Hard Homogenous Spaces and Commutative Supersingular Isogeny based Diffie-Hellman

Joël Felderhoff

Hard Homogenous Spaces and Commutative Supersingular Isogeny based Diffie–Hellman

Joël Felderhoff

1 Introduction

1.1 Motivation and problematic

When two parties want to communicate privately, a standard way to do it is to agree on a key and then use that key (or a quantity derived from it) to encrypt the messages using a symetric cryptosystem such as AES. An important algorithmic question is then "how can two parties share a key privately". This question gives raise to a fundamental topic which is at the heart of many cryptosystems used on a daily basis: the **key sharing**. It was introduced by Diffie and Hellman in [DH76] and the idea is to create a shared secret between two parties (usualy **Alice** and **Bob**) by publishing information such that an eavesdropping attacker could not get the shared secret.

The classical way to do such a sharing is the **Diffie–Hellman Key Exchange** explained in Figure 1. In this setting, G a group of prime order p and a generator g are chosen publicly. The notation $\xleftarrow{\$}$ denotes a uniform sampling.

Alice		Bob
$a \xleftarrow{\$} (\mathbb{Z}/p\mathbb{Z})^\times$		$b \xleftarrow{\$} (\mathbb{Z}/p\mathbb{Z})^\times$
$g_a \leftarrow g^a$		$g_b \leftarrow g^b$
	$\xrightarrow{g_a}$	
	$\xleftarrow{g_b}$	
$g_s \leftarrow g_b^a$		$g_s \leftarrow g_a^b$

Figure 1: The Diffie–Hellman key exchange

Alice and Bob are presumed authenticated for this protocol, else there is an obvious "Man in the Middle" attack. At the end of the exchange, both participants share $g_s = (g^a)^b = (g^b)^a$, and hence have a shared secret that has never transited througt the network. The question is then "how can a malicious listener find the shared secret given the public information (i.e., G, g and all the data sent throught the network)?". This question can be formulated in the following computational problem.

Definition 1.1.1. ***Computational Diffie–Hellman (CDH) problem***
Given $G = \langle g \rangle, g^a, g^b$ for $a, b \in (\mathbb{Z}/p\mathbb{Z})^\times$ compute g^{ab}.

This problem exists in different versions, each applying to particular cases: the secret key of Alice can be static (there will be multiple requests with the same a) or ephemeral (each new iteration of the protocol, a and b change). For the static case, one could think that only the public key g^a of Alice is revealed. The problem of finding its secret key from only this information is the **Discrete Log Problem**.

Definition 1.1.2. ***Discrete logarithm (DLOG) Problem***
Given $G = \langle g \rangle, g^a$ for $a \in \mathbb{Z}/\#G\mathbb{Z}$, find a.

The difficulty of those problems depends on the group considered. If $G = (\mathbb{Z}/p\mathbb{Z}, +)$, then those problems are trivially broken by modular division (solved by the Euclidean algorithm), but it is supposed to be classically hard (e.g., there is no known algorithm solving this problem in less than exponential time) if G is a cyclic subgroup of prime order of the group of points of an elliptic curve over a finite field (except for a little and very well known set of curves[1]). A third, intermediate, example is the group $\mathbb{F}_p^*$ for large p. Even though there is no known polytime algorithm for solving this problem, subexponential attacks exist (see for example [GM16]).

The problem with this protocol (and of all the protocols based on the discrete logarithm problem) is that with Shor's algorithm (decribed in [Sho94]), the DLOG problem is broken in polynomial time. Then, there were several attempts to build quantum resistant cryptosystems. Those systems are said to be **post-quantum**.

1.2 State of the art

In 2016, the American National Institute of Standards and Technology (NIST) initiated a call for project to try to create a standard for post-quantum cryptography. Several propositions where submitted and analysed by the institute and pairs. Among the ones which passed the first round, there are 3 types: cryptosystems based on multivariate polynomials, on lattices, and on isogenies.

The typical problem on which isogeny cryptography is based is the problem of finding an isogeny between two elliptic curves, given that such an isogeny exists. There are several versions of this problem, with differents class of curves and with more or less information about the underlying isogenies. The isogeny-based key encapsulation scheme that was taken for the second round is SIKE [SIK]. It is based on the **Supersingular Isogeny Diffie–Hellman (SIDH)** [JDF11] key exchange. In 2008, Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes published a preprint about another key exchange based on supersingular isogenies called **CSIDH** (for "commutative" SIDH) [CLM+18]. This protocol has an advantage comparing to SIDH, that is it can be studied and understood in a setting introduced by Couveignes in [Cou06], called **Hard Homogenous Spaces**, that enables more natural security hypothesis, an easier study of their complexities, and the adaptation of previously used protocol from security assumptions like **DLOG** or **CDH**.

1.3 Context of the internship

This internship was realised in the INRIA team GRACE at LIX under the supervision of Ben Smith. I also had the chance to attend the Winter School "Mathematical foundations of asymetric cryptography", organized by the LIP laboratory. During this school, I attended several talks about my research topic and about other public key cryptography topics, such as lattice-based cryptography.

During the my internship, I had the opportunity to prepare several associative events about the public computer science education. As a member of the association Prologin, I prepared and attended (as staff) the final of the programming contest Prologin[2], and I prepared (as principal organizator) the event Girls Can Code! Lyon 2019[3], a programming Summer camp for undergraduate girls. I was also invited to the Alkindi contest's final[4] to make an activity about public key

[1] Pairing-friendly and anomalous curves
[2] `https://prologin.org`
[3] `https://gcc.prologin.org`
[4] `http://www.concours-alkindi.fr`

cryptography for the finalists of the event.

1.4 Productions

My first work during the internship was to make myself clear about the general theory of elliptic curves in general and about the public key cryptography based on isogenies in particular. I then studied and implemented in **Rust**[5] the CSIDH algorithm along with a library to manipulate elliptic curves over prime fields and some protocols such as the OT protocol of Appendix A. During this time, some computational problems appeared (such as **2-Inv-Approx**) and started to investigate the difficulties of them compared to classical HHS problems such as CDH-HHS. I studied various reductions and protocols in of the discrete log litteratures, and I tried to adapt them to the HHS case, which seemed to be an easier framework to study them. During that time, I implemented the differents protocols with my CSIDH libraries and I tried to figure to what extends CSIDH could be studied via the HHS framework.

1.5 Organization of the report

In Section 2, I will introduce the notion of Hard Homogenous Space (HHS), and the problems associated with them. I will then discuss the relative computational difficulty of those problems, making reduction between them in different cases. In Section 3, I will study the major **HHS** of my internship, namely **Commutative Supersingular Isogeny Diffie–Hellman**. I will first introduce several theoretical notions, and I will make the link between **CSIDH** and the general theory of **HHS**, lighting their differences and their common points. The appendix will present algorithms and protocols that I studied and implemented during the internship.

2 Hard Homogenous Spaces

In this subsection I am going to introduce various definitions about (Hard) Homogenous Spaces. First I will explain what a **HHS** is, and then I will give the definitions of various computational problems about them, then I will talk about the relative difficulties of those problems. We will see that it depends on the structure of the **HHS** itself.

2.1 Definitions and notations

Let's start by defining some notation and objects for what comes next. In the rest of this report, G is a finite abelian group which is acting on X (a finite set).

2.1.1 Homogenous Spaces

In this document, I will use the capital letters for groups and sets of elements ($G, X \dots$), Greek letters for elements of the group acting on X ($\alpha, \beta, \gamma \dots$), letters of the end of the alphabet ($x, y, z, \dots$) to denote the elements of X and letters of the beginning of the alphabet to denote integers ($a, b, c, \dots$). In some case, like in example I will use group denoted multiplicatively, in that case the group name will be denoted by a gothic letter (e.g., $\mathfrak{H}$).

Elements of G will be denoted additively to make the analogy with vectors acting on a set of points. The action of an element $\alpha \in G$ on an element $x \in X$ will be denoted $[\alpha] \cdot x$. The neutral element will be denoted 0_G. Note that these notations differ from the usual CSIDH litterature, but

[5] Code avaible at `https://github.com/jetSett/Elliptic-curve-algorithms`

as the concept of homogenous space is studied here and not only CSIDH, I chosed those notations to make the analogy with both the discrete log setting in the case of elliptic curves and the case of vectors acting on a set of points of the space.

In our setting, the action is:

- Identic: $\forall x \in X,\ [0_G] \cdot x = x$
- Compatible: $\forall \alpha, \beta \in G,\ x \in X,\ [\alpha + \beta] \cdot x = [\alpha] \cdot [\beta] \cdot x.$
- Faithful and transitive: $\forall x, y \in X,\ \ \exists! \alpha \in G$ s.t.,$y = [\alpha] \cdot x.$

X is then said to be a Principal Homogeneous Space (or **PHS**) for G. In particular, $\#G = \#X$.

It is assumed that the following operations are efficiently computable (eg polytime in the size of G) to do.

- Given $x \in X$ and $\alpha \in G$, compute $[\alpha] \cdot x$ (**group action**).
- Given $\alpha, \beta \in G$, compute $\alpha + \beta$ (**group law**).

The most evident example of **PHS** is a group G acting on itself by its own group law, but this is not of great interest. Another example would be a vector space acting on its underlying affine space.

Example 1. *If E is a vector space and $\mathscr{E}$ is its underlying affine space, then $(E, \mathscr{E})$ is a **PHS** by the following action:*

$$[\vec{v}] \cdot x = x + \vec{v}$$

This example is not used in practice for reasons that will be explained in the next sections. Instead, another example is far more important for this report.

Example 2. *Let $\mathfrak{H}$ be a commutative group (written multiplicatively) of prime order p, then one can associate a **PHS** to it, that I will denote by $Exp(\mathfrak{H})$.*

$$Exp(\mathfrak{H}, p) = (\mathbb{Z}/p\mathbb{Z}^\times, \mathfrak{H} \setminus \{1_\mathfrak{H}\})$$

The action of $\alpha \in \mathbb{Z}/p\mathbb{Z}^\times$ on $h \in \mathfrak{H}$ is $[\alpha] \cdot h = h^\alpha$.

It must be emphasized that Example 2 is fundamental. The interest of the **PHS** problems are that they are generalisation of the **Discrete Log** ones and then they enable one to protocols, problems and reductions from the discrete log point of view to the **PHS** one.

The last example is a good candidate for Hard Homogenous Space suited for post-quantum cryptography, it will be more deeply developed in Section 3 of the report (including all the definitions of the terms used in it).

