Radboud University

Master Thesis Mathematics

Isogeny cryptography with twists and smooth primes

Author:
Maaike Heijdenrijk

Student number:
S4528301

Supervisosr:
Prof. Lejla Batina

Second reader:
Dr. Wieb Bosma

July 19, 2021

Abstract

In 2019, Craig Castello proposed a new approach to supersingular isogeny, named B-SIDH [16]. This is a version of SIDH where isogenies are computed over both the $p-1$ and $p+1$ torsion of $\mathbb{F}_{p^2}$. To make this cryptographic protocol work in real life, there need to be sufficiently large smooth primes. The search for these primes is currently very active, and in this thesis we suggest a new way of finding them. This can be found in Chapter 5. Furthermore we provide a full analysis of the active attack of Galbraith, Petit and Shani for B-SIDH [30] in Chapter 7, and discuss its consequences for extra security measures. In Chapter 6 we analyse the velusqrt algorithm for B-SIDH, including different optimal strategies for B-SIDH, which can be seen as an extension of the optimal strategies described in [2].

Acknowledgements

This thesis has been a real roller coaster to write. And that could never have happened without the help of many amazing people. First of all, Lejla my supervisor, thank you for sticking with me to the end and bringing me everywhere when that was still possible. Joost and Krijn, thank you for answering all my questions so fast and thoughtfull. Giacuomo for making the prime search really happening. Wouter Castryck and Cristophe Petit for helping with the mathematical challenges that came up. Wieb for saying yes everytime I needed help. And a big big thanks to my boyfriend and all my friends and family who dragged me through it all.

Contents

Notation list

$a\|b$	a divides b, for $a, b \in \mathbb{N}$
$\mathbb{A}^n(K)$	affine n-space over a field K
$\bar{K}$	algebraic closure of a field K
$\mathrm{Aut}(E)$	automorphism ring of E
char	characteristic of a field
C	curve
$\hat{\phi}$	dual of an isogeny
E	elliptic curve
E/K	elliptic curve defined over field K
$\mathrm{End}(E)$	endomorphism ring of E
$\mathbb{F}_p$	finite field of prime order p
$\mathbb{F}_q$	finite field with q elements
$\mathbb{F}_{p^k}$	finite field with p^k elements, with p prime
K	general field
R	general ring
g	genus of a curve
μ_q	group of all q^{th} roots of unity
ϕ	isogeny between two elliptic curves
$E[m]$	m-torsion group of E
R^*	multiplicative subgroup of a ring R
$\#E$	number of points on elliptic curve E
$\mathbb{Q}$	field of rational numbers
$R_1 - R_2$	ring R_1 that excluding the ring R_2
$\mathbb{Z}$	ring of integers
$\mathbb{Z}_n$	ring of integers modulo n
$\mathcal{O}$	point at infinity of an elliptic curve E
$\mathbb{P}^n(K)$	projective n-space over a field K
π	q-power Frobenius endomorphism $(x, y) \mapsto (x_q, y_q)$ over a field F_{q^k}
ζ_q	root of unity for a number q
$[m]P$	scalar multiplication of an point P on an elliptic curve by $m \in \|Z$
$E(K)$	set of K-rational points on E
t	trace of the Frobenius morphism
$\otimes$	tensor product
e_m	Weil pairing over torsion points $E[m]$

1. Background information

In this thesis we will first treat the background information on cryptography and quantum computers, that give us the context of why quantum cryptography is being developed. Then we will give all mathematical background needed to understand the elliptic curves computations that are performed in the supersingular isogeny cryptographic protocols. We then treat different elliptic curve based cryptographic protocols, and go on to the search of specific prime numbers, before we continue to the B-SIDH model and its security analysis. In this chapter we first mention the history and give some background of cryptography, and then we explain the most important topics on quantum computers.

1.1 Cryptography

In this section we treat the foundations of cryptography. Cryptography is a large field on protecting data using certain codes. One of the main aspects is the design of cryptographic protocols that prevent third parties from reading private messages. It has been used for millennia, and its development has taken a flight since the development of computers.

1.1.1 Historical overview

To get a sensitive message across a kingdom was a dangerous task, there would be spies and thugs throughout the way, and when could you be sure that the message received was the same as the message sent? You could not always trust the messenger to not read the message and deliver it safely. Therefore, already a long time ago people came up with ways to conceal or encrypt messages to keep them secret. A way was writing with invisible ink, that only becomes visible when held by candle light. A classical story tells that a Greek king used to shave the head of a slave, tattoo the message on his head, and waited until his hair was grown back to send the slave to deliver the message. These are good tactics when your opponent doesn't know the methods you use: security by secrecy. But of course only one maid needs to spill the story of the shaven head to the enemy, and this method can never be safely used again. Another example of this is the famous Ceasar cipher. You write your message, but shift each letter of each word for a certain amount of letters in the alphabet. For instance A = F, B = G. The letter now becomes unreadable nonsense, unless you know the way to transform the letters back. Over the centuries, more advanced methods became available, like using a specific sentence as a key, or permutation, etc. It is of course a cat and mouse game, with smart people on both sides trying to beat the others. These methods are called *security through obscurity*, as the only thing that keeps the message secure is the fact that the adversary does not know how to decode it. Already in the 19th century it was established that obscurity cannot be the only way of securing a message. One of the most famous more modern day examples of cryptography is the Enigma code the Germans used to encrypt their messages. Like the Titanic was deemed to be unsinkable, the Enigma code was deemed to be unbreakable. They used advanced methods of encrypting their messages, and changed keys every day to prevent the allies of using previously decoded messages to decode new ones. There was however a flaw in their setup. They ended their messages with the same text every day. Knowing this repetition that came back in all messages, it gave the English a huge

advantage of cracking this day's key, and therefore being able to deduce the secret messages they sent. This is seen as a large step to a fast allied victory of World War II.

1.1.2 Modern day cryptography

Nowadays cryptography is used to secure our online activities and other digital communications. Signing into bank accounts, communicating via a messenger app, all actions require a way of preventing an adversary to gain access to our data.

There are two different types of cryptography used in computers nowadays, public key and private key. If Alice and Bob want to have a private conversation, they will encrypt their messages to each other with a shared key, that only they know. This process is called symmetric key cryptography, and uses stream or block ciphers to encrypt the message. To decode the message the same process, with certain steps inverted, is used as to encode the message. The system that is currently the NIST-standard is AES, invented by Joan Daemen and Vincent Rijmen. [24]. The United States National Institute of Standards and Technology (NIST) is the authority in the field of establishing the standards for cryptographic protocols. They set the standards for what should be implemented in devices worldwide.

A message m is encrypted to a ciphercode c using an encryption function Enc, with as input a shared secret key K. The ciphercode is then decoded using the reversed protocol of the encryption function, the decoding function Dec with as input the same key K, to reveal the message m.

$$\mathrm{Enc}_K(m) = c, \quad \mathrm{Dec}_K(c) = m.$$

A key can be split into a *validation key* and a *session key*. The session key can be used to secure the channel using symmetric key cryptography, while the validation key is used during the secret key establishment to assure the receiver of the key that they are dealing with the correct person, and not an adversary.

A *Hash function* is a non-invertible function that sends a string of random length to a string of specific length n.

$$\{0,1\}^* \mapsto \{0,1\}^n$$

Hash functions are used everywhere to send and store information safely. For instance passwords are not stored themselves, but rather their hash function (with additional data). To check if a password is entered correctly, the password entered is hashed and compared to the stored hash function. A good hash function is random, meaning that the output does not depend on the input, and has a negligible probability of sending two strings to the same output.

An *oracle* is a "black box" that is able to solve a specific decision problem in a single operation. An oracle can either return a yes or no (or 1 or 0), or return the outcome of a function. A *random oracle* returns a specific randomised outcome for any input. One can make queries to the oracle, for a specific problem.

A *pseudo random function* (PRF) is a function that simulates a random oracle, in a way that no efficient algorithm would be able to distinguish between a PRF and a random oracle. The input of a PRF is calles a *random seed*. A *key derivation function* (KDF) is a function that derives a key from a secret value, such as a password or a master key, using a PRF. It can be used to put longer keys into a specific length key.

To be able to send messages using symmetric key cryptography, there needs to be an established secret key first. This is often done using public key cryptography. Public key cryptography is based on so called "trap-door" functions. These mathematical functions are easy to compute when you have access to a specific piece of information, and hard to compute if this piece of information is missing. The main idea is that Alice and Bob both will pick a secret key, that they use as input for this trap door function. The output of the function becomes their public key. These public keys can be exchanged through an insecure channel. Using the received public key, they compute their shared secret key using a cryptographic protocol. A cryptographic protocol is a set of rules designed to allow secure communication under a specific set of circumstances. In a public-key protocol Alice and Bob will only exchange information that the whole world is allowed to see -the

public keys- while both doing certain computations in order to end up with a shared secret key, that they do not have to communicate to each other.

Retrieving the key using only the public information is a mathematically hard problem, and when the key is large enough, this problem should be infeasible to solve. 128-bit security is seen as secure, meaning that it would take 2^{128} bit operations to find the secret shared key. The historically most used public key cryptography systems are RSA and Diffie-Hellman. The difficulty of RSA [47], is based on finding the prime factors p and q of a large number $N = pq$, nowadays Elliptic Curve Cryptography (ECC) is the NIST standard. RSA and standard Diffie-Hellman we describe below, ECC is treated in Chapter 4.

Diffie Hellman key exchange The Diffie Hellman key exchange is based on taking discrete logarithms of a number modulo a prime p. It was proposed by Ralph Merkle, Whitfield Diffie and Martin Hellman [27]. Assume Alice and Bob want to establish a shared secret key for their communications. They agree on a prime p and a number g coprime to p. Now Alice picks a secret integer a, and bob a secret integer b. Alice computes $g^a \mod p$, and Bob computes $g^b \mod p$. They exchange these new values, and repeat the process. Since taking powers is a commutative action, we get $g^{ab} \mod p = g^{ba} \mod p$ and they end up with the same key. It is a mathematically hard problem to retrieve the key not knowing a or b.

Different types of keys A *static key* is a key that is reused all the time. It can for instance be hardcoded into a device (like a smart fridge or cell phone). If this device sends a message, it will always be encrypted using the same key. If an attacker finds out the key, they can read all messages sent. On the other hand, there are *ephemeral keys*, that are only used for one specific instance. The one-time pad is an example of an ephemeral key, or a secret key chosen for a one-time key-exchange as described below. Static keys can also be used to generate ephemeral keys, where one would use public key cryptography to generate a static key, which is used as a master key to generate different ephemeral keys using private key cryptography. This is for instance used in secure communication channels, where a new ephemeral key is created for each session. It is cheaper than generating a new key using public key cryptography each time, and safer than reusing the same key over multiple sessions. There are also keys used in a more hybrid setting, where a static key is replaced with a new one after a certain period of time or amount of uses. This happens for instance with internet security certificates like SSL (the key lock in the address bar), where a trusted third party gives a certificate to a company assuring that communication with that website is secure, using a specific private key. These keys are then reset when the certificate is renewed.

1.2 Post quantum cryptography

As will be explained in the next section, it is not clear when -or if - the first full scale quantum computer will be built. But it is of utmost importance to have the possibility for quantum-safe communication as soon as possible. For instance, take diplomatically or military sensitive information that is supposed to remain a secret for a long time, that is being communicated to an overseas base at this moment. An adversary would for be able to store this data and decrypt this the moment they have access to a quantum computer. To be secure against quantum computers we need *post-quantum cryptography*: cryptographic systems that are secure against both quantum and classical computers.

To establish a new cryptographic protocol that can be used world wide, on every computer, one cannot simply take their preferred protocol and use it. NIST launched a competition in 2016. looking for the best post-quantum cryptographic protocol. In the first round, people from all over the world can then send in their created protocols. All information on these protocols is freely accessible, and researchers are encouraged to improve these protocols and look for potential weaknesses. In July 2020, the finalists of this competition were announced. One of the applications is SIKE, by (Jao et al.) [4]. It is a further developed version of the cryptographic scheme SIDH

that is reviewed in this thesis. SIKE is Up until the writing of this thesis, no attacks that break this scheme are known. SIKE has reached the final of the NIST competition as an Alternate Candidates in the category 'Public-key Encryption and Key-establishment Algorithms'. In the case of SIKE, this means that the protocol is seen as promising, but more research is needed before it should be implemented.

SIKE is a cryptographic protocol based on Supersingular Isogeny Diffie Hellman (SIDH). In this thesis we will look into SIDH, and mainly in a specific type of supersingular key exchange called B-SIDH. SIDH is explained in Chapter 4, B-SIDH in chapter 6.

1.3 Quantum computers

The main reference for this section is [55]. If not specified otherwise, this book is used as the main source. An undergraduate level of linear algebra and algebra is assumed.

Classical computers do computations using strings of *bits*. Bits are objects that, like a light switch, can be either in a state 1, or "on", or in a state 0, or "off". These states can be represented as vectors in the following way.

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Quantum computers do computations on another kind of bits, that can be in an infinite amount of states between 0 and 1. These bits are called *qubits*. A singe bit is represented as follows

$$|x\rangle = \begin{bmatrix} c_0 \\ c_1 \end{bmatrix},$$

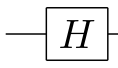
with c_0 and c_1 complex numbers such that $|c_0|^2 + |c_1|^2 = 1$. Since in our physical world complex numbers do not exist, to find the state of a specific bit we need to *measure* the bit. When a qubit is measured, it becomes an ordinary bit, $|0\rangle$ or $|1\rangle$, just like Schrödingers cat is both alive and dead until you open the box. The probability that the qubit after measurement will be found in state $|0\rangle$ is given by $|c_0|^2$, and the chance that it will be found in state $|1\rangle$ is given by $|c_1|^2$.

Just as on a classical computer, we can build logical gates on a quantum computer -these are called quantum logical gates. Classical logical gates cannot all be used in a quantum computer, since quantum logical gates have the requirement that they must be inversible. The classical logical gate AND is not inversible, as you cannot retrieve the input deterministically from the output. The most interesting quantum logical gates are described below.

The Hadamard gate The Hadamard gate brings a qubit in *superposition*. It makes a qubit "half $|0\rangle$ and half $|1\rangle$." It is represented by the following matrix

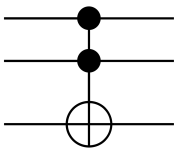
$$H = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix}$$

and is normally drawn in a circuit diagram as



.

The Toffoli gate The most common quantum logical gate is the Toffoli gate. It takes a qubit $|x, y, z\rangle$ and sends it to $|x, y, x \oplus y \oplus z\rangle$. x and y are called the control bits. In a circuit diagram it looks as follows:



1.3.1 Quantum algorithms

Security of cryptographic public key protocols relies on mathematically hard functions, as described in section 1.1 of this chapter. But the security for current protocols is only guaranteed for classical computers. In the next section we will discuss Grover's algorithm and Shor's algorithm, that can both be adapted to retrieve a private key significantly faster for all current public key protocols.

Grover's Algorithm In 1996 Lov Grover designed a quantum algorithm to speed up database searches [32]. It optimises the search for a specific, known, object in a database. Mathematically, this equation can be described as follows. Given a function $f : \{0,1\}^n \to \{0,1\}$ such that $f(\mathbf{x}) = \mathbf{1}$ if $\mathbf{x} = \mathbf{x_0}$ and 0 if $\mathbf{x} \neq \mathbf{x_0}$ for exactly one binary string $\mathbf{x_0}$, the goal is to find $\mathbf{x_0}$.

For classical computers, this would take at maximum 2^n evaluations of f, and on average $2^{n/2}$ evaluations. We will show that using Grover's algorithm it will take a quantum computer only $2\sqrt{n/2}$ evaluations on average. The algorithm

Algorithm 1 Grover's Algorithm

Data: A state $|\mathbf{0}\rangle$
Result: string $\mathbf{x_0}$ such that $f((\mathbf{x_0}) = \mathbf{1}$
Apply $H^{\otimes n}$ on $\mathbf{x_0}$
for *i in* $\sqrt{2^n}$ **do**
  Apply phase inversion operation $U_f(I \otimes H)$ on $\mathbf{x_0}$
  Apply inversion about mean operation $-I + 2A$ on $\mathbf{x_0}$
end
Measure $\mathbf{x_0}$

Here $\otimes$ is the tensor product of two matrices, and $H^{\otimes n}$ the Hadamard matrix tensored with itself n times.

U_f is the matrix that does the following:

$$|\mathbf{x}, y\rangle \mapsto \mathbf{x}, f(\mathbf{x} \oplus y\rangle \tag{1.1}$$

.

A is the matrix of size n with on every entry $\frac{1}{2^n}$. Multiplying a matrix $\mathbf{x_0}$ by (-I + 2A) will inverse the values about the average of the string $\mathbf{x_0}$. This is done to boost the separation of the phases.

As an example, consider the vector $[8, 8, 8, 8, 8]$. We now apply phase inversion operation on the fourth vector: $[8, 8, 8, -8, 8]$. Then calculating the inversion about the average gives $[1.6, 1.6, 1.6, 17.6, 1.6]$. The difference between the numbers is now 16. Applying the phase operation $U_f(I \otimes H)$ and inversion about the average again gives $[-6.08, -6.08, -6.08, 13.12, -.08]$. The difference between the fourth and the other elements is now -19.2. So we have separated the numbers further. Research shows that doing this $\sqrt{2^n}$ times is needed to find $\mathbf{x_0}$. Doing it more times will be too much.

While this algorithm speeds op searches a lot, it is no exponential improvement. Still, this algorithm is important because for many post-quantum cryptographic protocols, it is one of the best algorithms to get the private key knowing only the public keys.

The following algorithm by Shor is the one that actually breaks current public key cryptography.

Shor's Factoring algorithm A paper published by Shor, [50], describes an algorithm that can solve RSA and discrete logarithm problems in polynomial time instead of exponential time, essentially breaking public key cryptography. The algorithm described below can solve ordinary discrete logarithms, for adaptations of the algorithm that can solve Elliptic curve discrete logarithms, see [45]. For an adaptation to solve integer factorisation, see [50].

As described in section 1.1.2, assume we have numbers $g, y \in \mathbb{Z}$, with a prime p such that $g, y < p$. Given that $g^x = y \mod p$, we need to find x.

The idea of Shor's quantum algorithm is to find the period of a function f. In the case of discrete logarithms, the function f is described by:

$$f : \mathbb{Z}_p \times \mathbb{Z}_p \to \mathbb{Z}_p, \quad f(a, b) = g^a y^{-b}.$$

It is clear that the kernel of this function is generated by $(x, 1)$. Thus, if we find the kernel of f, we find our desired x.

To find the kernel of f, we will try to find its period. This uses the quantum computer property that evaluates f at all points simultaneously. A representation for this algorithm is to visualise two sine functions, one with period g and the other one with period y, and to look for a point where the combination of the two sine functions starts to repeat itself. This point will be (a multiple of) x.

Note that a quantum algorithm is always probabilistic, not deterministic. This means we can find x with great probability, but not always. Therefore the algorithm has to be repeatedly executed to find x with sufficient certainty.

Algorithm 2 Shor's Algorithm

Data: Three registers of length $p - 1$ of qubits in state $|\mathbf{0}\rangle$
Result: x such that $g^x = y \mod p$
Bring the first two registers of qubits in superposition using the Hadamard gate.
In the third register, compute $g^a y^b$.
Apply Quantum Fourier Transformation to both of the two first registers.
Measuring the first two registers should give a pair a', b' such that $x = b'(a')^{-1}$.

The first step is to bring the two registers of qubits into superposition:

$$\frac{1}{q} \sum_{a=0}^{p-1} \sum_{b=0}^{p-1} |a, b\rangle \tag{1.2}$$

.

Adding the third register gives

$$\frac{1}{q} \sum_{a=0}^{p-1} \sum_{b=0}^{p-1} |a, b, g^a y^b\rangle \tag{1.3}$$

,

in which the last part can also be written as $g^a y^b = g^a (g^x)^b = g^{a_0}$. The order of g is p, making this equal to $a + xb \equiv a_0 \mod p$. This gives that for each b there is exactly one solution. We find for the state of the first two registers:

$$\frac{1}{q} \sum_{b=0}^{q-1} |a_0 - xb, b\rangle \tag{1.4}$$

.

The next step is to apply quantum Fourier transformation. We will not describe in detail how this process works, for more explanation on this see [55].

The Fourier transform acts on base states as:

$$|z\rangle \mapsto \frac{1}{\sqrt{p}} \sum_{z'=0}^{p-1} \omega_p^{zz'} |z'\rangle \text{ with } \omega = e^{2\pi i/p}. \tag{1.5}$$

Combining 1.5 and

$$\frac{1}{\sqrt{p}}\frac{1}{p}\sum_{a',b'=0}^{p-1}\sum_{b=0}^{p-1}\omega_p^{(a_0-xb)a'}\omega_p^{bb'}|a',b'\rangle \tag{1.6}$$

.

