p + 1$ points with coefficients in $\mathbb{F}_p$, is said *Supersingular*, otherwise

is said Ordinary. The set of all the points of the elliptic curve E having coefficients in $\mathbb{F}_p$ is denoted by $E(\mathbb{F}_p)$ and it is called the set of rational points. $E(\mathbb{F}_p)$ is a subgroup of $(E, +)$;

- Given the elliptic curve $E : y^2 = x^3 + ax + b$ over a field with characteristic different than 2 and 3, its j-invariant is defined as follows:

$$j(E) = 1728 \cdot \frac{4a^3}{4a^3 + 27b^2}$$

The j-invariant is an important identifier for an elliptic curve and, for this reason, it is commonly used in practical ECC implementations.

At this point we can introduce the concept of Elliptic Curve Discrete Logarithm Problem (ECDLP). This problem is considered hard for classical computers, an assumption upon which the security of ECC schemes is based on:

Definition 2.1.2. ECDLP Given an elliptic curve E and two of its points G, P, find an integer k "if it exists" such that $P = [k]G$.

In figure 2.1 we can see two common affine representations of elliptic curves defined over the field $\mathbb{R}$: on the left $a = -3$, $b = 1$, on the right $a = -2$, $b = 2$.

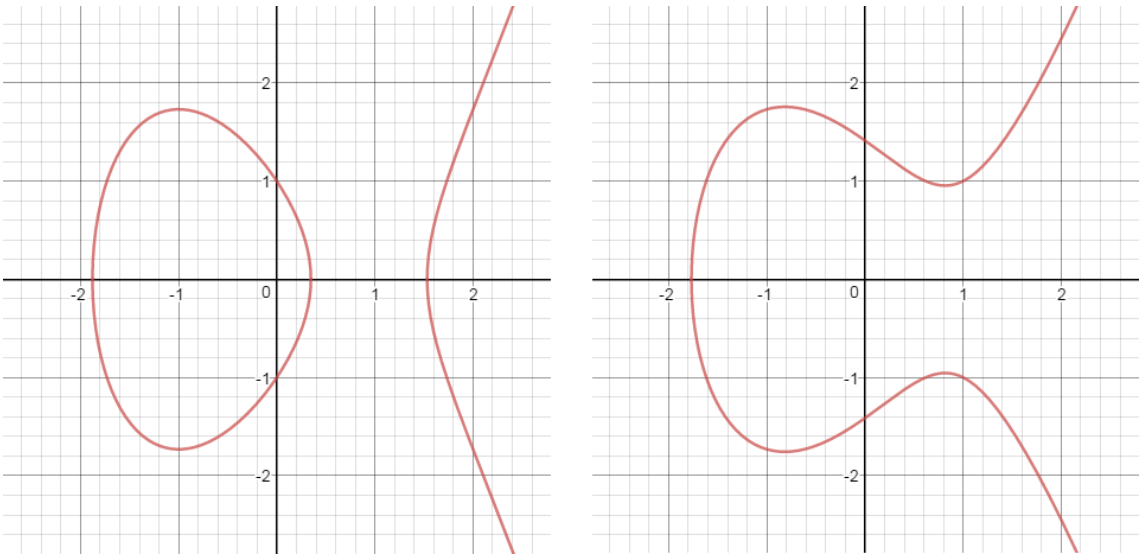

Figure 2.1: Two affine elliptic curves

2.2 Important concepts

In this section many fundamental mathematical concepts that will be used in the next chapters are listed. Where not better specified, the reader may find additional information in [22–25].

2.2.1 m-torsion groups

Given an elliptic curve E defined over a field K with characteristic p and an integer $m \neq 0$, the m-torsion group $E[m]$ is defined as follows:

$$\{P \in E \mid [m]P = \mathcal{O}\}$$

This set forms a subgroup for which holds:

$$E[m] \simeq (\mathbb{Z}/m\mathbb{Z})^2 \text{ if } p \nmid m$$

Otherwise in case we have $p \mid m$ then:

$$E[p^i] \simeq \begin{cases} \mathbb{Z}/p^i\mathbb{Z} & \forall i \geq 0 \text{ if } E \text{ is Ordinary} \\ \{\mathcal{O}\} & \forall i \geq 0 \text{ if } E \text{ is Supersingular} \end{cases}$$

As a consequence of the above, a supersingular curve has no points of order p.

2.2.2 Isomorphic curves

Given two groups $(G, \circ)$, $(H, *)$, the function $\phi : G \to H$ is a group homomorphism if it holds:

$$\phi(g \circ h) = \phi(g) * \phi(h) \quad \forall g, h \in G$$

If the homomorphism is also bijective then it is called *isomorphism* then we write $G \simeq H$ and the two groups have the same algebraic structure.
Since the points of an elliptic curve form a group [25], it is natural to extend the concepts of homomorphism and isomorphism to elliptic curves. In particular, it turns out that two elliptic curves over $\mathbb{K}$ are isomorphic over the algebraic closure $\overline{\mathbb{K}}$ if and only if they have the same j-invariant.

2.2.3 Isogenies

An isogeny is a surjective group homomorphism between two elliptic curves which is expressed by rational functions. Moreover, given two elliptic curves E, E' and the isogeny $\phi : E \to E'$, when the isogeny ϕ satisfies an extra property, named separability, its kernel is finite. Two elliptic curves E, E' are thus said *isogenous* if there exists an isogeny between them. Moreover, the isogeny preserves the cardinality of each curve: two elliptic curves are isogenous over an extension $\mathbb{F}$ of $\mathbb{K}$ if and only if $\#E(\mathbb{F}) = \#E'(\mathbb{F})$, where $E(\mathbb{F})$ denotes the subgroup of points with coefficients in $\mathbb{F}$ and "defined over $\mathbb{F}$" means that the rational functions defining ϕ have coefficients in $\mathbb{F}$.

Isogenies can be computed via Velù's formulae: let $E : y^2 = x^3 + ax + b$ be an elliptic curve over a finite field $\mathbb{F}_q$, P a generic point of E, $\langle K \rangle$ a cyclic group generated by K and let $\phi : E \to E'$ be an isogeny with kernel $\langle K \rangle$. The arriving curve E' is usually denoted as $E/\langle K \rangle$. Let G a finite subgroup of E, then there is a *unique* elliptic curve E' and a unique separable isogeny ϕ such that $ker\ \phi = G$ and $\phi : E \to E'$. Consequently, the notation $E' = E/G$ uniquely denotes the range of ϕ. It can be proven that the isogeny ϕ is unique (modulo composition with isomorphisms) and $\phi(P) = (x_\phi, y_\phi)$ is defined as [6]:

$$x_\phi = x(P) + \sum_{Q \in \langle K \rangle \setminus \{\mathcal{O}\}} \Big(x(P+Q) - x(Q) \Big) \tag{2.2}$$

$$y_\phi = y(P) + \sum_{Q \in \langle K \rangle \setminus \{\mathcal{O}\}} \Big(y(P+Q) - y(Q) \Big) \tag{2.3}$$

Note that with $x(P)$ and $y(P)$ we define respectively the x and y coordinate of P. The image curve $E' = E/\langle K \rangle : y^2 = x^3 + a'x + b'$ has parameters a', b' defined as follows:

$$a' = a - 5 \cdot \sum_{Q \in \langle K \rangle \setminus \{\mathcal{O}\}} \Big(3x(Q)^2 + a \Big) \tag{2.4}$$

$$b' = b - 7 \cdot \sum_{Q \in \langle K \rangle \setminus \{\mathcal{O}\}} \Big(5x(Q)^3 + 3ax(Q) + b \Big) \tag{2.5}$$

The formulas in 2.2, 2.3, 2.4 and 2.5 require a small constant number of computations for each $Q \in \langle K \rangle \setminus \{\mathcal{O}\}$ hence the computational complexity of Velù's formulae is

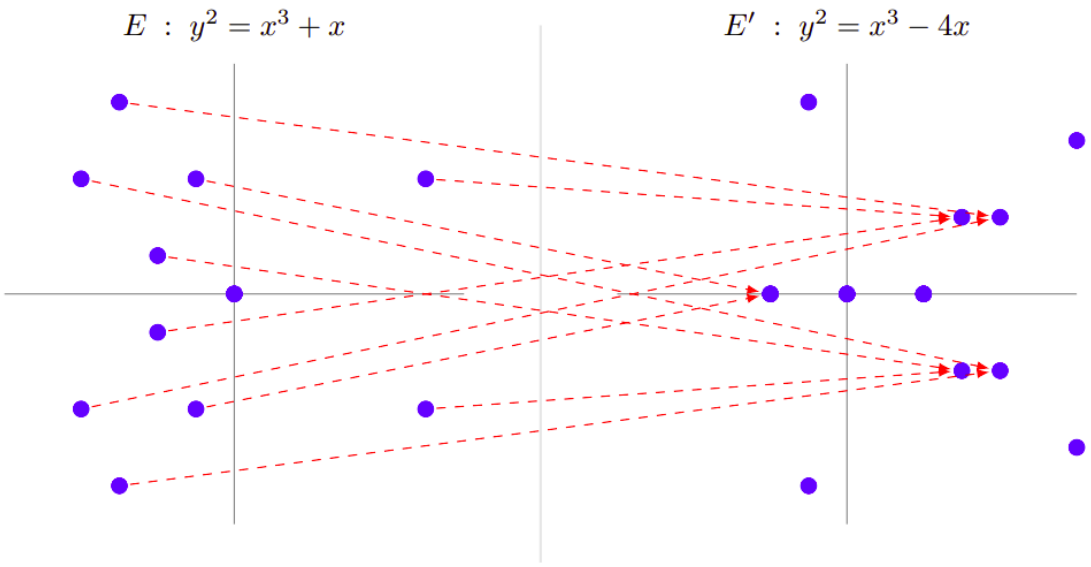

Figure 2.2: Isogeny mapping between E and E' [10]

$\mathcal{O}(\#\langle K\rangle)$ where $\#\langle K\rangle$ denotes the cardinality of $\langle K\rangle$.
An example of isogeny [10, 26] is shown in figure 2.2 where the left curve E, defined over $\mathbb{F}_{11}$ by $y^2 = x^3 + x$ is mapped on the right curve E', defined over $\mathbb{F}_{11}$ by $y^2 = x^3 - 4x$. The kernel is the point $(0, 0)$ in E and the isogeny is defined by:

$$\phi(x, y) = \left(\frac{x^2 + 1}{x}, y\frac{x^2 - 1}{x^2}\right).$$

Degree

Following [6, 26], we define the degree of a separable isogeny $\phi : E \to E'$ as the finite cardinality of its kernel. It is common in literature to call an isogeny ϕ of degree $deg(\phi) = d$ a d-isogeny and so we will do hereafter.

Dual Isogenies

For any isogeny $\phi : E \to E'$ between two elliptic curves there exists a dual isogeny [26] $\widehat{\phi} : E' \to E$ such that $\widehat{\phi} \circ \phi = \phi \circ \widehat{\phi} = [deg\, \phi]$. Furthermore it is important to

note that $deg\,\widehat{\phi} = deg\,\phi$, and $\widehat{\widehat{\phi}} = \phi$.

2.2.4 Endomorphisms

Isogenies from a curve to itself are called *endomorphisms*. Given an elliptic curve E and an integer m, the scalar multiplication-by-m defined by $[m] : P \to [m]P$ is an endomorphism of E. Its kernel is exactly the m-th torsion group $E[m]$.

Frobenius Endomorphism

Let E be an elliptic curve defined over a finite field $\mathbb{F}_q$ with q elements. Its Frobenius endomorphism is the map $\pi : P = (x, y) \to (x^q, y^q)$.
It holds that:

- $ker(\pi) = \{\mathcal{O}\}$
- $ker(\pi - 1) = E(\mathbb{K})$

2.2.5 Bases and Weil Pairing

Given an elliptic curve E defined over a finite field $\mathbb{F}_q$, with $q = p^n$, and an integer $m \neq p$, we know that $E[m] = \mathbb{Z}_m \times \mathbb{Z}_m$. A basis for $E[m]$ is a pair $(P, Q) \in E[m] \times E[m]$ of linearly independent points. Indeed, the set of all their linear combinations is equal to $E[m]$. In that case we write $\langle P, Q\rangle = E[m]$. For the sake of readability, we recall that two points P, Q of an elliptic curve are linearly independent when $[v]P + [u]Q = \mathcal{O}$ if and only if $v = u = 0$.
It is possible to check the linear independency of two points using the Weil pairing [25].