Example 3. *Let p be a prime number of the form $p = 4 \cdot \prod_i l_i - 1$ with l_i small primes, let $\mathcal{O}$ be the maximal order (i.e., the ring of integers) of the quadratic field $\mathbb{Q}(\sqrt{-p})$ and $G = Cl(\mathcal{O})$ be its class group.*

Now let $X = Ell_p(\mathcal{O})$ be the set of supersingular elliptic curves over $\mathbb{F}_p$ with endomorphism ring isomorphic to $\mathcal{O}$. Then (X, G) is an homogenous space.

With those **PHS**, one can desing a key exchange similar to the Diffie–Hellman key exchange of Figure 1. This key exchange is presented in Figure 2, where X, G is a **PHS** and x is a public parameter.

Alice		Bob
$\alpha \xleftarrow{\$} G$		$\beta \xleftarrow{\$} G$
$x_a \leftarrow [\alpha] \cdot x$		$x_b \leftarrow [\beta] \cdot x$
	$\xrightarrow{x_a}$	
	$\xleftarrow{x_b}$	
$x_s \leftarrow [\alpha] \cdot x_b$		$x_s \leftarrow [\beta] \cdot x_a$

Figure 2: The PHS Diffie–Hellman key exchange

2.1.2 Problems and Hard Homogenous Spaces

Now that Homogenous Spaces are defined, I am going to define some algorithmic problems about them. A major subject of my internship was to study the relative difficulty of those. In order to define the difficulty of the problems of this part, I introduce the parameter λ, and I will assume $\#G = \Theta(2^\lambda)$. The complexity of the algorithms and problems will expressed in terms of λ (e.g., a polytime algorithm will be read as "polytime in λ").

Definition 2.1.1. ***Vectorization*** *(Figure 3)*
Given $x, y \in X^2$, compute $\alpha \in G$ such that $y = [\alpha] \cdot x$.

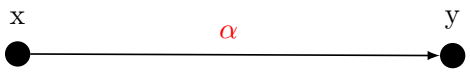

Figure 3: Vectorization

One can see that depending on the **PHS** one is considering, this problems' difficulties are on a line between "constant time" and "no polytime algorithm known". In the setting of Example 1, any instance of this problem can be efficiently solved since it suffices to return $y - x$ on input $(x, y) \in \mathscr{E}$. In the case of Example 2, where $(X, G) = Exp(\mathfrak{H}, p)$, an instance (x, y) of **Vectorization** is exactly a **Discrete Log** instance $DLOG_x(y)$ in the group $\mathfrak{H}$. At that point, it becomes evident that the difficulty of **Vectorization** changes a lot depending on the homogenous space one takes.

Definition 2.1.2. ***DLOG-Vectorization***
Given that $G = \langle\gamma\rangle$ is cyclic, $x, y \in X^2$ and $\gamma \in G$ a generator, find $n \in \mathbb{Z}_{\geq 0}$ such that $y = [n\gamma] \cdot x$.

One could note that **DLOG-Vectorization** is clearly harder than **Vectorization** but that there is a quantum reduction between those two because of the Shor algorithm ([Sho94]).

Definition 2.1.3. ***CDH-HHS*** *(Figure 4)*
Given $x, [\alpha] \cdot x, [\beta] \cdot x \in X^3$ compute $[\alpha + \beta] \cdot x$.

This problem is called **Parallelization** in [Cou06]. To take a look at what those looks like in the Example 2 will help to undestand the intrinsic difficulty of those problems.

Definition 2.1.4. *An homogenous space (X, G) is said to be a **Hard Homogenous Space (HHS)** if **CDH-HHS** is a hard problem for (X, G) e.g., there is no known polytime algorithm solving **CDH-HHS** for any arbitrary input $x, y, z \in X$.*

This distinction is important since the difficulty of the **CDH-HHS** problem will impact the cryptographic difficulty of the scheme that will be developed over it. For example in Example 2

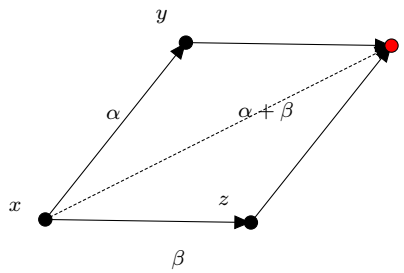

Figure 4: cdh-hhs

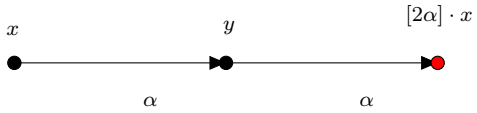

Figure 5: Square-HHS

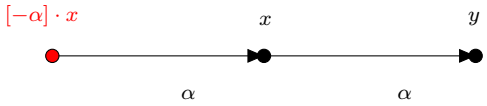

Figure 6: Inv-HHS

is classically hard with suitable groups (e.g., groups of points on elliptic curves over finite fields). In [Cou06], Couveigne asks for both **CDH-HHS** and **Vectorization** to be hard for a PHS to be a HHS, but as it will be explained in the following, **CDH-HHS** is easier than **Vectorization**, so the hardness of **CDH-HHS** implies the **Vectorization**'s one.

In the setting of Example 2, when $(G, X) = Exp(\mathfrak{G}, p)$ (and then $G = (\mathbb{Z}/p\mathbb{Z})^\times$), an instance of **CDH-HHS** is an instance of the **Computational Diffie–Hellman Problem**: given $(x, [\alpha] \cdot x, [\beta] \cdot x)$ it is asked to compute $[\alpha\beta] \cdot x$ (at least when $\alpha, \beta \neq 0$). This problem is known to be hard and it has been studied for a long time. One can then try to introduce the same kind of problem in the **HHS** settings and see how the relation between those problems behaves. In that setting, **CDH** is equivalent to **Inv-CDH** (given $g \in \mathfrak{H}, g^a$ compute $g^{a^{-1}}$) and **Square-CDH** (given $g \in \mathfrak{H}, g^a$ compute g^{a^2}). On can define analogs of those problems into the more general **HHS** setting.

Definition 2.1.5. ***Square-HHS*** *(Figure 5)*
Given $x, [\alpha] \cdot x$ *compute* $[2\alpha] \cdot x$.

Definition 2.1.6. ***Inv-HHS*** *(Figure 6)*
Given $x, [\alpha] \cdot x$ *compute* $[-\alpha] \cdot x$.

Both of these problems can be generalized for any scalar multiplication.

Definition 2.1.7. ***Scalar-HHS***
Given $x, [\alpha] \cdot x, n \in \mathbb{Z}$ *compute* $[n\alpha] \cdot x$.

By examining the **OT** scheme of [Vit19] (which is presented in Appendix A), other problems appeared. These problems are "approximate versions" of **Inv-HHS**.

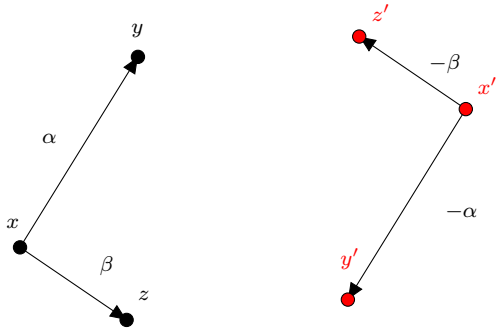

Figure 7: 2-Inv-Approx-HHS

Definition 2.1.8. ***k*-Inverse-Approx-HHS** *(Figure 7) Given* $x, (x_i)_{1\geq i\geq k} \in X \times X^k$ *with* $x_i = [\alpha_i] \cdot x$*, find* $x', (x'_i)_{1\geq i\geq k} \in X \times X^k$ *such that* $x_i = [-\alpha_i] \cdot x$*.*

All the problems seen so far can be sumarized in Table 1.

Name	Input	Output
DLOG-Vectorization	$G = \langle\gamma\rangle, x, y \in X$	$n \in \mathbb{Z}$ s.t., $y = [n\gamma] \cdot x$
Vectorization	$x, y \in X$	$\alpha \in G$ s.t., $y = [\alpha] \cdot x$
CDH-HHS	$x, y = [\alpha] \cdot x, z = [\beta] \cdot x \in X$	$[\alpha + \beta] \cdot x$
Square-HHS	$x, y = [\alpha] \cdot x \in X$	$[2\alpha] \cdot x$
Inv-HHS	$x, y = [\alpha] \cdot x \in X$	$[-\alpha] \cdot x$
Scalar-HHS	$x, y = [\alpha] \cdot x \in X, n \in \mathbb{Z}$	$[n\alpha] \cdot x$
***k*-Inverse-Approx-HHS**	$x, (y_i = [\alpha_i] \cdot x)_{1\leq i\leq k} \in X$	$x', (y'_i = [-\alpha_i] \cdot x')_{1\leq i\leq k}$

Table 1: HHS problems

2.2 Relative difficulty of HHS problems

First I will state some trivial ordering on the difficulties of the problems defined in the previous part, then I will explain more subtle links. All reductions and equivalence (unless stated otherwise) are **classical equivalences**. In this part (G, X) is a **PHS**.

2.2.1 Easy and known links between problems

Galbraith, Panny, Smith and Vercauteren proved that **CDH-HHS** and **Vectorization** are quantumly equivalent ([GPSV18]).

One can also notice that an oracle for the **CDH-HHS** problem and the data of a single point x gives a group structure to X with x as neutral element (this is the underlying idea of the proof of [GPSV18]). The law in question is the following. If one wants to compute $y \star z$, write $y = [\alpha] \cdot x$, $z = [\beta] \cdot x$ and use the **CDH-HHS** oracle to compute $w = [\alpha+\beta] \cdot x$, then define $y \star z = w$. This group law gives access to the structure of G via the elements of X. It is then easy to performs the operations asked in the problems **Square-HHS**, **Inv-HHS**, **Scalar-HHS** and ***k*-Inverse-Approx-HHS** - for example $CDH(x, y = [\alpha] \cdot x, y) = [2\alpha] \cdot x$. **Square-HHS**, **Inv-HHS**, etc... then reduces to **CDH-HHS**.

I should also state some other easy reductions: ***k*-Inverse-Approx-HHS** reduced to ***k'*-Inverse-Approx-HHS** with $k' \geq k$ and ***k*-Inverse-Approx-HHS** reduces to **Inv-HHS**.

Lemma 2.2.1. 1-***Inverse-Approx-HHS*** *is trivial.*

Proof. Let x, y be a 1-**Inverse-Approx-HHS** instance. One can simply return the tuple (y, x).