The sum over b gives $p\omega_q^{a_0a'}$ if $b \equiv xb'(\mod p)$ and vanishes otherwise.
This gives

$$\frac{1}{\sqrt{p}}\sum_{a'=0}^{p-1}\omega_p^{a_0a'}|a',b' \equiv xa' \mod q\rangle \tag{1.7}$$

.

It is clear that the probability of measuring a basis state is independent of a_0. So irregardless of the state that is measured in the third register, we will obtain a pair a', b' that can give us d as long as $a' \neq 0$.

In this thesis we will not go explain on how to execute these algorithms using quantum logical gates. In [48] it is estimated that to break elliptic curve cryptography at 128-bit security, you would need at least 2330 qubits and $1.26 * 10^{11}$ Toffoli gates, and factoring 3072-bit RSA would require almost 6200 qubits and $1.86 * 10^{13}$ Toffoli gates.

1.3.2 Quantum computers today

One of the questions that remains is when to expect full scale quantum computers to work. As of July 2021, there are already exciting innovations happening in the field, with the outlook on more. IBM has published a roadmap that shows their intention of having a 1000-qubit computer by 2025. Currently their best computer has 127 qubits. As of 16-06-2021, in Germany a research institute, the Fraunhofer-Gesellschaft, officially bought the first commercial quantum computer from IBM [43]. It has 27 qubits and works as a full quantum computer. As described in the section above, they are not able to crack current cryptography with it, but it is a start. Google has already in 2019 published an article [5], where it shows quantum supremacy on a specific computation done in 200 seconds on a 53-qubit quantum computer, for which a classical computer would have needed around 10.000 years. Some other prospects are less promising, as for instance the Microsoft-backed research into qubits based on Majorana particles turned out to be unsuccesfull, the Majorana parts that were thought to be examined turned out to be just a statistical error. For now, quantum computers are nowhere near being able to break security like RSA. However, development is ongoing and progress is being made by many different companies. These examples show that large-scale quantum computers might be realised within the next decennium, making it extremely relevant to implement quantum cryptography as soon as possible.

2. Elliptic Curves and Isogenies

In this section we give a background on the mathematics needed to formally define an elliptic curve, and the basic properties of an elliptic curve and some extra theorems needed later on in the thesis. An undergraduate mathematics level of the reader is assumed, specifically on fields and number theory. If not mentioned otherwise, the source is [51].

Definition 2.0.1. An *Affine n-space* over a field K is the set of $n-$tuples

$$\mathbb{A}^n = \mathbb{A}^n(\bar{K}) = \{P = (x_1, ..., x_n) : x_i \in \bar{K}\}.$$

where $\bar{K}$ is the closure of K. The set of $K-$rational points of $\mathbb{A}^n$ is obtained in a similar way

$$\mathbb{A}^n = \mathbb{A}^n(K) = \{P = (x_1, ..., x_n) : x_i \in K\}.$$

Definition 2.0.2. Let $\bar{K}[X] = \bar{K}[X_1...X_n]$ be a polynomial ring in n variables, and $I \subset \bar{K}[X]$ an ideal. We associate a subset of $\mathbb{A}^n$ to I:

$$V_I = \{P \in \mathbb{A}^n : f(P) = 0 \ \forall \ P \in V\}.$$

An *affine algebraic set* is any set of the form V_I.
For all algebraic sets V, the ideal of V is given by

$$I(V) = \{f \in \bar{K}[X] : f(P) = 0, \ \forall \ P \in V\}.$$

V is called an *algebraic variety* if $I(V)$ is a prime ideal in $\bar{K}[X]$.

Definition 2.0.3. *Projective n-space* over K is defined as $\mathbb{P}^n = \mathbb{P}^n(\bar{K})$, the set of $n+1$ tuples

$$P = (x_0, ...x_n) \in \mathbb{A}^{n+1},$$

such that not all x_i are zero, modulo the equivalence relation $\sim$:

$$(x_0, ...x_n) \sim (y_0, ..., y_n) = Q.$$

if there is a $\lambda \in \bar{K}$ such that $P = \lambda Q$. An equivalence class $\{(\lambda x_1, ..., \lambda x_n) : \lambda \in \bar{K}\}$ is denoted $[x_1 : ... : x_n]$. The points x_i are called *homogeneous coordinates.*

Definition 2.0.4. A polynomial f is *homogeneous of degree d* if

$$f(\lambda X_0, ..., \lambda X_n) = \lambda^d f(X_0, .., X_n), \ \forall \ \lambda \in \bar{K}.$$

An ideal $I \in \bar{K}[X]$ is homogeneous if it is generated by homogeneous polynomials.

Analogous to the affine definition, we can now define a projective algebraic variety.

Definition 2.0.5. Associate a subset of $\mathbb{P}^n$ to a homogeneous ideal I as follows:

$$V_i = \{P \in \mathbb{P} : f(P) = 0, \ \forall \text{ homogeneous } F \in I\}.$$

A projective algebraic set is any set of the form V_I with I homogeneous.

We denote the homogeneous ideal of V by

$$I(V) = \{f \in \bar{K}[X] : f \text{ homogeneous and } f(P) = 0,\ \forall\ P \in V\}$$

A *projective variety* is a projective set of which its homogeneous ideal $I(V)$ is a prime ideal in $\bar{K}[X]$.

Definition 2.0.6. Define the *affine coordinate ring* of a variety V defined over K (Denoted V/K) as

$$K[V] = \frac{K[X]}{I(V)}$$

The local ring of V at P, denoted $\bar{K}[V]_P$ is given by

$$\bar{K}[V]_P = \{F \in \bar{K}(V) : F = f/g \text{ for some } f, g \in \bar{K}[V] \quad \text{with} \quad g(P) \neq 0.\}.$$

The functions in $\bar{K}[V]_P$ are called *regular at* P, and just *regular* if they are regular at all P.

Definition 2.0.7. If V is a projective variety, and $P \in V$ such that $P \in \mathbb{A}^n \subset \mathbb{P}^n$ is *regular* at P if it is in the local ring of $V \cap \mathbb{A}^n$ at P.

Definition 2.0.8. An affine variety V is *nonsingular* at $P \in V$, if the matrix

$$\left(\frac{\partial f_i}{\partial X_j}(P)\right)_{1 \leq i \leq m,\ 1 \leq j \leq n}$$

has rank $n - \dim(V)$, with $f_1, ..., f_m$ a set of generators for $K[X]$. The dimension of V is the transcendence degree of $\bar{K}(V)$ over $\bar{K}$. A projective variety is nonsingular at P if $V \cap \mathbb{A}^n$ is nonsingular at P. If a function is not nonsingular at P, it is called singular at P.

Definition 2.0.9. Let $V_1, V_2 \subset \mathbb{P}$ be projective varieties. A *rational map* from V_1 to V_2 is a map of the form

$$\phi : V_1 \longrightarrow V_2 \quad \phi = [f_0, ..., f_n],$$

where all $f_i \in \bar{K}(V_1)$ have the property that for all $P \in V_1$ that they are defined, and that

$$\phi(P) = [f_0(P), ..., f_n(P)] \in V_2.$$

ϕ is called *regular* at $P \in V_1$ if there is a function $g \in \bar{K}(V_1)$ such that

1. each $g \cdot f_i$ is regular at P;
2. $g \cdot f_i(P) \neq 0$ for some i.

If such a g exists, we write

$$\phi(P) = [(gf_0)(P), ..., gf_n)(P)].$$

A rational map that is regular at every point is called a *morphism.*

We define two varieties V_1, V_2 to be *isomorphic*, written $V_1 \cong V_2$, if there are morphisms $\phi : V_1 \to V_2$ and $\psi : V_2 \to V_1$ such that $\phi \circ \psi$ and $\psi \circ \psi$ are the identity maps on V_1 and V_2.

2.1 Curves

Definition 2.1.1. A *curve* is an projective variety of dimension 1.

Theorem 2.1.2. Let $\phi : C_1 \to C_2$ be a morphism of curves. Then ϕ is either constant or surjective.

Let C_1 and C_2 be curves defined over K, with $\phi : C_1 \to C_2$ a nonconstant rational map defined over K. Composition with ϕ induces an injection of function fields,

$$\phi^* : K(C_2) \longrightarrow K(C_1), \quad \phi^* f = f \circ \phi.$$

Definition 2.1.3. Let ϕ be a map of curves defined over K. If ϕ is constant, we define the *degree* of ϕ to be 0. Else, we define its degree to be

$$\deg \phi = [K(C_1) : \phi^* K(C_2)].$$

This is always finite, see [33] II.6.8. We call ϕ separable, inseparable or purely inseparable if the field extension $K(c_1)/\phi^* K(C_2)$ has that property.

For C a curve and $P \in C$, We can define a valuation on the local ring of C at P $\bar{K}[C]_p$ as follows

$$\mathrm{ord}_P : \bar{K}[C]_p \longrightarrow \mathbb{N} \cup \infty \, \mathrm{ord}_P(f) = \sup\{d \in \mathbb{Z} : f \in M_P^d\}.$$

Here M_P^d is a maximal ideal of $\bar{K}[V]$ given by

$$M_P = \{f \in \bar{K}[V] : f(P) = 0\}.$$

Defining $\mathrm{ord}_P(f/g) = \mathrm{ord}_P(f) - \mathrm{ord}_P(g)$, we can extend ord_P to $\bar{K}(C)$:

$$\mathrm{ord}_P : \bar{K}(C) \longrightarrow \mathbb{Z} \cup \infty.$$

A *uniformizer for C at P* is any function $t \in \bar{K}(C)$ with $\mathrm{ord}_P(t) = 1$

The *divisor group* of a curve C is the free abelian group generated by the points of C. A divisor is a sum

$$D = \sum_{P \in C} n_P(P), \quad n_p \in \mathbb{Z}.$$

We have tat n_P is zero for almost all P. The *degree of D* is defined by

$$\deg D = \sum_{P \in C} n_P.$$

Let C be smooth, and let a function $f \in \bar{K}(C)^*$, then the divisor of f is given by

$$\div(f) = \sum_{P \in C} \mathrm{ord}_P(f)(P).$$

Definition 2.1.4. For a curve C, the *space of differential forms* on C, denoted Ω_C, is the $\bar{K}-$ vector space generated by symbols of the form dx, with $x \in \bar{K}(C)$, with the relations $d(x+y) = dx + dy$, $\quad d(xy) = xdy + ydx$ and $da = 0$, for all $x, y \in \bar{K}(C), a \in \bar{K}$.

For an element $\omega \in \Omega_C$ we can associate a divisor to ω

$$\div(\omega) = \mathrm{ord}_P(\omega)(P) \in \mathrm{Div}(C).$$

Definition 2.1.5. A *canonical divisor class* of a curve C is the image of $\mathrm{div}\, \omega$ in the Picard group $\mathrm{Pic}(C)$. Any divisor in this divisor class is called a *canonical divisor*

A divisor is *positive*, denoted $D \geq 0$, if $n_P \geq 0$ for every $P \in C$. Similarly, for divisors D_1, D_2, we write $D1 \geq D_2$ to say $D_1 - D_2$ is positive.

For a divisor D *in* $\mathrm{Div}(C)$ we can associate to it a set of functions

$$Ł(D) = \{f \in \bar{K}(C)^* : \mathrm{div}(f) \geq -D\} \cup 0.$$

Its dimension is given by

$$\ell(D) = \dim_{\bar{K}} Ł(D).$$

Theorem 2.1.6. [Riemann-Roch] Let C be a smooth curve and let K_C be a canonical divisor on C. Then there is an integer $g \geq 0$, such that

$$\ell(D) - \ell(K_c - D) = \deg D - g + 1$$

for all $D \in \mathrm{Div}(C)$.

The integer g is called the *genus* of the curve C.

2.2 Elliptic curves

Definition 2.2.1. An *elliptic curve* is a pair $(E, \mathcal{O})$ with E a curve of genus 1 and $\mathcal{O} \in E$ a fixed point. Usually it is shortened by writing just E and assuming $\mathcal{O}$.

All elliptic curves can be written as triples (X, Y, Z) that are solutions to the following cubic equation in $\mathbb{P}^2$,

$$Y^2Z + a_1XYZ + a_3YZ^2 = X^3 + a_2X^2Z + a_4XZ^2 + a_6Z^3. \tag{2.1}$$

This equation, and all following, are called *Weierstrass equations*. Using the substitution $x = X/Z$ and $y = Y/Z$ we can rewrite this equation to

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6. \tag{2.2}$$

together with the base point $\mathcal{O} = [0, 1, 0]$ at infinity. if the constants a_i lay in a field K, we say E is defined over K, denoted as E/K.

To an elliptic curve we can assign certain invariants. the *discriminant* $\Delta = -b_2^2b_8 - 8b_4^3 - 27b_6^2 + 9b_2b_4b_6$, where $b_2 = a_1^2 + 4a_4, b_4 = 2a_4 + a_1a_3, b_6 = a_3^2 + 4a_6$ and $b_8 = a_1^2a_6 + 4a_2a_6 - a_1a_3a_4 + a_2a_3^2 - a_4^2$, and the *j-invariant* $j = \frac{b_2^2 - 24b_4}{\Delta}$.

In [51] III.1.4 it is proven that two elliptic curves are isomorphic over $\bar{K}$ if and only if they have the same j-invariant. On the other hand, for all points $j_0 \in \bar{K}$ there is an elliptic curve defined over $K(j_0)$ with a j-invariant equal to j_0.

In this section we assume E is an elliptic curve over a field K, specified by a Weierstrass equation. Points P on E are given by affine coordinates (x, y), together with the point at infinity $\mathcal{O}$. Since the equation is of degree 3, any line through a point P has two more intersection points on E, Q and R. P, Q and R do not have to be distinct, as a line can be tangent to the curve.

We define a group law $\oplus$ on E as follows

Let L be the line through P and Q, or the line tangent to P (if $Q = P$). This line intersects E in one more point R. A second line L' runs through R, the point at infinity $\mathcal{O}$ and a third point. We call this point $P \oplus Q$. The group law is explained schematically in 2.1.

This is indeed well defined, and can be seen as group addition, as justified in [51] III 2.2.

We define $E(K)$ as the subgroup of E that contains all zero points (x, y) of the Weierstrass equation that are defined over K, together with $\mathcal{O}$.

Addition on E is not trivial. If we have points $P = (x_0, y_0)$, $Q = (x_1, y_1)$ we have the following formulae for addition

- $P + \mathcal{O} = \mathcal{O} + P = P$,

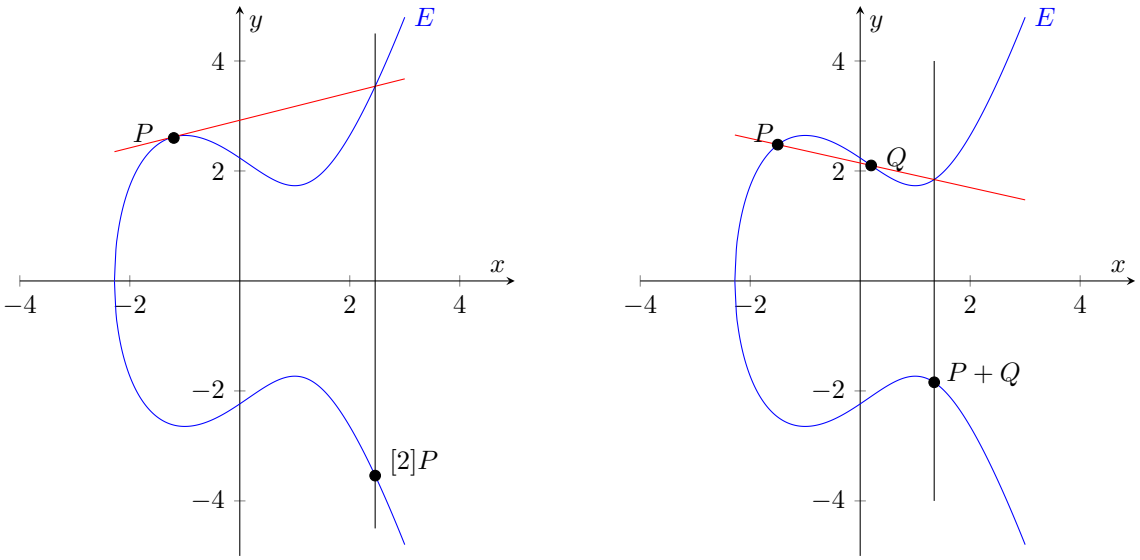


Figure 2.1: Addition of two points P and Q, and doubling of a point P on an elliptic curve $E : y^2 = x^3 - 3 * x + 5$.

- $-P = (x_0, -y_0 - a_1x_0 - a_3)$
- for $x_0 \neq x_1$ define $\lambda = \frac{y_1 - y_0}{x_1 - x_0}$ and $\nu = \frac{y_0x_1 - y_1x_0}{x_1 - x_0}$
- If $Q = P$ and $y_1 \neq 0$, define $\lambda = \frac{3x_0^2 + 2a_2x_0 + a_4 - a_1y_0}{2y_0 + a_1x_0 + a_3}$ and $\nu = \frac{-x_0^3 + a_4x_1 + 2a_6 - a_3y_1}{2y_0 + a_1x_0 + a_3}$
- Then the line L through P and Q is defined by $L : y = \lambda x + \nu$. The point $P + Q = (x_2, y_2)$ is thus given by

$$x_2 = \lambda^2 - a_1\lambda - a_2 - x_1 - x_0$$
$$y_2 = -(\lambda + a_1)x_2 - \nu a_3.$$

An elliptic curve E is, as shown in Equation 2.1, a plane in $\mathbb{P}^2$, with coordinates $(X : Y : Z)$. We make a surjective map ϕ to $\mathbb{P}^1$ given by

$$\phi : E \to \mathbb{P}^1$$

$$(X : Y : Z) \mapsto \begin{cases} (X : Z) & \text{if } Z \neq 0 \\ (1 : 0) & \text{if } Z = 0. \end{cases}$$

Using this map we can establish a bijection between $\mathbb{P}^1$ and $E/\{\pm 1\}$. We denote the projective line with the structure of E by the *Kummer line* of E.

Later we will see that in cryptographic applications, it is possible to work over the Kummer line of an elliptic curve, making computations faster.

2.3 Isogenies

Definition 2.3.1. Given two elliptic curves E_1 and E_2, we define an *isogeny* as a morphism

$$\phi : E_1 \longrightarrow E_2 \quad \text{satisfying} \quad \phi(\mathcal{O}) = \mathcal{O}.$$

If $E_2 = E_1$, we call ϕ an *endomorphism*. The invertible elements of the ring of endomorphisms $\mathrm{End}(E)$ form the group $\mathrm{Aut}(E)$, the automorphisms of the curve E.

As shown in 2.1.2 we either have either $\phi(E_1) = \mathcal{O}$ or $\phi(E_1) = E_2$. Two curves E_1 and E_2 are isogenous if there is an isogeny from E_1 to E_2 such that $\phi(E_0) \neq \mathcal{O}$. in Definition 2.3.6 we will see this is an equivalence relation.

Since all isogenies besides the zero map are surjective, they are finite maps of curves. This gives the usual injection of function fields

$$\phi^* : \bar{K}(E_2) \longrightarrow \bar{K}(E_1).$$

We define the degree of the isogeny ϕ as the degree of the finite extension $\bar{K}(E_1)/\phi^*\bar{K}(E_2)$, and give the zero map degree 0.

Definition 2.3.2. The *multiplication-by-m isogeny* is given by

$$[m]P = \underbrace{P + ... + P}_{\text{m times}}.$$

This map is an isogeny.

Definition 2.3.3. Given an elliptic curve E, and $m \in \mathbb{Z}$ with $m \geq 1$. The *m-torsion subgroup* of E, $E[m]$ is the set of points of order m:

$$E[m] \cong \{P \in E : [m]P = \mathcal{O}\}.$$

Theorem 2.3.4. For E an elliptic curve over a field K with $\text{char}(K) = p > 0$, $m \in \mathbb{Z}$, with $m \neq 0$ and $p \nmid m$, we have

$$E[m] = \frac{\mathbb{Z}}{m\mathbb{Z}} \times \frac{\mathbb{Z}}{m\mathbb{Z}}$$

Isogenies have certain useful properties.

Theorem 2.3.5. Let $\phi : E_1 \longrightarrow E_2$, $\psi : E_1 \longrightarrow E_2$ be isogenies, with ϕ separable, and let Φ be a finite subgroup of E_1. Then we have the following

1. $\deg(\phi) = \#\ker\phi$
2. for all $P, Q \in E$, $\phi(P + Q) = \phi(P) + \phi(Q)$
3. If ϕ and ψ are nonconstant, and $\ker\phi \subset \ker\psi$ then there is a unique isogeny $\lambda : E_2 \longrightarrow E_3$ such that $\psi = \lambda \circ \phi$
4. There is a unique elliptic curve E' and a separable isogeny $\phi : E_1 \longrightarrow E'$ such that $\ker\phi = \Phi$.

