Implementation of the isogeny-based key-exchange protocol CRS

Table of content

1 Introduction

Post-quantum cryptography (PQC) is a young field anticipating the availability of large-scale quantum computers in the context of digital security. Seen by many as the next logical step to supersede classical computers, the quantum paradigm offers algorithms that could endanger classical public key cryptography. Possibly the most famous one, Shor's algorithm [] brings integer factorization down to a polynomial problem. This has obvious dire consequences for classical RSA and Diffie-Hellman (DH) protocols. The performances of current quantum computers are still far from reaching usability in attack scenarios. But cautiousness calls for quantum-resistant algorithms that could secure classical computers. In that regard, the NIST (National Institute of Standards of Technology) has organized competitions to foster developpment in PQC.

Different usages require different cryptosystems and as such there are multiple classes of post-quantum protocols investigated. Of interest in this paper is public-key cryptography (PKC) also known as asymmetric cryptography. The benefit of PKC stems from the fact that no pre-sharing of information is needed to establish a secure channel, hence making it very convinient. To achieve this, a Diffie-Hellman key-exchange protocol is usually used. In our case we use elliptic curve isogenies and associated hard homogeneous spaces (HHS) to build such a protocol.

More precisely, we investigate the Couveignes-Rostovtsev-Stolbunov protocol (CRS). This protocol works with isogenies bewteen ordinary curves with fixed complex multiplication over a finite field of prime order $\sim 2^{256}$. Key-exchanges using this scheme are comparatively slower than their supersingular counterpart such as SIDH protocols. However, CRS heuristically gives less room for potential attacks than SIDH. This is due to the fact that supersingular endomorphism rings embed as orders in quaternion algebras whereas ordinary ones embed in imaginary quadratic fields. As such, more structure is available for exploitation in an attack of SIDH protocols. This kind of structural attack has been demonstrated against the NTRU protocol and led to Prime-NTRU. Another advantage of CRS is that it is truely symmetric compared to SIDH where parties have to agree on specific primes to use.

The main focus of this project was to implement as efficiently as possible the CRS protocol following the 2018 article of Luca De Feo, Jean Kieffer and Benjamin Smith []. Recent publications on the subject of isogeny computation allow for very efficient walking algorithm in the isogeny graphs. On the one hand, *radical isogenies* [] are used to take k steps in the l-isogeny graph in time complexity $\mathcal{O}(k)$ for primes 3, 5 and 7. On the other hand, the $\sqrt{}$-*Velu* algorithm [] is used to take walk in the remaining graphs with time complexity $\mathcal{O}(k\sqrt{l})$. The reason this subject was choosen specifically is after a previous unoptimized JULIA implementation of these techniques. It gave promising results (around 35% faster than the 2018 implementation) but was neither optimized nor using the expected quadratic-twist symmetry. The second issue was completely resolved by our C implementation while drastic optimizations were found.

This paper is divided in three sections. The first one mainly accounts for the minimal mathematical background surrounding isogeny-graphs and complex multiplication. An exposition on the CRS protocol is also given. The second part tackles the architecture of the program. The complete project is around 4000 lines long. Its source code is available at []

along with a documentation. Finally, the third part reports prime-by-prime timing results, optimized bound for the secret keys as well as some inner data observed at runtime.

2 Theoretical background

In this section we introduce the necessary mathematical material surrounding the CRS protocol. It is assumed the reader has some familiarities with elliptic curves and their arithmetic. Still, we recall the fundamentals in order to underline some important aspect of the algorithms.

2.1 Elliptic curves and models

Recall the standard definition of an elliptic curve over k:

Definition. An elliptic curve E over a field k is a smooth curve given by a (dehomogenized) polynomial $F \in k[x, y]$ of the form

$$F = y^2 + a_1xy + a_3y - (x^3 + a_2x^2 + a_4x + a_6).$$

This presentation of the curve is usually called the *Weierstrass* form. It is said to be in *short Weierstrass* form if the coefficients a_1, a_2 and a_3 are null. The following proposition allow one to only consider short Weierstrass equations most of the time.

Proposition. An elliptic curve E over a field k of caracteristic different from 2 and 3 is isomorphic to an elliptic curve in short Weierstrass form.

Proof. See Silverman III.1. [] □

The (short-)Weierstrass models admit explicit formulas that reflect the group law of the elliptic curve. It turns out that other models have better, faster properties when it comes to group operations. The following *Montgomery* model has one of the fastest doubling operation of all. It also has a fast scalar multiplication which we talk about in the next section.

Definition. A *Montgomery curve* or an elliptic curve in *Montgomery* form over a field k is an elliptic curve $E_{A,B}$ given by a polynomial $F \in k[x, y]$ of the form

$$F = By^2 - x(x^2 + Ax + 1)$$

with $A, B \in k$ satisfying $b \neq 0$ and $A^2 \neq 4$.