If it is required that the returned tuple is not (y, x) (which seems fair), the problem is still trivial if it is assumed that one has access to a nontrivial α in G; then one can return $[\alpha]\cdot y, [\alpha]\cdot x$, which will be a valid 1-**Inverse-Approx-HHS** solution (indeed, every solution for 1-**Inverse-Approx-HHS** will have this form). □

2.2.2 Equivalence of Square, Inverse and Scalar multiplication

Now, let's show that the **Square-HHS**, **Inv-HHS** and **Scalar-HHS** are, as in the discrete logarithm case, equivalent. The proof coming from the DLOG setting (see [BDZ03]) transfers quite easily to the HHS setting.

Proposition 2.2.2. ***Square-HHS*** *and* ***Inv-HHS*** *are equivalent.*

Proof. Let $x, y = [\alpha] \cdot x$ be an instance of one of those problems.

If one has access to S, a **Square-HHS** oracle, then she can solve **Inv-HHS** by:

$$S(y, x) = S([\alpha] \cdot x, [-\alpha + \alpha] \cdot x) = [-2\alpha + \alpha] \cdot x = [-\alpha] \cdot x.$$

If one has access to I, a **Inv-HHS** oracle, then she can solve **Square-HHS** by:

$$I(y, x) = I([\alpha] \cdot x, [-\alpha + \alpha] \cdot x) = [\alpha + \alpha] \cdot x = [2\alpha] \cdot x.$$

□

Theorem 2.2.3. ***Square-HHS*** *and* ***Scalar-HHS*** *are equivalent.*

Proof. **Square-HHS** is a subproblem of **Scalar-HHS** so the interesting thing to prove is the reduction from **Scalar-HHS** to **Square-HHS**.

Let's assume that one has a **Square-HHS** oracle S, that is to say an oracle wich on input $x, y = [\alpha] \cdot x$ returns $[2\alpha] \cdot x$.

Let $x, y = [\alpha] \cdot x \in X, n \in \mathbb{Z}_{\geq 0}$ be an instance of **Scalar-HHS** and observe that for all $a, b \in \mathbb{Z}$, $O([a\alpha] \cdot x, [b\alpha] \cdot x) = [(2b - a)\alpha] \cdot x$. One can then build a recursive polytime (in $\log(n)$) square-and-multiply algorithm to compute $[n\alpha] \cdot x$. This is Algorithm 2.1.

Algorithm 2.1 Compute $[n\alpha] \cdot x$ on input x, $y = [\alpha] \cdot x$ and $n \in \mathbb{Z}_{\geq 0}$

```
1:  procedure SCALAR(x, y, n)
2:      if n = 0 then return x
3:      else if n = 1 then return y
4:      if n is even then
5:          x' ← SCALAR(x, y, n/2)
6:          w ← S(x, x') = [(2·n/2 − 0)α] · x = [nα] · x
7:          return w
8:      else
9:          x' ← SCALAR(x, y, (n+1)/2)          ▷ (n + 1)/2 < n here.
10:         w ← S(y, x') = [(2·(n+1)/2 − 1)α] · x = [nα] · x
11:         return w
```

This algorithm is clearly polytime in $\log(n)$ and solves the problem. □

Figure 8 makes a global diagram of the problems studied so far.

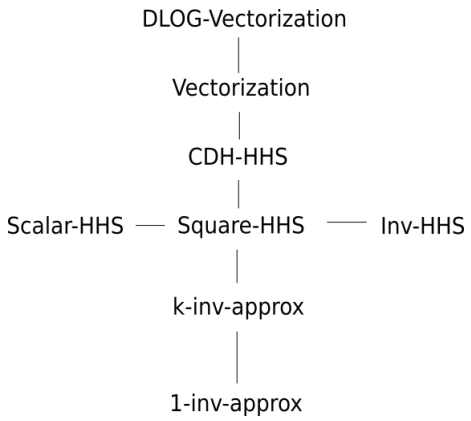

Figure 8: Relative difficulties of HHS problems in the general setting

2.2.3 Comparison between Scalar Multiplication and CDH-HHS

The interesting question is to know whether this scalar multiplication is as hard as the **CDH-HHS** problem. In the DLOG setting, **Scalar** and **CDH** are equivalent (see [BDZ03]), but the proof do not transfer to the HHS setting, in fact tt turns out that this question seems to be related with the structure of the group.

Theorem 2.2.4. *If one has access to an **Square-HHS** oracle and if $x, x_\alpha = [\alpha] \cdot x, x_\beta = [\beta] \cdot x$ a **CDH-HHS** instance is given such that the order of α is odd and known, then there is a poly-time algorithm computing $[\alpha + \beta] \cdot x$.*

*In particular, if (G, X) is a **HHS** with G having odd order and if this order is known, then **Square-HHS** is equivalent to **CDH-HHS** for this HHS.*

Proof. Let $(x, x_\alpha = [\alpha] \cdot x, x_\beta = [\beta] \cdot x)$ be a **CDH-HHS** instance. Assume that the order of α is $2n + 1$ for some n and let S be a **Square-HHS** oracle S and $Scal$ a **Scalar-HHS** oracle.

Algorithm 2.2 Compute $[\alpha + \beta] \cdot x$ on input x, x_α, x_β given Square and Scalar HHS oracles

1: $x_{2\alpha} \leftarrow S(x, x_\alpha)$
2: $x_{2\alpha-\beta} \leftarrow S(x_\beta, x_\alpha)$
3: $x_{2\alpha+\beta} \leftarrow S(x_{2\alpha-\beta}, x_{2\alpha})$
4: $w \leftarrow Scal(x_\beta, x_{2\alpha+\beta}, n + 1)$
5: **return** w

Algorithm 2.2 computes $[\alpha + \beta] \cdot x$, and is clearly polytime. One can check that at each step $x_\gamma = \gamma \cdot x$ for every $\gamma \in G$ considered. The last step $Scal(x_\beta, x_{2\alpha+\beta}, n+1) = Scal(x_\beta, [2\alpha] \cdot x_\beta, n+1)$ computes

$$(2n + 1 + 1)\alpha \cdot x_\beta = [\alpha] \cdot x_\beta = [\alpha + \beta] \cdot x$$

□

Remark 2.2.5. *The above reduction works even if an odd multiple of the order of α is known, but it is not possible to use it if the orders of α and β are even.*

Indeed, the following proposition shows that if the orders of α and β are even then there are cases where it is impossible to solve **CDH-HHS** with only an oracle for **Square-HHS**.

Proposition 2.2.6. *Let $(x, [\alpha] \cdot x, [\beta] \cdot x)$ be an instance of **CDH-HHS** where α and β have even order and are independent ($a\alpha = b\beta \implies a\alpha = b\beta = 0$). Then it is impossible to give an answer to this instance of **CDH-HHS** with only calls to an instance of **Square-HHS**.*

Proof. Let S be an oracle for **Square-HHS**. Let $(x, [\alpha] \cdot x = x_\alpha, [\beta] \cdot x = x_\beta)$ be an instance as in the statement.

Now consider Y, the set of points that one can generate with calls to S and the points x, x_α, x_β. I claim that the only points that can be generated by those calls are the points of the form $[a\alpha + b\beta] \cdot x$ with $a, b \in \mathbb{Z}$. I claim also that there are restrictions on a and b in that case, namely:

$$Y \subseteq \{[a\alpha + b\beta] \cdot x, \quad ab \equiv 0 \mod 2\}$$

Note that it makes sense to look at a and b and their parities only because α and β are independent and a (resp b) is defined modulo the order of α (resp β) which is even. Now recall that $S([a\alpha + b\beta] \cdot x, [a'\alpha + b'\beta] \cdot x) = [(2a' - a)\alpha + (2b' - b)\beta] \cdot x$, so the parity of a, b is preserved by the application of S. Since I only start with points whose group element parity are $(0, 0), (1, 0)$ and $(0, 1)$, a point with parity $(1, 1)$ will never be created by repeated applications of S. □

2.3 Reductions exploiting group structure

Previous results used few assumptions about the group. In the general case, one could think that an attacker would have a better knowledge of G than just its order.

For example, with only the public presentation of a group G, a quantum attacker could derive its whole structure using using Kitaev's generalisation of Shor's algorithm of [Kit95]. This justify the study of **HHS** when the group has a particular (and known structure).

2.3.1 The cyclic case

If G is cyclic with generator γ (which is known), then the problem of **DLOG-Vectorization** turns out to be easier than one could expect in view of Proposition 2.2.6.

Proposition 2.3.1. *If G is a cyclic group with order a power of two, then there is a reduction from **DLOG-Vectorization** to **Scalar-HHS** in this setting.*

Proof. Let x, y be an instance of vectorization, and assume that the order of G is 2^t. Let *Scal* be an oracle for **Scalar-HHS**.

Algorithm 2.3 On input $x, y = [\beta] \cdot x, \gamma, t$, compute a if $\beta = a\gamma$, else fail.

```
1:  a ← 0, α ← 0_G                          ▷ The current approximation of the discrete log in γ
2:  z ← y
3:  for i = 0 to t − 1 do                    ▷ Loop invariant: [α] · z = y
4:  |   z' ← Scal(x, z, 2^{t−1−i})
5:  |   if z' ≠ x then
6:  |   |   a ← a + 2^i
7:  |   |   α ← α + 2^i γ
8:  |   |   z ← [−2^i γ] · z
9:  if z ≠ x then
10: |   Fail
11: else
12: |   return a
```

If $y = [\sum_{i=0}^{t-1} 2^i a_i \gamma] \cdot x$ with all $a_i \in \{0,1\}$, then Algorithm 2.3 computes a_i at the step i. The loop invariant to consider is:

$I(i)$:"At the begining of step i, $\alpha = \sum_{j=0}^{i-1} 2^j a_j \gamma$ and $z = [\alpha] \cdot x$".

$I(0)$ is clearly true. If $I(i)$ is true, let's show that $I(i+1)$ is true too. There are two cases to consider: either $a_i = 0$ or $a_i = 1$. By $I(i)$,

$$z' = [2^{t-1-i}(\sum_{j=i}^{t-1} 2^j a_j)\gamma] \cdot x = [\sum_{j=i}^{t-1} 2^{j+t-1-i} a_j \gamma] \cdot x = [2^{t-1} a_i \gamma] \cdot x.$$

Then $z' = x$ if and only if $2^{t-1} a_i \gamma = 0_G$ i.e. $a_i = 0$. This concludes the proof that $I(i+1)$ is true.

It is easily shown that $\alpha = a\gamma$ at every point of the algorithm. At the end, the algorithm fails if and only if β is not a multiple of γ. □

The two precedent proofs can then be combined to describe a reduction from **CDH-HHS** to **Scalar-HHS**.