Proof. See [51]III 4.10, 4.11 and 4.12 □

This theorem is quintessential for supersingular isogeny cryptography, as we will see in 4.2. Assume we have a certain subgroup H of an elliptic curve of order $n = p_1^{k_1}...p_r^{k_r}$. Using Theorem 2.3.5 we can find a unique separable isogeny ϕ that has H as kernel. We know the degree of this isogeny is n, and using Theorem 2.3.5.3 we can show that if we take a point in H with degree p_i, we can create a chain of isogenies ψ_i such that $\phi = \psi_1 \circ ... \circ \psi_s$, with $s = \sum_{i=1}^{r} k_i$.

Definition 2.3.6. There exists a unique isogeny

$$\hat{\phi} : E_2 \longrightarrow E_1,$$

such that

$$\hat{\phi} \circ \phi = [m] \text{ and } \phi \circ \hat{\phi} = [m].$$

This isogeny is called *the dual isogeny of* ϕ.

The proof of this statement can be found in [51]III.6.1. As proven in [51] II.6.3, the degree of the dual of a function ϕ is the same as the degree of ϕ, and the dual of the dual of ϕ is just ϕ. Given that every isogeny has a dual, being isogenous becomes an equivalence relation.

2.4 The Frobenius morphism

Definition 2.4.1. Given a field K of characteristic $p > 3$, $q = p^r$ and an elliptic curve E/K, $E : y^2 = f(x)$. Then the *Frobenius morphism* of an elliptic curve is defined by

$$\phi_q : E \to E^{(q)}, \quad (x, y) \mapsto (x^q, y^q),$$

where $E^{(q)}$ is the curve defined by taking the coefficients of the equation for E to the q^{th} power.

Note that when $K = \mathbb{F}_q$, ϕ_q is actually the identity on K, so $E^{(q)} = E$. The set of points fixed by ϕ_q is exactly $E(\mathbb{F}_q)$.

Theorem 2.4.2. Let E be an elliptic curve defined over a field $\mathbb{F}_q$. We have

1. $|\#E(\mathbb{F}_q) - q - 1| \leq 2\sqrt{q}$. **(Hasse)**
2. For $a = q + 1 - \#E(\mathbb{F}_q)$, we have for the Frobenius morphism ϕ_q

$$\phi_q^2 - a\phi_q + q = 0 \quad \text{in } \mathrm{End}(E).$$

We therefore also call a the trace of the Frobenius morphism, denoted $\mathrm{tr}(\phi_q)$

Proof. See [51] V.1.1 and V.2.3. □

2.5 Miscellaneous

Hard Homogenous Spaces

In [22] the notion of *Hard Homogenous Spaces* is introduced. It is said to be an efficiently computable action

$$* : G \times S \to S$$

for G a finite commutative group and S a set. It has the following properties:

- for all $s \in S$, * acts free: $gs = s \to g = \mathrm{id}_G$
- for all pairs $(s_1, s_2) \in S^2$, * acts transitive: there is a group element $g \in G$ such that $gs_1 = s_2$
- Given $s_0, s_1 \in S$, it should be hard (non-polynomial) to find a $g \in G$ such that $g * s_0 = s_1$
- Given $s_0, s_1, s_2 \in S$, such that $s_1 = g * s_0$, it should be hard to find an $s_3 = g * s_2$.

To be able to make computations on hard homogenous spaces, the following operations are required to be easy (polynomial time):

- Compute the group operation in G
- Sample randomly from G with almost uniform distribution
- Compute the action $*$ of a group element $g \in G$ on some $s \in S$

The Chinese remainder theorem

Theorem 2.5.1. For $N = n_1 \cdot ... \cdot n_k$, with all n_i pairwise coprime, the map

$$x \bmod N \mapsto (x \bmod n_1, ..., x \bmod n_k)$$

defines a ring isomorphism

$$\mathbb{Z}/N\mathbb{Z} \cong \mathbb{Z}/n_1\mathbb{Z} \times ... \times \mathbb{Z}/n_k\mathbb{Z}. \tag{2.3}$$

Proof. First we prove that if there exists an x that satisfies equation (2.3) then x is unique, and then prove its existence.

Suppose x and y are both integers that are solutions to all congruence relations. This means they give the same remainder when divided by n_i, so their difference $x - y$ is a multiple of each n_i. All n_i are pairwise coprime, so $x - y$ is therefore also divisible by N. This means x and y are equivalent modulo N. This proves uniqueness.

For existence, we give a constructive proof that we can use to find x.

$$x \equiv a_1 \bmod n_1$$
$$x \equiv a_2 \bmod n_2$$

There exist integers m_1 and m_2 such that $m_1 n_1 + m_2 n_2 = 1$. A solution for x is then given by

$$x = a_1 m_2 n_2 + a_2 m_1 n_1$$

More general systems of k equations can be solved by iterating this process $k - 1$ times. □

3. Supersingular curves

We continue with giving specific properties of elliptic curves, and properties of maps between elliptic curves. After that, we will focus on the properties of supersingular elliptic curves and Montgomery curves. We end with a section on isogeny graphs. The main sources for this sections are [51] and [33].

3.1 Supersingular curves

Definition 3.1.1. An *algebra* over a field K is a vector space A over K equipped with a binary operation

$$\cdot : A \times A \longrightarrow A$$

such that the following equations hold:

1. $(x + y) \cdot z = x \cdot z + y \cdot z$
2. $z \cdot (x + y) = z \cdot x + z \cdot y$
3. $(ax) \cdot (by) = (ab)(x \cdot y)$

for $x, y, z \in A, a, b \in K$.

Definition 3.1.2. Let $\mathcal{K}$ be a finitely generated $\mathbb{Q}$-algebra. An *order* $\mathcal{R}$ of $\mathcal{K}$ is a subring of $\mathcal{K}$ that is finitely generated as a $\mathbb{Z}$-module, and satisfies $\mathcal{R} \otimes \mathbb{Q} = \mathcal{K}$.

A *quaternion algebra* is a $\mathbb{Q}$ - algebra of the form

$$\mathcal{K} = \mathbb{Q} + \mathbb{Q}\alpha + \mathbb{Q}\beta + \mathbb{Q}\alpha\beta$$

with

$$\alpha^2, \beta^2 \in \mathbb{Q}, \quad \alpha^2, \beta^2 < 0, \quad \beta\alpha = -\alpha\beta.$$

Theorem 3.1.3. Given an elliptic curve E over a field K, with $\mathrm{char}(K) = p > 0$, the endomorphism ring $\mathrm{End}(E)$ of E is either an order in an imaginary quadratic field, or an order in a quaternion algebra. In the first case, we call E *ordinary*. In the second case, E is *supersingular*.

Proof. See [51]III.9.4. □

Theorem 3.1.4. If E is supersingular, $j(E) \in \mathbb{F}_{p^2}$.

Proof. See [51]V.3.1 □

This theorem proves that there are only finitely many supersingular elliptic curves over $\mathbb{F}_q$. The next theorem shows exactly how many, and also how to check if a curve is supersingular.

Theorem 3.1.5. Let $\mathbb{F}_q$ be a field of characteristic $p > 3$.

1. Let $E/\mathbb{F}_q$, with $q = p^r$ be defined by the Weierstrass equation

$$E : y^2 = f(x).$$

Then E is supersingular if and only if the coefficient of $x^{(p-1)}$ in $f(x)^{(p-1)/2}$ is zero.

2. There is only one supersingular curve in characteristic 3, and for $p \geq 5$, the number of supersingular curves over a field of characteristic p is given by

$$\left[\frac{p}{12}\right] + \begin{cases} 0 \; if \; p \equiv 1 \mod 12 \\ 1 \; if \; p \equiv 5 \mod 12 \\ 1 \; if \; p \equiv 7 \mod 12 \\ 2 \; if \; p \equiv 11 \mod 12 \end{cases}$$

Proof. See [51]V.4.1. □

Theorem 3.1.6. Given an elliptic curve E over a field $K/\mathbb{F}_q$, with $\mathrm{char}(K) = p$. Then E is supersingular if and only if the trace of the Frobenius morphism is equal to

$$\mathrm{tr}(\phi_q) \equiv 0 \mod p.$$

Proof. See [51] Exercise 5.10a. □

Corollary 3.1.7. Given an elliptic curve E defined over $\mathbb{F}_p$, E is supersingular iff $\#E = p + 1$. For E defined over $\mathbb{F}_{p^2}$, we have, denoting the p^2-Frobenius morphism with ϕ,

$$\begin{aligned} \mathrm{tr}(\phi) = \pm 2p, \quad \#E = (p \pm 1)^2 \\ \mathrm{tr}(\phi) = \pm p, \quad \#E = p^2 \pm p + 1 \\ \mathrm{tr}(\phi) = 0, \quad \#E = p^2 + 1 \end{aligned}$$

Proof. Combining theorem 2.4.2 and 3.1.6 we get $|\mathrm{tr}(\phi_q)| \leq 2\sqrt{q}$, and $\mathrm{tr}(\phi_q)$ can only be a multiple of p. In the case $q = p$,this only leaves the option $\mathrm{tr}(\phi_q) = 0$, so using 2.4.2 we get $\#E(\mathbb{F}_p) = p+1$. For $q = p^2$ there are five options for $\mathrm{tr}(\phi_q)$, as described above. □

3.1.1 Ideal class groups

Here we will briefly introduce ideal class groups in the setting of submodules of $\mathrm{End}(E)$ of a supersingular elliptic curve E. For more information on modules and fractional ideals, we refer to [6]. As described in Theorem 3.1.4, a supersingular curve has an endomorphism ring $\mathrm{End}(E)$ isomorphic to an order of a quaternion algebra. Here we are concerning ourselves with a subring $\mathcal{O}$ of $\mathrm{End}(E)$ that is isomorphic to an order of an imaginary quadratic field, in particular $\mathcal{O} = \mathbb{Z}[\pi]$, where π is the Frobenius morphism. In particular, $\mathcal{O}$ is commutative. Note that this is also the form of the endomorphism ring of ordinary elliptic curves. While this is not the case for all supersingular curves, it can be shown (see [12]) that there exist supersingular curves that have such a subring in their endomorphism ring. We define $\ell(\mathcal{O}, \pi)$ to be that set of elliptic curves that have $\mathcal{O}$ as a subgroup of their endomorfism ring.

Definition 3.1.8. The *norm* of an $\mathcal{O}$-ideal $\mathfrak{a}$ is defined as

$$N(\mathfrak{a} = |\mathcal{O}/\mathfrak{a}| = \gcd(N(\alpha)|\alpha \in \mathfrak{a}).$$

Especially, we have

$$N(\mathfrak{ab}) = N(\mathfrak{a})N(\mathfrak{b}).$$

In the next paragraph we will introduce the ideal class group of $\mathcal{O}$. A *fractional $\mathcal{O}$-ideal* is an $\mathcal{O}$-submodule of $\mathrm{End}(E)$ of the form $\alpha\mathfrak{a}$, with $\alpha \in \mathrm{End}(E)$ nonzero. A fractional $\mathcal{O}$-ideal is *invertible* if there exists a fractional $\mathcal{O}$-ideal $\mathfrak{b}$ such that $\mathfrak{a}\mathfrak{b} = \mathcal{O}$. All principal fractional $\mathcal{O}$-ideals are invertible. We now get two groups, the set of invertible fractional ideals $I(\mathcal{O})$, and its subgroup of principal fractional ideals $P(\mathcal{O})$.

Definition 3.1.9. The *ideal class group* of $\mathcal{O}$ is defined as the quotient

$$\mathrm{cl}(\mathcal{O}) = I(\mathcal{O})/P(\mathcal{O}).$$

The size of the ideal class group therefore is an indication of how much $\mathcal{O}$ is not a principal ideal domain. Below we will prove some properties of the ideal class group.

Theorem 3.1.10. Let $\mathcal{O}$ be an order in an imaginary quadratic field and $\pi \in \mathcal{O}$ such that $\ell_p(\mathcal{O}, \pi)$ is non-empty. Here $\ell_p(\mathcal{O}, \pi)$ is the set of elliptic curves defined over $\mathbb{F}_p T$ with $End_p(E) \cong \mathcal{O}$ such that π corresponds to the $\mathbb{F}_p$ Frobenius endomorphism of E. Then the ideal-class group $\mathrm{cl}(\mathcal{O})$ acts freely and transitively on the set $\ell_p(\mathcal{O}, \pi)$ via the map

$$\begin{aligned} \mathrm{cl}(\mathcal{O}) \times \ell_p(\mathcal{O}, \pi) &\to \ell_p(\mathcal{O}, \pi) \\ ([\mathfrak{a}, E] &\mapsto E/\mathfrak{a} \end{aligned}$$

Where $E/\mathfrak{a}$ is defined by a $N(\mathfrak{a})$ degree isogeny

$$\phi_{\mathfrak{a}} : E \to E/\mathfrak{a}.$$

Proof. See [54].4.5. □

The curve $E/\mathfrak{a}$ is often written as $[\mathfrak{a}] * E$ or $[\mathfrak{a}]E$.

Theorem 3.1.11. Let $p \geq 5$ be a prime such that $p \equiv 3 \mod 8$, and let $E/\mathbb{F}_p$ be a supersingular elliptic curve. Then there is a subring $\mathbb{Z}[\pi]$ of the endomorphism ring $\mathrm{End}(E)$ if and only if there exists a $A \in \mathbb{F}_p$ such that E is $\mathbb{F}_p$ isomorphic to the curve $E : y^2 = x^3 + Ax^2 + x$. If such an A exists, it is unique

Proof. See [12] Proposition 8. □

To finish this section, we define a specific type of prime that helps split the ideal class group in smaller parts.

Definition 3.1.12. An *Elkies prime* is a prime ℓ that splits $\mathcal{O}$: There exist $\mathfrak{l}, \mathfrak{l}' \in \mathcal{O}$ such that $\ell\mathcal{O} = \mathfrak{l}\mathfrak{l}'$. The ideal $\mathfrak{l}$ is generated as $\mathfrak{l} = (\ell, \pi - \lambda)$, where λ is an eigenvalue of the Frobenius morphism on the ℓ torsion. The conjugate of $\mathfrak{l}$ is $\mathfrak{l}'$.

3.2 Twists of curves

Definition 3.2.1. Given an elliptic curve E/K, a *twist* of E is a curve $E^{(t)}$ such that E and $E^{(t)}$ are not isomorphic over K, but are isomorphic over $\bar{K}$. If they are isomorphic over K^2, $E^{(t)}$ is called a *quadratic twist* of E.

Given an elliptic curve E/K, defined as $E : y^2 = f(x)$, we can create a quadratic twist of E as follows. For d nonsquare in K, let $K(\sqrt{d})$ be a quadratic extension of K. A quadratic twist of E is now given by

$$E^{(t)} : dy^2 = f(x).$$

It can be seen that for all $x \in K$, for K not a field of order 2, $(x, f(x))$ lies on either E or $E^{(t)}$, as either $f(x)$ is a square in K and $(x, f(x))$ lies on E, or $f(x)$ is not a square and lies on $E^{(t)}$, by resizing the y-coordinate. (In a field of order two all points are squares). There is one exception

in the case $y = 0$. In this case $(x, f(x))$ is a point on both E and $E^{(t)}$. Noting this, we can count the points on E and $E^{(t)}$, using that if $(x, y) \in E, (x, -y) \in E$. For a field $\mathbb{F}_q$, this gives

$$\#E + \#E^{(t)} = 2q + 2.$$

Using Theorem 2.4.2, this gives us that the Frobenius traces t_E of E and $t_E^{(t)}$ of $E^{(t)}$ are related as follows

$$t_E = -t_E^{(t)}. \tag{3.1}$$

Using equation (3.1) we can now see that for supersingular curves over $\mathbb{F}_{p^2}$, as described in Corollary 3.1.7, the curves with Frobenius trace $-p$ or $-2p$ are the quadratic twists of curves with trace p respectively $2p$.

Interestingly, while these curves are isomorphic over $\mathbb{F}_{p^4}$, they are not isogenous over $\mathbb{F}_{p^2}$, as from 3.1.7 we can see that they do not have the same number of points on the curve. This means that every point on the isogeny graph is actually represented by two different twists over $\mathbb{F}_{p^2}$, as will be described in section 3.3.

3.3 Isogeny graphs

Definition 3.3.1. A graph G is a pair $G = (V, E)$, where $V = \{v_1, ..., v_n\}$ is a set of vertices, or points, and $E \subseteq (V, V)$ is a set of edges, or lines between these points.

A graph is *directed* if $(v_i, v_j) \neq (v_j, v_i)$ when $i/neqj$, and *undirected* otherwise. A *multigraph* is a graph that allows multiple edges (v_i, v_j), and also loops (v_i, v_i). A graph is called *connected* if for all vertices v_i and v_j, there is a path of edges from v_i to v_j.

A graph is called *k-regular* if all vertices have k edges connected to it.

The *Adjacency matrix* A of a graph is defined to be the $n \times n$ graph that has a 1 at the ij-th entry if and only if there is an edge from v_i to v_j, and a 0 if this is not the case. For all matrices, all eigenvalues satisfy the bound $|\lambda| \leq k$. Note that the adjacency matrix of an undirected graph is symmetrical.

Given a graph G, and a subset of the vertices A, the *vertex boundary* of a subset is

$$\delta_v(A) = \{v \in G - A | \text{ there is an edge between } v \text{ and a vertex in } A\}.$$

Let E be the set of edges (x, y) in G. The edge boundary of A in G is

$$\delta_e(A) = \{(x, y) \in E | x \in A, y \in G - A\}$$

Definition 3.3.2. An *expander graph* is a k-regular graph G with $\#V$ vertices such that there exists a $c \in \mathbb{R}_{\geq 0}$, such that for all subsets $A \subseteq V$ with $\#A \leq \#G/2$

$$\#\delta_v(A) \geq c \cdot \#A$$

We will now introduce *Ramanujan graphs*. For the source on this part of the text, see [41]

Definition 3.3.3. A graph has the *Ramanujan property* if for all nontrivial eigenvalues that are not equal to $-k$,

$$|\lambda| \leq 2\sqrt{k-1}.$$

Theorem 3.3.4. Ramanujan graphs are expander graphs.

Proof. If G is a regular graph, then according to Definition 3.3.2

$$\#\delta_v(A) \geq \#A \cdot \frac{k - \lambda_1(G)}{2},$$

when $\#A \leq \#G/2$. Here λ_1 is the smallest eigenvalue of G. Thus

$$\#\delta_v(A) \geq (\frac{1}{2} - \frac{\lambda_1(G)}{2k}) \cdot \#A$$

so take $c = \frac{1}{2} - \frac{\lambda_1(G)}{2k}$, this shows that Ramanujan graphs are expander graphs. □

An *isogeny graph* is a graph with its vertices consisting of the isomorphism classes of elliptic curves over a field F, and the edges are the isogenies between the elliptic curves. A *supersingular isogeny graph* is a graph with only isomorphism classes of supersingular elliptic curves over a field F. Since all supersingular elliptic curves are isomorphic to a curve over $\mathbb{F}_{p^2}$, supersingular isogeny graphs are finite. For edges, we define two isogenies ϕ_1, ϕ_2 to be isomorphic if there exists an automorphism $\alpha \in \mathrm{Aut}(E)$ such that $\phi_1 = \alpha\phi_2$.

Theorem 3.3.5. A supersingular isogeny graph with vertices and edges being isomorphism classes of elliptic curves and degree ℓ isogenies, is a connected $\ell+1$ regular multigraph with the Ramanujan property.

Proof. See [42] theorem 1. □

In Figure 3.1 an example of an isogeny graph is given.

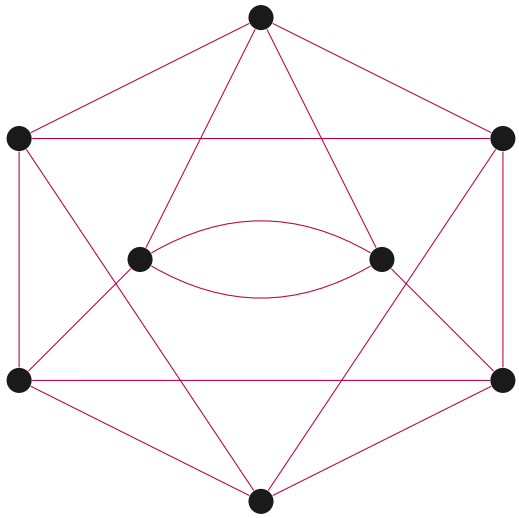


Figure 3.1: Isogeny graph of degree 3 over $\bar{\mathbb{F}}_{97}$.

Theorem 3.3.6. A supersingular isogeny graph over $K = \mathbb{F}_{p^2}$ is undirected if and only if $p \equiv 1 \mod 12$.