All Mongomery curves admit a short Weierstrass model via the variable change $x := x/B$ and $y := y/B$. The following proposition shows which short Weierstrass curves admit a Montgomery model.

Proposition. The elliptic curve $y^2 = x^3 + ax + b$ over a field k admits a Montgomery model if

- The polynomial $X^3 + aX + b$ has a root w in k

- $3w^2 + a$ is a quadratic residue in k.

Proof. Under the above conditions, set $u = (3w^2 + a)^{-\frac{1}{2}}$. The (admissible) variable change $x := u(x - w)$, $y := uy$ show that the curve has Montgomery model $E_{A,B}$ with $A = 3wu$ and $B = u$. $\square$

Associated to an elliptic curve E is the fundamental quantity called *j-invariant.* For a Mongomery curve $E_{A,B}$ over a field k of caracteristic different from 2 and 3 one has

$$j = \frac{256(A^3 - 3)^3}{A^2 - 4}.$$

Elliptic curves can be shown to be isomorphic if and only if they have the same j-invariant. The crucial thing to remember is that this classification only happens over the algebraic closure $\bar{k}$ of the base field k. Curves can be isomorphic over $\bar{k}$ while being non-isomorphic over k or a finite extension of k. In fact, the B parameter in the definition of a Mongomery curve account for two isomorphism classes: take $A, B, B' \in k$ and consider the curve $E_{A,B}$ and $E_{A,B'}$. One easily show that they are isomorphic at best over $k(\sqrt{B/B'})$. Hence they are isomorphic over k if and only if B/B' is a quadratic residue. We call $E_{A,B'}$ a *quadratic twist* of $E_{A,B}$.

The next and last model of elliptic curve we use is called the *Tate normal* form. It will be used when dealing with radical isogenies.

Definition. An Elliptic curve in *Tate normal* form over a field k is an elliptic curve E given by a polynomial $F \in k[x, y]$ of the form

$$F = y^2 + (1 - c)xy - by - x^2(x - b)$$

on which the point $(0, 0)$ has order at least 4. We also say E is in Tate normal form when given by

$$F = y^2 + (1 - c)xy - by - x^3$$

where $(0, 0)$ is of order 3.

The following lemma and its proof show how to transform a Mongomery curve along with a point P of order $N \geq 4$ into Tate normal form. We demonstrate the lemma with a general Weierstrass curve but only the Montgomery case with $B = 1$ interests us in the implementation. It is only a matter of setting some a-coefficients to 0 in the formulas.

Lemma 2.1.1. Let E be an elliptic curve over a field k and let $P = (x, y) \in E$ be a point of order $N \geq 3$. There exist a Tate normal model for E such that P is sent to $(0, 0)$..

Proof. We consider E in general Weierstrass form

$$Y^2 + a_1XY + a_3Y = X^3 + a_2X^2 + a_4X + a_6.$$

A translation from P to $(0, 0)$ allows us to remove a_6 and write the curve as

$$Y^2 + a_1XY + b_3Y = X^3 + b_2X^2 + b_4X.$$

where $b_2 = 3x + a_1$, $b_3 = 2y + a_1x + a_3$ and $b_4 = 3x^2 + 2a_1x + a_4$. The coefficient b_3 is non-zero as P does not have order 2. This can be read on the duplication formula. Let us remove the b_4 coefficient by the (admissible) change of variable $Y := Y + b_4/b_3X$. We get the representation

$$Y^2 + c_1XY + b_3Y = X^3 + c_2X^2.$$

where $c_1 = 2b_4/b_3 + a_1$ and $c_2 = b_2 - a_1b_4/b_3$.

Now introduce two free scaling variables $\alpha, \beta \in k$. The scaling $X = \alpha X$, $Y = \beta Y$ gives a model that is Tate normal if

$$\alpha^3 = \beta^2$$
$$\beta b_3 = c_2\alpha^2.$$

Following the standard negation formula and $N = 3$ being equivalent to $2P = -P$ we see that $c_2 = 0$ is also equivalent to $N = 3$. Assuming that $N \geq 4$, one find $\alpha = (b_3/c_2)^2$ and $\alpha = c_2/b_3$. Thus setting $b = -c_2(b_3/c_2)^2$ and $c = 1 - c_1(c_2/b_3)^2$ gives the Tate normal form. If $N = 3$, the curve is already in Tate normal form. □

Notice however that in the case $N = 3$, the coefficient b_3 should be computed to get b and c. This in turns involves knowing the value of y. As we will see with x-only arithmetic, over Mongomery curves it is better to use our free system in α and β. Setting $\alpha = b_3^3$ and $\beta = b_3^2$ we find a presentation which only involves y^2. In the end we find $b = -1/b_3^2$ and $c = 1 - 2b_4/b_3^2$. This will avoid us extracting a square root of $x^3 + ax^2 + 1$ to get a y value. This used to be a time-consuming task in the previous implementation that slowed down the computations. Following these steps in reverse order transforms a Tate normal form into a Montgomery curve.