Theorem 2.3.2. *In the setting of cyclic groups with known generators γ, there is an effective reduction from **CDH-HHS** to **Scalar-HHS**.*

Proof. Let $(x, [\alpha] \cdot x = x_\alpha, [\beta] \cdot x = x_\beta)$ be an instance of **CDH-HHS**. Let's assume that the access to a **Scalar-HHS** oracle *Scal* is granted. Let's assume G has known order $N = 2^s \cdot r$.

The idea is first to vectorize the even part of α and β and then to apply the reduction of Theorem 2.2.4. Let $A(x, y, \gamma)$ be an oracle for **DLOG-Vectorization** in the group spanned by γ of order a power of two. This oracle can be constructed from the *Scal* oracle using Algorithm 2.3.

Algorithm 2.4 Compute $[\alpha + \beta] \cdot x$ on input $x, [\alpha] \cdot x = x_\alpha, [\beta] \cdot x = x_\beta$

1: $\gamma_2 \leftarrow r\gamma$ ▷ γ_2 is a generator of the 2th power part of G.
2: $x_\alpha^{(2)} \leftarrow S(x, x_\alpha, r)$ and $x_\beta^{(2)} \leftarrow S(x, x_\beta, r)$ ▷ The odd part of α and β is killed.
3: $a \leftarrow A(x, x_\alpha^{(2)}, \gamma_2)$ and $b \leftarrow A(x, x_\beta^{(2)}, \gamma_2)$
4: $x_\alpha^{(1)} \leftarrow [-a\gamma] \cdot x_\alpha$, $x_\beta^{(1)} \leftarrow [-b\gamma] \cdot x_\beta$ ▷ The even part of α and β is killed.
5: $\gamma_1 \leftarrow 2^t \gamma$ ▷ γ_1 is a generator of the odd part of G.
6: compute $x_{\alpha+\beta}^{(1)}$ with the reduction of Theorem 2.2.4 in the group generated by γ_1 (of odd order).
7: **return** $[(a+b)\gamma] \cdot x_{\alpha+\beta}^{(1)}$

In Step 3, a and b are computed such that $r\alpha = a\gamma_2$ and $r\beta = b\gamma_2$. Hence in Step 4,

$$x_\alpha^{(1)} = [-a\gamma] \cdot x_\alpha = [\alpha - a\gamma] \cdot x \quad \text{and} \quad r(\alpha - a\gamma) = a\gamma_2 - a\gamma_2 = 0_G.$$
$$x_\beta^{(1)} = [-b\gamma] \cdot x_\beta = [\beta - b\gamma] \cdot x \quad \text{and} \quad r(\beta - b\gamma) = b\gamma_2 - a\gamma_2 = 0_G.$$

Hence $x_\alpha^{(1)} = [\alpha'] \cdot x$ and $x_\beta^{(1)} = [\beta'] \cdot x$ with α' and β' of odd order, hence in the cyclic group generated by γ_1, which justifies the use of the reduction of Theorem 2.2.4.

At the end, $x_{\alpha+\beta}^{(1)} = [\alpha' + \beta'] \cdot x = [\alpha + \beta - (a+b)\gamma] \cdot x$ so the value $[(a+b)\gamma] \cdot x_{\alpha+\beta}^{(1)}$ is equal to $[\alpha + \beta] \cdot x$, which is the desired output of the algorithm. □

2.3.2 Smooth-order groups

Algorithm 2.3 gives a reduction from **DLOG-Vectorization** to **Scalar-HHS** in the case of cyclic groups of order 2^t. But it can be straightforwardly adapted to any group of order p^t for a small p. I'm assuming that a generator γ of the group is given along with its order p^t.

Algorithm 2.5 On input $(x, y = [\beta] \cdot x, \gamma, p, t)$ compute b if $\beta = b\gamma$, else fail

```
Let a ← 0, α ← 0_G                          ▷ The current approximation of the discrete log in γ
z ← y
for i = 0 to t − 1 do                        ▷ Loop invariant: [α] · z = y
    z' ← S(x, z, p^{t−1−i})
    for k = 0 to p − 1 do
        if [−kp^{(t−1)}γ] · z' = x then
            a ← a + k · p^i
            α ← α + k · p^i γ
            z ← [−kp^i γ] · z
            break
if z ≠ x then
    Fail
else
    return a
```

This algorithm gives a reduction in time $O(p \cdot t)$. This is going to enable us to give a reduction from **DLOG-Vectorization** to **Scalar-HHS** for cyclic groups of smooth order. The algorithm is a straightforward generalisation of Pohlig–Hellman algorithm.

Theorem 2.3.3. *If G is cyclic of order $N = \prod_{i=1}^{n} p_i^{t_i}$, then there is a $O(\min_i (t_i p_i))$ reduction from **DLOG-Vectorization** to **Scalar-HHS** described in Algorithm 2.6.*

Algorithm 2.6 Compute the discrete log of α in base γ on input $x, y = [\alpha] \cdot x, \gamma, (p_i)_{i=1\ldots n}, (t_i)_{i=1\ldots n}$

Proof.
```
for i = 1 to n do
    Compute a_i the discrete log of Scal(x, y, N/p_i^{t_i}) in base N/p_i^{t_i} γ
Use EEA to compute u_1, ..., u_n ∈ ℤ such that Σ_{i=1}^n (N/p_i^{t_i}) u_i = 1.
return Σ_{i=1}^n (a_i N u_i)/(p_i^{t_i})  mod N
```

As before, *Scal* is a **Scalar-HHS** oracle. At each step, (a_i) are computed such that $N/p_i^{t_i}\alpha = (N/p_i^{t_i})a_i\gamma$. This is a valid operation since $N/p_i^{t_i}\gamma$ is in the $p_i^{t_i}$-part of G which is generated by $N/p_i^{t_i}\gamma$. By the CRT, $\alpha = \sum_{i=1}^{n} u_i(\frac{N}{p_i^{t_i}}\alpha) = \sum_{i=1}^{n} u_i(a_i \frac{N}{p_i^{t_i}}\gamma)$. Then the DLOG of α in base γ is $\sum_{i=1}^{n} \frac{a_i N u_i}{p_i^{t_i}}$. □

One can see that this result is legitimate, since in cyclic group **Scalar-HHS** and **CDH-HHS** are equivalent, and that with a **CDH-HHS**, X can be endowed with a group structure.

In cyclic groups, the relative difficulties of the problems are represented in Fig. 9.

2.4 Non-cyclic groups

The case of non-cyclic groups is more subtle. Of course some previous reductions still work (namely all the reductions of Section 2.2.2), but others seem to be more difficult to maintain.

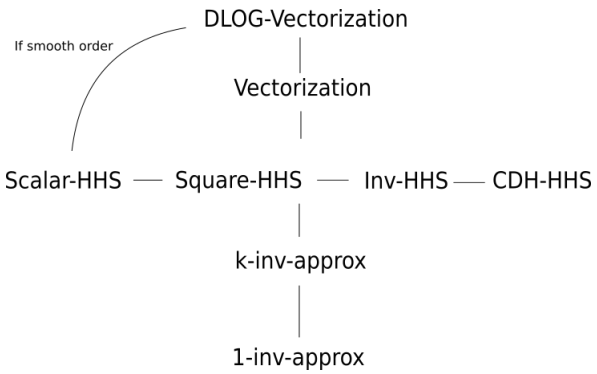

Figure 9: Relative difficulties of HHS problems in the cyclic group setting

For example, a good reason to think that something similar to Proposition 2.3.1 does not apply in the general case is that giving access to a **Square** oracle for the group $G = (\mathbb{Z}/2\mathbb{Z})^n$ acting on some set does not give any information, since every element of G has order 2.

Proposition 2.4.1. *If $G = G_1 \times G_2$ with G_1 of odd order r and G_2 of order 2^t, and if generators of G_2 are known, then there is a $(polylog(r) + 2 \cdot 2^t)$ reduction from **CDH-HHS** to **Scalar-HHS**.*

Proof. The idea of the reduction is to make an exhaustive search for the even part of the elements and then to use the reduction of Theorem 2.2.4. Let *Scal* be a **Scalar-HHS** oracle. Algorithm 2.7 computes the reduction. □

Algorithm 2.7 Compute $[\alpha + \beta] \cdot x$ on input $x, x_\alpha = [\alpha] \cdot x, x_\beta = [\beta] \cdot x$

1: $x_\alpha^{(2)} \leftarrow Scal(x, x_\alpha, r)$ and $x_\beta^{(2)} \leftarrow Scal(x, x_\beta, r)$ ▷ The odd part of α and β are killed.
2: Make an exhaustive search over the elements of G_2 to find α_2 and β_2 such that $x_\alpha^{(2)} = [\alpha_2] \cdot x$ and $x_\beta^{(2)} = [\beta_2] \cdot x$
3: $x_\alpha^{(1)} \leftarrow Scal(x, x_\alpha, 2^t)$ and $x_\beta^{(1)} \leftarrow Scal(x, x_\beta, 2^t)$ ▷ $x_\alpha^{(1)}$ and $x_\beta^{(1)}$ have odd order by CRT
4: Use the reduction of Theorem 2.2.4 to compute y^1, the **CDH-HHS** output on input $(x, x_\alpha^{(1)}, x_\beta^{(1)})$
5: Compute $u, v \in \mathbb{Z}$ such that $u2^t + vr = 1$
6: $w \leftarrow [u\alpha_2 + u\beta_2] \cdot Scal(x, y^1, v) = [ur\alpha + ur\beta + 2^t v\alpha + 2^t v\beta] \cdot x = [\alpha + \beta] \cdot x$
7: **return** w

Corollary 2.4.2. *The exaustive search of the previous algorithm can be replaced by Algorithm 2.3 if the even part of the group appears to be cyclic. On would then have a polytime reduction from **CDH-HHS** to **Scalar-HHS**.*

3 Commutative Supersingular Isogeny Diffie–Hellman

If one takes a prime p of the form $p = 4 \prod_i l_i - 1$ with (l_i) some odd primes, then she can consider a group action that can be modeled as an **HHS**.

- $X = Ell_p(\mathcal{O})$ with $\mathcal{O} = \mathbb{Z}[\pi]$ up to $\mathbb{F}_p$-isomorphism. Each class of curves is represented by its Montgomery coefficient, which makes sense as $p \equiv 3 \mod 8$.

- $G \subset Cl(\sqrt{-p})$ (more precisely G is the subgroup of $Cl(\sqrt{-p})$ generated by the ideals $([l_i] = (l_i, \pi + 1))_i$).