Proof. A supersingular isogeny graph is undirected if for every isogeny $\phi : E_1 \to E_2$ there exist a unique dual isogeny $\hat{\phi} : E_2 \to E_1$, up to automorphism. But it can be the case that the automorphism group $\mathrm{Aut}(E_1)$ does not have the same size as $\mathrm{Aut}(E_2)$. In [51]III.10 it is shown that this problem arises if one of the curves has complex multiplication by either $\sqrt{-1}$ or $e^{2\pi i/3}$. What happens then is that two isogenies $E_1 \to E_2$ can have equivalent dual isogenies, even though the original isogenies are not equivalent. If a curve does not have either of these complex multiplications, $\mathrm{Aut}(E) = \{\pm 1\}$. In [42], Proposition 4.6, it is proven that if p splits in both $\mathbb{Z}[\sqrt{-1}]$ and $\mathbb{Z}[e^{2\pi i/3}]$ the curve will not have this complex multiplication, and that this is the case if and only if $p \equiv 1 \mod 12$. □

Theorem 3.3.7. Let G be a Ramanujan graph. A random walk of length at least $\frac{\log 2\#G/\#S^{1/2}}{\log(k/\lambda_1)}$ starting from v will end in S with probability between $\frac{1}{2}\frac{\#S}{\#G}$ and $\frac{3}{2}\frac{\#S}{\#G}$.

Proof. see [34] Lemma 2.1 □

3.3.1 Isogeny graphs of twisted curves

As described in Corollary 3.1.7, there are five different isogeny classes for supersingular elliptic curves defined over $\mathbb{F}_{p^2}$, all corresponding to a different Frobenius trace. Two curves are isogenous

only if $\#E_1 = \#E_2$. Therefore, if we take a curve E with $\#E = (p+1)^2$, and its twist $E', \#E = (p-1)^2$, they will never lay on the same isogeny graph over $\mathbb{F}_{p^2}$. Define the isogeny graph of G_{2p} and G_{-2p} as the graphs with as vertices the j-invariants (defined over $\mathbb{F}_{p^2}$), and with edges the isogenies up to automorphism between the elliptic curves.

Theorem 3.3.8. The graphs G_{2p} and G_{-2p} are isomorphic, and they are both isomorphic to the general isogeny graph over $\bar{\mathbb{F}}_{p^2}$

Proof. See [1]. □

3.4 Montgomery arithmetic

In this part of the section we describe what is the Montgomery form of an elliptic curve, and how we can use these kind of forms to compute isogenies and point multiplications effectively.

3.4.1 Montgomery curves

In 1987, Peter Montgomery proposed in [40] a specific form of elliptic curves, that have special, beneficial, addition properties. These curves are called *Montgomery curves* and are described by the equation below.

$$E/K : by^2 = x^3 + ax^2 + x, \tag{3.2}$$

Under the conditions that $b \neq 0$ and $a^2 \neq 4$.

All Montgomery curves share certain useful properties.

- The point $(0, 0)$ has order two
- There are two points Q_4 of order four given by

$$Q_4 \in \{(1, \pm\sqrt{(a+2)/b}), (-1, \pm\sqrt{(a-2)/b})\} \tag{3.3}$$

  with $[2]Q_4 = (0, 0)$
- There are two other points P of order two, with x-coordinate given by $x_p^2 + ax_p + 1 = 0$.

As will be proven below, a great asset of Montgomery curves is that addition of points and isogenies can be computed on the Kummer line, so we need only the X and Z coordinates of a point, and we can disregard the Y-coordinate. This is useful as it gives the opportunity to use more efficient algorithms for addition, scalar multiplication and isogeny computation, without the need to compute inverses or square roots.

Theorem 3.4.1. For any Montgomery curve E over a field K, a point $P \in E$, for any integers n, m the two points given by $P_n = [n]P = (X_n : Z_n)$, and $P_m = [m]P = (X_m : Z_m)$ can be added to obtain the point $P_{n+m} = [n+m]P$, with the following coordinates:

$$X_{m+n} = Z_{m-n}((X_m + Z_m)(X_n + Z_n) + (X_m + Z_m)(X_n - Z_n))^2$$
$$Z_{m+n} = X_{m-n}((X_m - Z_m)(X_n + Z_n) + (X_m + Z_m)(X_n - Z_n))^2.$$

The costs for adding two points then is $3M + S + 5a$. Here M is a multiplication, in K, S a squaring in K and a an addition in K.

In the case that $m = n$, the operation is equal to doubling a point. Equations are then given by

$$X_{2n} = (X_n + Z_n)^2(X_n - Z_n)^2$$
$$Z_{2n} = ((X_n + Z_n)^2 - (X_n - Z_n)^2)(X_n - Z_n)^2 + ((a+2)/4)((X_n + Z_n)^2 - (X_n - Z_n)^2).$$

And the costs are $2S + 2M + 5a$ plus one scalar addition.

Proof. See [40], §10. □

.

Efficient algorithms for these computations are given for instance in [20].

3.4.2 The Montgomery ladder

Using Theorem 3.4.1 one can create a fast algorithm for multiplying a point on an elliptic curve, as described in [40]. This process is called the *Montgomery ladder*. The advantage of using the Montgomery ladder over double-and-add for multiplying a point is that the Montgomery ladder always terminates in a fixed amount of time, therefore it leaks no power consumption data during a side-channel attack, and is also not vulnerable to timing attacks. In Algorithm 3 below the process of multiplying a point using the Montgomery ladder is described. Here double and add represent the algorithms for doubling and adding a point as described in Theorem 3.4.1.

Algorithm 3 Montgomery Ladder

Data: A point $P \in E$ and an integer $d \in \mathbb{N}$.
Result: A point $[d]P \in E$.
$R_0 = 0, R_1 = P$
Write d in binary representation: $d = d_0 + 2d_1 + 2^2 d_2 + ... + 2^m d_m$
for $i = i$ *downto* 0 **do**
  if $d_i = 0$ **then**
    $R_1 = \text{add}(R_0, R_1)$
    $R_0 = \text{double}(R_0)$
  end
  else
    $R_0 = \text{add}(R_0, R_1)$
    $R_1 = \text{double}(R_1)$
  end
end

The costs of the Montgomery ladder come down to $\log(d)$ additions and doublings, costing in total $3S + 5M + 10a$ plus one scalar multiplication per round.

If we would like to know a point $[m]P+Q$, one could first compute $[m]P$ using the Montgomery ladder and then add Q. However, this is quite slow. Using specific properties of the Montgomery curves, we can compute this a lot faster using *differential addition*, where we use the difference $P - Q$ to compute $[m]P + Q$.

Theorem 3.4.2. Given two projective points $P = (X_P, Z_P), Q = (X_Q, Z_Q)$ and their difference $P - Q = (X_\ominus, Z_\ominus)$, the projective point $P + Q = (X_\oplus, Z_\oplus)$ is given by

$$X_\oplus = Z_\ominus(X_P - Z_P)(X_Q + Z_Q) - (X_P + Z_P)(X_Q - Z_Q))^2$$
$$Z_\oplus = X_\ominus X_P - Z_P)(X_Q + Z_Q) + (X_P + Z_P)(X_Q - Z_Q))^2$$

Proof. See [40] 10.3.1. □

This can be efficiently computed, see for instance [20] Algorithm 3.2.1. This algorithm will be referred to as **dADD**. The costs are $4\mathbf{M}+ 2\ \mathbf{S} + 6\mathbf{A}$ (one multiplication less if $Z_\ominus$ is normalised to 1).

A way to compute $[m]P+Q$ is then given by [25] Algorithm 4.1.1, and is also given in Algorithm 4. It is constructed in the same way as the Montgomery ladder.

Algorithm 4 Computing [d]P+Q

Data: Points $P, Q, P-Q \in E$, given as projective coordinates, and an integer $d \in \mathbb{N}$.
Result: A point $[d]P + Q \in E$.
$A = 0, B = P, C = P$
Write d in binary representation: $d = d_0 + 2d_1 + 2^2 d_2 + ... + 2^m d_m$
for $i = i$ *downto* 0 **do**
  if $d_i = 0$ **then**
    $A = \text{double}(A)$
    $B = \mathbf{dADD}(A, B, P)$
    $C = \mathbf{dADD}(A, C, Q)$
  end
  else
    $A = \mathbf{dADD}(A, B, P)$
    $B = \text{double}(B)$
    $C = \mathbf{dADD}(B, C, P - Q)$
  end
end

The costs of computing $[d]P + Q$ is

$$C = \lceil \log(d) \rceil \cdot (C_{\text{double}} + 2C_{\text{dADD}}) = 16 \log(d) \tag{3.4}$$

in field multiplications if we assume a squaring equals a multiplication and ignore the additive costs.

In the following sections we will denote addition of two multiples of P by add, and the doubling of a point by double.

Theorem 3.4.3. Given an elliptic curve E over a field K with $\text{char}(K) \neq 2$, if E or its twist has a point of order 4, E is isomorphic over $\bar{K}$ to a Montgomery curve.

Proof. See theorem 3.2 and 3 in [7]. □

So, if it is given that for an elliptic curve E, the group $E(K)$ has group structure $(\mathbb{Z}/(p \pm 1)\mathbb{Z})^2$, and its twist E' has group structure $E'(K) = (\mathbb{Z}/(p \mp 1)\mathbb{Z})^2$, it is clear that E and E' are isomorphic to Montgomery curves, since p is always an odd number.

3.5 Velu formulae

In this section we will look at a famous algorithm for computing isogenies, due to Velu [53]. First we will give the general theorem, and then give a simplified version that is sufficient for most isogeny computations on Montgomery curves.

Theorem 3.5.1. Let E be an elliptic curve over a field K defined by

$$F(x, y) = x^3 + a_2 x^2 + a_4 x + a_6 - (y^2 + a_1 xy + a_3 y) = 0. \tag{3.5}$$

Let G be a finite subgroup of $E(\bar{K})$. Let G_2 be the set of points in $G \backslash \{\mathcal{O}_E\}$ of order 2 and let G_1 be such that $\#G = 1 + \#G_2 + 2\#G_1$ and

$$G = \{\mathcal{O}_E\} \cup G_2 \cup G_1 \cup \{-Q | Q \in G_1\}.$$

Write the two partial derivatives of F as follows

$$F_x = 3x^2 + 2a_2 x + a_4 - a_1 y, \quad F_y = -2y - a_1 x - a_3.$$

Define for a point $Q = (x_Q, y_Q) \in G_1 \cup G_2$ two functions as follows

$$u(Q) = (F_y(Q))^2 = (-2y_Q - a_1 x_Q - a_3)^2$$
$$t(Q) = \begin{cases} F_x(Q), & \text{if } Q \in G_2. \\ 2F_x(Q) - a_1 F_y(Q), & \text{if } Q \in G_1. \end{cases}$$

Then define

$$t(G) = \sum_{Q \in G_1 \cup G_2} t(Q)$$
$$w(G) = \sum_{Q \in G_1 \cup G_2} (u(Q) + x_Q t(Q))$$

now set $A_1 = a_1,\ A_2 = a_2,\ A_3 = a_3,\ A_4 = a_4 - 5t(G),\ A_6 = a_6 - (a_1^2 + 4a^2)t(G) - 7w(G)$.

Then the map $\phi(x, y) \mapsto (X, Y)$, with

$$X = x + \sum_{Q \in G_1 \cup G_2} \frac{t(Q)}{x - x_Q} + \frac{u(Q)}{(x - x_Q)^2} \tag{3.6}$$

$$Y = y - \sum_{Q \in G_1 \cup G_2} u(Q)\frac{2y + a_1 + a_3}{(x - x_Q)^3} + t(Q)\frac{a_1(x - x_Q) + y - y_Q}{(x - x_Q)^2} + \frac{a_1 u(Q) - F_x(Q)F_y(Q)}{(x - x_Q)^2} \tag{3.7}$$

is a seperable isogeny from E to

$$E' : Y^2 + A_1XY + A_3Y = X^3 + A_2X^2 + A_4X + A_6,$$

with kernel G.

Proof. See [53] □

This function can be used to create isogenies $E \to E/G$. The problem with this function is that it is quite slow, especially for larger degree isogenies. Also it gives no specific algorithm that maps curves with specific properties to other curves with those properties, for instance it does not necessarily map Montgomery curves to Montgomery curves. This made dealing with 2-degree isogenies more difficult for instance, as described in [25]. Recent improvements have been made in this by Castello and Hisil in [17], and especially for 2-degree isogenies by Renes in [46]. However, the fastest improvemnt so far is proposed by Bernstein and De Feo in [8], and is more optimised by [2]. In the next sections we will give both the algorithms as proposed by Castello and Hisil that work best for small degrees isogenies over Montgomery curves, and the algorithm as proposd by Bernstein and De Feo.

3.5.1 Isogenies between Montgomery curves

In [17] it is described how we can compute isogenies between Montgomery curves not using the more costly Vélu formulas [53], but instead using a cheaper formula. This works for all odd-degree isogenies. In [46] it is shown that with a little adaptation, this formula can also be applied to specific degree 2 isogenies. The theorem as proven in [17] is given below.

Theorem 3.5.2. For a field K with $\text{char}(K) \neq 2$, let $P \in E(\bar{K})$ be a point of order $\ell = 2d + 1$ on the Montgomery curve $E/K : by = x^3 + ax^2 + x$ and write $\sigma = \sum_{i=1}^{d} x_{[i]P}$, $\bar{\sigma} = \sum_{i=1}^{d} \frac{1}{x_{[i]P}}$ and $\pi = \prod_{i=1}^{d} x_{[i]P}$. The Montgomery curve

$$E'/K : b'y^2 = x^3 + a'x^2 + x \tag{3.8}$$

with

$$a' = (6\bar{\sigma} - 6\sigma + a) \cdot \pi \text{ and } b \ = b \cdot \pi^2 \tag{3.9}$$

is the codomain of the ℓ-isogeny $\phi : E \to E'$ with $\ker(\phi) = \{P\}$, which is defined by the coordinate maps

$$\phi : (x, y) \mapsto (f(x), y \cdot f'(x)), \tag{3.10}$$

where

$$f(x) = x \cdot \sum_{i=1}^{d} (\frac{x \cdot x_{[i]P-1}}{x - x_{[i]P}})^2 \tag{3.11}$$

with derivative $f'(x)$

Proof. See Section 3 in [17]. □

Given any arbitrary odd prime p we can use Theorem 3.5.2 to write an algorithm that computes a fast isogeny. This algorithm splits computation on three parts. First, the points $[s]P$ will be computed and stored. This algorithm is generally known as KPS, stemming from Kernel Points, as it is used to generate the points in the kernel of the isogeny. Then there can be two algorithms run in parallel, XISOG and XEVAL, where the first computes the coefficient A' for the new codomain E' of the isogeny $\phi : E \to E'$, and the other computes the image Q' for a point $Q' = \phi(Q)$.

First, we generate the kernel points $[i]P$ and store them. This is the KPS algorithm. These points can be computed using the Montgomery ladder.

Take $p = 2d + 1$. Using the Kummer line, we represent points P on curve $E/K : y^2 = x^3 + ax^2 + x.$ as $(X_p : Z_p)$. As stated before in this section, Montgomery curves have three points of order 2: $P_0 = (0 : 1), P_\alpha = (X_\alpha : Z_\alpha)$ and $P_{1/\alpha} = (Z_\alpha : X_\alpha)$, with $a = -(\alpha^2 + 1)/\alpha$. Rewriting the last part in $\mathbb{P}^1$ and writing $a = \frac{A}{C}$ gives us the computation of xISOG:

$$(a : 1) = (A : C) = (X_\alpha^2 + Z_\alpha^2 : X_\alpha Z_\alpha). \tag{3.12}$$

This makes it clear that we can represent the curve E using the point $(X_\alpha : Z_\alpha)$ as representative. This way, we can do isogeny computation, using only the calculations on specific points.

For any point $(X : Z)$ on the Kummer line of E, we can now find the corresponding point (X', Z') on the curve E', the codomain of the isogeny ϕ, by computing $f(X : Z)$. This is the xEVAL computation. The coordinates (X', Z') are given by

$$X' = X \cdot (\Pi_{i=1}^{d}(X \cdot X_i - Z_i \cdot Z))^2$$
$$Z' = Z \cdot (\Pi_{i=1}^{d}(X \cdot Z_i - X_i \cdot Z))^2.$$

.

We can rewrite these equations following Montgomery [40], making the computation more efficient.

$$X' = X \cdot (\Pi_{i=1}^{d}((X - Z)(X_i + Z_i) + (X + Z)(X_i - Z_i)))^2$$
$$Z' = Z \cdot (\Pi_{i=1}^{d}((X - Z)(X_i + Z_i) - (X + Z)(X_i - Z_i)))^2$$

.

Here $(X_i : Z_i)$ are the coordinates of the i-th multiple of P, $[i]P$, on the Kummer line. These $(X_i : Z_i)$ are the Kernel points, and we find them with the KPS algorithm. To find the equation for the new curve E', we can compute $(X'_\alpha : Z'_\alpha)$, and retrieve a using Equation 3.12. The curve is then given by

$$E' : y^2 = x^3 + ax^2 + x.$$

The problem with Vélû's function is that it becomes slower for larger degree isogenies. As described in Theorem 2.3.5, isogenies can be split into smaller isogenies under circumstances. However, if the degree of an isogeny is prime, it cannot be split. In B-SIDH, there will be computations of large prime degree isogenies. To compute these in more reasonable time, an improved way of calculating large degree isogenies is proposed by [8], and is more optimised by [2]. The idea of these optimisations is based on a result by Pollard [44] on evaluating polynomials whose roots are powers. The name for this new Vélu function is $\sqrt{}$élu formulae, and the algorithm is explained in the section below.

3.6 $\sqrt{}$élu formulae

We will begin by stating a version of Velu's theorem, adapted to Montgomery curves.

Theorem 3.6.1. Let $E : y^2 = x^3 + Ax + x$ be a Montgomery curve, with $A^2 \neq 4$. Let $P \in E$ be a point with $\deg(P) = \ell$, with ℓ an odd prime number. There exists a separable isogeny $\phi : E \to E'$ with kernel $G = \langle P \rangle$ and $E' : y^2 = x^3 + A'x^2 + x$. We define the functions

$$d = \left(\frac{A-2}{A+2}\right)^{\ell} \left(\frac{h_S(1)}{h_S(-1)}\right)^{S}, S = \{1, 2, .., \ell - 2\} \tag{3.13}$$

$$h_S(X) = \prod_{s \in S} (X - x([s]P). \tag{3.14}$$

Then A' is given by

$$A' = 2\frac{1+d}{1-d}. \tag{3.15}$$

The evaluation of the x-coordinate Q_x of a point $Q \in E$ on E' is given by

$$\phi_x(Q_x) = X^{\ell} \frac{h_S(1/Q_x)^2}{h_S(Q_x)^2}. \tag{3.16}$$

Proof. See [2] □

By far the largest part of computation of A' and Q_x is from computing $h_S(X)$. To speed up this computation, a baby-step giant-step algorithm is used. In the rest of this section, we will give a more detailed look into the algorithm that computes the isogeny. Just like the algorithm given in the previous section, this algorithm is divided into three parts, KPS, xISOG and xEVAL.

3.6.1 KPS

To compute the kernel points, we will split the set S as given in Equation (3.13) by an index set. In the end this algorithm will output three sets I, J and K that together store all points $[s]P$.

Definition 3.6.2. An *index system* for a set S is given by sets I, J, such that the maps $I \times J \to S$ given by $(i, j) \mapsto i + j$ and $(i, j) \mapsto i - j$ are injective and their images $I + J$ and $I - J$ are disjoint. We write $I \pm J = (I + J) \cup (I - J)$, and $K = S \backslash (I \pm J)$, such that $S = I \pm J \cup K$. Note that I and J themselves need not be subsets of S, only $I + J$ and $I - J$.

The index set used in the KPS algorithm for $S = \{1, 3, 5..., m\}$ is given as follows. Define $I = \{2b(2i+1) | 0 \leq i < b'\}$ and $J = \{2j+1 | 0 \leq j < b\}$. Here $b = \lfloor \sqrt{m+1}/2 \rfloor$, $b' = \lfloor (m+1)/(4b) \rfloor$ if $b \neq 0$, and $b' = 0$ if $b = 0$. K is given by $\{4bb' + 1, ..., m-2, m\}$.

In the case of $m = 19$, we get $b, b' = 2$ and $I = \{4, 12\}$, $J = \{1, 3\}$ $K = \{17, 19\}$. This gives for the sets $I + J = \{5, 7, 13, 15\}$ and $I - J = \{1, 3, 9, 11\}$, so indeed $S = K + (I \pm J)$.

Now the KPS algorithm is given by 5

Algorithm 5 KPS in $\sqrt{}$élu

Data: An elliptic curve $E/\mathbb{F}_q$ and a point $P \in E(\mathbb{F}_q)$ of order ℓ, an odd prime.
Result: An index system $\mathcal{I}, \mathcal{J}, \mathcal{K}$ for multiples of P.
$b \leftarrow \lfloor \sqrt{\ell - 1}/2 \rfloor$, $b' \leftarrow \lfloor (\ell - 1)/(4b) \rfloor$
$I = \{2b(2i+1) | 0 \leq i < b'\}$
$J = \{2j + 1 | 0 \leq j < b\}$
$K = \{4bb' + 1, ..., \ell - 4, \ell - 2\}$
$\mathcal{I} = \{x([i]P) | i \in I\}$
$\mathcal{J} = \{x([j]P) | j \in J\}$
$\mathcal{I} = \{x([k]P) | k \in K\}$

3.6.2 xISOG and xEVAL

Now we'll use the sets found in KPS to compute the coefficient A' of $E' = x^3 + A'x^2 + x$, and the x-coordinate α' of the point $\phi(Q) \in E'(\mathbb{F}_q)$, with $Q \in E(\mathbb{F}_q)$. In Algorithms 6 and 7 respectively the steps to compute these values are described. There are a few things to note regarding these computations. First of all, we need to compute resultants of polynomials over a finite field.