2.2 Isogenies

We fix once and for all a prime number p. Unless indicated otherwise we assume throughout the section that curves are defined over $\mathbb{F}_q$, a field of caracteristic p with q elements. Recall the definition of an isogeny, which is essentialy an homomorphism between varieties preserving the group structure.

Definition. Let E and E' be two elliptic curves defined over $\mathbb{F}_p$. An isogeny $\phi : E \to E'$ is a non-trivial algebraic map sending O_E to $O_{E'}$.

The isogenies from E to E, together with the trivial map form a ring called the endomorphism ring of E and noted $\mathrm{End}(E)$. For a positive integer m, the *multiplication-by-m* map over E is an endomorphism sending a point P to the sum of m copies of P. This map will be denoted by $[m]$.

Example. Of interest in the CRS protocol is the *Frobenius* endomorphism:

$$\pi : (x, y) \in E \mapsto (x^q, y^q) \in E.$$

This endomorphism satisfies the relation $\pi^2 - t\pi + q = 0$, where t is the trace of E. This trace is linked to the cardinality of $E(\mathbb{F}_q)$ via $\#E(\mathbb{F}_q) = q + 1 - t$ and can be computed effectively using the PARI/GP library. When the trace is divisible by p, we say that the curve is *supersingular*. Otherwise it is called *ordinary*. The CRS protocol only uses ordinary curves.

The *degree* of an isogeny is its degree as an algebraic map. For instance, the multiplication-by-m map is of degree m^2 when $m \wedge p = 1$. The degree of the frobenius endomorphism is q. We say an isogeny of degree l is an *l-isogeny*.

For the purpose of CRS, we mainly consider *separable* isogenies between elliptic curves. These include isogenies of degree coprime to p and are in one-to-one correspondance with their kernels up to isomorphism. This means that for any finite subgroup G of E of order l, there is a unique elliptic curve (up to isomorphism) denoted E/G and an l-isogeny

$$E \to E/G$$

whose kernel is G. Recall finally that for each l-isogeny $\phi : E \to E'$, there is a unique l-isogeny $\hat{\phi} : E' \to E$ called the *dual isogeny* such that

$$\phi \circ \hat{\phi} = [l]_{E'} \text{ and } \hat{\phi} \circ \phi = [l]_E.$$

The important thing to remember from this is that being l-isogenous is a *symmetric relation* and being isogenous is an *equivalence relation*. Isogenous curves share some common properties. For instance, they both have the same number of points over $\mathbb{F}_q$ and thus also have the same trace. This brings us to the core component of the CRS protocol.

2.3 Isogeny graphs

For an integer l, the *l-isogeny graph* over a field k is the graph whose vertices are elliptic curve isomorphism classes (over k) and whose edges represent l-isogenies. Recall that for l coprime to p, the l-torsion group of E denoted $E[l]$ is isomorphic to $\mathbb{Z}/l\mathbb{Z} \times \mathbb{Z}/l\mathbb{Z}$. This way, if l is prime and different from p, $E[l]$ contains $l+1$ distinct cyclic subgroups of order l. This in turns mean that there are exactly $l+1$ (separable) l-isogenies whose domain is E. Thus, a connected component in the l-isogeny graph over $\overline{\mathbb{F}_q}$ is an $l+1$-regular graph.

As mentionned before, isomorphism classes are determined by j-invariant over $\overline{\mathbb{F}_q}$. However in CRS we will use isogeny graphs over $\mathbb{F}_q$ which gives more complex structures. Let us analyze the isogeny graph in this case, starting by computing the incidence degree of the graph's vertices. We consider a prime l different from p.

It can be shown that an isogeny is defined over $\mathbb{F}_q$ if and only if the frobenius π stabilizes its kernel. We are only interested in cyclic isogenies, those with cyclic kernels. As such can see the action of π on the kernel as a scalar multiplication due to the fact that π commutes with multiplication-by-m maps. Notice further that $E[l] = \mathbb{Z}/l\mathbb{Z} \times \mathbb{Z}/l\mathbb{Z}$ is a 2-dimentional $\mathbb{F}_l$-vector space. The problem is hence reduced to understanding the stable subspaces of π as an endomorphism of $E[l]$. This calls for a study of the eigenvalues of π, whose characteristic polynomial is $X^2 - tX + q \bmod l$. Counting the stable subgroups, we get:

- If π has no eigenvalues, then E has no l-isogenies.

- If π has one eigenvalue of geometric multiplicity one, there is exactly one l-isogeny from E.

- If π has one eigenvalue of geometric multiplicity two, there are exactly $l+1$ l-isogenies from E.
- If π has two distinct eigenvalues, there are exactly two l-isogenies from E.

Only that last case is of interest for use. We call primes l that satisfy this condition *Elkies* primes. Note that since isogenous curves have same trace, their frobenius has same characteristic polynomial. As such, isogenous curves have the same Elkies primes. This implies that for l an Elkies prime connected components of the l-isogeny graph over $\mathbb{F}_q$ are 2-cycles.