Definition 2.2.1. Weil Pairing Let E be an elliptic curve over a field $\mathbb{K}$ and let n be an integer not divisible by the characteristic of $\mathbb{K}$. Then $E[n] \simeq (\mathbb{Z}/n\mathbb{Z})^2$. Let $\mu_n = \{x \in \overline{\mathbb{K}} | x^n = 1\}$ be the group of n-th roots of unity in $\overline{\mathbb{K}}$, which is a cyclic group of order n and any generator ζ of μ_n is called a *primitive n-th root of unity*. The Weil pairing is hence a bilinear map $e_n : E[n] \times E[n] \to \mu_n$.

A bilinear map is a function that combines elements of two groups, two instances of $E[n]$ in our case, and returns an element of a third group, μ_m, $E[n] \times E[n]$, and is linear in both its arguments.
For a basis (P, Q) of $E[n]$ the Weil pairing $e_n(P, Q)$ is a primitive n-th root of unity. Consequently, to check if (P, Q) is a basis for $E[m]$ is sufficient to check the order of $e_n(P, Q)$, in particular if it is n.

2.3 Mathematical improvements

This section aims to sum up some of the many improvements to the concepts seen in section 2.2 and commonly used in ECC practical implementations.

2.3.1 Montgomery Curves

A Montgomery curve over a field $\mathbb{K}$ is a special form of an elliptic curve with affine equation:

$$M_{A,B} : By^2 = x(x^2 + Ax + 1) \tag{2.6}$$

where the parameters A, B are in $\mathbb{K}$ and satisfy $B \neq 0$, $A^2 \neq 4$. In projective coordinates $(X : Y : Z)$ with $x = X/Z$, $y = Y/Z$ the equation becomes:

$$M_{A,B} : BY^2Z = X(X^2 + AXZ + Z^2) \tag{2.7}$$

The latter has a unique point at infinity $\mathcal{O} = (0 : 1 : 0)$ i.e., it is the **only** point satisfying the equation where $Z = 0$.

In practical implementations, many computational improvements come from choosing a Montgomery curve over a general elliptic curve given by its Weierstrass equation. For a complete discussion about these special curves it is recommended to read [8]; for this thesis the most relevant speed up are:

- reduced number of operations for the group law;
- only x-coordinates computations are possible thus halving the overall operations when computing a scalar multiplication;

- the point addition can be further improved: when adding the points P, Q, the knowledge of a third point $Q - P$ reduces the overall computations. In this case we talk about Differential Addition.

Several variants of the Montgomery curve coefficients are used when implementing elliptic curve cryptographic schemes [11] to achieve better performances. Let E_a be the curve E_A defined over $\mathbb{F}_{p^2}$ by the equation $y^2 = x^3 + ax^2 + x$, then we can write the pair $(A : C)$ to denote a pair equivalent to $(a : 1)$ in $\mathbb{P}^1(\mathbb{F}_{p^2})$. Furthermore, we define the equivalences:

$$\begin{aligned}(A_{24}^+ : C_{24}) &\longleftrightarrow (A + 2C : 4C)\\ (A_{24}^+ : A_{24}^-) &\longleftrightarrow (A + 2C : A - 2C)\\ (a_{24}^+ : 1) \quad &\longleftrightarrow (A + 2C : 4C)\end{aligned}$$

2.3.2 Weil Pairing

The base algorithm for computing the Weil pairing e_n requires, roughly, $\mathcal{O}(n)$ operations which is greatly inefficient for large-order bases; as an example, typical orders [7, 25] are 2^{250} and 3^{159}. We now show a faster way to run the computation as described in [7].
Let P, Q be two points both of order mn of an elliptic curve E, so that they are in $E[mn]$; the n-th power of the pairing $e_{mn}(P, Q)$ can be computed as follows [7, 25]:

$$e_{mn}(P, Q)^n = e_m([n]P, [n]Q)$$

We now consider an example: let $m = 4$, $n = 2^{370}$ such that $mn = 2^{372}$, then:

$$e_{mn}(P, Q)^n = e_{4\cdot 2^{370}}(P, Q)^{2^{370}} = e_4([2^{370}]P, [2^{370}]Q)$$

Together with the assertion that P and Q both have order $mn = 2^{372}$, the assertion that the Weil pairing $e_4([2^{370}]P, [2^{370}]Q)$ has order 4, proves that P and Q have Weil pairing of order 2^{372}. If indeed P and Q have order 2^{372}, the points $P' = [2^{370}]P$ and $Q' = [2^{370}]Q$ both have order 4. In that case, $e_4(P', Q')$ is of order less than 4 if and only if P' and Q' are independent. Finally, this is equivalent to say that P and Q are linearly independent [7].

To prove this last step we can exploit another speed up: all the described operations are possible considering only the x coordinates of input points P, Q. For the following procedure we are going to denote with $x(P)$ the affine x-coordinate of the point P, and with $X(P)$, $Z(P)$ its projective X- and Z-coordinate respectively such that $x(P) = \frac{X(P)}{Z(P)}$.
At this point it is mandatory to check $P' \neq Q'$ that is $x(P') \neq x(Q')$. Since ECC implementations work with projective points we can transform the latter expression into $\frac{X(P')}{Z(P')} \neq \frac{X(Q')}{Z(Q')}$ and obtain $X(P')Z(Q') \neq X(Q')Z(P')$ which is called projective cross-multiplication.
Lastly, we need to check:

- Said $\left(\overline{X} : \overline{Z}\right) = x([2]P')$, then it must hold true that $\overline{Z} \neq 0$ meaning that $[2]P'$ is not the point at infinity;
- Said $\left(\overline{\overline{X}} : \overline{\overline{Z}}\right) = x([4]P')$, then it must hold true that $\overline{\overline{Z}} = 0$ meaning that $[4]P'$ is the point at infinity.

The same checks have to be performed on Q'.
At this point if both P' and Q' have passed all these tests then $[4]P' = [4]Q' = \mathcal{O}$ finally proving that P and Q both have order 2^{372} and are linear independent [7]. The pair (P, Q) is then a basis of order 2^{372}.

2.3.3 Isogenies

We have seen in Section 2.2.3 that, being $\langle K \rangle$ a cyclic subgroup of points of an elliptic curve E, the worst drawback of using Velù's formulae for computing an isogeny from E having $\langle K \rangle$ as kernel, is that they require $\mathcal{O}(\#\langle K \rangle)$ operations hence making their implementation intractable when $\#\langle K \rangle$ is big. A great improvement is possible when the order of K is smooth, meaning, for example, it is a power of a prime number, via composition of small degree isogenies [7, 9] obtaining a cost drop to $\tilde{\mathcal{O}}(log(\#G))$. Since typical applications use kernels of order 2^n or 3^m, this speed up is fundamental for practical implementations.

Costello et al. [7] write about efficiently computing large degree isogenies via small

degree isogenies. We now describe their method to compose isogenies.
Given an elliptic curve E defined over $\mathbb{F}_{p^2}$, a cyclic subgroup $\langle R \rangle \subseteq E[l^e] \subseteq E(\mathbb{F}_{p^2})$ of order l^e, with l prime, there is a unique isogeny $\phi_R : E \to E/\langle R \rangle$ of degree l^e defined over $\mathbb{F}_{p^2}$ with kernel $\langle R \rangle$. Set $E_0 = E$, $R_0 = R$, such isogeny can be computed by composing e l-isogenies [12], obtaining intermediate curves $E_{i+1} = E_i/\langle [l^{e-i-1}]R_i \rangle$ and intermediate points $R_{i+1} = \phi_i(R_i)$ for $0 \leq i < e$. Note that the point R_i is an l^{e-i}-torsion point and so the point $[l^{e-i-1}]R_i$ has order l. The resulting isogeny is then $\phi_R = \phi_{e-1} \circ \ldots \circ \phi_0$ having degree l^e as required. Indeed, the composition of two separable isogenies is a separable isogeny, and it is possible to prove that the degree of $\phi \circ \varphi$ is $deg(\phi)\, deg(\varphi)$.
Even though we may compute the curve E_{i+1} and its isogeny ϕ_i via Velù's formulae, this strategy is still too onerous. A better strategy is a particular isogeny walk, as shown in figure 2.3 for the case $e = 6$ [12]. Bullets "•" represent points: those at the same horizontal level have the same order, those lying on the same left-diagonal belong to the same curve. Dashed edges "− − −" are directed from top to bottom: leftward edges represent a multiplication-by-l, rightwards edges represent the evaluation of an l-isogeny.
At the beginning of the algorithm only the root R is known and our aim is to compute all the points on the bottom line.

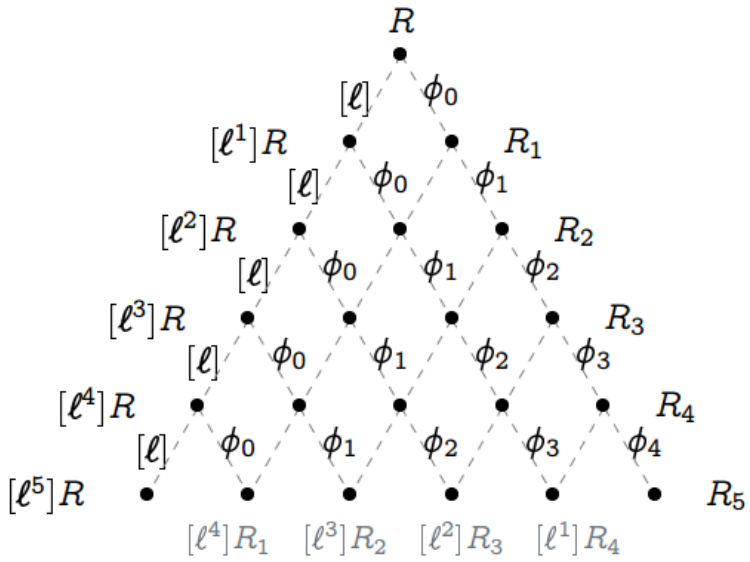

Figure 2.3: l-isogeny walks up to degree l^5

At this point we need a "*strategy*" [7,12] to minimise the operations in descending the graph in figure 2.3 down to the bottom leaves. We may define a strategy as a binary tree topology: a tree in which each node has only one parent node (directed from above in oriented trees with the root on top) and two or zero child node (directed to the bottom direction). From each of its nodes two sub-trees, or sub-strategies, originate on strictly smaller leaf sets. The overall cost for the strategy is the sum of the sub-strategy costs plus the cost for moving down the tree from the root on top along the edges to the roots of the sub-strategies. Given the costs of all sub-strategies, one can select then a strategy by choosing the one with minimum cost hence named *optimal*.

A strategy can be stored in a simple list L of integers, with L having length equal to the total number of leaves. The i-th element, namely $L[i]$, in the list characterises the sub-strategy on a graph with i leaves.
For the given graph there are seven optimal formed strategies as shown in figure 2.4 [12].

Figure 2.4: Optimal strategies for $e = 4$

Among all strategies, there are optimal ones which minimise the number of operations. These can be precomputed offline and, for large smooth degree isogeny computations, say for l^e-isogeny, are performed with complexity $\mathcal{O}(e \log e)$.

Small degree isogenies

In this section we show the formulae alternative to that of Velù's used in real applications. There, only two different kinf of isogenies are usually used: 2- and 3-isogenies. In the former case, a further speed up is possible if we consider 4-isogenies which halves the total operations needed [11].