Definition 3.6.3. The *resultant* of two polynomials $f = a_n x^n + ... + a_0, g = b_m x^m + ... + b_0 \in \mathbb{F}[x]$ with roots $f(\alpha_i) = 0$ and $g(\beta_i) = 0$ respectively, is given by

$$\mathrm{Res}(f, g) = a_n^m b_m^n \prod_{i,j} (\alpha_i - \beta_j).$$

Note that the resultant is 0 if and only if f and g share a root.

Computing the resultant in general is quite tedious, but if the factorisation of $f = a \prod_{0 \leq i < n} (x - \alpha_i)$ is known, this can be done efficiently, and we can rewrite the formula as

$$\mathrm{Res}(f, g) = a^m \prod_{0 \leq i < n} g(\alpha_i).$$

The resultant is now given by first computing each $g(\alpha_i)$, which can be done using continued fractions (see 5.3.1) and then multiplying the parts together.

The main difference with evaluating polynomials in [44] is that the x-map of a coordinate is not a homomorphism. However, there exists a relation between $x(P), x(Q), x(P+Q)$, and $x(P-Q)$. This relation is described by

$$Z - x(P+Q))(Z - x(P-Q)) = Z^2 + \frac{F_1(x(P), x(Q))}{F_0(x(P, x(Q))} X + \frac{F_2(X(P), X(Q))}{F_0(x(P), x(Q))}.$$

The three polynomials in $\mathbb{F}[X, Y]$ are given by:

$$F_0(X, Y) = X^2 - 2XY + Y^2$$
$$F_1(X, Y) = -2(X^2 Y + (Y^2 + 2AY + 1)X + Y)$$
$$F_2(X, Y) = X^2 Y^2 - 2XY + 1.$$

Using this information, we now describe the algorithms to compute xISOG and xEVAL. The results are due to [8]. Note that there is a lot of overlap between the two algorithms, cheapening the computation.

Algorithm 6 xISOG in $\sqrt{}$élu

Data: An elliptic curve $E/\mathbb{F}_q$ and a point $P \in E(\mathbb{F}_q)$ of order ℓ - an odd prime, and $\mathcal{I}, \mathcal{J}, \mathcal{K}$ from KPS

Result: $A' \in \mathbb{F}_q$, with $E'/\mathbb{F}_q = x^3 + A'x^2 + x$ the image of $E/\langle P \rangle$.

$h_I \leftarrow \prod_{x_i \in \mathcal{I}}(X - x_i) \in \mathbb{F}_q[X]$

$E_{0,J} \leftarrow \prod_{x_j \in \mathcal{J}}(F_0(X, x_j) + F_1(X, x_j) + F_2(X, x_j)) \in \mathbb{F}_q[X]$

$E_{1,J} \prod_{x_j \in \mathcal{J}} \leftarrow (F_0(X, x_j) - F_1(X, x_j) + F_2(X, x_j)) \in \mathbb{F}_q[X]$

$R_0 \leftarrow \mathrm{Res}_X(h_I, E_{0,j}) \in \mathbb{F}_q$

$R_1 \leftarrow \mathrm{Res}_X(h_I, E_{1,j}) \in \mathbb{F}_q$

$M_0 \leftarrow \prod_{x_k \in \mathcal{K}}(1 - x_k) \in \mathbb{F}_q$

$M_1 \leftarrow \prod_{x_k \in \mathcal{K}}(-1 - x_k) \in \mathbb{F}_q$

$d \leftarrow \left(\frac{A-2}{A+2}\right)^{\ell} \left(\frac{M_0 R_0}{M_1 R_1}\right)^8$

$A' = 2\frac{1+d}{1-d}$

Algorithm 7 xEVAL in $\sqrt{}$élu

Data: An elliptic curve $E/\mathbb{F}_q$ and a point $P \in E(\mathbb{F}_q)$ of order ℓ - an odd prime, and $\mathcal{I}, \mathcal{J}, \mathcal{K}$ from KPS, the x-coordinate $\alpha \neq 0$ of a point $Q \in E(\mathbb{F}_q) \backslash \langle P \rangle$.

Result: The x-coordinate α' of a point $\phi(Q) \in E'(\mathbb{F}_q)$, with $E'(\mathbb{F}_q)$ the image of $E/\langle P \rangle$.

$h_I \leftarrow \prod_{x_i} \in \mathcal{I}(X - x_i) \in \mathbb{F}_q[X]$

$E_{0,J} \leftarrow \prod_{x_j \in \mathcal{J}}(F_0(X, x_j)/\alpha^2 + F_1(X, x_j)\alpha + F_2(X, x_j)) \in \mathbb{F}_q[X]$

$E_{1,J} \prod_{x_j \in \mathcal{J}} \leftarrow (F_0(X, x_j)\alpha^2 - F_1(X, x_j)\alpha + F_2(X, x_j)) \in \mathbb{F}_q[X]$

$R_0 \leftarrow \mathrm{Res}_X(h_I, E_{0,j}) \in \mathbb{F}_q$

$R_1 \leftarrow \mathrm{Res}_X(h_I, E_{1,j}) \in \mathbb{F}_q$

$M_0 \leftarrow \prod_{x_k \in \mathcal{K}}(1/\alpha - x_k) \in \mathbb{F}_q$

$M_1 \leftarrow \prod_{x_k \in \mathcal{K}}(\alpha - x_k) \in \mathbb{F}_q$

$\alpha' = \left(\frac{M_0 R_0}{M_1 R_1}\right)^2$

The costs of computing an isogeny with this method are described in [2], and amount in total to approximately

$$cost(b) = 4(9b^{\log_2(3)}(1 - 2\frac{2}{3}^{\log_2(b)+1}) + 2b\log_2(b)) + 3((1 - \frac{1}{3^{\log_2 b+1}}b^{\log_2(3)} + 37b + 3\log_2(b) + 16, \tag{3.17}$$

Where b is given in Section 3.6.1, and is given by $\lfloor \frac{\ell-1}{2}$ As described in [2], currently this method beats the traditional Velu formula when the isogeny has degree $\ell \geq 87$.

Remark 3.6.4. In some cases we will not work over the base field $\mathbb{F}_p$, but over $\mathbb{F}_{p^2}$. Multiplication in $\mathbb{F}_{p^2}$ equal 3 multiplications in $\mathbb{F}_p$. In $\mathbb{F}_{p^2}$ squaring has a cost of 0.8 multiplications, so equation 3.6 becomes a bit less than $3 \cdot cost(b)$ in $\mathbb{F}_p$. How much exactly it costs is not yet computed.

3.7 Weil pairing

Definition 3.7.1. Given a ring R, with R-modules M, N, L. A *pairing* is a R-bilinear map

$$e : M \times N \to L$$

The Weil pairing is a function we can compute on the torsion groups of elliptic curves. As we know, the group of m-torsion points of elliptic curve E has the form $E[m] \cong \mathbb{Z}/m\mathbb{Z} \times \mathbb{Z}/m\mathbb{Z}$.

Now take a point $T \in E[m]$, and a point T' such that $[m]T' = T$. Then there is a function $g \in \bar{K}(E)$ such that

$$\div(g) = \sum_{R \in E[m]} (T' + R) - (R). \tag{3.18}$$

Definition 3.7.2. A m^{th} *root of unity* in a field K is a number defined ζ such that $\zeta^m = 1$. ζ is a primitive root of unity if for all $1 \leq i$ *lem* $\zeta^i \neq 1$. The *group of* m^{th} *roots of unity* $\mu_{\mathbf{m}}$ is the cyclic group generated by a primitive m^{th} root of unity. $\mu_{\mathbf{m}} \subseteq \mathbf{K}^*$

For the proof of the last statement, see [51]III.8.1.1.

Definition 3.7.3. The *Weil pairing* is defined as

$$e_m : E[m] \times E[m] \to \mu_{\mathbf{m}} \tag{3.19}$$

with

$$e_m(S,T) = \frac{g(X+S)}{g(X)}, \text{ for any point } X \in E. \tag{3.20}$$

This function is constant, independent of the chosen $X \in E$, and maps to the roots of unity of m, $\mu_{\mathbf{m}}$.

Theorem 3.7.4. Given points $S, S_1, S_2, T, T_1, T_2 \in E[m]$ then

1. e_m is bilinear: $e_m(S_1+S_2,T) = e_m(S_1,T)+e_m(S_2,T)$, $e_m(S,T_1+T_2) = e_m(S,T_1)+e_m(S,T_2)$.
2. $e_m(T,T) = 1$.
3. If $e_m(S,T) = 1$ for all $S \in E[m]$, then $T = \mathcal{O}$.

Proof. See [51] III.8.1. □

Theorem 3.7.5. For m prime, given two points $S, T \in E[m]$, both not $\mathcal{O}$, S and T are

- if $\langle S,T\rangle = E[m]$, $e_m(S,T)$ is a primitive root of unity,
- linearly dependent if $e_m(S,T) = 1$.

Proof. If S and T are linearly dependent, then assume we can write S as $S = [n]T$. Theorem 3.7.4.1 gives $e_m(S,T) = e_m(T,T)^n$, and then Theorem 3.7.4.2 gives $e_m(S,T) = e_m(T,T)^n = 1$. Assume S,T generate $E[m]$ and $e_m(S,T)$ not a primitive root of unity. Then there must exist a $r < m$ such that $e_m(S,T)^r = 1$. using linearity $e_m(S,t)^r = e_m([r]S,T) = 1$. But note that for any $P \in E[m]$ we have $P = [a]S + [b]T$. This gives

$$e_m([r]S,P) = e_m([r]S,[a]S+[b]T) = e_m([r]S,[a]S) \cdot e_m([r]S,[b]T) = 1^{ar} \cdot e_m([r]S,T)^b = 1 \cdot 1^b = 1$$

Using 3.7.4.3 this gives that $[r]S = \mathcal{O}$. But S is part of a basis for $E[m]$ so $\text{ord}(S) = m \neq r$. This proofs that $e_m(S,T)$ is a primitive root of unity, and therefore also that $\text{ord}(e_m(S,T) = m$. □

Proposition 3.7.6. Let $\phi : E_1 \to E_2$ be an isogeny of elliptic curves. Then for all points $S \in E_1[m]$, $T \in E_2[m]$

$$e_m(S, \hat{\phi}(T)) = e_m(\phi(S), T)$$

Proof. See [51]III.8.2 □

Theorem 3.7.7. Given an isogeny $\phi : E \to E'$ of degree D and two points $P, Q \in E[m]$,

$$e_m(\phi(P), \phi(Q)) = e_m(P,Q)^D$$

Proof. Using Theorem 3.7.6 we get

$$e_m(P, \hat{\phi}(\phi(Q))) = e_m(\phi(P), \phi(Q))$$

and 2.3.6 gives $\hat{\phi}(\phi(Q)) = [\deg(\phi)]Q$, so applying 3.7.4.1 this gives

$$e_m(\phi(P), \phi(Q)) = e_m(P, \hat{\phi}(\phi(Q))) = e_m(P, [m]Q) = e_m(P,Q)^{\deg(\phi)}$$

□

Miller's algorithm With the information above, it is hard to compute a Weil pairing for two points. Miller [39] constructed an algorithm to efficiently compute the Weil pairing of two points. For this an alternative definition of the pairing is required. See [51], Exercise 3.16 for this definition.

Define a function $h_{P,Q}$ as follows, for two points P, Q on an elliptic curve $E : y^2 + a_1xy + a^3y = x^3 + a_2x^2 + a_4x + a_6$, with $P = (x_p, y_P), Q = (x_Q, y_Q)$, and λ the slope of the tangent line between P and Q:

$$h_{P,Q} = \begin{cases} \frac{y - y_p - \lambda(x - x_p)}{x + x_P + x_Q - \lambda^2 - a_1\lambda + a_2}, \text{ if } \lambda \neq \infty \\ x - x_P \text{ if } \lambda = \infty \end{cases}$$

Then the algorithm is defined as:

Algorithm 8 Miller's algorithm

Data: Two points $S, T \in E[m]$
Result: A root of unity $x \in \mu_{\mathbf{m}}$
$S = P, T = Q$ and $f = 1, g = 1$
for $i = t - 1$ *to* 0 **do**
  $f = f^2 \cdot h_{S,S}$
  $P = 2P$
  if $\epsilon = 1$ **then**
    $f = f \cdot h_{S,P}$
    $S = S + P$
  end
end
Repeat for T and g.
Pick a point $R \in E$ such that $R \notin \langle T, S \rangle$
$e_m = \frac{f(R+T)}{f(R)} / \frac{g(S-R)}{g(-R)}$

4. Cryptography on Elliptic Curves

In this Chapter we first introduce the Elliptic Curve Diffie Hellman key exchange, and then discuss different aspects of Supersingular Isogeny Diffie Hellman.

4.1 Elliptic Curve Diffie Hellman key exchange protocol

Another, more secure, way of obtaining a shared secret key for two persons using the Diffie Hellman key exchange is using addition of points on an elliptic curve instead of computing powers of numbers modulo a prime. This scheme is called elliptic curve Diffie Hellman and is usually shortened to ECDH. ECDH is first proposed by Miller [39] and is now widely implemented worldwide. In this section we will explain how we can use elliptic key addition to obtain a shared secret key. Alice and Bob want to create a shared secret key using the elliptic curve group operation. First they agree on an elliptic curve E/K to work on. This key is usually part of public parameters and optimised to offer good security. For more info on preferred elliptic curves for ECDH, see [3]. They then agree on a point $P = \langle x, y \rangle$ that generates a subgroup of $E[K]$

Now Alice and Bob pick a secret random integer, a respectively b, and calculate their public keys, $Q_A = [a]P$ and $Q_B = [b]P$. They then exchange their public keys. Multiplying by b, respectively a, provides them with their secret shared key $[ab]P$. Retrieving a from only Q_A and the given parameters is a harder problem than solving a discrete logarithm in a cyclic group, see [39]. The ECDH protocol is described below in Figure 4.1 using a commutative diagram.

$$\begin{array}{ccc} P & \xrightarrow{\times a} & Q_A = [a]P \\ \Big\downarrow{\scriptstyle \times b} & & \Big\downarrow{\scriptstyle \times b} \\ Q_B = [b]P & \xrightarrow[\times a]{} & Q_{AB} = [ab]P \end{array}$$

Figure 4.1: The commutative diagram of the Elliptic Curve Diffie-Hellman key exchange protocol.

It should be noted that these examples, among all other 'textbook' explanations of cryptographic protocols, should never be implemented in the form they are explained in this text. This way they are vulnerable to all kinds of attacks. Security measures like for instance *padding* and *hashing* should always be taken to prevent any adversary to take advantage of the public data.

4.2 Supersingular Isogeny Diffie Hellman

As described in Section 1.2, with the prospect of life-scale, functioning quantum computers, there is the need for more advanced assymetric key cryptography to secure electronic communication. One of the suggestions for a new cryptographic scheme is based on supersingular isogeny Diffie

Hellman (SIDH). In this section we will explain how this protocol works and prove its correctness. SIDH lies at the basis of the NIST proposal SIKE. Using supersingular curves for cryptographic uses is first proposed in [22], and the first properly working key exchange protocol is described in [25], by Jao and DeFeo.

As described in Section 1.3.1, the Shor algorithm exploits periodic functions, like elliptic curve addition. SIDH therefore does not work with a periodic function, but uses the hardness of computing large degree isogenies as explained in 4.2.1. Security aspects of SIDH will be discussed in Chapter 7.

4.2.1 SIDH protocol

The first step for Alice and Bob is to agree on a supersingular elliptic starting curve E_0 over a field $\mathbb{F}_q$, with $q = p^2$. p is a prime of the form $p = \ell_A^{e_A}\ell_B^{e_B}f \pm 1$, with f a cofactor to make p prime. The curve will have cardinality $(\ell_A^{e_A}\ell_B^{e_B}f)^2$. In the SIKE submission, ℓ_A is taken to be 2, and ℓ_B to be 3, and the curve is taken to be $y^2 = x^3 + 6x^2 + x$. In Theorem 3.1.5 it is described how to test if a curve is supersingular.

Besides the curve E_0, the public parameters also include points, P_A and P_B, Q_A and Q_B, such that $\langle P_A, Q_A\rangle = E_0[\ell_A^{e_A}]$ and $\langle P_B, Q_B\rangle = E_0[\ell_B^{e_B}]$. Thus the points are chosen to generate the $\ell_A^{e_A}$- and $\ell_B^{e_B}$-torsion groups.

Alice then chooses her secret integers m_A and n_A, not both divisible by $\ell_A^{e_A}$, so that $R_A = [m_a]P_A + [n_A]Q_A$ has order $\ell_A^{e_A}$. She computes an isogeny $\phi_A : E_0 \longrightarrow E_A$ with $E_A \cong E_0/\langle R_A\rangle$, as according to Theorem 2.3.5 every subgroup A of E_0 gives way to a unique curve $E_0/A = E_A$ and a unique isogeny between them. Bob acts *mutatis mutandis*. How to compute the isogenies between elliptic curves is described in the next section. Alice also computes the images of P_B and Q_B on E_A

Alice and Bob exchange their public keys; curve E_A, and the points $\phi(P_A), \phi_A(Q_B)$, are send to Bob, Alice receives $E_B, \phi_B(P_B), \phi_B(Q_A)$. The isogenies ϕ_A, ϕ_B and the points R_A, S_A stay private.

In the last phase Alice computes an isogeny $\phi'_A : E_B \longrightarrow E_{AB}$ with kernel $[m_a]\phi_B(P_A) + [n_A]\phi_B(Q_A)$. Bob proceeds in the same way to generate $E_{BA} \cong E_{AB}$. They can then use the common j-invariant of E_{AB} as a secret shared key, since isomorphic curves have the same j-invariant.

The SIDH protocol is explained in schematic form in Figure 4.2.

SIDH Key Exchange protocol

Public parameters :

$E_0,\ p,\ \ell_A,\ \ell_B,\ P_A,\ P_B,\ Q_A,\ Q_B$

Alice		**Bob**
$R_A = [m_A]P_A + [n_A]Q_A$		$R_B = [m_B]P_B + [n_B]Q_B$
$\phi_A : E \to E_A = E/\langle R_A \rangle$		$\phi_B : E \to E_B = E/\langle R_B \rangle$
	$\xrightarrow{E_A, \phi_A(P_B), \phi_A(Q_B)}$	
	$\xleftarrow{E_B, \phi_B(P_A), \phi_B(Q_A)}$	
$E_{AB} = E_B/\langle [m_A]\phi_B(P_A), [n_A]\phi_B(Q_A) \rangle$		$E_{BA} = E_A/\langle [m_B]\phi_A(P_B), [n_B]\phi_A(Q_B) \rangle$
Output: $j(E_{AB})$		**Output:** $j(E_{BA})$

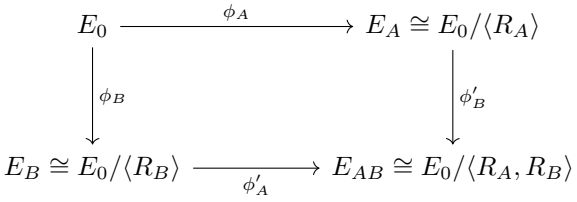


Figure 4.2: The SIDH protocol explained schematically.

One difference that is immediately noticeable when looking at this key-exchange, is that there is more information exchanged than in normal Diffie Hellman key exchanges. Besides the curves E_A and E_B, Bob and Alice must also exchange the auxiliary points $\phi(P_A), \phi_A(Q_B)\phi_B(P_B), \phi_B(Q_A)$. The reason for this is that the endomorphism ring is not commutative, as proven in Theorem 3.1.3. This unfortunately makes the public key larger and may open this protocol to attacks. Proposals to use ordinary elliptic curves, with commutative endomorphism rings, are also made, but have thus far always proven vulnerable to attacks [9], [14]. In the next section the proof is given that with these extra auxiliary points, SIDH is indeed a commutative protocol.

4.3 Commutativity

Theorem 4.3.1. The curves E_{AB} and E_{BA} are isomorphic.