2.4 The structure of End(E)

In this part we show that endomorphism rings of ordinary curves are orders in imaginary quadratic fields. Let E be an ordinary elliptic curve over $\mathbb{F}_q$ with Frobenius element π. Remember that the ring $\mathbb{Z}$ embeds in $\mathrm{End}(E)$ via multiplication-by-m elements. As such we will identify elements $[m]$ and m. Let us start with the definition of an order in an algebra;

Definition. Let A be a finite dimentional algebra over $\mathbb{Q}$. A subring $\mathcal{O}$ of A is an *order* if:

- $\mathcal{O} \otimes \mathbb{Q} = A$.
- $\mathcal{O}$ is a $\mathbb{Z}$-lattice in A.

This essentialy means that $\mathcal{O}$ is a free abelian group containing a $\mathbb{Q}$-basis of A. The following theorem gives the remarkably simple classification of endomorphism rings of elliptic curves. The heavy lifting of the proof is done using Tate-modules theory. See [theorem Silverman III.9.3].

Theorem 1. Let E be an elliptic curve over a field k. Then $\mathrm{End}(E)$ is either $\mathbb{Z}$, an order in an imaginary quadratic field or an order in a quaternion algebra.

We now show that our ordinary elliptic curve E falls in the second category. For that we need some generalities about $\mathrm{End}(E)$.

Proposition. The ring $\mathrm{End}(E)$ is torsion free of characteristic 0 and has no non-trivial zero divisors.

Proof. Recall that for $m \neq 0$, the multiplication-by-m map $[m]$ is non-constant. Take $\phi \in \mathrm{End}(E)$ and m such that

$$[m] \circ \phi = [0].$$

The multiplicativity of the degree map gives

$$\deg([m])\deg(\phi) = 0.$$

Thus either $m = [0]$ or $\phi = [0]$ and $\mathrm{End}(E)$ is torsion free of caracteristic 0. Now suppose for $\phi, \psi \in \mathrm{End}(E)$ that $\phi \circ \psi = [0]$. Then taking the degree,

$$\deg(\phi)\deg(\psi) = 0.$$

Thus either $\phi = [0]$ or $\psi = [0]$ and $\mathrm{End}(E)$ has no non-trivial zero divisors. □

The following lemma already shows that $\mathbb{Z} \subsetneq \mathrm{End}(E)$.

Lemma 2.4.1. The Frobenius endomorphism $\pi \in \mathrm{End}(E)$ is not in $\mathbb{Z}$.

Proof. Assume the converse so that $\pi = [n]$ for some integer n. Then taking the degree gives

$$q = \deg(\pi) = \deg([n]) = n^2.$$

This implies that $d := [\mathbb{F}_q : \mathbb{F}_p]$ is even and that $n = \pm p^{d/2}$. Hence using the roots-coefficients relations on the characteristic polynomial of π we get $t = 0 \bmod p$, which is absurd as E is ordinary. □

Let $\mathrm{End}^0(E) := \mathrm{End}(E) \otimes \mathbb{Q}$. The next lemma will be used to show that every isogeny of $\mathrm{End}^0(E)$ commutes with π.

Lemma 2.4.2. Let $n \geq 1$. There exist $a, b \in \mathbb{Z}$ satisfying $a \neq 0 \bmod p$, $b = 0 \bmod p$ and such that

$$\pi^n = a\pi + b.$$

Proof. We use induction on n. The case for $n = 1$ is verified for $a = 1$ and $b = 0$. Assume the lemma holds for every $1 \leq m \leq n$. Then using the fact that $\pi^2 - t\pi + q = 0$, we have for some $a, b \in \mathbb{Z}$ satisfying the hypothesis

$$\pi^{n+1} = \pi(a\pi + b) = b\pi + a(t\pi - q).$$

Rearranging the equation one gets

$$\pi^{n+1} = c\pi + d$$

where $c = at + b$ and $d = -aq$. These verify the conditions since $c = at \neq 0 \bmod p$, E being ordinary, and $d = 0 \bmod p$. □

This lemma shows that $\pi^n \notin \mathbb{Q}$ for any integer $n \geq 1$. Now any element of $\mathrm{End}^0(E)$ can be written as $m\phi$ with $m \in \mathbb{Q}$ and $\phi \in \mathrm{End}(E)$. Being a rational map, the isogeny ϕ is defined over a finite extension of $\mathbb{F}_q$, say $\mathbb{F}_{q^d}$. Writting ϕ in its reduced form $(r(x), ys(x))$ we have

$$\phi\pi^d = (r(x^{q^d}), y^{q^d} s(x^{q^d})) = (r(x)^{q^d}, (ys(x))^{q^d}) = \pi^d\phi.$$

Hence every element of $\mathrm{End}^0(E)$ commutes with π^d for some $d \geq 1$. And this last element is in fact in $\mathbb{Q}(\pi)$ according to the previous lemma. The next lemma shows that this is enough to prove the inclusion $\mathrm{End}^0(E) \subset \mathbb{Q}(\pi)$, hence the equality.