2-isogenies

Let $E_{A,B}$ be an elliptic curve in Montgomery form, $(x_2, y_2) \in E_{A,B}$ be a point of order 2 having $x_2 \neq 0$. Let $\phi_2 : E_{A,B} \to E_{A',B'}$ be the unique 2-isogeny originating from $E_{A,B}$ with kernel $\langle (x_2, y_2) \rangle$. The coefficients of the image curve can be computed as follows:

$$\left(A', B'\right) = \left(2 \cdot \left(1 - 2x_2^2\right), \quad Bx_2\right)$$

For any point $P = (x_P, y_P) \notin \langle (x_2, y_2) \rangle$ of $E_{A,B}$, its image $(x_{\phi_2}, y_{\phi_2}) = \phi_2(x_P, y_P)$ can be computed as:

$$x_{\phi_2} = \frac{x_P^2 x_2 - x_P}{x_P - x_2}$$
$$y_{\phi_2} = y_P \cdot \frac{x_P^2 x_2 - 2x_P x_2^2 + x_2}{\left(x_P - x_2\right)^2}$$

4-isogenies

Let $E_{A,B}$ be an elliptic curve in Montgomery form, $(x_4, y_4) \in E_{A,B}$ be a point of order 4 having $x_4 \neq \pm 1$. Let $\phi_4 : E_{A,B} \to E_{A',B'}$ be the unique 4-isogeny originating from

$E_{A,B}$ with kernel $\langle(x_4, y_4)\rangle$. The coefficients of the image curve can be computed as follows:

$$\big(A', B'\big) = \Big(4x_4^4 - 2, \quad -x_4(x_4^2 + 1) \cdot B/2\Big)$$

For any point $P = (x_P, y_P) \notin \langle(x_4, y_4)\rangle$ of $E_{A,B}$, its image $(x_{\phi_4}, y_{\phi_4}) = \phi_4(x_P, y_P)$ can be computed as:

$$x_{\phi_4} = \frac{-x_P(x_P x_4^2 + x_P - 2x_4) \cdot (x_P x_4 - 1)^2}{(x_P - x_4)^2 \cdot (2x_P x_4 - x_4^2 - 1)}$$
$$y_{\phi_4} = y_P \cdot \frac{-2x_4^2(x_P x_4 - 1)\Big(x_P^4(x_4^2 + 1) - 4x_P^3(x_4^3 + x_4) + 2x_P^2(x_4^4 + 5x_4^2) - 4x_P(x_4^3 + x_4) + x_4^2 + 1\Big)}{(x_P - x_4)^3(2x_P x_4 - x_4^2 - 1)^2}$$

3-isogenies

Let $E_{A,B}$ be an elliptic curve in Montgomery form, $(x_3, y_3) \in E_{A,B}$ be a point of order 3. Let $\phi_3 : E_{A,B} \to E_{A',B'}$ be the unique 3-isogeny originating from $E_{A,B}$ with kernel $\langle(x_3, y_3)\rangle$. The coefficients of the image curve can be computed as follows:

$$\big(A', B'\big) = \Big(\big(Ax_3 - 6x_3^2 + 6\big)\, x_3, \quad Bx_3^2\Big)$$

For any point $P = (x_P, y_P) \notin \langle(x_3, y_3)\rangle$ of $E_{A,B}$, its image x_{ϕ_3}, y_{ϕ_3} by ϕ_3 can be computed as:

$$x_{\phi_3} = \frac{x_P(x_P x_3 - 1)^2}{(x_P - x_3)^2}$$
$$y_{\phi_3} = y_P \cdot \frac{(x_P x_3 - 1)(x_P^2 x_3 - 3x_P x_3^2 + x_P + x_3)}{(x_P - x_3)^3}$$

2.4 Isogeny-based Cryptography

Traditional Elliptic Curve Cryptography (ECC) relies for its security on the Discrete Logarithm Problem (DLP). That said, Shor's algorithm represents a threat to DLP-based cryptography since it can compute discrete logarithms in polynomial time. Isogeny-based cryptography [6] is a young field, begun in the 2000s but stayed mostly discrete until the second half of 2010 when the cryptographic community realised the possibly near arrival of a general purpose quantum computer. While the capabilities of quantum computers would render ECC worthless, isogeny-based

cryptography seems to resist much better. A quantum-resistant approach involves the use of the *computational supersingular isogeny problem* stated as follows: "*given two isogenous supersingular elliptic curves E and E', find an isogeny ϕ such that $\phi : E \to E'$*". For the given problem does not exist any efficient quantum attack thus making it post-quantum resistant.

Chapter 3

SIDH-based OT

The protocol studied and implemented in this thesis relies on an adaptation of the Supersingular Isogeny Diffie-Hellman (SIDH) to construct an Oblivious Transfer (OT) protocol.
First things first, we separately explain how SIDH [7, 11, 12] and a general OT work, then how it is possible to use the former to obtain an instantiation of the latter [9].

3.1 SIDH Key Exchange

The Supersingular Isogeny Diffie-Hellman is a protocol developed by Jao and De Feo in 2011 [11] which offers a great ratio efficiency-security having keys smaller than other post-quantum protocols (lattice-based and code-based) and than traditional Diffie-Hellman public keys [7].
The scheme begins with a shared supersingular elliptic curve and two parties, Alice and Bob, who get assigned two different torsion groups. After few isogenies and data being exchanged both parties have two curves with the same j-invariant. The latter is then used as shared key hence used, for example, to encrypt and decrypt data between the parties. Let's now describe the protocol in deeper details.

All the supersingular elliptic curves used in SIDH are defined over $\mathbb{F}_{p^2}$ where p is a prime of the form $p = l_A^{e_A} l_B^{e_B} \pm 1$, l_A and l_B are small primes, e_A and e_B are integers. All SIDH implementations consider $l_A = 2$, $l_B = 3$ and so we will henceforth.

Typical choices for e_A fall within the range $[100, 500]$ according to the security level desired; e_B is then chosen such that $l_A^{e_A} \approx l_B^{e_B}$. In order to simplify the notation we will be using n instead of e_A and m instead of e_B.
The initial public parameters are:

- A supersingular elliptic curve E defined over $\mathbb{F}_{p^2}$ having $\#E(\mathbb{F}_{p^2}) = (l_A^{e_A} l_B^{e_B})^2$;
- A bases (U, V) for $E[2^n]$ which lies in $E(\mathbb{F}_{p^2})$;
- A bases (P, Q) for $E[3^m]$ which lies in $E(\mathbb{F}_{p^2})$.

The protocol proceeds as follows:

Alice	**Bob**
Chooses $x_A, y_A \in \mathbb{Z}/2^n\mathbb{Z}$ randomly, at least one of them coprime to 2	Chooses $x_B, y_B \in \mathbb{Z}/3^m\mathbb{Z}$ randomly, at least one of them coprime to 3
Computes $R_A = x_A U + y_A V$	Computes $R_B = x_B P + y_B Q$
Computes the curve $E_A = E/\langle R_A \rangle$	Computes the curve $E_B = E/\langle R_B \rangle$
Computes the isogeny $\phi_A : E \to E_A$ having $\langle R_A \rangle$ as kernel	Computes the isogeny $\phi_B : E \to E_B$ having $\langle R_B \rangle$ as kernel
Computes $P' = \phi_A(P)$, $Q' = \phi_A(Q)$	Computes $U' = \phi_B(U)$, $V' = \phi_B(V)$
$\longleftarrow$ Exchange curves E_A, E_B and points U', V', P', Q' $\longrightarrow$	
Computes $E_{BA} \simeq E_B/\langle x_A U' + y_A V' \rangle$	Computes $E_{AB} \simeq E_A/\langle x_B P' + y_B Q' \rangle$

Since $E_{AB} \simeq E_{BA}$, at the end of the protocol both parties have the same j-invariant which can be used as shared secret.
Figure 3.1 shows the protocol graphically to ease the comprehension.

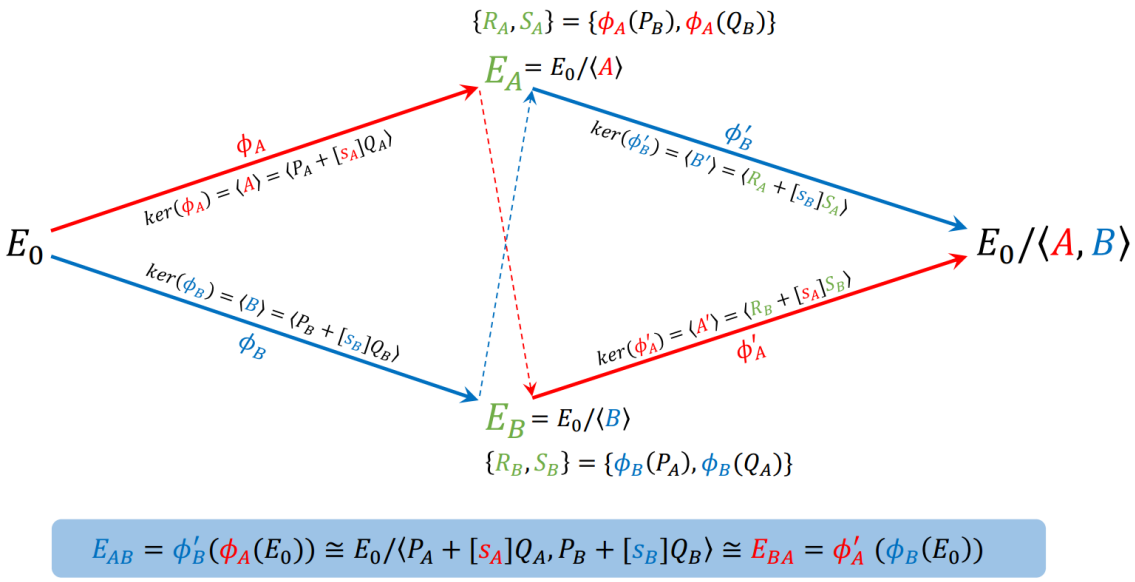

Figure 3.1: SIDH schema. Alice private parameters are in red, Bob's are in blue. Public parameters are green.

3.1.1 Correctness

This brief section shows how Alice and Bob compute two curves isomorphic to one another thus yielding the same j-invariant.

Alice computes:

$$\begin{aligned} E_{BA} &= E_B / \langle x_A U' + y_A V' \rangle \\ &= E_B / \langle x_A \phi_B(U) + y_A \phi_B(V) \rangle \\ &= E_B / \langle \phi_B(x_A U + y_A V) \rangle \\ &= E_B / \langle \phi_B(R_A) \rangle \\ &= \left(E / \langle R_B \rangle\right) / \langle \phi_B(R_A) \rangle \\ &= E / \langle R_B, R_A \rangle \end{aligned}$$

Bob computes:

$$\begin{aligned} E_{AB} &= E_A / \langle x_B P' + y_B Q' \rangle \\ &= E_A / \langle x_B \phi_A(P) + y_B \phi_A(Q) \rangle \\ &= E_A / \langle \phi_A(x_B P + y_B Q) \rangle \\ &= E_A / \langle \phi_A(R_B) \rangle \\ &= \left(E / \langle R_A \rangle\right) / \langle \phi_A(R_B) \rangle \\ &= E / \langle R_A, R_B \rangle \end{aligned}$$

Therefore $E_{BA} = E / \langle R_B, R_A \rangle = E / \langle R_A, R_B \rangle = E_{AB}$ proving that Alice and Bob both have a curve isomorphic to their party's curve.

3.1.2 Security

Before introducing the problem which SIDH's security relies on it is important to state the *Standard Isogeny* problem.
"Given a prime $p = 2^n 3^m \pm 1$, two supersingular elliptic curves E, E_A over $\mathbb{F}_{p^2}$, determine a 2^n-isogeny $\phi_A : E \to E_A$ if it exists".
The problem exploited for SIDH's security is similar to the *Standard Isogeny* problem but it gives some information to the attacker. The problem is stated as follows [13]:
"Given a prime $p = 2^n 3^m \pm 1$, two supersingular elliptic curves E, E_A over $\mathbb{F}_{p^2}$, determine the 2^n-isogeny $\phi_A : E \to E_A$, if it exists, **also** knowing a basis $(P, Q) \in E[3^m]$ and $P' = \phi_A(P)$, $Q' = \phi_A(Q)$".
This problem is assumed to be as hard as the *Standard Isogeny* problem and all the known classical and quantum attacks against it have exponential complexity. In fact the fastest known attack is Tani's Claw Finding attack which requires $\mathcal{O}(\sqrt[4]{p})$ operations on a classical computer and $\mathcal{O}(\sqrt[6]{p})$ on a quantum computer [7].

SIDH's primes p (one for each security level)were initially selected considering only the running time of the Claw Finding attack without considering its space complexity amounting to $\mathcal{O}(\sqrt[6]{p})$ qubits.
A helpful notation for the following disclosure is to express an n-bit length prime number p as p_n.