This setting (more precisely the key exchange derived from this setting) is called **CSIDH**.

In this section, first I will explain the mathematical background important to understand this group action, then I will explain to what extent my discussion about the **HHS** carries over to the **CSIDH** framework.

3.1 Elliptic Curves and Isogenies

The theory of elliptic curves comes from the number theory world and has many applications in computer science, for example factorization algorithms, key exchanges, or attacks on cryptosystems. There are many ways to present their theory but here I will try to stick to the simplest and to introduce only the objects I will need. In the following, $\mathbb{F}$ will be a field of characteristic different from $2, 3$. The interested reader will find a more precise and exhaustive presentation in [Sil09] or a presentation focused on the application of isogeny-based cryptography in [De 17].

Definition 3.1.1. *Let $a, b \in \mathbb{F}$, an elliptic curve $\mathcal{E}$ defined over $\mathbb{F}$ (we will just note $\mathcal{E}/\mathbb{F}$) is the curve defined by the equation $y^2 = x^3 + ax + b$. Its set of points is*

$$\mathcal{E}(\mathbb{F}) = \left\{(x, y) \in \mathbb{F}^2 : y^2 = x^3 + ax + b\right\} \cup \{\infty\}$$

.

Where ∞ stands for a point "at infinity".

This point at infinity has a proper meaning in terms of projective geometry, but it is not interesting for this short presentation. Elliptic curves are **algebraic curves** (set of points defined by a single polynomial equation). It turns out that elliptic curves also have a group structure, by the "chord and tangent rule" (Figure 10).

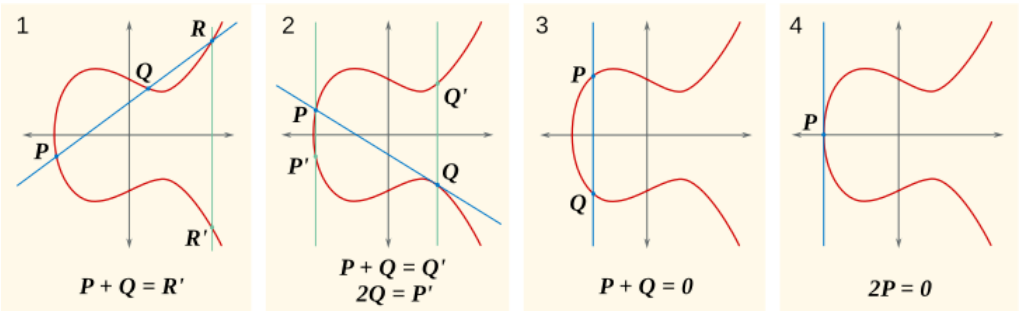

Figure 10: Operations on elliptic curves.[6]

Proposition 3.1.2. *[Sil09, Prop. III.2.2] The operations of Figure 10 define a group law on any elliptic curve $\mathcal{E}(\mathbb{F})$, this group law is defined by algebraic operations and its neutral element is ∞.*

Remark 3.1.3. *If $\mathbb{K}$ is an extension of $\mathbb{F}$ (e.g., $\mathbb{K} = \overline{\mathbb{F}}$ the algebraic closure of $\mathbb{F}$), we can see $\mathcal{E}$ as a curve over $\mathbb{K}$ and then $\mathcal{E}(\mathbb{F})$ will be a subgroup of $\mathcal{E}(\mathbb{K})$. A way to see this is to say that $\mathcal{E}$ is a functor from the category of extensions of $\mathbb{F}$ to the category of Abelian groups.*

One can see that elliptic curves have two "facets" : they are algebraic objects and groups. An **isogeny** is then a morphism that maintains those two aspects.

[6] Figure by SuperManu [CC BY-SA 3.0 (https://creativecommons.org/licenses/by-sa/3.0)]

Definition 3.1.4. *Let $\mathcal{E}_1/\mathbb{F}, \mathcal{E}_2/\mathbb{F}$ be two elliptic curves. A map $\phi : \mathcal{E}_1 \to \mathcal{E}_2$ is said to be a* ***morphism of algebraic varieties*** *if it is defined by algebraic equations over $\mathbb{F}$. It is said* ***defined over a field*** *$\mathbb{F}$ if its equations have coefficients in $\mathbb{F}$. This map is said to be a homomorphism if, in addition, it sends the point at infinity of $\mathcal{E}_1$ into the point at infinity of $\mathcal{E}_2$.*

A map $\phi : \mathcal{E}_1 \to \mathcal{E}_2$ is said to be an ***isogeny*** *if it is a nonzero homomorphism.*

I can then give the most fundamental examples of isogenies. Let $\mathcal{E}/\mathbb{F}$ be an elliptic curve.

Definition 3.1.5. *Let $n \in \mathbb{Z}$, then*

$$\begin{array}{rccc} [n]: & \mathcal{E} & \longrightarrow & \mathcal{E} \\ & P & \longmapsto & \underbrace{P + \cdots + P}_{n \text{ times}} \end{array}$$

is an isogeny if and only if $n \neq 0$.

The second example is less straightforward but also very important.

Definition 3.1.6. *Let $\mathbb{F}$ be the finite field with q elements $\mathbb{F}_q$. Then the q-Frobenius endomorphism is*

$$\begin{array}{rccc} \pi_q: & \mathcal{E} & \longrightarrow & \mathcal{E} \\ & P = (x, y) & \longmapsto & (x^q, y^q) \end{array}$$

and it is an isogeny.

One can see that these are isogenies from $\mathcal{E}$ to $\mathcal{E}$. They are **endomorphisms** (homomorphism from $\mathcal{E}$ to $\mathcal{E}$). As endomorphisms are stable under composition and additions, they form a ring denoted $End_{\mathbb{F}}(\mathcal{E})$. The structure of this ring very restricted.

Proposition 3.1.7. *[Sil09, Prop. III.4.2] The ring $End_{\mathbb{F}}(\mathcal{E})$ contains $\mathbb{Z}$ by the morphism $n \mapsto [n]$ i.e., the map $n \mapsto [n]$ is an injection from $\mathbb{Z}$ into $End_{\mathbb{F}}(\mathcal{E})$.*

One can go further and say that when $\mathbb{F} = \mathbb{F}_q$, $End_{\mathbb{F}}(\mathcal{E})$ contains $\mathbb{Z}[\pi]$. Isogenies have a very restricted definition, and there are several facts about them.

Theorem 3.1.8. *[Sil09, Prop. III.4.12] If $\phi : \mathcal{E}_1 \to \mathcal{E}_2$ is an isogeny, then its kernel is a finite subgroup of $\mathcal{E}_1(\overline{\mathbb{F}})$ and there is correspondence between subgroups of $\mathcal{E}_1(\overline{\mathbb{F}})$ and isogenies starting from $\mathcal{E}_1$.*

More precisely, if $H \subset \mathcal{E}_1(\overline{\mathbb{F}})$ is finite, then there is an unique (up to $\overline{\mathbb{F}}$-isomorphism) elliptic curve $\mathcal{E}_H/\overline{\mathbb{F}}$ and an isogeny $\phi_H : \mathcal{E}_1 \to \mathcal{E}_H$ defined over $\overline{\mathbb{F}}$ such that the kernel of ϕ_H is H. Moreover, if one requires ϕ_H to be ***separable*** *(a notion that is beyond the scope of this report), then the correspondance is one-to-one.*

Proposition 3.1.9. *In Theorem 3.1.8, if H is a subgroup stable by the action of the Galois group of $\overline{\mathbb{F}}/\mathbb{F}$ (e.g., a subgroup of $\mathcal{E}(\mathbb{F})$), then both ϕ_H and $\mathcal{E}_H$ are defined over $\mathbb{F}$.*

What Theorem 3.1.8 along with Proposition 3.1.9 are saying is that if one works with isogenies defined over $\mathbb{F}$, she only have to look at curves **up to $\mathbb{F}$-isomorphism**. This will be important for the algorithm I am going to consider in the report. The class of curve defined by a subgroup H is called the **quotient curve** of $\mathcal{E}$ by H.

Remark 3.1.10. *The notations here are quite ambiguous and I emphasize that the group of points of a quotient curve $(\mathcal{E}/H)(\mathbb{F})$ is* ***not*** *equal to the group quotient $\mathcal{E}(\mathbb{F})/H$. Indeed, if $\mathbb{F}$ is a finite field, $|(\mathcal{E}/H)(\mathbb{F})| = |\mathcal{E}(\mathbb{F})|$.*

From this point, the only fields I am going to consider are $\mathbb{F}_p$ and its algebraic closure $\overline{\mathbb{F}}_p$ for a prime p. For the following, $\mathcal{E}$ is a curve defined over $\mathbb{F}_p$.

Proposition 3.1.11. *[Sil09, Th. V.2.3.1] There exist t such that $\mid t \mid \leq 2\sqrt{p}$ For every $P \in \mathcal{E}(\overline{\mathbb{F}}_p)$,*

$$\pi_p(P)^2 - t\pi_p(P) + p = 0$$

The integer t is called the trace of the Frobenius.

The value of t has a great influence on the structure of the endomorphism rings of a given elliptic curve. Here, in order to stick to the **CSIDH** cryptosystem, I will focus on curves with $t = 0$ (a particular case of **supersingular curves**). For the following, $\mathcal{E}$ has Frobenius' trace 0.

It is interesting to note that in that case, π_p follows the equation $X^2 + p = 0$, meaning that if it is seen as a purely algebraic object it can be seen as $\sqrt{-p}$.

Theorem 3.1.12. *[Sil09, Th. III.9.3] The ring $End_{\mathbb{F}_p}(\mathcal{E}) = \mathcal{O}$ is an order of $\mathbb{Q}(\sqrt{-p})$ i.e.,*

$$\mathcal{O} \otimes \mathbb{Q} \cong \mathbb{Q}(\sqrt{-p}).$$

Corollary 3.1.13. *If $\mathcal{O}_K$ is the ring of integers of $\mathbb{Q}(\sqrt{-p})$, then*

$$\mathbb{Z}[\pi_p = \sqrt{-p}] \subset End_{\mathbb{F}_p}(\mathcal{E}) \subset \mathcal{O}_K.$$

The structure of endomorphism ring is well studied and well known. It can be used to study the actions of endomorphism on a certain curve. In the following, I will denote $End_{\mathbb{F}_p}(\mathcal{E})$ by $\mathcal{O}$.