Lemma 2.4.3. Let $\alpha, \beta \in \mathrm{End}^0(E)$. If these elements commute and $\alpha \notin \mathbb{Q}$ then $\beta \in \mathbb{Q}(\alpha)$.

Proof. Introduce the $\mathbb{Q}$-linear trace map $T : \alpha \in \mathrm{End}^0(E) \mapsto \alpha + \hat{\alpha} \in \mathbb{Q}$. Its codomain being justified by equalities

$$T\alpha = 1 - \alpha\hat{\alpha} - (\alpha - 1)\widehat{(\alpha - 1)} = 1 - [\deg(\alpha)] - [\deg(\alpha - 1)] \in \mathbb{Q}.$$

We may replace α by $\alpha - \frac{1}{2}T\alpha$ to have $T\alpha = 0$. Since $T\alpha = 0$ and $\mathrm{End}^0(E)$ is without non-trivial zero divisors, we get $\alpha^2 \in \mathbb{Q}^*$. Replacing β by $\beta - \frac{1}{2}T\beta - \frac{1}{2\alpha^2}T(\alpha\beta)\alpha$, we can assume further that

$$T\beta = T\alpha\beta = 0.$$

The combined equations $T\alpha = T\beta = T(\alpha\beta) = 0$ now give $\alpha\beta = -\beta\alpha$. All the substitutions were done while keeping the commutativity hypothesis thus we have $2\alpha\beta = 0$. As $\mathrm{End}^0(E)$ is without non-trivial zero divisors, we find $\beta = 0 \in \mathbb{Q}(\alpha)$ since $\alpha \notin \mathbb{Q}$. $\square$

In the end we have proved that $\mathrm{End}(E)\otimes\mathbb{Q} = \mathbb{Q}(\pi)$. The field $\mathbb{Q}(\pi)$ is quadratic imaginary as the Frobenius' characteristic polynomial is of degree 2. Another way to think of this field is to see it as $\mathbb{Q}(\sqrt{\Delta_\pi})$ where Δ_π is the discriminant of the characteristic polynomial of the Frobenius. This concludes the proof that ordinary curves have endomorphism rings isomorphic to orders in imaginary quadratic fields.

2.5 The group action

In this section we focus on defining the group action that will be used to exchange keys Diffie-Hellman style. We keep our notations. The previous section showed that $\mathrm{End}(E)$ is an order in $\mathbb{Q}(\Delta_\pi)$. Using isogeny-volcanos theory, one can show that for Elkies primes l, two l-isogenous curves have the same endomorphism ring. As such we label $\mathcal{O}$ the endomorphism ring of E and we say that a curve with endomorphism ring $\mathcal{O}$ has *complex multiplication* (CM) by $\mathcal{O}$.

Recall that by a *fractional ideal* $\mathfrak{a}$ we mean a $\mathcal{O}$-submodule of $\mathbb{Q}(\sqrt{\Delta_\pi})$. This fractional ideal is said to be *invertible* if there exists another fractional ideal $\mathfrak{b}$ such that $\mathfrak{a}\mathfrak{b} = \mathcal{O}$. For such an invertible ideal $\mathfrak{a}$, we define the $\mathfrak{a}$-torsion subgroup of E as

$$E[\mathfrak{a}] = \{P \in E \mid \sigma P = O \text{ for all } \sigma \in \mathfrak{a}\}.$$

This defines a subgroup of E and hence the kernel of a unique isogeny denoted $\phi_{\mathfrak{a}}$. Its codomain will be denoted as $\mathfrak{a} \cdot E$ to underline a group action.

Let $\mathrm{Ell}_q(\mathcal{O})$ be the set of elliptic curves with complex multiplication by $\mathcal{O}$ up to isomorphism. One can extend the $\mathfrak{a}$-torsion construction into an action of the fractional ideals of $\mathcal{O}$ on $\mathrm{Ell}_q(\mathcal{O})$. Principal ideals then act trivially under this action and thus induce an action of the class group of $\mathcal{O}$, denoted $\mathcal{C}(\mathcal{O})$, on $\mathrm{Ell}_q(\mathcal{O})$. The *main theorem of complex multiplication* states that this action is simply transitive, hence a bijection

$$\text{Ideal class of } \mathfrak{a} \text{ in } \mathcal{C}(\mathcal{O}) \mapsto \text{Isomorphism class of } \mathfrak{a} \cdot E \text{ in } \mathrm{Ell}_q(\mathcal{O}).$$

This bijection implies that the space on the left has $h(\mathcal{O})$ elements, where $h(\mathcal{O})$ is the class number of $\mathcal{O}$. Over $\mathbb{F}_q$ this space decomposes into two components corresponding to curves and their quadratic twist. Under this bijection, ideal norms correspond to isogeny degrees.