As an example [17], in NIST's Competition's Round 1 introduced in Section 1.2.1, in order to achieve a b-bit security level against known classical and quantum attacks SIDH primes p were selected with bitlength approximately equal to $6b$. Hence the 751-bit prime $p_{751} = 2^{372}3^{239} - 1$ was proposed for the 128-bit of classical security and the Quantum Level I. Moreover the 964-bit prime $p_{964} = 2^{486}3^{301} - 1$ was proposed for the 160-bit classical security level.
For the second round of the NIST's competition, few revisions were made thus reaching: p_{434} for 128-bit classical and Quantum Level II security, p_{503} for 160-bit classical and Quantum Level III security, p_{610} for 192-bit classical and Quantum Level IV security, finally p_{751} for 256-bit classical and Quantum Level V security. The 964-bit curve was hence discarded.
More details about attacks and examples can be found in [14–16].

3.2 Oblivious Transfers

An *oblivious transfer*, or **OT**, is a cryptographic scheme in which two or more parties are involved.
There exist different instantiations of the oblivious transfer primitive each of them achieving different yet similar goals. The base idea is to send one of many pieces of information to a second party while the sender has no knowledge of which piece has been sent. A classical instantiation is the *Rabin OT* [35] in which Alice, with a probability of 1/2, sends a simple bit to Bob. This scheme leaves Alice "oblivious" of whether Bob has received it or not.

There exists another variation proposed by Shimon Even, Oded Goldreich and Abraham Lempel [28] called "*1 out of 2 Oblivious Transfer*", often written as $\binom{2}{1}$-OT, which can easily be generalised to "*1 out of n OT*". In this variation, a party B, receives one out of two (alternatively n) pieces of information but the sender party A, which owns both pieces, does not know which piece B has received.
As shown in figure 3.2, A sends the pieces b_0, b_1 to B which, in turn, chooses a random bit c. At the end of the transfer B will have knowledge of the piece b_c coming from A. It is fundamental to observe that b_0 and b_1 are masked, and that B is only

able to recover one of them, i.e. b_c.

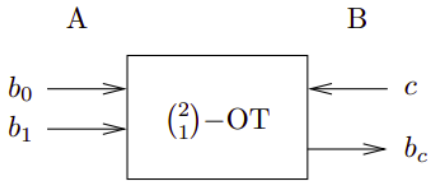

Figure 3.2: Simple $\binom{2}{1}$-OT

A typical OT application is the secure function evaluation where every party holds a part of an input for a given function. In this scenario, the output should be computed in a way such that no party has to reveal unnecessary information about their piece of information. Correctness of the protocol is usually proved with a zero knowledge proof.

3.3 SIDH-based OT

In this chapter we introduce the SIDH-based oblivious transfer protocol proposed by Vanessa Vitse in [9]. The goal is to construct oblivious transfer protocols based on SIDH scheme described in Section 3.1.
The initial public parameters are:

- A supersingular elliptic curve E defined over $\mathbb{F}_{p^2}$, with p prime of the form $2^n 3^m \pm 1$ and having $\#E(\mathbb{F}_{p^2}) = (l_A^{e_A} l_B^{e_B})^2$;
- A bases (P, Q) of $E[3^m]$ with P, $Q \in E(\mathbb{F}_{p^2})$;
- A secure symmetric encryption protocol *Enc* for which we use the notation: $c = Enc(m, k)$, $m = Enc^{-1}(c, k)$, where m is the plain text message, k is the encryption key, c is the cipher text;
- A key derivation function *KDF* such that k = *KDF(seed)* for a given initial input *seed*.

For simplicity we consider a $\binom{2}{1}$-OT in which Alice has *two* secrets $\{s_0, s_1\}$. The protocol then proceeds as follows:

Alice **Bob**

Computes *two* bases for $E[2^n]$:
$(R_0, T_0), (R_1, T_1)$

Computes *two* curves and *two* isogenies:
$E_{A,0} = E/\langle R_0 \rangle\,,\ \phi_{A,0} : E \to E_{A,0}$
$E_{A,1} = E/\langle R_1 \rangle\,,\ \phi_{A,1} : E \to E_{A,1}$

Computes *two* pairs of points:
$P_0 = \phi_{A,0}(P),\ Q_0 = \phi_{A,0}(Q),$
$P_1 = \phi_{A,1}(P),\ Q_1 = \phi_{A,1}(Q)$

$\xrightarrow{\text{Sends Bob } \textit{two} \text{ tuples } \{E_{A,0},\, P_0,\, Q_0\},\ \{E_{A,1},\, P_1,\, Q_1\}}$

Chooses an integer $k \in \{0, 1\}$
Chooses an integer $b \in \mathbb{Z}/3^m\mathbb{Z}$

Computes the curve and j-invariant:
$E_B = E/\langle P + bQ \rangle\,,\ j_B = j(E_B)$

Computes *two* bases:
$(U_0,\, V_0)$ for $E_{A,0}[2^n]$
$(U_1,\, V_1)$ for $E_{A,1}[2^n]$
having the same Weil pairing:
$e_{2^n}(U_0,\, V_0) = e_{2^n}(U_1,\, V_1)$

Computes the curve and isogeny:
$E'_B \simeq E_{A,k}/\langle P_k + bQ_k \rangle\,,\ \phi'_B = E_{A,k} \to E'_B$

Computes the points:
$U'_k = \phi'_B(U_k),\ V'_k = \phi'_B(V_k)$

$\xleftarrow{\text{Sends Alice } \{E'_B,\, U'_k,\, V'_k\} \text{ and the } \textit{two} \text{ basis } (U_0,\, V_0),\ (U_1,\, V_1)}$

Computes *two* pairs $x_i,\ y_i \in \mathbb{Z}$, with $i = 0, 1$:
$(x_0,\ y_0)\ s.t.\ \phi_{A,0}(T_0) = x_0U_0 + y_0V_0$
$(x_1,\ y_1)\ s.t.\ \phi_{A,1}(T_1) = x_1U_1 + y_1V_1$

Computes *two* curves and their correspondent j-invariants:
$F_0 = E'_B/\langle x_0U'_k + y_0V'_k\rangle,\ j_0 = j(F_0)$
$F_1 = E'_B/\langle x_1U'_k + y_1V'_k\rangle,\ j_1 = j(F_1)$

Encrypts her *two* secrets:
$S_0 = Enc(s_0,\ KDF(j_0))$
$S_1 = Enc(s_1,\ KDF(j_1))$

$\xrightarrow{\text{Sends Bob the } two \text{ encrypted secrets } S_0,\ S_1}$

Decrypts its chosen k-th secret:
$s_k = Enc^{-1}(S_k,\ KDF(j_B))$

The protocol described above can be generalised to an $\binom{n}{1}$-OT with little adjustments: every "*two*" in the protocol should be changed to "n"; Alice's pairs $x_i,\ y_i \in \mathbb{Z}$, would have $i = 0, \ldots, n-1$; Bob's k should be chosen in the finite set $\{0, n-1\}$.

3.3.1 Correctness

How can Bob be able to decrypt the k-th Alice's secret? Since Alice encrypts her secrets with j_i, Bob must have that $j_k = j_B$. In order for this to happen, Alice's curve F_k must be isomorphic to the curve E_B computed by Bob. We now show the correctness of the protocol. For the sake of simplicity we assume $k = 0$ without loss of generality.

$$\begin{aligned}
F_0 &= E'_B/\langle x_0U'_0 + y_0V'_0\rangle \\
&= E'_B/\langle x_0\phi'_B(U_0) + y_0\phi'_B(V_0)\rangle \\
&= E'_B/\langle \phi'_B(x_0U_0 + y_0V_0)\rangle \\
&= E'_B/\langle \phi'_B(\phi_{A,0}(T_0))\rangle \\
&= \left(E_{A,0}/\langle P_0 + bQ_0\rangle\right)/\langle \phi'_B(\phi_{A,0}(T_0))\rangle \\
&= E_{A,0}/\langle P_0 + bQ_0, \phi_{A,0}(T_0)\rangle \\
&= E_{A,0}/\langle \phi_{A,0}(P) + b\phi_{A,0}(Q), \phi_{A,0}(T_0)\rangle
\end{aligned}$$

$$
\begin{aligned}
&= E_{A,0} \big/ \langle \phi_{A,0}(P + bQ), \phi_{A,0}(T_0) \rangle \\
&= \big(E \big/ \langle R_0 \rangle\big) \big/ \langle \phi_{A,0}(P + bQ), \phi_{A,0}(T_0) \rangle \\
&= E \big/ \langle R_0, P + bQ, T_0 \rangle \\
&\simeq E \big/ \langle R_0, T_0 \rangle \big/ \langle P + bQ \rangle \\
&= E \big/ E[2^n] \big/ \langle P + bQ \rangle \\
&= E \big/ \langle P + bQ \rangle \\
&= E_B
\end{aligned}
$$

3.3.2 Security

For this section it is important to first introduce few hard problems that we are going to use to analyse the security of the protocol introduced in the previous section. We will follow [9, 12].

XDSSI - *Extended Decisional Supersingular Isogeny* problem

Given two supersingular elliptic curves E and E' defined over $\mathbb{F}_{p^2}$, the pairs (U, V), (U', V') such that (U, V) is a basis for $E[2^n]$ and (U', V') is a basis for $E'[2^n]$, and having Weil pairing $e_{2^n}(U, V)^{3^m} = e_{2^n}(U', V')$; determine if there exists a 3^m-isogeny $\phi : E \to E'$ such that $\phi(U) = U'$ and $\phi(V) = V'$.

DSSI - *Decisional Supersingular Isogeny* problem

Given the prime p of the form $l_A^{e_A} l_B^{e_B} \pm 1$ where l_A and l_B are small primes, e_A and e_B are integers, the supersingular curves E_0 and E_A defined over $\mathbb{F}_{p^2}$, decide whether E_A is $l_A^{e_A}$-isogenous to E_0.

CSSI - *Computational Supersingular Isogeny* problem

Consider a supersingular elliptic curve E defined over $\mathbb{F}_{p^2}$, the bases (U, V) for $E[2^n]$ and (P, Q) for $E[3^m]$, an integer $a \in \mathbb{Z}/2^n\mathbb{Z}$, the isogeny $\phi_A : E \to E_A$ with kernel $ker\, \phi_A = \langle U + aV \rangle$. Given E_A and the points $\phi_A(P)$, $\phi_A(Q)$, determine the isogeny kernel $ker\, \phi_A$.

SSCDH - *Supersingular Computational Diffie-Hellman Isogeny* problem

Let E_0 be a supersingular elliptic curve defined over $\mathbb{F}_{p^2}$ and let $\phi_A : E_0 \to E_A$ be an isogeny with kernel $\langle [m_A]P_A + [n_A]Q_A \rangle$ where (P_A, Q_A) is a basis for $E_0[2^n]$ and m_A and n_A are chosen at random from $\mathbb{Z}/2^n\mathbb{Z}$ and not both divisible by 2. Let $\phi_B : E_0 \to E_B$ be an isogeny with kernel $\langle [m_B]P_B + [n_B]Q_B \rangle$ where (P_B, Q_B) is a basis for $E_0[3^m]$ and m_B and n_B are chosen at random from $\mathbb{Z}/3^m\mathbb{Z}$ and not both divisible by 3.
Given the curves E_A, E_B and the points $\phi_A(P_B)$, $\phi_A(Q_B)$, $\phi_B(P_A)$, $\phi_B(Q_A)$, find the j-invariant of $E_{AB} = E_0/\big\langle [m_A]P_A + [n_A]Q_A,\, [m_B]P_B + [n_B]Q_B \big\rangle$.

SSDDH - *Supersingular Decision Diffie-Hellman* problem

Given a tuple sampled with probability 1/2 from one of the following two distributions:

- $\Big(E_A, E_B, \phi_A(P_B), \phi_A(Q_B), \phi_B(P_A), \phi_B(Q_A), E_{AB}\Big)$, where all the parameters are as in the *SSCDH* problem and $E_{AB} = E_0/\big\langle [m_A]P_A + [n_A]Q_A,\, [m_B]P_B + [n_B]Q_B \big\rangle$;
- $\Big(E_A, E_B, \phi_A(P_B), \phi_A(Q_B), \phi_B(P_A), \phi_B(Q_A), E_C\Big)$, where all the parameters are as in the *SSCDH* problem except for $E_C = E_0/\big\langle [m'_A]P_A + [n'_A]Q_A,\, [m'_B]P_B + [n'_B]Q_B \big\rangle$ where m'_A, n'_A (resp. m'_B, n'_B) are chosen at random from $\mathbb{Z}/2^n\mathbb{Z}$ (resp. $\mathbb{Z}/3^m\mathbb{Z}$) and not both divisible by 2 (resp. 3);

determine from which distribution the tuple is sampled.