Proposition 3.1.14. *If $\mathfrak{a}$ is an ideal of $\mathcal{O}$, then the set $\mathcal{E}[\mathfrak{a}] = \{P \in \mathcal{E}, \phi(P) = \infty \ \forall \phi \in \mathfrak{a}\}$ is a subgroup of $\mathcal{E}(\mathbb{F}_p)$.*

One could try to take the quotient curve by $\mathcal{E}[\mathfrak{a}]$. It will be denoted by $\mathcal{E}/\mathcal{E}[\mathfrak{a}]$.

Proposition 3.1.15. *The curve $\mathcal{E}/\mathcal{E}[\mathfrak{a}]$ has the same endomorphism ring as $\mathcal{E}$.*

With this knowledge, one can construct a map giving the action of the set of ideals of $\mathcal{O}$ (called $\mathbb{I}$ here) on the set of Elliptic curves over $\mathbb{F}_p$ with endomorphism ring $\mathcal{O}$ up to $\mathbb{F}_p$-isomorphism (called $Ell_p(\mathcal{O})$ here).

$$\begin{array}{rccc} : & \mathbb{I} \times Ell_p(\mathcal{O}) & \longrightarrow & X(\mathcal{O}) \\ & (\mathfrak{a}, \mathcal{E}) & \longmapsto & \mathcal{E}/\mathcal{E}[\mathfrak{a}] \end{array}$$

This is not a group action because $\mathbb{I}$ is a monoid and not a group, but it appears that we can transform it into a group action.

Proposition 3.1.16. *The action of any principal ideals (x) is trivial i.e., $\mathcal{E}/\mathcal{E}[(x)]$ is isomorphic to $\mathcal{E}$.*

The quotient of $\mathbb{I}$ by the set $\mathbb{P}$ of principal ideals is a well known number theoretic object called the **Class group** of $\mathcal{O}$ and denoted $Cl(\mathcal{O})$.

Theorem 3.1.17. *[CLM+18] The quotient action of $Cl(\mathcal{O})$ on $Ell_p(\mathcal{O})$ is faithful and transitive. Hence $(Cl(\mathcal{O}), X(\mathcal{O}))$ is a principal homogenous space.*

3.2 The CSIDH Protocol

Now that the action of $Cl(\mathcal{O})$ on $Ell_p(\mathcal{O})$ has been made explicit, an efficient algorithm to compute this action is needed. The first thing to do is to compute the quotient curve by a given subgroup.

Proposition 3.2.1. *If $\mathcal{E}$ is an elliptic curve over a finite field $\mathbb{F}$, and H is a finite subgroup of $\mathcal{E}(\mathbb{F})$, then one can compute the morphism ϕ_H and the equation of the curve $\mathcal{E}/H$ in $O(\#H)$ operations.*

Proof. See Appendix B for the algorithm. □

The idea of **CSIDH** is indeed to take primes and curves that make those computation easy. The choice has been made to take $p = 4\prod_{i=1}^{m} l_i - 1$ with $(l_i)_i$ small odd primes, and supersingular elliptic curves with trace of Frobenius equal to 0, because their number of points is equal to $p+1 = 4\prod_i l_i$. The elements of the class group that are to be considered are those represented by the ideals $\mathfrak{l}_i = (l_i, \pi_p - 1)$. The action of such an ideal is then computed with Algorithm 3.1. Note that the action of $\mathfrak{l}_i^{-1}$ can be computed in a similar way (taking advantage of the quadratic twist and the representation of the curve), but it is beyond the scope of this report.

Algorithm 3.1 Compute $[\mathfrak{l}_i] \cdot \mathcal{E}$ on input $\mathcal{E}$, l_i

$P \xleftarrow{\$} \mathcal{E}(\mathbb{F}_p)$
while $(4\prod_{j\neq i} l_j)P = \infty$ **do**
  $P \xleftarrow{\$} \mathcal{E}(\mathbb{F}_p)$
$Q \leftarrow 4(\prod_{j\neq i} l_j)P$ ▷ Q is of order l_i
$H \leftarrow \langle Q \rangle$ be the group generated by Q
$\mathcal{E}' \leftarrow \mathcal{E}/H$ be the curve computed from Velu formula
return $\mathcal{E}'$

The private key is then a tuple $(e_i) \in \mathbb{Z}^m$, which will represent the ideal $\prod_{i=1}^{m} \mathfrak{l}_i^{e_i}$. The action of the class group element represented by (e_i) is then computed by Algorithm 3.2.

Algorithm 3.2 Compute $\prod_{i=1}^{m} \mathfrak{l}_i^{e_i} \cdot \mathcal{E}$ on input $\mathcal{E}$, (e_i)

$\mathcal{E}' \leftarrow \mathcal{E}$
for $i = 1$ to m **do**
  for $j = 1$ to e_i **do**
    $\mathcal{E}' \leftarrow [\mathfrak{l}_i] \cdot \mathcal{E}$
return $\mathcal{E}'$

Remark 3.2.2. *Algorithm 3.2 is neither optimized nor constant-time (meaning that its execution time depends on the secret key, so it is vulnerable to side-channel attack). A more optimized version is in Appendix B, and a constant time version is a current research topic.*

The whole key exchange is then the following. $\mathcal{E}_0$ is a public supersingular elliptic curve, $p = 4\prod_{i=1}^{m} l_i - 1$ and $k \in \mathbb{Z}_{\geq 0}$ are public parameters too.

3.3 CSIDH viewed as an instance of HHS

It is clear that CSIDH is a HHS instance, but one could argue that in that sense that not the action of all elements of the class group that are computed, but only the action of the $\mathfrak{l}_i$. This is true and it comes from the fact that no efficient algorithm is known for computing the action of an arbitrary

Alice		Bob
$(a_i) \xleftarrow{\$} [0,k]^m$		$(b_i) \xleftarrow{\$} [0,k]^m$
$\mathcal{E}_a \leftarrow [\prod_{i=1}^m \mathfrak{l}_i^{a_i}] \cdot \mathcal{E}_0$		$\mathcal{E}_b \leftarrow [\prod_{i=1}^m \mathfrak{l}_i^{b_i}] \cdot \mathcal{E}_0$
	$\xrightarrow{\mathcal{E}_a}$	
	$\xleftarrow{\mathcal{E}_b}$	
$\mathcal{E}_{ab} \leftarrow [\prod_{i=1}^m \mathfrak{l}_i^{a_i}] \cdot \mathcal{E}_b$		$\mathcal{E}_{ab} \leftarrow [\prod_{i=1}^m \mathfrak{l}_i^{b_i}] \cdot \mathcal{E}_a$

Figure 11: The Commutative Supersingular Diffie–Hellman key exchange

element of the class group on $Ell_p(\mathcal{O})$. Indeed, the complexity of computing the action of a given isogeny is exponential in the size of its kernel which is equal to the norm of the ideal representing it in $Cl(\mathcal{O})$. In order to keep a reasonable amount of computation, one can either use only elements smooth in the $\mathfrak{l}_i$ or to smooth any given element with respect to the $\mathfrak{l}_i$ with lattice-based techniques such as LLL.

It should be noticed too that in almost all of the reduction presented in Section 2 informations about the group structure (e.g., its size or a multiple of it) are used. This is the second difference between the abstract HHS model and CSIDH: the structure of the class group is an information one does not have in the general case. There are algorithms to compute the structure of the class group of quadratic orders, the best known has (under GRH) heuristic complexity $L_p(1/3)$ (see [Bia12]). It is based on **sieving**: finding relations betweens elements choosen in a smart way. For the parameters of the original **CSIDH** paper [CLM+18], the class group order and structure were computed during my internship and presented in [BKV19]. Those parameters were the prime $p = 4\prod_{i=1}^{74} l_i - 1$, with $l_1, \ldots, l_{73}$ the first 73 odd primes and $l_{74} = 587$ the smallest prime making the product prime. The size of p was then around 512 bits, making the classical attacks (such as birthday paradox method) impratical. The class group was discovered to be cyclic of order $N = 37 \cdot 1407181 \cdot 51593604295295867744293584889 \cdot 31599414504681995853008278745587832204909$.

The structure of the class group should then be considered as a "secret but computable" information for the protocol, and it is interesting to note that (at least in this setting) all the reduction for cyclic group presented in Section 2 applies.

3.4 Difference between SIDH and CSIDH

As mentioned in the state of the art, another protocol using supersingular elliptic curve is currently under study. This protocol is called **SIDH** [JDF11] and, even if it seems similar with **CSIDH** (even in its name), it has a lot of differences and, in particular, does not fit into the **HHS** modelization.

The idea of **SIDH** is to take primes of the for $p = 2^a 3^b f - 1$ (with f a small cofactor) and a public supersingular curve $\mathcal{E}_0$ with trace of its Frobenius equal to 0 over $\mathbb{F}_{p^2}$. It can be shown that the 2^a (resp. the 3^b-torsion) of $\mathcal{E}(\mathbb{F}_{p^2})$, denoted $\mathcal{E}[2^a]$ (resp. $\mathcal{E}[3^a]$) is of order 2^a (resp. 3^b), is generated by two points. Those point are fixed and publicly known, denoted P_a, Q_a (resp. P_b, Q_b). The key exchange is then explicited in Figure 12. At the end of it, $\mathcal{E}_{ab} = \mathcal{E}_{ba}$ is the shared secret.

Even if the basic aspects of the key sharing are similar in **SIDH** and **CSIDH** (the shared secret is a curve obtained by the application of two isogenies), these protocol are deeply differents. First, the amount of information given in SIDH is far greater than in CSIDH: the whole torsion basis of the curves is transmited throught the network in SIDH. The second difference is that no group is acting on the curves in SIDH. TODO: insert a thing about walking on the graph of curves.