Now to walk on this $\mathrm{Ell}_q(\mathcal{O})$ space, recall that for an Elkies prime l, the Frobenius restricted to $E[l]$ has two distinct eigenvalues λ and μ. Let $\mathfrak{a} = (\pi - \lambda, l)$ and $\hat{\mathfrak{a}} = (\pi - \mu, l)$ be two ideals in $\mathcal{C}(\mathcal{O})$. The space $E[\mathfrak{a}]$ is the eigenspace of π which defines an l-isogeny $\phi_{\mathfrak{a}}$. It turns out that $\mathfrak{a}\hat{\mathfrak{a}} = \hat{\mathfrak{a}}\mathfrak{a} = (l)$ in $\mathcal{C}(\mathcal{O})$. As such, these fractional ideals are inverse of each other and the corresponding isogenies are dual of one another. These two eigenvalues hence define the two direction on the l-isogeny cycle independently of the curve E. We now have a way to take steps in the isogeny graph.

2.6 The key-exchange protocol

We describe the CRS key-exchange protocol using the group action of the previous section. As a set of vertices, we use $X = \mathrm{Ell}_q(\mathcal{O})$ for some order $\mathcal{O}$ and q. Let $G = \mathcal{C}(\mathcal{O})$ and S be a set of ideal of G with small norms. Consider the directed graph $\mathcal{G}$ whose vertices are the points of X. We connect two vertices (classes of curves) $[E]$ and $[E']$ if and only if there is an element of S such that $s \cdot E = E'$. If S is sufficiently large, i.e. it generates the group $\mathcal{C}(\mathcal{O})$, then the graph will be connected.

In a nutshell, the keyspace is a very large graph composed of cycles whose vertices are elliptic curves. To walk on this graph means to take a certain path made of isogenies of various degrees on this graph. This is all backed up by the main CM theorem states that the operation of taking steps is commutative and bijective. A secret key can then be a walk on the graph starting from a fixed curve and the corresponding public key the destination curve. The problem of creating a connecting isogeny between the starting curve and the destination one is believed to be quantum-hard, hence the security.

To set up the protocol, a good order $\mathcal{O}$ is needed. For quantum security using this protocol, a keyspace of size at least 256 bits is required. Estimating the class number is a very hard problem but on average

$$h(\mathcal{O}) \approx 0.461\sqrt{|\Delta_\pi|}.$$

Since the keyspace is of size $\#X = \#\mathrm{Ell}_q(\mathcal{O}) = h(\mathcal{O})$ and $\Delta_\pi \sim q$, we have to look for a curve over $\mathbb{F}_q$ with q around 512 bits. This is complicated by the fact that we need a generating set S of ideals with small norm. Thousands of computer hours have been spent generating some usable protocol parameters: sufficiently many primes of small norm and a large enough keyspace. To actually take a step $\mathfrak{a}$ starting from a vertex $[E]$ in this graph we explicitly generate the kernel $E[\mathfrak{a}]$ and compute the resulting isogeny.

The kernels correspond to random points of order l. Generating these is the most time-consuming part of our implementation and will be described in the next section. To effectively compute isogenies from these points two methodes are used. The fastest, using *radical isogenies*, transform the starting curve in Montgomery form into a Tate normal curve. Then one can iterate very efficiently over small rational expessions and take k steps in the graph in one go. This is only possible for primes 3, 5 and 7. The second one, $\sqrt{}$*Velu* cleverly uses some structure on the kernel polynomials and fast multi-evaluation to take one step at a time.

3 Program architecture

3.1 Global architecture of the code

Here we introduce the relations between the different files and folders of the code. At the top level there are three folders: `example/`, `optimization/` and `src/`

Folder `example/` holds a working instance of a key-exchange using the algorithm. A *bash* script is provided to compile the example. This script accepts two flags:

- -DVERBOSE which will print some live information on the key-exchange; and
- -DTIMING which will output timings for the isogeny steps in *json* format.

This second flag is used to perform optimization on the l-primes bounds.

The `optimization/` folder contains two pyhon scripts. The `optimize.py` script loads a *timings.json* file located in subfolder `files` and created using the -DTIMING flag. It uses the *GEKKO* optimizer to find the best combination of bounds on the steps for each l-primes. This optimizes the *timing* of the global algorithm while keeping the *keyspace* size above 256 bits. The results can then be found in `files/`. A plotting script is also available to graph the speed of steps against l. See the third section on results for more details.

Last and most importantly, the `src/` folder contains all the necessary functions and structures implementing the CRS protocol. It is once again divided in four subfolders.