2-inv-CSSI - *2-inverse Computational Supersingular Isogeny* problem

Let E, E_0, E_1 be three supersingular elliptic curves defined over $\mathbb{F}_{p^2}$ such that E_0 and E_1 are 2^n-isogenous to E with the corresponding isogenies denoted by $\phi_0 : E \to E_0$, $\phi_1 : E \to E_1$. Let (P, Q) be a basis of $E[3^m]$ and for each $i = \{0, 1\}$, let (U_i, V_i) be a basis of $E_i[2^n]$ and (x_i, y_i) be the coordinates with respect to this basis of a generator of the kernel of the dual isogeny $\widehat{\phi_i}$. Given the points $P, Q, U_0, V_0, U_1, V_1, \phi_0(P), \phi_0(Q), \phi_1(P), \phi_1(Q)$, find three supersingular elliptic curves E', F_0, F_1 and a basis $(U', V') \in E'[2^n]$ such that $F_0 \simeq E'/\langle x_0U' + y_0V' \rangle$ and $F_1 \simeq E'/\langle x_1U' + y_1V' \rangle$.

2-inv-DSSI - *2-inverse Decisional Supersingular Isogeny* problem

This problem uses the same notations of the *2-inv-CSSI* problem for the following parameters: curves E, E_0, E_1, points $P, Q, U_0, V_0, U_1, V_1, \phi_0(P), \phi_0(Q), \phi_1(P), \phi_1(Q)$. We consider the following game between a party Bob and a challenger oracle:

- Bob initially knows E, E', U', V', and can only query the oracle;
- Bob sends to the challenger oracle a supersingular elliptic curve E' and the basis (U', V') for $E'[2^n]$;
- the oracle chooses four integers x_0, y_0, x_1, y_1 and the random points W_0 and W_1 of $E'[2^n]$. Then it computes the supersingular curves $F_0 = E'/\langle x_0U' + y_0V' \rangle$, $F_1 = E'/\langle x_1U' + y_1V' \rangle$, $F_0' = E'/\langle W_0 \rangle$, $F_0' = E'/\langle W_1 \rangle$;
- the oracle chooses uniformly and independently two bits b_0, b_1. Then it outputs two pairs (C_0, C_0') and (C_1, C_1') of supersingular curves such that:

$$(C_0, C_0') = \begin{cases} (F_0, F_0') \text{ if } b_0 = 0 \\ (F_0', F_0) \text{ if } b_0 = 1 \end{cases} \quad \text{and } (C_1, C_1') = \begin{cases} (F_1, F_1') \text{ if } b_1 = 0 \\ (F_1', F_1) \text{ if } b_1 = 1 \end{cases}$$

- Bob must guess whether $b_0 = b_1$ or $b_0 \neq b_1$.

Bob's advantage in this game is defined as "$\mathcal{P}$*(c) - 1/2*", where $\mathcal{P}$*(c)* is the probability of answering correctly. Then the *2-inv-DSSI* problem is hard if no algorithm can achieve a non-negligible advantage for Bob in probabilistic polynomial time.

We have now introduced all the necessary hard problems and can finally approach the security model for the protocol. Starting with Alice, we must ensure that she cannot discover Bob's secret k otherwise she would know which of her secrets Bob would know hence breaking the security offered by the oblivious transfer. In a similar way, we must ensure that it is infeasible for Bob to discover the other secrets of Alice. This case is worse than the latter since it would completely exposes Alice messages if Bob were successful.
The two cases described are named Malicious Alice and Malicious Bob respectively. We will now describe them both and prove that both cases are computationally infeasible thus making the SIDH-based OT a secure post-quantum protocol.

Malicious Alice

Malicious Alice's analysis focuses on Bob's message:

$$\xleftarrow{\text{Sends Alice } \{E'_B,\, U'_k,\, V'_k\} \text{ and the } \textit{two} \text{ basis } (U_0,\, V_0),\, (U_1,\, V_1)}$$

With this message, Alice knows $\{E'_B, U'_k, V'_k\}$, even more she knows that E'_B is 3^m-isogenous to one of her $E_{A,i}$: she might be able to recover Bob's secret k by finding which of her curves is isogenous to E'_B. This problem is formalised as the *Decisional Supersingular Isogeny* (*DSSI*) problem and is expected to be computationally intractable [12]. Nevertheless Alice has more information at her disposal: $\phi'_B(U_k)$, $\phi'_B(V_k)$ and the *two* pairs (U_i, V_i). It is possible now to consider a Weil pairing's property such that:

$$e(\phi'_B(U_k), \phi'_B(V_k)) = e(U_k, V_k)^{deg\, \phi'_B} = e(U_k, V_k)^{3^m}$$

Now recall when in Section 3.3 Bob was required to compute *two* bases (U_0, V_0), (U_1, V_1) for $E[2^n]$. The bases were required to be chosen such that they have the same Weil pairing e_{2^n}. What would happen if Bob calculated his *two* bases with different pairings? Alice would be able to check all bases until she finds the correct one for which the condition above holds, as a consequence she would know Bob's k. It is then mandatory for Bob to choose bases having the same Weil pairing.
In order to prevent this scenario, [9] considers the *XDSSI* problem and its computational intractability. Considering the protocol settings, the *XDSSI* is equivalent to the the *CSSI* problem. Since the latter is considered a hard problem then consequently the former *XDSSI* is hard too and the presented protocol is secure with respect to Bob's secret k.

Malicious Bob

The security in the random oracle model relies on the hardness of the *2-inv-CSSI* problem since if the generators of $ker(\phi_0)$ and $ker(\phi_1)$ can be efficiently computed, then it would be easy to obtain x_0, y_0, x_1, y_1 and solve the *2-inv-CSSI*. A malicious Bob should solve the *2-inv-CSSI* without computing x_0, y_0, x_1, y_1, that is, solving the *CSSI*. In fact it would require to find a curve E' and points U', V' such that U' (resp. V') is related to both U_0, U_1 (resp. V_0, V_1). The problem described is precisely

how the oblivious transfer works although it is expected to be computationally infeasible even on a quantum computer.
There could be another way for Bob to break the scheme and it requires him to send Alice a pair (U', V') that is not a $E'[2^n]$ basis. In this way Bob may choose a smaller basis thus limiting the possible values of $x_iU' + y_iV'$. For a small enough basis, it would be feasible for Bob to compute all the possible values of $x_iU' + y_iV'$. He would then be able to compute the curves all the possible curves of the form $F_j = E'_B/\langle x_iU' + y_iV' \rangle$, in particular he would find the curves F_0, F_1, the same computed by Alice! At this point Bob would need to compute the j-invariant for every curve F_j and use them as *seeds* for the *KDF*. By applying each seed to each of Alice's secrets, Bob would eventually be able to decrypt all of her secrets.
In order to avoid this situation, Alice should always perform a safety check on the received basis before proceeding further.

Ultimately it is assumed that the combination of the symmetric encryption scheme *Enc* and the *KDF* is IND-CPA (Indistinguishable under Chosen Plaintext Attack). In this case Bob is in the situation to guess whether his chosen secret is b_0 or b_1 hence he must solve the *2-inv-DSSI* problem. Even though this problem is expected to be easier than its computational version *2-inv-CSSI* there is no known reduction to the *SSDDH* problem therefore the *2-inv-DSSI* can be considered computationally intractable.

Chapter 4

Implementation details

The SIDH-based OT implementation strongly relies on the functions used in SIDH implementations. For the purposes of this these we have chosen the *Supersingular Isogeny Key Encapsulation* (SIKE) library [11] as a starting point in our implementation. Along with said library, additional functions have been coded from scratch in order to reproduce the exact behaviour of SIDH-based OT protocol.

4.1 SIKE Library

This section presents a brief disclosure on the SIKE library, a collection of functions and definitions to make the SIDH Key Exchange possible. For all the information in this section and in Section 4.1.1 the reader can refer to [11].
The library is presented as a collection of different base implementations: one for portable C; one for x64 platforms; two for ARM64 and FPGA optimising different aspects of the platform; finally a simple textbook implementation. Many of these are protected against timing and cache attacks at the software level. Moreover the optimised x64 implementation is further subdivided into *standard* and *compressed* versions, both of them support four different parameters sets: p_{434}, p_{503}, p_{610}, p_{751}. The latter implementation offers compressed parameters sets and optimised, state-of-the-art, functions and strategies to achieve the highest speed ups possible. Ultimately, our choice is the x64 optimised, compressed, *SIKEp503_compressed* which also grants us to easily and readily adapt our code to the other *compressed* param-

eters sets.

4.1.1 x64 optimised, compressed implementations

The software is written in portable C and, in addition to the standard optimised version, provides public key compression and key encapsulation.
The compression is performed for both the static and ephemeral public keys.
The uncompressed public key (resp. ciphertext) sizes corresponding to *SIKEp434* and *SIKEp503* are 330 and 378 (resp. 378 and 402) bytes, which is comparable to the 384-byte (3072-bit) modulus that is conjectured to offer 128 bits of classical security. Likewise, *SIKEp610* public key (resp. ciphertext) sizes are 462 (resp. 486) bytes, and the largest of our parameter sets, SIKEp751, has 564-byte uncompressed public keys and 596-byte ciphertexts. On NIST's Challenge Round II there have been a further public key and ciphertext compression; this reduces all of the above numbers to roughly 60% of their former size, for performance overheads ranging from 139% to 161% during public key generation, from 66% to 90% during encapsulation, and from 59% to 68% during decapsulation. Finally there are reduced public key sizes by 41% and reduced ciphertext sizes by 39%.

The implementation offers efficient algorithms for:

- isogeny computations and tree traversing strategies;
- elliptic curves computations using projective coordinates on Montgomery curves;
- scalar multiplication via 3-points Montgomery Ladder.

Field operations on $\mathbb{F}_{p^2}$ make use of Karatsuba and Lazy Reduction techniques: multiprecision multiplication is implemented with a fully rolled version of Comba while the modular reduction with a Montgomery reduction. Additionally this implementation is common to all the security levels hence offers great code reuse.
Protocol performances have been evaluated with a benchmark test on a 3.4GHz Intel Core i7-6700 processor running Ubuntu 16.04.3 LTS; TurboBoost disabled, clang 3.8.0 with the command "*clang -03*". Table 4.1 shows the comparison between the two implementations. For a memory analysis, in table 4.2 are reported the sizes in bytes of secret and public keys, ciphertexts and shared secrets used in SIKE.

Scheme	*Key Generation*	*Encapsulation*	*Decapsulation*
Optimised implementations			
SIKEp434	56.264	92.180	98.335
SIKEp503	86.067	141.891	150.879
SIKEp610	160.401	294.628	296.577
SIKEp751	288.827	468.175	502.983
Compressed implementations			
SIKEp434_compressed	16.542	20.045	18.930
SIKEp503_compressed	23.395	27.543	25.534
SIKEp610_compressed	40.386	47.099	45.449
SIKEp751_compressed	62.347	78.748	72.774

Table 4.1: SIKE performances in thousands of clock cicles (rounded down)

Scheme	*Secret Key*	*Public Key*	*Ciphertext*	*Shared Secret*
SIKEp434	374	330	346	16
SIKEp503	434	378	402	24
SIKEp610	524	462	486	24
SIKEp751	644	564	596	32
SIKEp434_compressed	239	196	209	16
SIKEp503_compressed	280	224	248	24
SIKEp610_compressed	336	273	297	24
SIKEp751_compressed	413	331	363	32

Table 4.2: SIKE inputs and outputs sizes in bytes

4.1.2 Implementation Choices

Most parameters used in our SIDH-based OT come from the SIKE library due to its high performances and optimisation. Therefore, in this section we are going to describe SIKE's choices and so, by reflection, ours. For the entirety of this section, the reader can refer to [7].