Alice		Bob
$m_a, n_a \xleftarrow{\$} \mathbb{Z}/2^a\mathbb{Z}$		$m_b, n_b \xleftarrow{\$} \mathbb{Z}/3^b\mathbb{Z}$
not both divisible by 2		not both divisible by 3
$S_a = [m_a]P_a + [n_a]Q_a$		$S_b = [m_b]P_b + [n_b]Q_b$
Compute $\mathcal{E}_a = \mathcal{E}/\langle S_a \rangle$		Compute $\mathcal{E}_b = \mathcal{E}/\langle S_b \rangle$
along with $\phi_a : \mathcal{E} \to \mathcal{E}_a$		along with $\phi_b : \mathcal{E} \to \mathcal{E}_b$
	$\xrightarrow{\mathcal{E}_a, \phi_a(P_b), \phi_a(Q_b)}$	
	$\xleftarrow{\mathcal{E}_b, \phi_b(P_a), \phi_b(Q_a)}$	
Let $R_a = [m_a]\phi_b(P_a) + [n_a]\phi_b(Q_a)$		Let $R_b = [m_b]\phi_a(P_b) + [n_b]\phi_a(Q_b)$
Compute $\mathcal{E}_{ab} = \mathcal{E}/\langle R_a \rangle$		Compute $\mathcal{E}_{ba} = \mathcal{E}/\langle R_b \rangle$

Figure 12: The Commutative Supersingular Diffie–Hellman key exchange

3.5 Attacks on CSIDH

The more evident classical and quantum attacks on CSIDH are presented in its founding paper [CLM+18]. The first class of attack is the "meet in the middle". The idea is to walk in the graph $Ell_p(\mathcal{O})$ randomly waiting to find a collision by the birthday paradox. This attack has complexity $O(|Cl(\mathcal{O})|^{1/2})$. As $Cl(\mathcal{O})$ is of size roughly $\sqrt{p}$, this attack has exponential complexity. For the quantum security, an attack based on Grover search and claw finding exists in $O(\sqrt[6]{p})$ is proposed too but recent work by Schanck shows that once memory management is taken into account, this complexity jumps from $O(\sqrt[6]{p})$ to $O(\sqrt[3]{p})$ (see [JS19]).

Another more problematic attack is the attack by the **abelian hidden-shift problem** which applies to CSIDH (in fact it would apply to any HHS instance), and uses $L(1/2)$ calls to the quantum oracle and the same amount of quantum memory. This attack is subexponential, but not that problematic since the amount of qubits needed to implement it is clearly a problem for now. In order to give a compairison, subexponential attacks on the discrete logarithm for finite fields with this families of complexity exists but the cryptosystem based on them are still used today.

I explored another way of attacking CSIDH during the internship. In CSIDH, the elliptic curves are considered in Montgomery form ($y^2 = x^3 + Ax^2 + x$), and are represented by their Montgomery coefficient (A). Not all curve can be written in that way, but it is the case for supersingular curves for the values of p considered in CSIDH. For further details on Montgomery curves and their arithmetic, see [Mon87] and [CS17]. One has at her disposition an involution on $Ell_p(\mathcal{O})$: the **quadratic twist**. This involution is very efficiently computable, since the Montgomery coefficient of the quadratic twist of the curve of coefficient A is $-A$. The interest of this involution is summed up in Proposition 3.5.1.

Proposition 3.5.1. *If $E = [\mathfrak{a}]E'$, and if the quadratic twist is denoted by $\tilde{\cdot}$, then $\tilde{E} = \mathfrak{a}^{-1}\tilde{E}'$.*

Remark 3.5.2. *If the curve defined by the equation $y^2 = x^3 + x$ is denoted $\mathcal{E}_0$, then $\mathcal{E}_0 = \tilde{\mathcal{E}_0}$.*

This involution can then be used to solve k-**Inverse-Approx-HHS**.

Proposition 3.5.3. *k-**Inverse-Approx-HHS** is poly-time in the CSIDH setting.*

Proof. I will do the proof only in the case of 2-**Inverse-Approx-HHS** since its generalization is straightforward.

Let $(E, E_0 = [\mathfrak{a}] \cdot E, E_1 = [\mathfrak{b}] \cdot E)$, then for any $\mathfrak{c} \in Cl(\sqrt{-p})$, $([\mathfrak{c}] \cdot \tilde{E}, [\mathfrak{c}] \cdot \tilde{E_0}, [\mathfrak{c}] \cdot \tilde{E_1})$ is a valid 2-**Inverse-Approx-HHS** solution by Prop 3.5.1. □

Corollary 3.5.4. *In the CSIDH setting, the **OT** scheme of figure 13 is broken for curious Bob.*

A single class of problem is then broken for CSIDH. In order to solve **CDH-HHS**, one would need to solve **Inv-HHS** and then apply the reduction algorithms presented in Section 2. I propose a new problem to be studied in order to work in this direction.

Definition 3.5.5. ***Group-twist***

Given the CSIDH parameters (p, the l_i, the structure of $Cl(\mathcal{O})$) and a curve $\mathcal{E} \in Ell_p(\mathcal{O})$, compute the unique $\mathfrak{a} \in Cl(\mathcal{O})$ such that $\tilde{\mathcal{E}} = [\mathfrak{a}] \cdot \mathcal{E}$.

If one can solve this problem, then I claim that she can solve **Inv-HHS** on CSIDH and then **CDH-HHS**.

Proposition 3.5.6. *For CSIDH, **Inv-HHS** reduces to **Group-twist**.*

Proof. Let $\mathcal{E}, \mathcal{E}' = [\mathfrak{a}] \cdot \mathcal{E}$ be a **Inv-HHS** instance and GT be a **Group-twist** oracle.

Let $\mathfrak{p} = GT(\mathcal{E})$. Then the curve $[\mathfrak{p}^{-1}] \cdot \tilde{\mathcal{E}'}$ is the solution to **Inv-HHS** for this instance: one has $\tilde{\mathcal{E}'} = [\mathfrak{a}^{-1}] \cdot \tilde{\mathcal{E}}$, then

$$[\mathfrak{p}^{-1}] \cdot \tilde{\mathcal{E}'} = [\mathfrak{a}^{-1}] \cdot [\mathfrak{p}^{-1}] \cdot \tilde{\mathcal{E}'} = [\mathfrak{a}^{-1}] \cdot \mathcal{E}$$

□

Solving the **Group-twist** problem could be a subject of further research.

4 Conclusion

4.1 Summary of the results

In this report, I presented the work done during the internship. I stated definitions and problems in the setting of Hard Homogenous Spaces, then I precised their relative difficulties. I showed that these relations depends on the structure of the group. Then I stated other reductions, trying to investigate the relations between **Scalar-HHS** and **CDH-HHS**. After this step I presented the **CSIDH** key exchange. I briefly introduced the theory of elliptic curves and compared **CSIDH** to the HHS model and to **SIDH** by explaining their common points and differences. Finaly I presented attacks on **CSIDH** and proposed a new way to attack it.

I produced a Rust implementation of the various elliptic curves algorithm presented in the report, some of them are presented in Appendix B.

4.2 Acknowledgements

I would like to thank Ben Smith for his help and flexibility during the whole duration of the internship. Thank also to all the GRACE team for their friendly welcome and their help when Ben could'nt be there.

I also want to thank Danaé (for being Danaé), all the team of Prologin and le Foyer de Paris for their help to escape the loneliness of being in a city far from home.

References

[BDZ03] Feng Bao, Robert H Deng, and Huafei Zhu. Variations of Diffie–Hellman problem. In *International conference on information and communications security*, pages 301–312. Springer, 2003.

[Bia12] Jean-François Biasse. Improvements in the computation of ideal class groups of imaginary quadratic number fields. *arXiv preprint arXiv:1204.1300*, 2012.

[BKV19] Ward Beullens, Thorsten Kleinjung, and Frederik Vercauteren. CSI-FiSh: Efficient isogeny based signatures through class group computations. Cryptology ePrint Archive, Report 2019/498, 2019. `https://eprint.iacr.org/2019/498`.

[CLM+18] Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. CSIDH: an efficient post-quantum commutative group action. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 395–427. Springer, 2018.

[Cou06] Jean-Marc Couveignes. Hard homogeneous spaces. *IACR Cryptology ePrint Archive*, 2006:291, 2006.

[CS17] Craig Costello and Benjamin Smith. Montgomery curves and their arithmetic: The case of large characteristic fields. *IACR Cryptology ePrint Archive*, 2017:212, 2017.

[De 17] Luca De Feo. Mathematics of isogeny based cryptography. *CoRR*, abs/1711.04062, 2017.

[DH76] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE transactions on Information Theory*, 22(6):644–654, 1976.

[GM16] Aurore Guillevic and François Morain. Discrete logarithms, 2016.

[GPSV18] Steven Galbraith, Lorenz Panny, Benjamin Smith, and Frederik Vercauteren. Quantum equivalence of the DLP and CDHP for group actions. *arXiv preprint arXiv:1812.09116*, 2018.

[JDF11] David Jao and Luca De Feo. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In *International Workshop on Post-Quantum Cryptography*, pages 19–34. Springer, 2011.

[JS19] Samuel Jaques and John M. Schanck. Quantum cryptanalysis in the ram model: Claw-finding attacks on SIKE. Cryptology ePrint Archive, Report 2019/103, 2019. `https://eprint.iacr.org/2019/103`.

[Kit95] A. Yu. Kitaev. Quantum measurements and the abelian stabilizer problem. *arXiv preprint quant-ph/9511026*, 1995.

[Mon87] Peter L Montgomery. Speeding the pollard and elliptic curve methods of factorization. *Mathematics of computation*, 48(177):243–264, 1987.

[Sho94] Peter W Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *Proceedings 35th annual symposium on foundations of computer science*, pages 124–134. Ieee, 1994.

[SIK] SIKE website. `https://sike.org/`.

[Sil09] Joseph H Silverman. *The arithmetic of elliptic curves*, volume 106. Springer Science & Business Media, 2009.

[Vit19] Vanessa Vitse. Simple oblivious transfer protocols compatible with kummer and supersingular isogenies. 2019.

A Oblivious Transfer with HHS

The discrete log version of this protocol was presented in the preprint [Vit19], but its generalisation to the HHS setting is straightforward and I present it here for reference. In the **Oblivious Transfer** settings there are two parties, namely Alice and Bob. Alice has two secrets and want to share only one of them with Bob, and Bob want to have access to one of those secrets, but do not want Alice to be able to tell which one of them. An analogy could be made with the medical records of patients. If Bob is a doctor, and Alice a server which has the records, Bob would want to check the record of a certain patient, but he wouldn't want the server to know which one.

The proposed way to implement such a protocol is the following (X, G is a **HHS**, and KDF is a **Key Derivative Function** for X, that is to say a function that will output a symetric cryptographic key with input an element of X).

Alice (has secret s_0, s_1)		Bob (wants secret k)
$\alpha_0, \alpha_1 \xleftarrow{\$} G$		$\beta \xleftarrow{\$} G$
$x \xleftarrow{\$} X$		
$x_0 \leftarrow [\alpha_0] \cdot x$		
$x_1 \leftarrow [\alpha_1] \cdot x$		
	$\xrightarrow{x, x_0, x_1}$	
		$y \leftarrow [\beta] \cdot x$
		$x' \leftarrow [\beta] \cdot x_k$
	$\xleftarrow{x'}$	
$c_0 \leftarrow Enc(s_0, KDF([-\alpha_0] \cdot x'))$		
$c_1 \leftarrow Enc(s_1, KDF([-\alpha_1] \cdot x'))$		
	$\xrightarrow{c_0, c_1}$	
		Retrieve $Enc^{-1}(c_k, KDF(y))$

Figure 13: OT protocol

B Velu's formulae and CSIDH algorithm

B.1 The formulae

The Velu's formulaes give an explicit isogeny and curves for the curve $\mathcal{E}/H$ is H is a subgroup of $\mathcal{E}(\mathbb{F}_q)$. For the recall, all the curve are given in **Montgomery form**: $y^2 = x^3 + Ax^2 + x$.