Proof. According to theorem 2.3.5.4 the curve E_{AB} is isomorphic to

$$E_{AB} \cong E_B/\langle [m_a]\phi_B(P_A) + [n_A]\phi_B(Q_A) \rangle.$$

Theorem 2.3.5.2 gives

$$\begin{aligned}[m_a]\phi_B(P_A) + [n_A]\phi_B(Q_A) &= \phi_B([m_a]P_A) + \phi_B([n_A]Q_A) \\ &= \phi_B([m_a]P_A + [n_A]Q_A) = \phi_B(R_A). \quad (4.1)\end{aligned}$$

Therefore

$$E_{AB} \cong E_B/\langle \phi_B(R_A) \rangle.$$

In the same way it follows that

$$E_B \cong E_0/\langle R_B \rangle.$$

Therefore we have that the kernel of

$$(\phi'_A \circ \phi_B) : E_0 \longrightarrow E_{AB} \quad \text{is } \langle R_A, R_B \rangle.$$

Since

$$E_{BA} \cong E_A/\langle [m_B]\phi_A(P_B) + [n_A]B\phi_A(Q_B) \rangle = E_A/\langle \phi_A(R_B) \rangle,$$

and $E_A = E_0/\langle R_A \rangle$, this gives for the kernel of

$$(\phi'_B \circ \phi_A) : E_0 \longrightarrow E_{BA}$$

(which is again an isogeny since both ϕ'_B and ϕ_A are surjective, making $\phi'_B \circ \phi_A$ a surjective homomorphism between elliptic curves, and therefore an isogeny) that

$$\ker((\phi'_B \circ \phi_A) : E_0 \longrightarrow E_{BA}) = \langle R_B, R_A \rangle.$$

Since $\langle R_B, R_A \rangle = \langle R_A, R_B \rangle$, it follows that the kernels of $\phi'_B \circ \phi_A$ and $\phi'_A \circ \phi_B$ are the same. The only thing left to prove is that the kernels are subgroups of E_0, then it follows from Theorem 2.3.5.4 that $\phi'_B \circ \phi_A = \phi'_A \circ \phi_B$, and that $E_0/\langle R_A, R_B \rangle$ again is an elliptic curve. We have

$$\text{order}(R_A) = \ell_A^{e_A} \neq \ell_B^{e_B} = \text{order}(R_B),$$

thus $\langle R_A \rangle + \langle R_B \rangle = \langle R_A, R_B \rangle$ is a subgroup of E_0. This proves that $E_{AB} \cong E_{BA}$. □

4.4 Computing Isogenies

As proven in Theorem 2.3.5.1, the size of the kernel of the isogeny is the degree of the isogeny. To compute the curve E_A we will have to compute a degree $\ell_A^{e_A}$ isogeny. As described in Sections 3.5 and 3.4, computing isogenies can be done using Vélu's formulas or by using special properties of Montgomery curves. In both scenarios, the costs of computing an isogeny increase significantly by increasing the degree of the isogeny. As shown in Theorem 2.3.5.3, there is a solution to this. We can split the isogeny into smaller isogenies, the size of ℓ_A, as long as the kernel of each isogeny is contained in the kernel of the larger isogeny. In [25] Jao and DeFeo describe a method to compute these isogenies, using multiplication of the kernel generator R. For any isogeny of degree ℓ^e, we can compute the isogeny sequentially using the following calculations.

$$E_{i+1} = E_1/\langle \ell^{e-i-1} R_i \rangle \tag{4.2}$$

$$\phi_i : E_i \to E_{i+1} \tag{4.3}$$

$$R_{i+1} = \phi_i(R_i) \tag{4.4}$$

A schematic overview of this method is shown in Figure 4.3.

To optimise the amount of computations, not all edges on this graph should be computed. In Section 4.2 it is shown that to calculate a new isogeny ϕ_{i+1}, we need the curve E_{i+1} and the generator of the kernel $\ell^{e-i} R_{i+1}$. So we need only the two outermost lines of the graph, and the nodes in the lowermost row. In [25] it is described how to find a strategy that uses the least amount of edges, and therefore computations, to find all these points. Assuming multiplying and computing an isogeny take an equal amount of computation, the best way of computation is a *balanced* strategy. Asymptotically this would require $\frac{1}{2 \log 2} n \log n$ left and right edges. If there is a discrepancy in cost between multiplication and isogeny computation, an optimal strategy gives the cheapest way of computing a large isogeny, balancing the amount of multiplications and isogeny computations required.

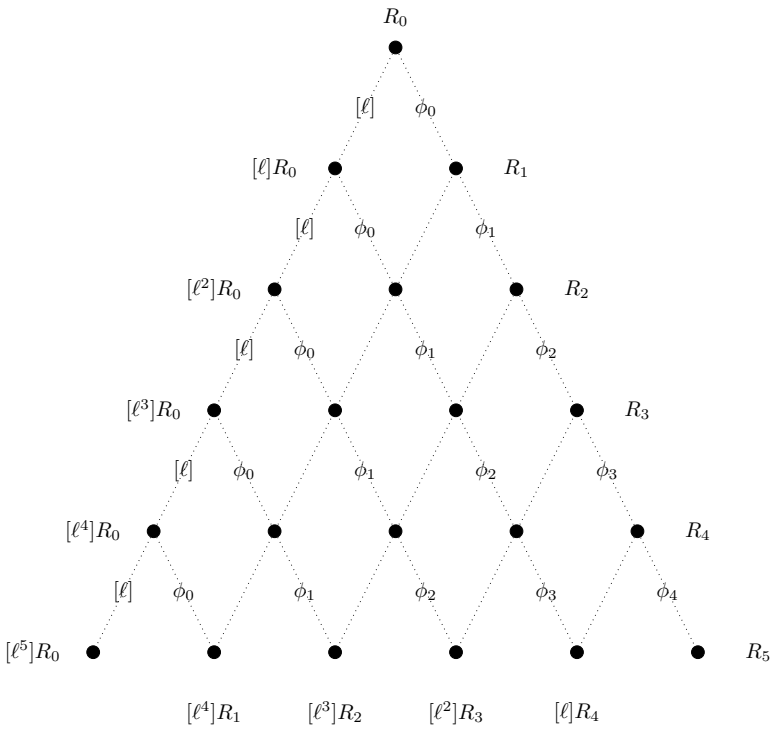


Figure 4.3: Computational structure of the construction of $\phi = \phi_4 \circ ... \circ \phi_0$

Optimal strategy Jao and De Feo describe in [25] an optimal strategy for computing isogenies of degree p^n. In [25] Lemma 4.5, it is proven that the cost for an optimal strategy of length n is

$$C_{p,q}(n) = \min_{i=1,..,n-1}(C_{p,q}(i) + C_{p,q}(n-i) + (n-i)p + iq). \tag{4.5}$$

Here p is the cost of one multiplication, q the cost of one isogeny, and n the length of the strategy (equal to the degree of the isogeny). Equation 4.5 is recursive. It takes the input of smaller triangles to compute the cost of the larger triangle. Here a triangle is a triangular subgraph with it's outer edges having a ramification on the outer edge of the larger triangle.

In [56] they give an efficient algorithm to compute the optimal strategy for any given n, p, q using Equation 4.5. This is described in Algorithm 9. The result of the algorithm is a strategy S, with S a list of length n, of values $s_i \in \{0, ..., n-1\}$ such that each s_i indicates how many multiplicative edges we have to take before we encounter a ramification or a leaf on the strategy drawn over the graph in Figure 4.3.

Algorithm 9 Optimal strategy

Data: The length of the strategy n, cost of multiplication p, cost of isogeny computation q
Result: An optimal strategy S with cost C

```
C, P = [1, ..., n+1] for k = 2 to e do
    j = 1, z = k − 1
     while j < z do
        m = j + ⌊(z − j)/2⌋
         w = m + 1
         t1 = C[m] + C[k − m] + (k − m) · p + m · q
         t2 = C[w] + C[k − w] + (k − w) · p + w · q
         if t1 ≤ t2 then
            z = m
             else
                j = w
            end
        end
    end
    C[k] = C[j] + C[k − j] + (k − j) · p + j · q
     S[k] = j
end
return S
```

So we end up with an optimal strategy given as a list P of integers a. The 2^n-degree isogeny can then be computed as described in Algorithm 10.

Algorithm 10 General overview of SIDH key exchange protocol

Data: A kernel point R_A of order 2^n, a strategy S of length $n-1$ with total amount of isogenies s a starting curve $E : y^2 = x^3 + Ax^2 + x$ and points P_B, Q_B, PQ_B.
Result: The constants A', C' that define the curve $E_A = E_0/E_A$, the images of P_B, Q_B, PQ_B on E_A.

```
for i is 1 to s do
    while j < s − i do
        m = S[s − i − j + 1]
         T = [2^m]R
         j+ = m
    end
    I, J, K = KPS(T, A, C)
     A, C = xISOG_2(T, A, C, I, J, K)
     R, P_B, Q_B, PQ_B = xEVAL_2(R, P_B, Q_B, PQ_B, A, C, I, J, K)
end
I, J, K = KPS_2(T, A, C)
 A, C = xISOG_2(T, A, C, I, J, K)
 P_B, Q_B, PQ_B = xEVAL_2(P_B, Q_B, PQ_B, A, C, I, J, K)
 A = A/C
```

4.5 Security basis of SIDH

The hardness of the SIDH protocol can be stated as two problems, that are given below.

1. (Supersingular isogeny problem) Given a finite field K and two supersingular elliptic curves E_1, E_2 defined over K such that $\#E_1 = \#E_2$, compute an isogeny $\phi : E_1 \to E_2$.

2. (Endomorphism ring computation) Given an elliptic curve E defined over a finite field K, compute its endomorphism ring.

There is an equivalence of categories between the set of supersingular curves and the set of maximal orders of a quaternion algebra, [36] Theorem 45. This means that, at least heuristically, problems 1 and 2 are the same, meaning that one can turn an algorithm that computes isogenies into an algorithm that computes the full endomorphism ring of an elliptic curve, and vice versa. Using the birthday paradox, there exists an algorithm for computing any endomorphism ring in $O(\sqrt{p})$ time, see [29].

Since in the protocol auxiliary points are sent, SIDH is vulnerable to active attacks [30]. In this paper it is described how by modifying the information, the information returned leaks the final bits of the secret key. A modification of this protocol that adapts this to B-SIDH is described in Section 7.

Another option would be to view the security of SIDH as a random walk on an isogeny graph, as described in Section 3.3. The drawback here is that for a walk to be random enough, as shown in Theorem 3.3.7, it should have length of order p. In the SIDH protocol the random walks are of length $\sqrt{p}$. Security claims made based on randomness of the random walk are therefore not strong. B-SIDH does not have the same issue as it uses walks of length p, as will be shown in the Chapter 6.

4.6 CSIDH

SIDH was not the first Isogeny-based cryptographic protocol proposed. The first instance of using isogenies in cryptography was independently found by Couveignes [22] and Rostovtsev and Stolbunov [49]. These proposals work over a field $\mathbb{F}_q$ of ordinary elliptic curves. As described in Section 3.1.1 In ordinary elliptic curves, the ideal-class group $\mathrm{cl}(\mathcal{O})$ is commutative and acts freely and transitively on the set of elliptic curves that have $\mathcal{O}$ as its endomorphism ring. While the results seemed promising, there were two major drawbacks: first, it can be solved by the abelian hidden shift problem. Secondly, it is extremely slow. Completing a 128-bit secure key exchange takes minutes on normal computers. In 2018, a new key-exchange was proposed, CSIDH, [**CSIDH**], based on the CRS method, that adapts this method to supersingular curves and uses a part of the endomorphism ring that is commutative as described in Section 3.1.1, so one can use it as a commutative scheme. The main benefits compared to SIDH are that there is no need to using auxiliary points, which may prove vulnerable to attacks, and that it can be used as a static key exchange. Its keys are smaller, and it is easy to validate public keys without fear for the active attack described in Chapter 7. In this section we will briefly describe the CSIDH protocol.

First we note that the Hard Homogenous Spaces as described in Section 2.5 give way to a Diffie-Hellman style key exchange: Alice and Bob agree on a element in a set $s \in S$. They have private keys $a, b \in G$. Their public keys are given by $a * s$ and $b * s$, their shared secret key is $ab * s = a(b * s) = b(a * s)$. The hardness assumptions given in Section 2.5 make that this is indeed a good Diffie-Hellman key exchange.

As described in Section 3.1.1, we can see the isogeny class action

$$\begin{aligned} \mathrm{cl}(\mathcal{O}) \times \ell_p(\mathcal{O}, \pi) &\to \ell_p(\mathcal{O}, \pi) \\ [\mathfrak{a}] * E &\mapsto [a] * E \end{aligned}$$

as a free and transitive action. Also, from [12] it follows that E is a supersingular curve with $\mathrm{cl}(\mathcal{O})$ as a subgroup of its endomorphism ring. Together with hardness properties as described in [12] that are required as stated in Section 2.5, this makes that we can see the ideal class action as described in Section 3.1.1 as a hard homogenous space. We can thus use it as a Diffie Hellman key exchange protocol. Below we will explain shortly the CSIDH protocol. For more details we refer to to the original CSIDH paper [12].

Any ideal $\mathfrak{a} \in \mathrm{cl}(\mathcal{O})$, can be represented as a product of small prime ideals, as described in [12]. Here it is also described that we can use Velu's formulae of Section 3.5 to compute a larger ideal

$\mathfrak{l} \in \mathrm{cl}(\mathcal{O})$ if we know its composition into smaller prime ideals. Now to start the key exchange, we first pick a large prime $p = 4 \cdot \ell_1 \cdot ... \cdot \ell_n - 1$ where the ℓ_i are small distinct odd primes. It can be proven that all these primes are Elkies primes. We pick a starting curve $E_0/\mathbb{F}_p$ with a subring of the endomorphism ring equal to $\mathbb{Z}[\pi]$, where π is the Frobenius morphism. We can then find ideals $\mathfrak{l}$ in the ideal-class group of the following form

$$\mathfrak{l} = \prod_i \mathfrak{l}_i^{e_i}.$$

We can expect different ideals of this form but with different parameters e_i to be in different classes of $\mathrm{cl}(\mathcal{O})$ almost always (see [12]). Assume we pick the e_i from a range $\{-m, ..., m\}$. We can then describe the ideal $\mathfrak{l}$ as a vector $(e_1, .., e_n)$.

For the key exchange, we now have a prime p and a starting curve E_0. For private keys, Alice and Bob pick a secret vector $(e_1, ..., e_n)$ and $(e'_1, .., e'_n)$ from a range $\{-m, ..., m\}$. They are then able to compute ideal classes $[\mathfrak{a}], [\mathfrak{b}]$ with $[\mathfrak{a}] = [\mathfrak{l}_1^{e_1}, .., \mathfrak{l}_n^{e_n}]$. Alice and Bob now compute $E_a = [\mathfrak{a} * E]$ and $E_b = [\mathfrak{b} * E]$. Their public keys are A and B, the unique values representing their curves as given in Theorem 3.1.11. To verify that the keys they received from eachother are honestly generated, they can verify if the curve of the other lies in $\ell_p(\mathcal{O}, \pi)$. An algorithm for this is given in [12]. For the shared secret key, they compute $E_1 = [\mathfrak{a}][\mathfrak{b}] * E_0 = [\mathfrak{a}]E_b$. The secret key is then the value S representing E_1. The key is the same since $\mathrm{cl}(\mathcal{O})$ is commutative and S is unique due to Theorem 3.1.11.

There are similarities and differences between CSIDH and SIDH. It uses the same Velu formula to compute its keys, but needs no auxiliary points to make the diagram commute. We will see in the Chapter 6 that there are even more similarities between B-SIDH and CSIDH, where both protocols work with different small prime degree isogenies.

5. Smooth primes

This chapter treats a specific kind of prime numbers, B-smooth primes, and describes a method to systematically find these prime numbers, that has not been used before in cryptography to find new smooth prime numbers. The definition for a B-smooth prime is derived from B-smooth numbers. A number is called *B-smooth* if it has no prime divisors larger than B. A number is often called *smooth* if it is B-smooth for a sufficiently small B.

Definition 5.0.1. A B-smooth prime is a prime number p such that both $p-1$ and $p+1$ are B-smooth.

For instance, $p = 10635661441913127573799$ is a 150-smooth prime, as $p+1 = 2^3 \cdot 5^2 \cdot 7^9 \cdot 11^2 \cdot 19^2 \cdot 41^2 \cdot 131 \cdot 137$, and $p-1 = 2 \cdot 3^4 \cdot 13^4 \cdot 29 \cdot 67 \cdot 71 \cdot 73 \cdot 103 \cdot 107 \cdot 139 \cdot 149$.

The task of finding large smooth primes is not an easy one. The fastest methods currently known first find *smooth neighbours*. A pair of numbers $m, m+1$ is a *B-smooth neighbour pair* if both m and $m+1$ are B-smooth. A smooth prime numbers is found by multiplying the smooth neighbours by two and checking if $2m+1$ is a prime number. Large smooth neighbours are rare, as described in Section 5.1, and there is no efficient way of computing them. In literature, there exist complete lists up to 200-smooth neighbours. Smooth neighbours can be found solving Pell equations, as described in Section 5.3 below. The drawback of this is that the amount of computations needed grows exponentially as the smoothness bound increases. In this paper we explore another option of finding smooth neighbours using the extended smooth neighbour technique.

5.1 Prevalence of smooth neighbour pairs

If we want to find a smooth neighbour pair m, $m+1$, we can translate this to finding a smooth number $m \cdot (m+1)$. Finding such a pair depends on the availability of B-smooth numbers.

Definition 5.1.1. The amount of B-smooth numbers m, with m at maximum N, is given by

$$\Psi(N, B) = \#\{1 \leq m \leq N : m \text{ is } B\text{-smooth.}\}.$$

There is no exact function to compute $\Psi(N, B)$ for all $N, B \in \mathbb{Z}$. There are multiple estimations on $\Psi(N, B)$ in literature, of which the Dickman-rho function as given in Theorem 5.1.2 below, is one of the most used.

Theorem 5.1.2 (Dickman-rho function)**.** For the amount of smooth numbers $\Psi(N, B)$ there exists a function $\phi(u)$ such that

$$\frac{\Psi(N, N^{1/u})}{N} \sim \rho(u) \text{ as } N \to \infty.$$

For $u > 1$ $\phi(u)$ is defined by

$$\rho'(u) = -\frac{\rho(u-1)}{u}$$

B	$\Psi(2^{128}, B)$	$\Psi(2^{129}, B)$	$\Psi(2^{129}, B) - \Psi(2^{128}, B)$	$(\Psi(2^{129}, B) - \Psi(2^{128}, B))/2^{128}$
100	$1.26263 \cdot 10^{11}$	$4.95798 \cdot 10^{11}$	$3.69534 \cdot 10^{11}$	$\cdot 1.08596 \cdot 10^{-27}$
200	$1.58239 \cdot 10^{16}$	$1.76536 \cdot 10^{16}$	$1.82970 \cdot 10^{15}$	$5.37705 \cdot 10^{-24}$
300	$2.55764 \cdot 10^{18}$	$3.07853 \cdot 10^{18}$	$5.20891 \cdot 10^{17}$	$1.53076 \cdot 10^{-21}$
400	$5.75545 \cdot 10^{19}$	$7.06651 \cdot 10^{19}$	$1.31105 \cdot 10^{19}$	$3.85285 \cdot 10^{-20}$
500	$5.19236 \cdot 10^{20}$	$6.54362 \cdot 10^{20}$	$1.35078 \cdot 10^{20}$	$3.96960 \cdot 10^{-19}$
600	$2.75653 \cdot 10^{21}$	$3.51636 \cdot 10^{21}$	$7.59831 \cdot 10^{20}$	$2.23294 \cdot 10^{-18}$
800	$3.12884 \cdot 10^{22}$	$4.09757 \cdot 10^{22}$	$9.68738 \cdot 10^{21}$	$2.84686 \cdot 10^{-17}$
1000	$1.73856 \cdot 10^{23}$	$2.83710 \cdot 10^{23}$	$1.09854 \cdot 10^{23}$	$3.22832 \cdot 10^{-16}$
2000	$1.72242 \cdot 10^{25}$	$3.42890 \cdot 10^{25}$	$1.70648 \cdot 10^{25}$	$5.01489 \cdot 10^{-14}$
5000	$2.02014 \cdot 10^{27}$	$4.02079 \cdot 10^{27}$	$2.00064 \cdot 10^{27}$	$5.87937 \cdot 10^{-12}$
10000	$3.59283 \cdot 10^{28}$	$7.17787 \cdot 10^{28}$	$3.58503 \cdot 10^{28}$	$1.05354 \cdot 10^{-10}$
15000	$1.54649 \cdot 10^{29}$	$3.06992 \cdot 10^{29}$	$1.52342 \cdot 10^{29}$	$4.47693 \cdot 10^{-10}$

Table 5.1: Estimation of the amount of B-smooth prime numbers between $a = 2^{128}$ and $b = 2^{129}$ and the probability that a number between a and b is B-smooth for different values of B.