Smooth Order Supersingular Elliptic Curves

Prime numbers in the form of $p = l_A^{e_A} l_B^{e_B} \pm 1$ have bases $l_A = 2$ and $l_B = 3$ fixed, the exponents are chosen such that the resulting numbers 2^{e_A} and 3^{e_B} have bit lengths slightly smaller than multiples of 64. This choice allows to have $2^{e_A} \approx 3^{e_B}$ in order to ensure that attacking one party is not considerably easier than attacking the other. This also serves to balance the computational costs for each party without disadvantaging either. This choice also supports efficient arithmetic on many platforms and allows a large variety of optimisations. Among all, it is well-known that prime numbers having a special form (like the former just described above) can improve algorithm performances for the underlying modular arithmetic. As a brilliant side effect, these primes ease the construction of supersingular elliptic curves E over $\mathbb{F}_{p^2}$ with the consequence of having smooth order $(l_A^{e_A} l_B^{e_B})^2$. Given $l \in \{l_A, l_B\}$ and $e \in \{e_A, e_B\}$, the l^e-torsion group on E is defined over $\mathbb{F}_{p^2}$; furthermore, since l is coprime to p then $E[l^e] \simeq (\mathbb{Z}/l^e\mathbb{Z})^2$. Let $(P, Q) \in E[l^e]$ be a basis of order l^e such that we have the isomorphism $\phi : (\mathbb{Z}/l^e\mathbb{Z})^2 \to E[l^e]$, such that $\phi(m, n) = [m]P + [n]Q$ (see Section 2.2.1). SIDH secret keys are the l^e-isogenies of E which are, in turn, in one-to-one correspondence with the cyclic subgroups of order l^e with $\ker(\phi) = \langle P, Q \rangle$. Finally, a point $[m]P + [n]Q$ has full order l^e if and only if at least either m or n are coprime to l.

Avoid Inversions

The idea behind the use of Montgomery projective curves is the combination of two techniques aiming to minimise the number of *inversions*, which is a high cost operation. First we consider elliptic curves in a projective space, then we exploit the fast arithmetic of Montgomery curves to efficiently compute points in $\mathbb{P}^1$. Following the

notations introduced in Section 2.2, with the application of Montgomery projective curves it has been possible to code more compact algorithms ultimately achieving:

- faster point arithmetic by ignoring the Y projective coordinate;
- faster and more efficient isogeny arithmetic by ignoring the Montgomery curve constant B and working with the pair $(A : C) \in \mathbb{P}^1$;
- key generation with only one inversion instead of three;
- shared secret (j-invariant) computation with only two inversions in the function $j\_inv : (A : C) \rightarrow j(E_{(A:B:C)})$.

Projective Isogenies

With this section we want to briefly summarise isogeny computations in SIKE library which are also used in our implementation.
The fundamental isogenies are 3- and 4-degree evaluated with an isogeny walk and a traversing strategy Section 2.3.3. Starting with the 3-isogenies case we define:

- the supersingular elliptic curve $E_{(A:C)}$ expressed in projective Montgomery form;
- the affine x-coordinate $x(P) = (X_3 : Z_3) \in \mathbb{P}^1$ such that the point P has order 3 in $E_{(A:C)}$;
- the supersingular elliptic curve $E'_{(A':C')} = E_{(A:C)}/\langle P \rangle$;
- the 3-isogeny $\phi : E_{(A:C)} \rightarrow E'_{(A':C')}$;
- the point $Q \in E_{(A:C)}$ such that $Q \notin ker(\phi)$, its affine x-coordinate $x(Q) = (X : Z) \in \mathbb{P}^1$ and its image $x(\phi(Q)) = (X' : Z') \in \mathbb{P}^1$

At this point we have two functions at our disposal: *get_3_isog* which computes the isogenous curve $E'_{(A':C')}$ only and *eval_3_isog* to evaluate a single point in the image curve. The first function takes the curve $E_{(A:C)}$ and the projective coordinates

$(X_3 : Z_3)$ of the point P as input and outputs the 3-isogenous curve $E'_{(A':C')}$. This curve is computed with the expression:

$$(A' : C') = \big((AX_3Z_3 + 6(Z_3^2 - X_3^2))X_3 : CZ_3^3\big)$$

which is independent from $E_{(A:C)}$ coefficients and it requires 3 multiplications, 3 squaring, 8 additions. Using a common notation we could also write $3\mathbf{M}+3\mathbf{S}+8\mathbf{a}$ where **M** stands for Multiplication, **S** for Squaring, **a** for Addition, eventually **I** for Inversion.
To evaluate the point Q, by *eval_3_isog*, we use:

$$(X' : Z') = \big(X(X_3X - Z_3Z)^2 : Z(Z_3X - X_3z)^2\big)$$

requiring $6\mathbf{M}+2\mathbf{S}+2\mathbf{a}$.

Passing on 4-isogenies we can reuse the above notations and only change the point P with a point of order 4 in $E_{(A:C)}$ with affine x-coordinate $x(P) = (X_4 : Z_4) \in \mathbb{P}^1$. In the isogeny walk used to compute a 4^e-isogeny, an isomorphism is needed for every 4-isogeny step except for the first one. The following formulae consider a curve $E_{(A:C)}$ normalised so that $A = a$, $C = 1$ where a is the Montgomery curve coefficient. For the first 4-isogeny, the image curve $E'_{(A':C')}$ is computed as:

$$(A' : C') = \big(2(a + 6) : a - 2\big)$$

while the image of a point Q is:

$$(X' : Z') = \big((X + z)^2(aXZ + X^2 + Z^2) : (2 - a)XZ(X - Z)^2\big)$$

requiring $4\mathbf{M}+2\mathbf{S}+9\mathbf{a}$.

For all the other 4-isogenies we get:

$$(A' : C') = \big(2(2X_4^4 - Z_4^4) : Z_4^4\big)$$

while the image of a point Q is:

$$(X' : Z') = \Big(X\big(2X_4Z_4Z - X(X_4^2 + Z_4^2)\big)(X_4X - Z_4Z) \; : \\ Z\big(2X_4Z_4X - Z(X_4^2 + Z_4^2)\big)(Z_4X - X_4Z)\Big)$$

Bases and Torsion Points

Throughout the protocol, Alice and Bob make use of two particular bases: Alice's basis is (P_A, Q_A) of order 2^n and Bob's (P_B, Q_B) of order 3^m. To better explain how these bases had been computed we are going to follow the example in [7] thus we set $n = 372$, $m = 239$.
Let $P_A \in E_0(\mathbb{F}_p[2^{372}])$ be a point described as $[3^{239}](z, \sqrt{z^3 + z})$ where z is the smallest positive integer such that $\sqrt{z^3 + z} \in \mathbb{F}_p$ and P_A is of order 2^{372}. To compute the second point Q_A we use a distortion map $\tau : E_0(\mathbb{F}_p[2^{372}]) \to E_0(\mathbb{F}_p[2^{372}])$, $(x, y) \to (-x, iy)$, hence $Q_A = \tau(P_A)$. Bob's basis is found similarly.

Having now two distinct bases, one may exploit a precise linear combination of basis points to sample a full order torsion point.
To sample a 2^{372}-order point $R_A \in E_0(\mathbb{F}_p[2^{372}])$, it is possible to choose a random integer $m' \in \{1, 2, \ldots, 2^{371} - 1\}$ and set $R_A = P_A + [2m']Q_A$.
A similar approach is used to sample 3^{239}-order points: choose a random integer $m' \in \{1, 2, \ldots, 3^{238} - 1\}$ and set $R_B = P_B + [3m']Q_B$.

4.2 SIDH-based OT implementation

In this section we will comment about what has been implemented for this thesis. The entire protocol requires less than 100 lines of code but many functions had been adapted or newly added to achieve our purpose. For a closer look to our algorithms we will refer to (appendix A); for the bottom implementation, the reader can still refer to SIKE's library [11].

Initial settings

In order to exploit the SIKE library, the first thing required is to include the header files *api.h*, *config.h*, *Pxxx_internal.h* and the C file *Pxxx.c*. The "*Pxxx*" stands for any curve belonging to the optimised, compressed implementation.

The protocol's public parameters have been set as follows:

- The supersingular elliptic curve E defined over $\mathbb{F}_{p^2}$ is hard-coded in the *Pxxx* chosen at compile time;
- The bases (P, Q) of $E[3^m]$ is hard-coded according to the *Pxxx* chosen;
- The secure symmetric encryption protocol *Enc* and the key derivation function *KDF* are coded into a single function Algorithm 6 in the main program.

The *main* program defaults to a $\binom{2}{1}$-OT but it is possible to run it for the generic $\binom{s}{1}$-OT at running time without the need to recompile it. We have used the letter s for the generic OT in order to avoid confusion when we introduce Alice bases having order 2^n.
Finally, four constants have been used to fully describe a Montgomery curve. Among those shown in Section 2.3.1, we used only $(A_{24}^+ : C_{24})$ computed accordingly to SIKE library: $A_{24}^+ = A + 2C$, $C_{24} = 4C$ where $A = 6$ and $C = 1$.

Part 1 - Alice

Alice initial settings require her to generate from *two* up to s secrets, according to the OT version chosen.
The scenario in now divided into three main functions:

- The first one generates *two* (resp. s) bases (R_i, T_i) for $E[2^n]$, with $i = 0, 1$ (resp. $i = 0, \ldots, s-1$), by adapting and exploiting the SIKE's function *get_2_torsion_entangled_basis_compression*;
- The second function computes the difference between the projective points. Its scope is to calculate the difference $T_i - R_i$ for each of Alice's bases in order to fully exploit the optimisation of isogeny computation as implemented in SIKE library;
- Finally the third and last function Algorithm 2 computes *two* (resp. s) isogenous curves $E_{A,i} = E/\langle R_i \rangle$, via the isogeny $\phi_{A,i} : E \to E_{A,i}$. For each one of these curves, four points are also computed which will be required further on. Three of these points represent the image of 3^m public base points and their difference: $\phi_{A,i}(P)$, $\phi_{A,i}(Q)$, and $\phi_{A,i}(Q - P)$. The last point is the image of T_i by $\phi_{A,i}$, namely $\phi_{A,i}(T_i)$.

For computing the isogeny we rely on SIKE implementation and on ad-hoc adaptations based on the required output. Each isogeny is implemented following the idea given in Section 2.3.3.

Part 2 - Bob

Bob starts with choosing a random integer k out of *two* (resp. s) and computing the point $P + bQ$, with b uniformly chosen in $\mathbb{Z}/3^m\mathbb{Z}$. Now he is able to compute the curve $E_B = E/\langle P + bQ \rangle$ and its j-invariant j_B. The function Algorithm 4 to compute the isogenous curve E_B has a different implementation compared to Algorithm 1 used by Alice. In this case the kernel is generated by a cyclic subgroup of order 3^m: this affects how the isogeny's traversing strategy is evaluated. Moreover we have now a sequence of m 3-isogenies and no image points are computed nor returned, since not required. This fact speeds up our function when compared to SIKE's original implementation.

The next step is to evaluate *two* (resp. s) bases for $E[2^n]$, namely (U_i, V_i), all having the same Weil pairing. We have coded the necessary functions, loops and checks in order to meet the latter requirements, in particular: two new functions have been implemented and two have been modified. The SIKE function to compute the Weil pairing is rather generic and allows for computing pairings of any order. Following Section 2.3.2, we took every basis (U_i, V_i) and multiplied it by 2^{n-1}, obtaining $([2^{n-1}]U_i, [2^{n-1}]V_i)$, allowing us to use a simple pairing of order 2. For each basis it has been mandatory also to compute the difference $V_i - U_i$ as it will be required in the next steps.
At this point it was finally possible to compute Algorithm 3 the last curve of Bob $E'_B \simeq E_{A,k}/\langle P_k + bQ_k \rangle$ and his last isogeny $\phi'_B = E_{A,k} \to E'_B$. To compute the isogeny, first the point $P_k + bQ_k$ has been computed and then a new function has been used for the isogeny. In this case, the function required three additional input points U_k, V_k, $V_k - U_k$ and returned their images $U'_k = \phi'_B(U_k)$, $V'_k = \phi'_B(V_k)$, and $\phi'_B(V_k - U_k)$.

Part 3 - Alice

Alice now has to find *two* (resp. s) linear combinations in terms of x_i, y_i such that $\phi_{A,i}(T_i) = x_i U_i + y_i V_i$. To approach this problem one may choose to set a value for x_i and increment y_i while the condition above does not hold. In a similar way, we opted to randomly choose values for both the terms until they do not meet the condition above Algorithm 5. Both approaches have the great drawback of requiring $\mathcal{O}(n)$ operations therefore being computationally intractable. For the moment we decided to select the first random values obtained and proceed with the protocol, leaving this problem to future developments.