In `EllipticCurves/` one will find all the functions and structures related to curve manipulation. The `models.h` file holds the type definition of elliptic curves in different forms and their corresponding points. All functions related to memory management are located in the `memory.c/h` files. The exact structures holding the elliptic curves are discussed in more details in the next section. The `arithmetic.c/h` files contain all the mathematical functions directly related to the curves. For instance the *Montgomery Ladder*, the extraction of random rational torsion points and the model transformation functions are located there. The `auxiliary.c/h` files only serve as shortcuts to unavailable FLINT functions. The `pretty_print.c/h` files define convenient debugging functions mainly used to check on the key-exchange.

Folder `Isogeny/` holds the two algorithms used to compute isogenies between elliptic curves. The `radical.c/h` files implement radical isogenies for primes 3, 5 and 7. Files `velu.c/h` account for the $\sqrt{}$-Velu algorithm described further in the paper. Finally `walk.c/h` holds the two handle functions used to walk in the isogeny graph using the aforementioned algorithms.

The `Polynomial/` folder contains the binary-tree based multi-evaluation algorithm in `multieval.c/h`. The binary-tree structure and functions themselves are defined in file `binary_trees.c/h`. Alongside with `roots.c/h` hosting a bruteforce square root extraction algorithm. This was necessary as no root-extraction methods are available for large fields in the FLINT library.

Finally the `Exchange/` folder hosts everything related to the global structure of the CRS protocol. The global environment setup is handled in `setup.c/h` while secret key generation happens in `keygen.c/h`. Applying a key to an elliptic curve is done through the single function located in the `dh.c` file. Auxiliary files `info.c/h` are there to print live information during the exchange and to get timings for optimization.

3.2 C structures and memory management

This section will first introduce the different custom types used in the code to handle context and mathematical objects. We have made the choice to clearly represent in C the objects we use. To that extent we defined a number of structures representing elliptic curves, their points and other less obvious data in memory. We followed the GMP/FLINT standard nomenclature identifying types with a "_t".

The file `EllipticCurves/models.h` defines the three types of elliptic curve models we use. These *structs* named respectively `SW_curve_t`, `MG_curve_t` and `TN_curve_t` stand for Short Weierstrass, Montgomery and Tate normal curves. Inside these *struct* lies a `const fq_ctx_t *` pointer to the field of definition of the curve. A pointer is used in order to not re-instantiate the whole field structure for every curves. The multiple `fq_t` parameters respectively named a, b; A, B; b, c are the respective models' parameters. The `fmpz_t` parameter l in the definition of type `TN_curve_t` is used to keep track of the order of the point $(0, 0)$ on the curve. These all strictly follow the definitions found in the first part of this paper.

The *structs* named respectively `SW_point_t` and `MG_point_t` represent points on the respective curves. These *structs* contain the coordinate of a point on a curve as `fq_t` elements. Said curve is linked to the point via a pointer of the respective type.

In file `Polynomials/binary_tree.h` we define a binary-tree structure used in the fast-multipoint polynomial evaluation. The *struct* type `fq_poly_btree_t` holds a `const fq_ctx_t *` pointer to keep track of the field of definition of its elements. It also holds a pointer to the head (or root) cell of the tree. This decision to distinguish the whole tree from its cells stems from fear of memory management issues.

The cells are represented by the `fq_poly_bcell_t` type. This structure also holds a pointer to the field k we are working with. Most importantly, it holds a `fq_poly_t` which is is a polynomial over k. The remaining fields are `fq_poly_bcell_t` pointers to the left and right child of this cell. Keyword *struct fq_poly_bcell_t* is used to recursively define these childs in the type definition. Both should be initialized to *NULL* to represent a leaf of the tree.

File `Exchange/setup.h` holds the `lprime_t` structure representing l-primes. A specific structure has been assigned for this task because of the numerous parameters surrounding l-primes. These consists only of an `fmpz_t` representing l and multiple *unsigned int*. Integers *lbound* and *hbound* account for the bounds on the number of steps allowed for the walk on the l-isogeny graph. It turns out that both upper and lower bounds are equal since we managed to avoid root extraction on the backward direction by using the quadratic twist. This structure also holds the extension degree r corresponding to the Frobenius eigenvalue's order mod l. Finally it also contains a uint serving as a boolean to allow or disallow backward walking. This is necessary as some l-primes have one eigenvalue of low order but another one of unusably high order.

In this same file, the global configuration type `cfg_t` is defined. This type holds the public starting curve as an `MG_curve_t`. Furthermore, it contains the number of l-primes to use in the protocol as well as the `lprime_t` array holding them. The base field and its extensions up to degree 9 are held in an `fq_ctx_t` array to avoid re-instantiation. Finally an unsigned integer defined as a seed for the random key generation. This last step allow for debugging but also for timing tests.

In file Exchange/keygen.h one can find the definition of type key__t representing a secret CRS key. This twisted nomenclature avoids the already defined key_t type while remaining as consistent as possible. A key holds the number of primes used in the protocol, the list of lprime_t used and an fmpz_t list holding the bounds on the steps to take for each prime. Note that these lists should all be ordered from the smallest working extension degree to the largest. The order of the l-primes themselves within the same extension degree does not matter. This is done to avoid downgrading the base field. This way we only upgrade (embed) the base field once per degree (refer to the MG_curve_update_field function in Exchange/dh.h for details).