Proof. See [23] Theorem 1.4.9. □

We can make an estimate for the upper bound of the chance that $m+1$ is B-smooth when we know m is B-smooth as follows. We compute

$$C = (\Psi(b, B) - \Psi(a, B))/(b - a)$$

for a, b suitable close to each other, and such that $a \leq m \leq b$. C then gives us the probability that any number $m \in [a, .., b]$ is B-smooth, so in particular, it gives an upper bound for the chance of $m + 1$ being B-smooth if we know m is B-smooth. In an ideal situation, we would be able to compute $\Psi(a, B)$ and $\Psi(b, B)$ as exact as possible, as we would have

$$D = \#B\text{-smooth } m = \Psi(b, B) - \Psi(a, B), \text{ for } a \leq m \leq b.$$

But as described above, we unfortunately only have approximations for $\Psi(b, B)$. Table 5.1 shows computed values using the Dickman-rho function to give an estimation for C and D for different values of B, taking $a = 2^{128}$ and b^{129} to give an upper bound on how hard it is to find a B-smooth neighbour pair in the range of 2^{128}. The values are computed using the **dickman rho** function of Sage. We can see that the chances are quite small for encountering B-smooth numbers, as expected, but the changes increase rapidly when B-gets larger. For $B \sim 15000$, a smooth prime of bitlength 253 has been found, see Section 5.4. Up until $B = 100$, (most probably) all smooth neighbour pairs are published in [38], and we know no smooth neighbour pair of around size 2^{128} exists, the largest pair having bitlength 65.

What Table 5.1 shows us is that the chance of finding a B-smooth number is small, and increases significantly when we make B larger. It is hard to say how the numbers given in Table 5.1, an upper bound for the amount of smooth neighbour pairs, relate to the actual amount of smooth neighbour pairs. But these numbers can give us a comparison on how hard it is to find a B-smooth neighbour pair in relation to other smoothness bounds. We can also clearly see from these upper bounds that finding a B-smooth neighbour pair is not an easy task, and it may very well be that throughout the years improved methods for finding them will be discovered, and larger smooth primes be found.

5.2 Use in cryptography

As explained in section 3.2, for every supersingular elliptic curve E defined over $\mathbb{F}_{p^2}$, there exists a twist E', such that $E(\mathbb{F}_{p^2}) = \mathbb{Z}_{p-1} \times \mathbb{Z}_{p-1}$ and $E'(\mathbb{F}_{p^2}) = \mathbb{Z}_{p+1} \times \mathbb{Z}_{p+1}$ or vice versa. To feasibly compute isogenies using both twists, $p - 1$ and $p + 1$ must both be smooth to a certain extent, as

isogeny computations become a lot more expensive as their degree becomes larger, as described in 3.6. [16] proposes a cryptographic protocol called B-SIDH using both twists, and will be discussed in section 6. The advantage is that in SIDH the torsion groups have order $\approx \sqrt{p}$, and in B-SIDH the torsion groups have order $\approx p$, meaning that quadratically smaller primes can be used to achieve the same level of security.

Thus, for the B-SIDH protocol to be applicable in real life cryptography, a smooth prime p with bitlength around 250 is needed. There is no strict smoothness bound for the prime, but as shown in Section 3.6, the costs of computing a larger degree isogeny grow in size quite quick, with costs of different isogeny degrees given in Section 6.7. Ideally one would therefore have prime degrees of maximum smoothness bound $B = 1000$. No such prime is known at this moment. We tried to find a large enough prime using the extending neighbours method as described in Section 5.5. The results are described in Section 5.8. Before we go to the extending neighbours method, we first describe two other methods that can be used to find prime numbers, Lenstra's method and the PTE method.

5.3 Lenstra's method

A standard way of computing smooth neighbours is by using Størmer's theorem [52]. This theorem proves that for every smooth bound B, there are only finitely many pairs $S, S+1$ that are B-smooth. Also there is an explicit way to find all of these numbers using the Pell equation.

Theorem 5.3.1 (Størmer)**.** Let $q_1 < q-2 < ... < q_m$ be a given set of m primes, and let Q be the set generated by them. Let Q' be the subset of all square-free members of Q. Let S be an integer such that both S and $S+2$ belong to Q. Then $S = x_n - 1$ where (x_n, y_n) is a solution of the Pell equation

$$x^2 - D \cdot y^2 = 1 \tag{5.1}$$

in which $1 < D \in Q$, $1 \leq n \leq \frac{q_m - 1}{2}$, $y_n \in Q$.

Proof. See [52]. □

To solve the Pell equation, a normal method is to use continued fraction, as described by [37].

5.3.1 Continued fraction

A continued fraction is a way of representing a number $r \in \mathbb{R}$ by an infinite fraction.

Any square root $\sqrt{x}$ can be written as follows

$$\sqrt{x} = 1 + \frac{x-1}{1+\sqrt{x}}$$

This gives for a continued fraction

$$\sqrt{x} = 1 + \cfrac{x-1}{2 + \cfrac{x-1}{2 + \cfrac{x-1}{2+\cdots}}}$$

Continued fractions for square roots are cyclic, which means that after a certain period the fractions start to repeat themselves. This is the case since the group of units of $\mathbb{Z}[\sqrt{d}]$ is the product of ± 1 and an infinite cyclic group. For $\sqrt{14}$, the cycle has length 4 as is shown in 5.2

$$\sqrt{14} = 3 + \cfrac{1}{1 + \cfrac{1}{2 + \cfrac{1}{1 + \cfrac{1}{3+\sqrt{14}}}}} \tag{5.2}$$

To find a rational approximation for $\sqrt{x}$, one can truncate the continued fraction after the first cycle. For the case $x = 14$, this would mean setting $\frac{1}{3+\sqrt{14}} = 0$ in the lowest fraction. This gives $\frac{15}{4} = 3.75$ as an approximation for $\sqrt{14} \approx 3.7416$. This first solution is called the *fundamental solution*. Other, more precise, solutions can be found by truncating at a later point in the continued fraction.

5.3.2 Pell equation

To solve a Pell equation $x^2 - D \cdot y^2 = 1$, one can rewrite the equation to $(x + \sqrt{d}y)(x - \sqrt{d}y) = 1$. Then applying the continued fraction method as described in the previous section, gives a fraction $\frac{a}{b}$ as an approximation for $\sqrt{d}$, and by setting $x = a$, $y = b$ this gives the first solution, the fundamental solution, to the Pell equation. Indeed $15^2 - 14 \cdot 4^2 = 1$. All other equations can be found using the following

$$x_n + y_n\sqrt{d} = (x_1 + y_1\sqrt{d})^n$$

Lenstra's algorithm using continued fraction to solve the Pell equation is given in algorithm 11

Algorithm 11 Lenstra's Simple Continued Fraction method

Result: A list of numbers x_i such that x_i and $x_i + 1$ are B-smooth
Create list of primes $X = \{q_i := q_1 < q_2 < ... < q_n = B, q_i \text{ prime}\}$
for q_i *in* X **do**
  $D = q_1^{j_1} \cdot ... \cdot q_n^{j_n}$, for $j_i \in \{0, 1\}$
  Solve Pell equation $x^2 - D \cdot y^2 = 1$ to obtain (x_1, y_1)
  $x^2 - D \cdot y^2 = (x + \sqrt{d}y)(x - \sqrt{d}y)$ Write $\sqrt{d}$ as continued fraction until first cycle is reached.
  Replace $\sqrt{d}$ with 1 in the fraction, set fraction as $\frac{a}{b}$. $(x_1, y_1) = (a, b)$ **for** $p < q_n/2$ **do**
    $x_p = x_1x_k + ny_1y_k$
    $y_p = x_1y_k + y_1x_k$
    if y_p *is* B*-smooth* **then**
      $(x_p - 1, x_p + 1)$ are B-smooth
    end
  end
end

5.3.3 Costs of Lenstra's method

As described in [37], the cost of the continued fraction method is at most $\sqrt{d}(1 + \log(d))^c$ for a certain c independent of d. Most importantly, it is exponentially slow and will fail to run in polynomial time. Unfortunately, this means that currently to compute B-smooth neighbour pairs for B rapidly becomes infeasible for $B > 150$.

5.4 PTE-Method

A paper by Costello et al. [19] approaches the search to new smooth primes differently. In this section we'll shortly explain this approach and their most important findings The method is based on solving the Prouhet-Tarry-Escott (PTE) problem, where for multisets $\{a_1, ..., a_n\}$, $\{b_1, ..., b_n\}$ the following equation hold for all $0 \leq i \leq n - 1$:

$$a_1^i + ... + a_n^i = b_1^i + ... + b_n^i.$$

These multisets then give rise to polynomials

$$a(x) = \prod_{i=1}^{n}(x - a_i) \quad b(x) = \prod_{i=1}^{n}(x - b_i).$$

It is proven that $a(x)$ and $b(x)$ will differ only by a constant $C \in \mathbb{Z}$. If you then can find $\ell \in \mathbb{Z}$ such that $a(\ell) \equiv b(\ell) \equiv 0 \mod C$, then you have that $\frac{a(\ell)}{C} - \frac{b(\ell)}{C} = 1$. Thus $a(\ell)$ and $b(\ell)$ are the smooth neighbours we are looking for.

This method resulted in the best bound $B < 2^{15}$ so far for a prime p within the range $2^{240} < p < 2^{256}$. One of the successful prime numbers found is

$$p_{PTE} = 265319464891319853876302880884726722210256475303002503310412276022343680 1,$$

with a bitlength of 241 and a smoothness bound of $B = 32029$. The prime of its neighbours are given below.

$$p_{PTE} - 1 = 2^1 2 \cdot 5^2 \cdot 7^2 \cdot 11^2 \cdot 13^2 \cdot 17 \cdot 29 \cdot 31 \cdot 43 \cdot 53 \cdot 103 \cdot 113 \cdot 181 \cdot 191 \cdot 211 \cdot 277 \cdot 557 \cdot 1093 \cdot 2663 \cdot 2897 \cdot 3347 \cdot 4783 \cdot 7963 \cdot 8623 \cdot 9787 \cdot 19841 \cdot 31489$$

$$p_{PTE} + 1 = 2 \cdot 3^2 \cdot 23^2 \cdot 41^2 \cdot 71^2 \cdot 83^2 \cdot 919^2 \cdot 1117^2 \cdot 1163^2 \cdot 1237^2 \cdot 6571^2 \cdot 11927^2 \cdot 18637^2 \cdot 32029^2$$

The main difference between this method and the others described in this section is that it calculates the probability to find a smooth neighbour pair using specific parameters in a search space, and using optimised parameters searches this search space until a satisfactory prime is found. As described in [19], the probability of finding new smooth neighbours using the PTE-method is higher than any other method used so far for prime numbers of the desired bitlength. The other two methods described in this chapter work with building a larger and larger set of smooth neighbours, with not the aim to find numbers of a specific length, but to find as many B-smooth neighbours as possible. This makes them slower and not applicable to specific search spaces, only by increasing B over time. The benefit there is that you get a more comprehensive list of B-smooth primes, which can give better results if you want to keep B as low as possible.

5.5 Extending neighbours method

In this section we will describe a new way of finding smooth prime numbers, that has never been used in such a way before. It is based on a method described in [15]. The idea is to start with a set of B-smooth numbers X, and to expand this set by finding smooth neighbours using already found smooth neighbours. First we will explain the algorithm, then we will describe the results and give an estimation for the complexity of the algorithm.

Assume we start with a set of numbers $X = \{1, ..., B\}$. These numbers are all B-smooth. Then take all pairs $(a, b) \in X^2$ and compute

$$\frac{a}{a+1} \cdot \frac{b+1}{b} = \frac{c}{d},$$

with c and d coprime. It is clear that both c and d are B-smooth. To find a smooth neighbour pair, check if $d = c + 1$. If this is the case, add c to X. This way a new set $X^{(1)}$ is created, that is the union of X and all such numbers c. Then iterate this process until for a certain n, $X^{(n)} = X^{(n+1)}$. The theorem below proves that all smooth neighbours can be found using this method.

Theorem 5.5.1. Given any B-smooth neighbour pair $c, c+1$, there exist a, b such that

$$\frac{a}{a+1} \cdot \frac{b+1}{b} = \frac{c}{c+1}. \tag{5.3}$$

Proof. Assume $u|c$, $v|(c+1)$ and $u < v$. Then write for a and b:

$$a = c - \frac{u}{v}(c+1), \quad b = \frac{v}{u}c - (c+1).$$

One can easily verify that a and b satisfy equation (5.3):

$$\frac{a}{a+1}\cdot\frac{b+1}{b}=\frac{c-\frac{u}{v}(c+1)}{c-\frac{u}{v}(c+1)+1}\cdot\frac{\frac{v}{u}c-(c+1)+1}{\frac{v}{u}c-(c+1)}=\frac{(c-\frac{u}{v}(c+1))(\frac{v}{u}c-(c+1)+1)}{(c-\frac{u}{v}(c+1)+1)(\frac{v}{u}c-(c+1))}=$$
$$\frac{c(\frac{v}{u}c-2c+\frac{u}{v}(c+1)-1)}{(c+1)(\frac{v}{u}c-2c+\frac{u}{v}(c+1)-1)}=\frac{c}{c+1}$$

□

The process is described as an algorithm below.

Algorithm 12 Using near 1 division to find new smooth neighbour pairs

Data: A list of B smooth numbers $X^0 = 1, ..., B$.
Result: A list of numbers x_i such that x_i and $x_i + 1$ are B-smooth
while $X^{(i)} \neq X^{(i+1)}$ **do**
  for x, y *in* $X^{(i)}$, $x < y$ **do**
    if $\frac{x}{x+1} * \frac{y+1}{y} = \frac{z}{z+1}$ *for a certain* $z \in \mathbb{Z}$ **then**
      $X^{(i)} = X^{(i)} \cup z$
    end
  end
  $X^{(i+1)} = X^{(i)}$
end

This algorithm works remarkably well, for the following reason. Computing the fractions gives

$$\frac{x}{x+1}\cdot\frac{y+1}{y}=\frac{xy+x}{xy+y}$$

This provides a smooth neighbour pair in the case that either $y - x = 1$, or when $y - x|xy + x$ and $y - x|xy + y$. The last case is true because we have $xy + x = k \cdot z$, $xy + y = k \cdot (z + 1)$, for a constant $k \in \mathbb{N}$, and combining the two equations gives $xy + y = xy + x + c$, so $c = y - x$. So for specific instances of x and y, this always returns a smooth neighbour pair.

Note that this method does not return a complete list of all B-smooth numbers. For instance, to find the largest 97-smooth number z, this method requires at smallest the use of two 227-smooth neighbours x and y [15]. This is possible since in the fraction both parts are divisible by 227. However, it does find almost all B-smooth numbers. Of the $13,374$ 97-smooth neighbours, only 37 were not found using $X = \{1, ..., 97\}$ as starting set.

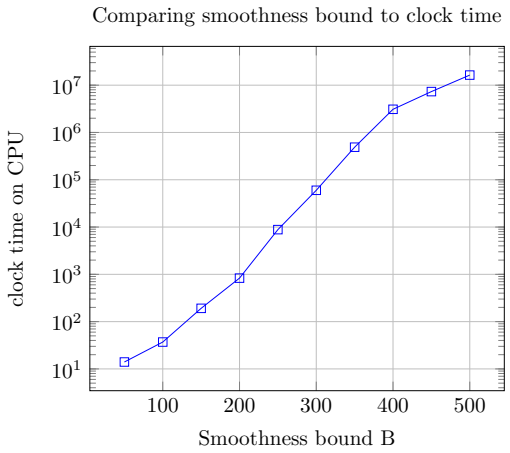
5.6 Costs of extending neighbours method

This algorithm is faster than Lenstra's method. It requires $O(n^2)$ multiplications for n the number of elements of the last set. The values of n for different values of B are described in Table 5.2. Using the methods described in Section 5.6.1 the algorithm is optimised, causing the CPU time to be as low as possible. In Figure 5.1 the real computational costs are shown and compared to the n^2 maximum multiplications. Unfortunately the data in computing the CPU time is incomplete, as for the last part from $B = 500$ to $B = 560$, the running time took so long the computer automatically reset itself, not saving the CPU time for the first part it ran. Because of the long run time of several weeks, it was decided not to rerun the algorithm to obtain this data. What we can see in Graph 5.1 is that the algorithm is nearly exponential, while the value n^2 grows less then exponential. One possible explanation for this is that due to the optimisations proposed in Section 5.6.1, relatively more pairs are compared for larger values of B than for lower values, as there are more pairs (x, y) that fit in the bound $x \leq 2y$. Therefore, there is a proposal for another

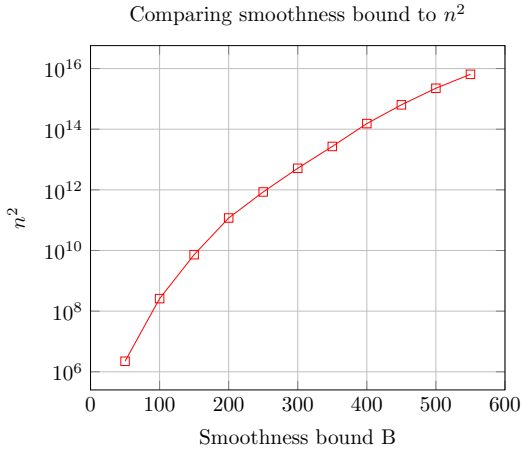
optimisation of the algorithm, to make it run faster for larger smoothness bounds B by setting to set even more restrictions on the value y/x during the running of the algorithm. So one could for instance set limits on the set size n, and when a certain limit is reached when running the algorithm, so when at least a specific amount M of smooth neighbour pairs is being compared, the maximum value y/x is set to be even smaller. You could start with for instance $y/x = 2$, and decrease it to smaller values, eventually even reaching small values like $y/x = 1.1$. Currently, a team from Microsoft research is looking into the possibilities this provides.

value of B	value of n	value of n^2
50	1495	$2,23 \cdot 10^6$
100	16096	$2,59 \cdot 10^8$
150	85291	$7,27 \cdot 10^9$
200	343808	$1,18 \cdot 10^{11}$
250	922864	$8,52 \cdot 10^{11}$
300	2268166	$5,14 \cdot 10^{12}$
350	5196435	$2,70 \cdot 10^{13}$
400	12352868	$1,53 \cdot 10^{14}$
450	25222339	$6,36 \cdot 10^{14}$
500	47348072	$2,24 \cdot 10^{15}$
550	80533790	$6,49 \cdot 10^{15}$

Table 5.2: Size of n for different smoothness bounds



(a) Costs of extending neighbours method for different values of B. Here the next step to B_{i+1} is always taken with the set of B_i-smooth primes precomputed



(b) Value of n^2 for different smoothness bounds B, as given in Table 5.2, where n is the total amount of smooth neighbour pairs found.

Figure 5.1: Comparing expected maximum amount of computations n^2 to the real CPU times when running the extended neighbour algorithm.

5.6.1 Reducing the amount of computations

One measure to reduce the amount of computations is described below.
We have

$$(y-x)|((x+1)y), \quad (y-x)|((y+1)x).$$

Also, $y, y+1$ are coprime, so the chance that $(x+1)\cdot y$ and $(y+1)\cdot x$ are divisible by the same number r decreases rapidly for increasing size of r. We can see in real-life results this means that almost for all successful attempts of finding a smooth neighbour pair, $x \leq y \leq 2x$. In 5.2 the distribution for x relative to y is given. One may even argue to reduce these boundaries further to $x \leq y \leq 1.5x$.

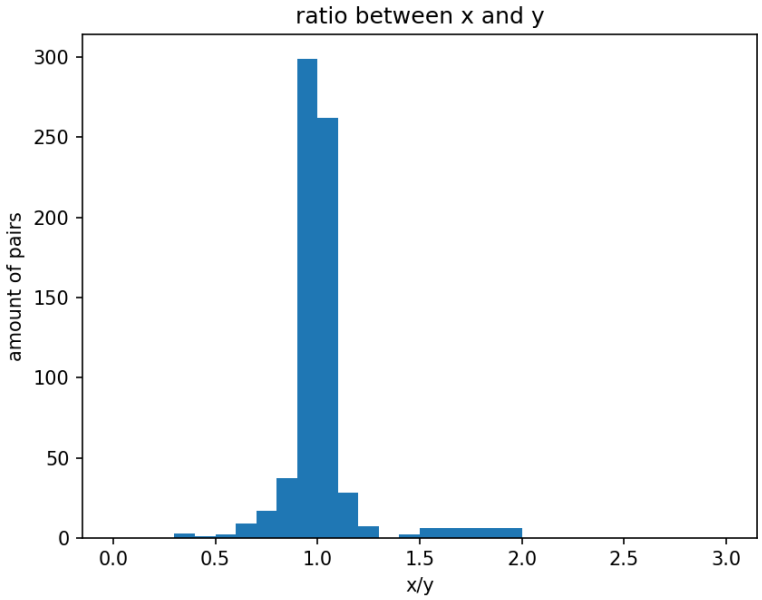


Figure 5.2: ratio x to y for pairs creating 59-smooth neighbours

To see the impact applying this measure has on the total of neighbour pairs found and the speed improvements, we have run the program for different restrictions on the compared values (x, y) for different smoothness bounds B. In Table 5.3 the main differences are shown. It is clear that when we are not putting any restrictions on the comparison of values the CPU time increases significantly, for $B = 200$ it is already a 40-fold increase compared to restricting $x \leq 2y$, while there is only a 0.4% increase in smooth neighbour pairs found. This justifies our restriction to compute only pairs (x, y) that lie significantly close to each other. The differences between $x \leq 1.5y$ and $x \leq 2y$ are less significant, and for now we chose to keep the boundary $x \leq 2y$, as it gives more smooth neighbour pairs for a small increase in costs.