Now, we computed the *two* (resp. s) isogeny kernels $x_i U'_k + y_i V'_k$, used each one to obtain the final curves $F_i = E'_B / \langle x_i U'_k + y_i V'_k \rangle$, exploiting yet another ad-hoc function Algorithm 1. In this case we only focused on computing the isogenous curve bypassing all supplementary points and images.
Only two operations remain at this point: computing the j-invariant for all the curves F_i and then encrypting Alice secrets with the *Enc* function. The former operation strictly relies on SIKE implementation, the latter is a new function that we implemented.
The *Enc* function Algorithm 6 takes two parameters, a secret and a j-invariant, and returns the encrypted secret. Obviously we may pass an encrypted secret and get the decrypted secret as output, this behaviour makes the function symmetric and does not require additional functions to be implemented. Lastly, we used the j-invariant parameter to generate a seed via the *shake256* function provided in SIKE library and based on SHA-3 implementation [36].

Part 4 - Bob

This final part is simply another application of the *Enc* function Algorithm 6 in which Bob submits the k-th Alice encrypted secret and his j-invariant j_B. The output is the k-th Alice decrypted secret and the protocol is now completed.

Chapter 5

Conclusions and Future Development

In this thesis we have studied thoroughly the paper [9] of Vanessa Vitse and decided to create the first implementation of her newly designed protocol. To accomplish our goal we run into several difficulties many of which are now overcome and only few of them still remain unsolved but we hope to find different solutions in future developments.
We have been able to study and use the best state-of-the-art optimisations regarding elliptic curve cryptography; we have exploited and adapted one of the best known post-quantum software library; finally, we have implemented the SIDH-based OT protocol in such a general way to automatically adapt to every other curve SIKE's library may introduce in the future.

One of the most important difficulty to be solved is the correct and efficient way to compute the linear combination $\phi(T) = xU + yV$ to complete Alice's part in the protocol. Minor changes can still be applied like: optimising the *Makefile* file to better take advantage of SIKE library and our compiled program; optimising function imports and inclusions; uniforming the forms of the point coordinates to the projective short form $(X : Z)$; avoiding frequent variable conversions between different functions; writing more of our own functions to better fit our requirements.

Finally it would be interesting to implement different post-quantum Oblivious Transfer protocols and benchmark them.

First of all we may cite a second Vanessa Vitse idea [9] based off the exponentiation-only OT scheme of Wu-Zhang-Wang [38]. This protocol is hard to implement since it requires to efficiently sample random supersingular elliptic curves over a given finite field, a problem which is difficult to overcome.

An easier protocol to implement is the Supersingular Isogeny Oblivious Transfer described in [39]. In a way similar to SIDH in Section 3.1, the public parameters are: a supersingular elliptic curve E; the bases (P_A, Q_A) for $E[l_A^{e_A}]$, (P_B, Q_B) for $E[l_B^{e_B}]$, (U, V) for $E_B[l_A^{e_A}]$; a symmetric encrypting function Enc as requested in 3.3. Alice chooses a random integer $r_A \in \mathbb{Z}/l_A^{e_A}\mathbb{Z}$ and computes and sends to Bob her public key $pk_A = \{E_A, G_A, H_A\}$, with the isogenous curve $E_A = E/\langle P_A + r_A Q_A \rangle$ and two image points $G_A = \phi_A(P_B)$, $H_A = \phi_A(Q_B)$. Bob chooses a random integer $r_B \in \mathbb{Z}/l_B^{e_B}\mathbb{Z}$ and computes his first public key $pk_B = \{E_B, G_B, H_B\}$ in a similar way to Alice. He also picks a random bit $\sigma \in \{0, 1\}$. Lastly, he computes and sends to Alice his second public key as $\hat{pk}_B = \{E_B, G_B - \sigma U, H_B - \sigma V\}$. Alice computes the isogeny $\phi'_{A_i} : E_B \to E_{BA_i}$ for each $i \in \{0, 1\}$, where $E_{BA_i} = E_B/\langle (G_B - \sigma U + iU) + r_A(H_B - \sigma V + iV) \rangle$. At this point she encrypts her i-th secret using the j-invariant of E_{BA_i} and send all her encrypted secrets to Bob. The protocol terminates with Bob been able to decrypt the σ-th secret using the j-invariant of $E_A B = E_A/\langle (G_A + r_B H_A) \rangle$.
This protocol shows no difficult implementation problems at first glance and could be faster than Vitse's SIDH-based OT since it does not require to generate random bases at run time, nor to compute any linear combinations in the form of our Algorithm 5.

Concluding our disclosure, many more OT protocols can be found in literature that could compete with the SIDH-based OT. Among others, two oblivious transfer protocols are described in [40] but both require additional mathematical knowledge than what we presented in Section 2.2, hence falling beyond the scope of this thesis.

Acknowledgements

Two years ago, I could hardly imagine to get here like I am today. My goals and determination have brought me far through this master degree but much much more I owe to all the people who have been around me this time.
First of all I believe my parents deserve a big thank for always being supportive in my decisions, and to my sisters for holding me tight to their hearts.
Secondly, I find it necessary to, at least, mention my professors Riccardo and Federico for helping me greatly in this thesis, Norberto Gavioli for giving tips from time to time and Gabriele Di Stefano for having me introduced into elliptic curves cryptography and still being earnestly interested in my personal growth.

I feel necessary to give a big shoutout to all the friends I met and are close to me. Even if anti climatic I am grateful for some exam circumstances for letting me sympathise with new fellowships giving me a jump start into the first lessons.
Roommates friends: genius, somewhat funny, comprehensive and wise have been at my side, helping me and bearing me all day, all days. Good friends come along good friends. A blessed friend, neverending boon so kind yet sometime short-tempered, ceaselessly there for me; the sun on the horizon line. Faraway friends, separated by distance but everlasting, old, great, best friends, always at arm's reach.

I love all of you.
And my newly catched Sylveon.

Bibliography

[1] Heyse, S. (2013). Post quantum cryptography: implementing alternative public key schemes on embedded devices. Doctorat, Bochum Germany.

[2] Szepieniec, A. (2018). Mathematical and Provable Security Aspects of Post-Quantum Cryptography.

[3] Mosca, M. (2018). Cybersecurity in an era with quantum computers: will we be ready?. *IEEE Security & Privacy, 16(5)*, 38-41.

[4] Legernæs, M. W. (2018). *On the Development and Standardisation of Post-Quantum Cryptography-A Synopsis of the NIST Post-Quantum Cryptography Standardisation Process, its Incentives, and Submissions* (Master's thesis, NTNU).

[5] Shor, P. W. (1999). Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review, 41*(2), 303-332.

[6] De Feo, L. (2017). Mathematics of isogeny based cryptography. *arXiv preprint arXiv:1711.04062.*

[7] Costello, C., Longa, P., & Naehrig, M. (2016, August). Efficient algorithms for supersingular isogeny Diffie-Hellman. In *Annual International Cryptology Conference* (pp. 572-601). Springer, Berlin, Heidelberg.

[8] Costello, C., & Smith, B. (2018). Montgomery curves and their arithmetic. *Journal of Cryptographic Engineering, 8*(3), 227-240.

[9] Vitse, V. (2019). Simple oblivious transfer protocols compatible with Kummer and supersingular isogenies.

[10] Luca De Feo. (2018). Isogeny based crypto: what's under the hood?

[11] Jao, D. (2019). Supersingular isogeny key encapsulation 2019. *NIST Round, 2*.

[12] De Feo, L., Jao, D., & Plût, J. (2014). Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. *Journal of Mathematical Cryptology, 8*(3), 209-247.

[13] Federico Pintore, (2019). Cryptographic primitives from elliptic curve isogenies. *De Cifris Athesis*

[14] Petit, C. (2017, December). Faster algorithms for isogeny problems using torsion point images. In *International Conference on the Theory and Application of Cryptology and Information Security* (pp. 330-353). Springer, Cham.

[15] Galbraith, S. D., Petit, C., Shani, B., & Ti, Y. B. (2016, December). On the security of supersingular isogeny cryptosystems. In *International Conference on the Theory and Application of Cryptology and Information Security* (pp. 63-91). Springer, Berlin, Heidelberg.

[16] Longa, P. (2018). A Note on Post-Quantum Authenticated Key Exchange from Supersingular Isogenies. *IACR Cryptology ePrint Archive, 2018*, 267.

[17] Cid, C., & Jacobson Jr, M. J. (Eds.). (2019). *Selected Areas in Cryptography–SAC 2018: 25th International Conference, Calgary, AB, Canada, August 15–17, 2018, Revised Selected Papers* (Vol. 11349). Springer.

[18] Daemen, J., & Rijmen, V. (2002). The design of Rijndael. Information security and cryptography.

[19] Rivest, R. L., Shamir, A., & Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2), 120-126.

[20] Certicom. (2004). An Elliptic Curve Cryptography (ECC) Primer. *Certicom Research*.

[21] Deutsch, D. (1985). Quantum theory, the Church–Turing principle and the universal quantum computer. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences, 400*(1818), 97-117.

[22] Jurišic, A., & Menezes, A. (1997). Elliptic curves and cryptography. *Dr. Dobb's Journal*, 26-36.

[23] Silverman, J. H. (2009). *The arithmetic of elliptic curves* (Vol. 106). Springer Science & Business Media.

[24] Silverman, J. H. (2006). An introduction to the theory of elliptic curves. *Brown University. June, 19*.

[25] Christophe Ritzenthaler, (2014). Introduction to elliptic curves. Université de Rennes I.

[26] Luca De Feo, (2019). Isogeny Graphs in Cryptography. Université Paris Saclay, UVSQ.

[27] Costello, C., & Hisil, H. (2017, December). A simple and compact algorithm for SIDH with arbitrary degree isogenies. In *International Conference on the Theory and Application of Cryptology and Information Security* (pp. 303-329). Springer, Cham.

[28] Even, S., Goldreich, O., & Lempel, A. (1985). A randomized protocol for signing contracts. *Communications of the ACM*, 28(6), 637-647.

[29] Kute, V. B., Paradhi, P. R., & Bamnote, G. R. (2009). A software comparison of rsa and ecc. *Int. J. Comput. Sci. Appl, 2*(1), 43-59.

[30] Bosamia, M., & Patel, D. (2018). Current Trends and Future Implementation Possibilities of the Merkel Tree. *International Journal of Computer Sciences and Engineering, 6*(8), 294-301.

[31] Zhang, W., & Tan, C. H. (2015, December). MI-T-HFE, a new multivariate signature scheme. In *IMA International Conference on Cryptography and Coding* (pp. 43-56). Springer, Cham.

[32] Sendrier, N. (2011). Niederreiter Encryption Scheme. 842-843.

[33] Biswas, B., & Sendrier, N. (2008, October). McEliece cryptosystem implementation: Theory and practice. In *International Workshop on Post-Quantum Cryptography* (pp. 47-62). Springer, Berlin, Heidelberg.

[34] Hoffstein, J., Pipher, J., & Silverman, J. H. (1998, June). NTRU: A ring-based public key cryptosystem. In *International Algorithmic Number Theory Symposium* (pp. 267-288). Springer, Berlin, Heidelberg.

[35] Rabin, M. (1981). *How to exchange secrets by oblivious transfer. Harvard Aiken Comp. Lab.* TR-81.

[36] Kelsey, J., Chang, S. J., & Perlner, R. (2016). *SHA-3 derived functions: cSHAKE, KMAC, TupleHash, and ParallelHash* (No. NIST Special Publication (SP) 800-185 (Draft)). National Institute of Standards and Technology.

[37] Bernstein, D. J., & Lange, T. (2017). Montgomery curves and the Montgomery ladder. *IACR Cryptology ePrint Archive, 2017*, 293.

[38] Wu, Q. H., Zhang, J. H., & Wang, Y. M. (2003, October). Practical t-out-n oblivious transfer and its applications. In *International Conference on Information and Communications Security* (pp. 226-237). Springer, Berlin, Heidelberg.

[39] Barreto, P., Oliveira, G., & Benits, W. (2018). Supersingular isogeny oblivious transfer. *arXiv preprint arXiv:1805.06589*.