Theorem B.1.1. *If H is a subgroup of $\mathcal{E}(\mathbb{F}_q)$, then the function*

$$\phi : P = (x_p, y_p) \mapsto (x_p + \sum_{Q \in H} (x_{P+Q} - x_Q), y_p + \sum_{Q \in H} (y_{P+Q} - y_Q))$$

is an isogeny from $\mathcal{E} : y^2 = x^3 + Ax^2 + x$ to the curve $\mathcal{E}' : y^2 = x^3 + A'x^2 + x$, of kernel H. A' is given by the formula $A' = \eta \cdot (A - \sigma)$ for

$$\eta = \prod_{Q \in H \setminus 0} x_Q, \quad \sigma = \sum_{Q \in H \setminus 0} (x_Q - \frac{1}{x_Q})$$

B.2 Implementations and optimizations

As elliptic curve cryptography has been studied for a long time and is widely used, all sorts of practical improvements has been developped to compute quicker that kind of opérations. The more usual idea is to take **projective coordinates** instead of affine one, and to use those coordinates to reduce the number of divisions in finite fields (because divisions are the longest operations on finite fields). The idea is also to only use the x coordinate, because in the case studied here (the scalar multiplication of a point), all the operations can be computed with only the x-coordinate. The algorithm used to compute $[n]P$ is the Montgomery ladder, decribed in Algorithm B.1.

Algorithm B.1 Montgomery ladder compute $[n]P$ on input n, P

procedure XADD($[x_P : z_P], [x_Q : z_Q], [x_{P-Q} : z_{P-Q}]$)
  $u \leftarrow (x_P - z_P)(x_Q + z_Q)$
  $v \leftarrow (x_P + z_P)(x_Q - z_Q)$
  $x \leftarrow u + v$
  $z \leftarrow u - v$
  return $[x_{P-Q} \cdot x^2 : z_{P-Q} \cdot z^2]$
procedure XDOUBLE($[x : z], A$)
  $u \leftarrow (x + z)^2$
  $r \leftarrow (x - z)^2$
  $s \leftarrow u - r$
  return $[4 \cdot u \cdot r : s \cdot (4 \cdot r + s \cdot (a + 2))]$
procedure MONTGOMERY(n, P)
  if $n < 0$ **then**
    return Montgomery($-n, -P$)
  else if $n == 0$ **then**
    return $[1 : 0]$
  $Q \leftarrow [1 : 0]$
  Decompose $n = \sum_{i=0}^{b} a_i 2^i$
  for $i = b$ to 0 **do**
    if $a_i = 0$ **then**
      $P \leftarrow$ xADD(P, Q, P)
      $Q \leftarrow$ xDBL(Q)
    else
      $Q \leftarrow$ xADD(P, Q, P)
      $P \leftarrow$ xDBL(P)
  return Q

The overall algorithm used to compute the action of an ideal of the form $\mathfrak{l} = (l, \pi - 1)$ is then described in Algorithm B.2. A piece of my implementation in Rust can be found in Code 1 (for the Velu formulae) and in Code 2 for the class group action.

Algorithm B.2 Computation of the action of $(l, \pi - 1)$ on the curve and on the point Q

procedure ACTION($l, \mathcal{E} : y^2 = x^3 + Ax^2 + x, P$ of order l, Q)
  $T \leftarrow P$
  $\eta \leftarrow [1 : 0]$, $\sigma \leftarrow 0$
  $x_R \leftarrow 0$, $y_R \leftarrow 0$
  for $i = 1 \dots (l-1)/2$ **do** ▷ The fact that $x_P = x_{-P}$ is used
    $\eta_x \leftarrow \eta_x \cdot T.x$
    $\eta_z \leftarrow \eta_z \cdot T.z$
    $\sigma \leftarrow \sigma + (T.x^2 - T.z^2)/(T.x \cdot T.z)$
    $x_R \leftarrow x_R \cdot (T.x \cdot Q.x - T.z \cdot Q.z)$
    $z_R \leftarrow z_R \cdot (Q.x \cdot T.z - T.x \cdot Q.z)$
    if $i = 1$ **then**
      $T \leftarrow$ XDBL(T)
    else
      $T \leftarrow$ XADD(T, P, P)
  $\eta_x \leftarrow \eta_x^2$ ▷ Here the values are squared or doubled
  $\eta_z \leftarrow \eta_z^2$
  $x_R \leftarrow x_R^2$
  $z_R \leftarrow z_R^2$
  $\sigma \leftarrow 2\sigma$
  return The curve $y^2 = x^3 + \frac{\eta_x}{\eta_z}(A - 3 \cdot \sigma)x^2 + x$ along with the point $[Q.x \cdot x_R : Q.z \cdot z_R]$

[h!]

Source Code 1: Implementation of Velu formulaes

```
fn isogeny_velu(ell : &EllipticCurve<Fp<N, Integer>>,
point : &UnsignedProjPoint<Fp<N, Integer>>,
mut q : UnsignedProjPoint<Fp<N, Integer>>, k : Integer) ->
Result<(EllipticCurve<Fp<N, Integer>>, UnsignedProjPoint<Fp<N, Integer>>), ()>{
  assert!(ell.is_montgomery());
  assert!(k>=Integer::from(3));
  assert!(&k%2 != Integer::from(0));

  let p = point;

  let mut i = Integer::from(1);

  let mut t = p.clone();  // t = [i]p will iterate over elements of <p>
  let mut t_minus_1 = UnsignedProjPoint::infinite_point();

  let mut pi = UnsignedProjPoint::finite_point(Fp::from_int(1));
  let mut sigma = Fp::from_int(0);

  let mut projection_x = Fp::from_int(1);
  let mut projection_z = Fp::from_int(1);

  while &Integer::from(2)*&i < k{
```

```
    if t.x == Fp::from_int(0){ // point of order 2, we abort
      return Err(());
    }

    pi.x *= t.x.clone();
    pi.z *= t.z.clone();

    sigma += (&t.x*&t.x - &t.z*&t.z)/(&t.z*&t.x);

    projection_x *= &t.x*&q.x - &t.z*&q.z;
    projection_z *= &q.x*&t.z - &t.x*&q.z;

    if i == Integer::from(1){
      let _temp = t.clone();
      t = ell.x_dbl(p.clone());
      t_minus_1 = _temp;
    }else{
      let _temp = t.clone();
      t = ell.x_add(t, p.clone(), t_minus_1);
      t_minus_1 = _temp;
    }
    i += Integer::from(1);
  }

  pi.x *= pi.x.clone();
  pi.z *= pi.z.clone();

  projection_x *= projection_x.clone();
  projection_z *= projection_z.clone();

  sigma *= Fp::from_int(2);

  pi = pi.normalize();
  Ok((
  EllipticCurve::new_montgomery(pi.x*(&ell.a_2 - &(Fp::from_int(3)*sigma))),
  UnsignedProjPoint{
    x: q.x*projection_x,
    z: q.z*projection_z
  }
  ))

}
```

Source Code 2: Implementation of the class group action

```
pub fn class_group_action(self : &CSIDHInstance<Fp<N, Integer>>,
pk : Fp<N, Integer>, mut sk : Vec<i32>) -> Fp<N, Integer>{
    let L = &self.l;
```

```
let P = &self.p;
let N_PRIMES = &self.n_primes;
let mut finished_total : [bool; 2] = [false, false];

let mut k_sign : [Integer; 2]= [Integer::from(1),
    Integer::from(1)]; // 1 -> >= 0;  0-> <= 0
let mut s_sign : [Vec<Integer>; 2] = [vec!(), vec!()];
let mut e_sign : [Vec<i32>; 2] = [vec!(), vec!()];
let mut finished_sign : [Vec<bool>; 2] = [vec!(), vec!()];

for i in 0..sk.len(){
    if sk[i] == 0{
    }else if sk[i] > 0{
        e_sign[1].push(sk[i]);
        s_sign[1].push(L[i].clone());
        finished_sign[1].push(false);

        k_sign[1] *= L[i].clone();
    }else{
        e_sign[0].push(sk[i]);
        s_sign[0].push(L[i].clone());
        finished_sign[0].push(false);

        k_sign[0] *= L[i].clone();
    }
}

let mut ell = EllipticCurve::new_montgomery(pk);

while !finished_total[0] || !finished_total[1] {
    // Sample the point
    let x = Fp::new(Integer::sample_uniform(&Integer::from(0),
                    &(P-Integer::from(1))));

    let s = CSIDHInstance::compute_rhs(&x, &ell.a_2).legendre_symbol() as i32;
    if s == 0{
        continue;
    }
    let sign_index = ((s + 1)/2) as usize;

    if finished_total[sign_index]{
        continue;
    }

    finished_total[sign_index] = true;

    let uns_p = UnsignedProjPoint::finite_point(x);
```

```
        let mut k = k_sign[sign_index].clone();

        let mut q_point = ell.scalar_mult_unsigned(
            (P.clone()+Integer::from(1))/k.clone(), uns_p);

        for j in 0..s_sign[sign_index].len(){

            // Trick from the original implementation, they started by the big primes
            let i = s_sign[sign_index].len()-1-j;

            if finished_sign[sign_index][i]{
                continue;
            }
            finished_total[sign_index] = false;

            let li : Integer = s_sign[sign_index][i].clone();

            let r_point = ell.scalar_mult_unsigned(&k/(&li), q_point.clone());

            if &r_point == &UnsignedProjPoint::infinite_point(){
                // The point sampled had too low l-index
                continue;
            }

            let (ell_, q_point_) = match Self::isogeny_velu(&ell,
                    &r_point, q_point.clone(), li.clone()){
                Err(()) => {
                    // This should never happend
                    println!("Erreur");
                    continue;
                    },
                Ok(pair) => pair
            };

            ell = ell_;
            q_point = q_point_;

            e_sign[sign_index][i] -= s;

            if e_sign[sign_index][i] == 0{
                finished_sign[sign_index][i] = true;
            }

            k /= li;
        }
    }
    ell.a_2
}
```