Memory management was the first concern of this project and as such was dealt with very rapidly. Since we are using GMP through FLINT, we decided to use the same *declare, init, set, clear* paradigme. That way almost all our functions start by a declaration of the variables used. Followed by an initialization, often using an fq_ctx_t field. Then the data is set and used accordingly then finally the memory is cleard. This is respected by all of our custom types. The corresponding management functions are named following the nomenclature *_init, *_set and *_clear where * is the type in question. Some additional practical functions like MG_curve_set_si or fq_poly_bcell_set_right allow for shortcuts. Still they were inspired by the FLINT ecosystem.

Most of our custom initialization and clearing functions are just wrappers for FLINT functions. This offers a safer and more comfortable handling of the many objects in the code. However there are multiple instances where manual memory allocation was favorable. This is the case for instance with binary trees, l-primes, keys and config types. The main reason was for allocating memory for arrays of custom data and/or not having to pre-declare objects. Sometimes both approaches were used. This can be seen in Exchange/keygen.c with key_init initializing a declared key and key_init_ returning a pointer to an initalized key.

3.3 Montgomery curve arithmetic

We chose to implement the Montgomery curve arithmetic as described by Craig Costello and Benjamin Smith []. It is called x-only arithmetic because any finite projective point $P = (X : Y : Z)$ of E can be mapped to a point $\mathtt{x}(P) = (X : Z) = (X/Z : 1) \in \mathbb{P}^1$ if $Z \neq 0$ or to $\mathtt{x}(P) = (1 : 0)$ if $Z = 0$. Any operation on a finite point P is done after discarding its Y coordinate. This coordinate can be recovered later if needed.

Lemma 3.3.1. Assume $X_{P-Q} \neq 0$.

If $P \neq Q$ then

$$\begin{cases} X_{P+Q} = Z_{P-Q}[(X_P - Z_P)(X_Q + Z_Q) + (X_P + Z_P)(X_Q - Z_Q)]^2 \\ Z_{P+Q} = X_{P-Q}[(X_P - Z_P)(X_Q + Z_Q) - (X_P + Z_P)(X_Q - Z_Q)]^2 \end{cases} \tag{1}$$

If $P = Q$ then

$$\begin{cases} X_{[2]P} = (X_P + Z_P)^2(X_P - Z_P)^2 \\ Z_{[2]P} = (4X_PZ_P)[(X_P - Z_P)^2 + \frac{A+2}{4}(4X_PZ_P)] \end{cases} \tag{2}$$

Using this lemma, we provide three generic pseudo-operations on Montgomery curve points that constitute the framework for x-only arithmetic in our implementation : addition, doubling and scalar multiplication.

These algorithms are called pseudo-operations because they operate over the quotient set of the curve E by the partition $\{\{P, -P\}, P \in E\}$. e.g. every point is identified with its opposite. This identification is implied in the following paragraphs.

xADD The pseudo-addition or differential addition `MG_xADD` returns $P + Q$ given $P, Q \in E$ and their difference $P - Q$ using the formula. Our implementation of xADD uses 4 multiplications, 2 squaring, 3 additions and 3 subtractions in $\mathbb{F}_p$, for a total of 12 base operations over $\mathbb{F}_p$. However the point addition xADD requires to first compute the difference $P - Q$, therefore it cannot be used as a generic point addition algorithm but can only be used in the specific context of a differential addition chain.

xDBL The pseudo-doubling `MG_xDBL` computes $[2]P$ from input P. The cost is 3 additions, 2 squaring, 2 subtractions, 3 multiplications and 1 division in $\mathbb{F}_p$, for a total of 11 operations. When calling xDBL multiple times for points on the same curve, one could cache or precompute the value $\frac{A+2}{4}$ from equation (2) and therefore shave off 1 addition and 1 division from the cost of every subsequent calls to xDBL.

Montgomery Ladder `MG_ladder` computes the point multiplication $[k]P$ calling xADD and xDBL. Write $k = \sum_{i=0}^{l-1} k_i 2^i$. The Montgomery Ladder follows a differential addition chain of length $2l - 1$ wherein the difference of consecutive terms is constant. The execution of `MG_ladder` requires l calls to xDBL and $l - 1$ calls to xADD.

```
P0 = P
P1 = xDBL(P)
for(int i=l-2; i>=0, i--) {
        if(k[i]==0){
                P1 = xADD(P0, P1, P)
                P0 = xDBL(P0)
        }
        else{
                P0 = xADD(P0, P1, P)
                P1 = xDBL(P1)
        }
}
return P0
```

In the specific case of the Montgomery Ladder Note that the Montgomery ladder can be implemented using a conditional constant-time swap to prevent branching or timing attacks.