[40] de Saint Guilhem, C., Orsini, E., Petit, C., & Smart, N. P. (2018). Secure Oblivious Transfer from Semi-Commutative Masking. *IACR Cryptology ePrint Archive, 2018*, 648.

Appendix A

Algorithms

In this chapter we will introduce the algorithms used in our implementation. Before we begin, we show the fundamental *structs* and variable *types* used stricktly following the naming system from SIKE library.

- **digit_t**: it represents an unsigned 32- or 64-bit integer. The bitlength depends on the machine the protocol is built on;
- **felm_t**: acronym of field element, this is an array of *digit_t* and its length depends on the machine the protocol is built on;
- **f2elm_t**: array of two *felm_t*. This is used to represent field elements with both real and complex parts;
- **point_proj_t**: it is a struct having two *f2elm_t* representing a point P on an elliptic curve. This struct represents the projective short form of P consisting in only $(X : Z)$ coordinates;
- **point_full_proj_t**: similarly to *point_proj_t*, it represents a point P in a more complete form, namely projective short form, consisting in the triple $(X : Y : Z)$. This struct is rarely used since the Y coordinate can be easily computed from the curve's equation and X, Z coordinates;
- **point_t**: this is the affine correspondence of *point_proj_t* and, in fact, represents the x and y affine coordinates of a generic field element.

Algorithm 1 is a lighter implementation of SIKE's *2_e_iso*. This simply computes a 2^n-isogeny returning the image curve's coefficients $A_{24}^{+'}$, C_{24}' only. The function *4_iso_curve* on line 3 is the same as SIKE's and computes the 4-isogenous *curve*. Once it is put inside the *for* loop (on line 1) it allows to effectively traverse the isogeny path, following a hard-coded traversing strategy, eventually leading to the 2^n-isogeny curve.

Algorithm 1 *get_2n_isogenousCurve_Only*

Static parameter: Public parameter n
Input: The isogeny kernel $K = (X_k : Z_k)$,
Initial curve constants A_{24}^{+}, C_{24}
Output: Final curve constants $A_{24}^{+'}$, C_{24}'

1: **for** $e = n - 2$ **down to** 0 **by** -2 **do**
2: $(X : Z) \leftarrow [2^e]K$
3: $(A_{24}^{+'}, C_{24}') \leftarrow 4\_iso\_curve(X : Z)$
4: **end for**
5: **return** $(A_{24}^{+'}, C_{24}')$

Algorithm 2 is a more general variation of Algorithm 1 having one additional input, a point T, and four additional outputs: the images of public points P, Q, $R = Q - P$ and the image of T.
The function *4_iso_eval* is the same as SIKE's library and computes the 4-isogeny of a given *point*. Once it is put inside the *for* loop (on line 1) it allows to effectively traverse the isogeny path, following a hard-coded traversing strategy, eventually leading to the 2^n-isogenous point. This function is used for each point required in output.

Algorithm 3 is an even more general variation of Algorithm 1. In this case we directly supply three points U, V, W that we want to map on the 2^n isogenous curve. As before, the functions *4_iso_curve* and *4_iso_eval* are the same as SIKE's library and compute the 4-isogeny of a given *curve* and *point* respectively.

Algorithm 2 *get_ 2n_ isogenousCurve*

Static parameter: Public parameters n, P, Q, R

Input: The isogeny kernel $K = (X_k : Z_k)$, a point T
Initial curve constants A_{24}^{+}, C_{24}

Output: Final curve constants $A_{24}^{+'}$, $C_{24}^{'}$
Image points $P^{'}$, $Q^{'}$, $R^{'}$, $T^{'}$

1: **for** $e = n - 2$ **down to** 0 **by** -2 **do**
2: $(X : Z) \leftarrow [2^e]K$
3: $(A_{24}^{+'}, C_{24}^{'}) \leftarrow 4\_iso\_curve(X : Z)$
4: $P' \leftarrow 4\_iso\_eval(P)$
5: $Q' \leftarrow 4\_iso\_eval(Q)$
6: $R' \leftarrow 4\_iso\_eval(R)$
7: $T' \leftarrow 4\_iso\_eval(T)$
8: **end for**
9: **return** $(A_{24}^{+'}, C_{24}^{'})$, $P^{'}$, $Q^{'}$, $R^{'}$, $T^{'}$

Algorithm 3 *get_ 2n_ isogenousCurve_ andPoints*

Static parameter: Public parameter n

Input: The isogeny kernel $K = (X_k : Z_k)$, three points U, V, W
Initial curve constants A_{24}^{+}, C_{24}

Output: Image points $U^{'}$, $V^{'}$, $W^{'}$
Final curve constants $A_{24}^{+'}$, $C_{24}^{'}$

1: **for** $e = n - 2$ **down to** 0 **by** -2 **do**
2: $(X : Z) \leftarrow [2^e]K$
3: $(A_{24}^{+'}, C_{24}^{'}) \leftarrow 4\_iso\_curve(X : Z)$
4: $U' \leftarrow 4\_iso\_eval(U)$
5: $V' \leftarrow 4\_iso\_eval(V)$
6: $W' \leftarrow 4\_iso\_eval(W)$
7: **end for**
8: **return** $(A_{24}^{+'}, C_{24}^{'})$, $U^{'}$, $V^{'}$, $W^{'}$

Algorithm 4 is analogue to Algorithm 1: it is a lighter implementation of SIKE's *3_ e_ iso* allowing us to compute the 3^m-isogenous curve to the input curve. In this algorithm we exploit SIKE's functions *xTPLe* (to perform the $[3^m]$ scalar multiplication) and *3_ iso_ curve* (to compute the 3-isogenous curve to the input curve). Differently from 2^n-isogeny functions, this is the only 3^m-isogeny function we need. Other variations, as seen in Algorithms 2 and 3, can be easily adapted to operate in 3-isogenies with small changes.

Algorithm 4 *get_ 3m_ isogenousCurve_ Only*

Static parameter: Public parameter m
Input: The isogeny kernel $K = (X_k : Z_k)$,
Initial curve constants A_{24}^+, C_{24}
Output: Final curve constants $A_{24}^{+'}$, C_{24}'

1: **for** $e = m - 1$ **down to** 0 **by** -1 **do**
2: $(X : Z) \leftarrow [3^e]K$
3: $(A_{24}^{+'}, C_{24}') \leftarrow 3\_iso\_curve(X : Z)$
4: **end for**
5: **return** $(A_{24}^{+'}, C_{24}')$

We have completely shown all the algorithms to perform 2^n- and 3^m-isogenies and we can now introduce a miscellaneous of algorithms needed in the SIDH-based OT. With Algorithm 5 we find two random factors, namely x and y, aiming to compute a specific linear combination. As a note, the *random* function on lines 3 and 4 is a generic, uniformly distributed, random choosing integer function. Since Vanessa Vitse in her paper [9] did not put an upper bound to this random factors, neither we did.
One may consider to put the upper bound to p since we are working on field extensions over $\mathbb{F}_{p^2}$. Algorithm 6 encrypts and decrypts a string secret. Firstly we need a key derivation function (KDF) which takes a curve's j-invariant as seed and returns an encrypted key. Secondly, and lastly, we use the KDF's key to encrypt our input secret via a simple bitwise XOR operation. For the KDF we chose the *shake256* function from the NIST publication [36] regarding SHA3. The final output is an

Algorithm 5 *linearCombination_Random*

Input: Total number of Alice secrets s
An array of $2s$ points B
An array of s points A for comparisons
Output: An array of $2s$ integers xy

1: **for** $i = 0$ **up to** s **by** $+1$ **do**
2: **do**
3: xy[$2is$] = random();
4: xy[$2is + 1$] = random();
5: **while** (A[$2i$] $\neq xy[2is] * B[2is] + xy[2is + 1] * B[2is + 1]$)
6: **end for**
7: **return** xy

encrypted string.
Thanks to the bitwise XOR, we are able to decrypt a given string by providing the same j-invariant to our function hence making it symmetric and avoiding the issue to implement a different decrypting algorithm.

Algorithm 6 *Enc*

Input: A string *secret*
A curve's j-invariant j
Output: The encrypted secret *out*

1: k = *shake256(secret)*
2: *out* = *secret XOR k*
3: **return** *out*

Other functions used in our implementation provide to implement:

- a *simple point multiplication* by a scalar. This was due to SIKE's library optimised to compute a point multiplication followed by an addition. In order to speed up our code we opted to implement a standard Montgomery Ladder [37];

- the *point addition* in both affine and projective coordinates for a similar reason as above;
- the *point difference* because many operations in SIKE are optimised to work on differential additions, requiring an ad-hoc function to compute the opposite of a point and then add it to another point via a point addition;
- a function to compute the *projective Y coordinate* from a point $P = (X : Z)$ on a Montgomery curve with coefficient A. The reason behind this implementation comes from compatibility issues in the many formats used in SIKE when we have a point represented in projective short form $(X : Z)$ but we need the full $(X : Y : Z)$ coordinates point. The projective Y coordinates can be easily computed from the curve equation once the coefficient A and a point P are known;
- a function to convert an affine point $P = (x, y)$ to its projective short form $P = (X : Z)$ simply by assigning the values $X = x$ and $Z = 1$;
- a function analogous to the above but used to convert an affine point $P = (x, y)$ to its projective full form $P = (X : Y : Z)$. This operation was done by assigning the values $X = x$, $Y = y$ and $Z = 1$;
- a function to convert a projective full form of a point $P = (X : Y : Z)$ to its short form simply by discarding the Y coordinate;
- a function to generate a 2^n-order basis by exploiting SIKE's *get_2_torsion_entangled_basis_compression* function and by adding a conversion function to obtain the basis in projective short and full form
- a last function to compute Weil pairing of order 2^n by exploiting SIKE's *Tate_pairings_2_torsion* function and the optimisation shown in Section 2.3.2. By joining together these two ingredients we were able to obtain only two output values instead of 2^n hence reducing enormously space and time complexities.

Unused algorithms

Even though the following algorithms are no longer used in current implementation, we considered important to describe them since they could be helpful in future developments.

Algorithm 7 generates a pair of points (T_1, T_2) having order 2^n. It also performs a cross-multiplication and the other checks, described in Section 2.3.2, to confirm the points have the desired order.

Some inputs are not explicitly used but they are required in mandatory sub-functions: input A is used on lines 3 and 5 to compute the SIKE's *LADDER3PT*; inputs A_{24}^{+} and C_{24}, are used on lines 11 and 12 in SIKE's function *xDBLe*.

Algorithm 7 *generate_2n_torsionPoints*

Input: Two linearly independent points P, Q of order 2^n,
a point representing their difference $Q - P$,
Montgomery curve constants A_{24}^{+}, C_{24}, A

Output: Two points T_1, T_2 of order 2^n

1: **do**
2: $k_1 \leftarrow$ *random chosen in* $\{0, \ldots, 2^{n-1} - 1\}$
3: $T_1 \leftarrow P + [2k_1]Q$
4: $k_2 \leftarrow$ *random chosen in* $\{0, \ldots, 2^{n-1} - 1\}$
5: $T_2 \leftarrow P + [2k_2]Q$

6: $X_1 \leftarrow X$*-coordinate of* $[2^{n-2}]T_1$
7: $X_2 \leftarrow X$*-coordinate of* $[2^{n-2}]T_2$
8: $Z_1 \leftarrow Z$*-coordinate of* $[2^{n-2}]T_1$
9: $Z_2 \leftarrow Z$*-coordinate of* $[2^{n-2}]T_2$

10: $linearDependent \leftarrow X_1Z_2 = X_2Z_1$
11: $T1\_notFullOrder \leftarrow ([2^{n-1}]T_1 \neq \mathcal{O})\, and\, ([2^n]T_1 = \mathcal{O})$
12: $T2\_notFullOrder \leftarrow ([2^{n-1}]T_2 \neq \mathcal{O})\, and\, ([2^n]T_2 = \mathcal{O})$
13: **while** (linearDependent or T1_notFullOrder or T2_notFullOrder)
14: **return** T_1, T_2

Lastly, a function enables to convert a point $M = (X : Y : Z)$ on a Montgomery curve with coefficient A, to a point W on a Weierstrass elliptic curve. The algorithm simply computes and returns $W = (X/A : Y : 1)$. This had not been used since we could just convert single points from projective to affine form without the need to convert the entire curve too.