B	$x \leq 1.5y$	$x \leq 2y$	no restrictions
50	15	15	9
100	37	37	125
150	160	189	3437
200	1193	1736	67781

(a) CPU time for different runs of the extending neighbours algorithm, for different smoothness bounds B and restrictions on comparing pairs (x, y).

B	$x \leq 1.5y$	$x \leq 2y$	no restrictions
50	1495	1496	1497
100	16037	16049	16093
150	85157	85284	85513
200	342770	343495	344867

(b) Number of smooth primes found for different runs of the extending neighbours algorithm, for different smoothness bounds B and restrictions on comparing pairs (x, y).

Table 5.3: Comparing the improvent of restricting the amount of pairs (x, y) compared versus the reduction in B-smooth neighbour pairs found for different values of B

5.7 Finding large smooth prime numbers

In this section we will discuss the search for large prime numbers and our results. We ran the algorithm given in [11] on the Dagobert computer of the ICIS institute at Radboud University. This computer has 4 E7-4870 v2 processors, of which we were able to use half. The algorithm was optimised by Giacomo Bruno. With this algorithm we were able to compute up to 560-smooth neighbour pairs, which took around 2 months. In total, we found 87026090 560-smooth neighbour pairs, of which 8368969 resulted in a smooth prime number. In this section we will first show the highest primes and smooth neighbours we found, then analyse the overall data gathered and give an estimation for future successes.

5.8 Finding new smooth primes

The largest smooth neighbour pair we found was 123-bit large. It is given by m, $m+1$ with

$$m = 896705136115967970985001321999051720 0.$$

Unfortunately, the largest 6 neighbour pairs found did not yield any smooth prime numbers. The largest smooth prime found is

$$p_{EN} = 206563233444570751827239872749937601, \tag{5.4}$$

which is 449-smooth and has a bitlength of 118. Its neighbours are

$$2 \cdot 3^4 \cdot 11^2 \cdot 29^2 \cdot 37^2 \cdot 43^2 \cdot 47^2 \cdot 127^2 \cdot 157^2 \cdot 193^2 \cdot 389^2 = 206563233444570751827239872749937602,$$

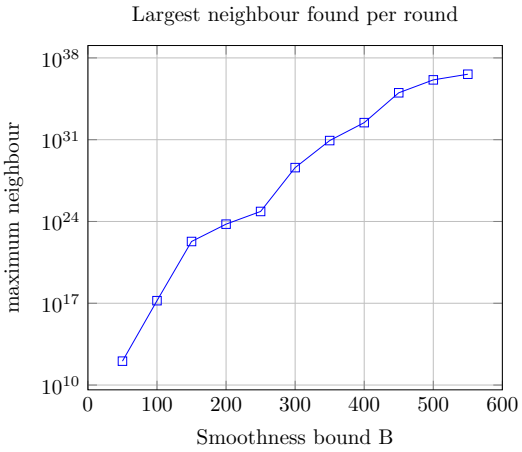
and

$$2^6 \cdot 5^2 \cdot 13^2 \cdot 17^2 \cdot 23 \cdot 53 \cdot 89 \cdot 151 \cdot 271 \cdot 277 \cdot 307 \cdot 317 \cdot 331 \cdot 353 \cdot 421 \cdot 449 = 206563233444570751827239872749937600.$$

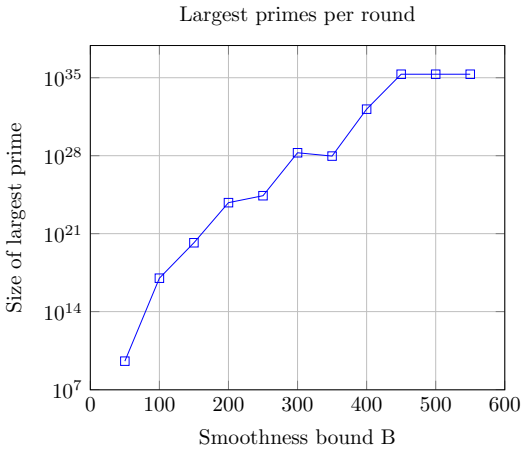
This is around half the required size of bit length 240 that is needed for B-SIDH primes in the current security standards required.

In Figure 5.3 the highest smooth neighbours and highest prime for different runs of the algorithm with smoothness bound B are shown. This is not necessarily the highest prime/neighbour found that has a smoothness bound of B, as it may be that some neighbours with a smoothness bound B are only found when running the algorithm for a higher smoothness bound $C < B$, as is described earlier in this section.

We see that in the beginning, there is an almost exponential increase in the size of the largest smooth neighbour and smooth prime found. Around $B = 450$ the curve flattens quite fast. This is also around the time where the computation time needed for the algorithm to complete became significantly longer, taking days or weeks to complete. Another important thing to notice is that the largest prime numbers found are significantly smaller than the largest neighbours found. This is due to the fact that only around 10% of the neighbour pairs produces a smooth prime. As we can see in Figure 5.4, where the largest 100 values per run of the algorithm are plotted, there are per round usually a few neighbours that are significantly larger than the rest, and the chance of one of those neighbour pairs resulting in a smooth prime is therefore small. however, sometimes we get lucky. This is what happened for prime p_{EN}, that has a smoothness bound of 449. It can be seen in Figure 5.4 that this point is significantly larger than is to be expected. This is what happens when the largest neighbour pair found does result in a smooth prime. This is promising for the future, as it is expected that this will happen in around 10% of the cases. While Figure



(a) For each run with the algorithm on a starting set $\{1, ..., B\}$, the largest smooth neighbour pair is shown. It may be that a B smooth neighbour is found only while running the algorithm for bound $C > B$.



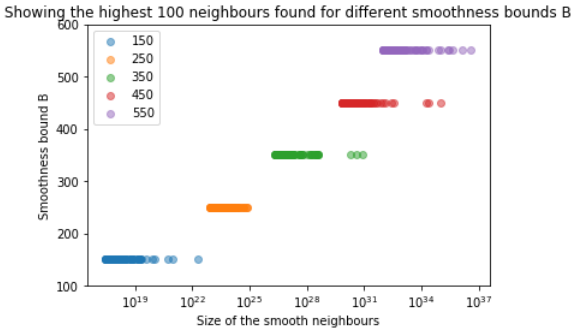
(b) For each run with the algorithm on a starting set $\{1, ..., B\}$, the largest smooth prime is shown. It may be that a B smooth prime is found only while running the algorithm for bound $C > B$.

Figure 5.3: Highest prime numbers and smooth neighbours found for specific smoothness bounds B.

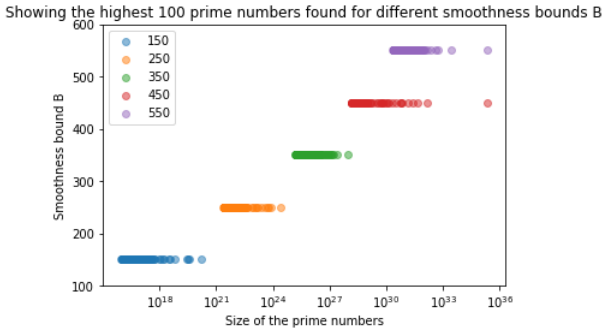
5.3 shows a more negative picture on the growth size, we can see that in case that we find such an outlier again, the largest prime found can increase in size rapidly.

As mentioned before, the largest B-smooth prime may be found when running the algorithm for a larger smoothness bound $C > B$. It is interesting to see the difference between the highest prime found per round and the highest prime per B overall. If there is a large difference between the two, it is (if computational time and space allow) interesting to look for a large B-smooth prime with a larger search bound C, and then disregard the primes that are not B-smooth. In Figure 5.5 we can see that for almost all smoothness bounds B below 400, we find a higher value for the largest smooth neighbour and/or the largest prime. This indicates that quite a few large B-smooth neighbours are found by searching on bigger bounds, as the chance of a finding a smooth prime is quite low, as discussed above. For the smoothness bounds B larger than 400 there are no larger values found, which is within expectations as there are fewer bounds $C > B$ that have been searched that could possibly result in larger B-smooth values. But overall, the results are almost never significantly higher when the larger smoothness bounds are included. This means that if you want to find a large B-smooth prime, it is usually sufficient to look with smoothness bound B.

Another interesting question to look into is the distribution of the smooth neighbour pairs and primes for a smoothness bound B. We plotted the density distributions for smooth neighbours found for different runs of the algorithm for a smoothness bound B in histogram plots in Figure 7.1, given in Appendix A. Both axis are on logarithmic scale. For the y axis this is because this way we can show more clearly the largest values found, as there are usually only few and on a linear scale this would hardly be visible. For the x-axis this is again because of the outliers, as the largest result may be 10 times as large as the second-to-largest value found, skewing up distribution. Also, this way we can better visualise the bounds where we expect to find the most B-smooth primes. So note that while Figure 7.1 shows a bell curve, the median of the values is actually more to the right of the data. We made the same plots but then for B-smooth primes in Figure 7.2, also given in Appendix A. We can see that the B-smooth primes found follow have the same density distribution as the B-smooth neighbours found, indicating a uniform chance of a number being prime. Furthermore we see some outliers on the right, which match with those seen in Figure 5.4.

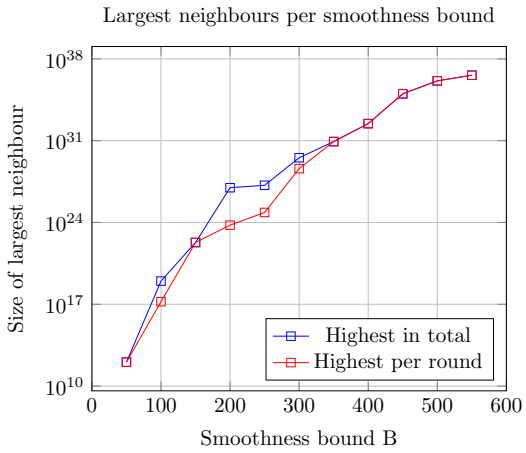


(a) Highest values found for smooth neighbours per smoothness bound B

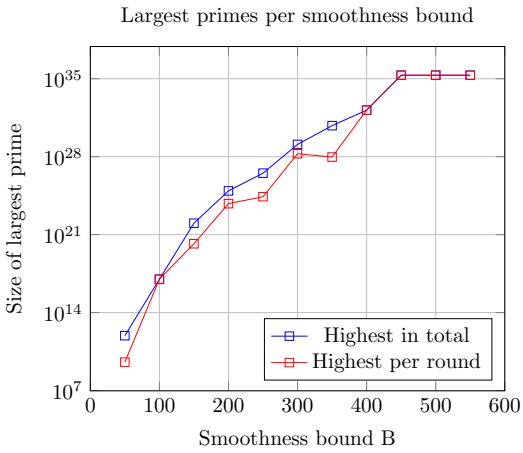


(b) Highest values found for smooth primes per smoothness bound B

Figure 5.4: Per run of the algorithm for a smoothness bound B, the 100 highest primes and neighbours.



(a) In total the largest smooth neighbour pair for each smoothness bound B.



(b) In total the largest smooth prime for each smoothness bound B.

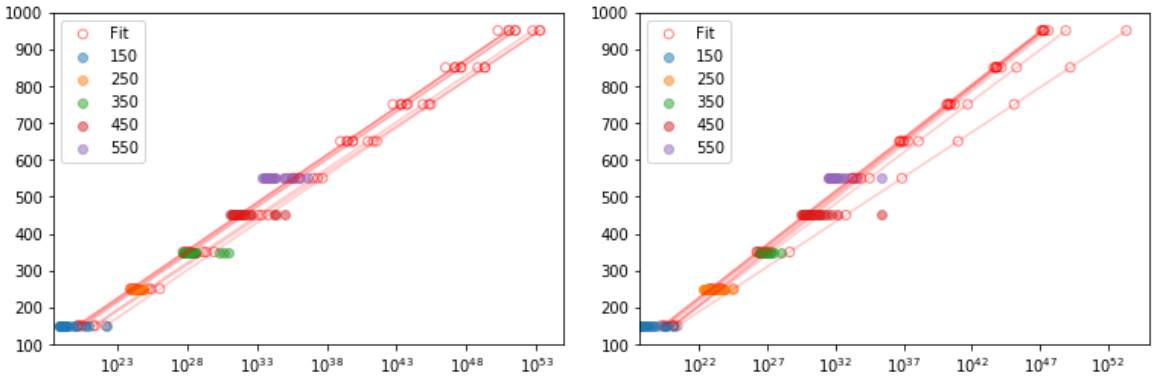
Figure 5.5: Largest smooth neighbour pair and prime for each bound B. These values were usually found for running the algorithm with a smoothness bound $C > B$.

As all these graphs have a normal distribution, it is expected that for larger smoothness bounds B, this will still be the case, and we can keep expecting high outliers. What these graphs also show is how many large smooth prime numbers there are, just a bit smaller than the largest smooth primes found. Maybe this is currently not relevant for B-SIDH, but there may be instances where large smooth prime numbers are needed of lower bit length than 240, and this data shows that the extending neighbours method will provide plenty examples. Who knows what applications this will have in the future. And if there are some computational speed-ups and we are able to run this algorithm for significantly larger B, this will also for cryptographic purposes provide plenty of choice for the prime p needed, which is interesting due to implementation optimalisation requirements (see Section 6.7 for an example of this). Not only for B-SIDH, but maybe also for SQIsign [26] this may be interesting.

The graphs of Figure 7.1 seem to indicate a certain bound on the size of B-smooth neighbours. While there may be some outliers, almost all numbers stay lower than a specific bound. This raises an interesting question on whether this is a universal property for all B-smooth primes, and whether we can establish this bound. While that research goes beyond the scope of this thesis, it

is highly relevant for future searches for smooth primes for B-SIDH.

To make a start with this, we did a regression analysis on a possible expected largest smooth neighbour and smooth prime to be found for smoothness bound B. The results are shown in Figure 5.6. From 5.4 it can be seen that the size of the largest B-smooth prime numbers does not increase linearly compared to B. However, making a regression analysis can be quite tricky, so we chose to make a linear regression analysis to provide an upper bound for the expected size of a 1000-smooth prime number. Since the dataset of only the largest prime/neighbour itself is quite small, we took the 100 largest primes/neighbours for different values of B. The results are computed using the numpy linear regression function.

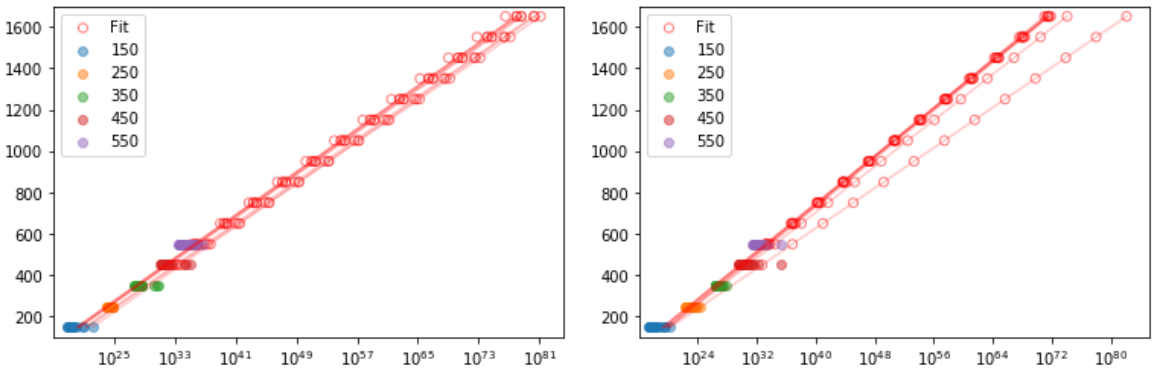


(a) A linear regression analysis for the largest prime of smoothness bound $B = 1050$.

(b) A linear regression analysis for the largest neighbour pair of smoothness bound $B = 1050$.

Figure 5.6: Linear regression analysis for smooth neighbours and smooth primes up until smoothness bound $B = 1050$ using as data the 100 largest values for specific smoothness bounds B.

As we can see from Figure 5.6, a maximum estimation for the largest smooth neighbour pair $m, m+1$ would be around size $m = 10^{50}$, with possible outliers until $m = 10^{52}$. An estimation of the largest smooth prime would be around $p = 10^{46}$. This corresponds to a prime of bit length 152, nowhere near enough for being useable in a B-SIDH key exchange. To make an estimation on the minimum bound B so that we could use the B-smooth prime p for B-SIDH, we made a larger regression analysis, as shown in Figure 5.7. We assumed here that a minimum bit length of 230 would be required, corresponding to a 68-digit prime number.



(a) A linear regression analysis until smoothness bound $B = 1750$ for smooth neighbours.

(b) A linear regression analysis until smoothness bound $B = 1750$ for smooth primes.

Figure 5.7: Linear regression analysis for smooth neighbours and smooth primes to find an upper bound for a 68-digit smooth prime.

Here the limitations of our analysis clearly show, as in Figure 5.7a the estimated size of the largest smooth neighbour pair of smoothness bound $B = 1750$ is around 10^{77}, while for the largest smooth prime this smoothness bound correspond to an expected size of 10^{68}. But as we are giving an upper bound here, not an exact estimation, we can still say that we do not expect to find a prime suitable for B-SIDH key exchange that is smoother than 1750-smooth.

5.8.1 Conclusion on the extending neighbours method

As a conclusion, this method initially proved very promising in finding new smooth primes. It is a lot faster than using Lenstra's method, and has as great benefit that it builds up from previous results, so you do not need to compute sets of smooth neighbour pairs again. It is also very complete, while not as complete as Lenstra's method it manages to give a comprehensive list of smooth neighbour pairs. When this method is run for large enough starting sets, it can give a clear indication of the expected bit lengths of B-smooth primes for a specific B. In the search for smoother primes this can be a great asset. Also, if there is access to a faster computer, the search for B-smooth primes with B larger could be continued. We do not expect to find any 250-bit primes with this method, but at least 150-bit primes should be a reasonable estimation. As discussed in Section 5.6.1, the difference between a and b could be investigated further. Now our computing capacity didn't allow us to research this for larger primes, but we think this could be really valuable information.

However, in the end, even with many improvements made to the algorithm, it still has exponential growth. And while we hoped that the costs were low enough to make computations possible for at least until $B = 1000$, at $B = 500$ our supercomputer reached its maximum capacity. After this value it took almost a week already to compute the next neighbour pairs with smoothness bound $B = 520$. Also the results were no longer as promising as in the beginning, as is shown in Figure 5.2. In the end we hoped to find at least one prime number with at least a bitlength of 128, but so far this has not happened. A few things are here to consider. The algorithm itself is still very valuable in finding smooth neighbour pairs, as it managed to find a stunning 87026090 pairs of smoothness bound $B = 560$. If there is a wish to continue the search for large smooth neighbour pairs, this method could be used, albeit it being on a significantly faster computer. When looking for smooth primes that can be used in B-SIDH key-exchanges, we advise to look for primes that have a minimal smoothness bound of $B = 1750$.

6. B-SIDH

In this section we will treat an alternative to the standard SIDH protocol, using smooth primes. We will first describe how to create a commutative protocol using twisted elliptic curves. Then we will explain the adapted SIDH protocol, called BSIDH, that was introduced by Costello in [16].

6.1 Background on isogenies on twists of elliptic curves

6.1.1 Introduction

The standard SIDH protocol as described in SIKE uses primes of the form $p = c \cdot 2^e \cdot 3^f \pm 1$. This has advantages for calculating isogenies fast, as larger isogenies can be split into multiple smaller computations using the prime factors of the isogeny, as described in Theorem 2.3.5. As seen in Section 3.5, the smaller the isogeny, the cheaper the computation. In this chapter we will evaluate the benefits and drawbacks of using more generalised isogenies with higher degree calculations, in exchange for the opportunity of using smaller prime fields.

6.1.2 Generalised isogenies

As seen in Section 3.3.8, each j-invariant over $\bar{F}_{p^2}$ is represented by E and its twist E'. In the case of supersingular isogeny cryptography, the Frobenius traces of these curves are $2p$ and $-2p$, such that $\#E(\mathbb{F}_{p^2}) = (p+1)^2$ and $\#E'(\mathbb{F}_{p^2}) = (p-1)^2$. E and E' are isomorphic over $\mathbb{F}_{p^4}$. The torsion groups are given by

$$E(\mathbb{F}_{p^2}) = \mathbb{Z}_{p+1} \times \mathbb{Z}_{p+1}$$

and

$$E(\mathbb{F}_{p^2}) = \mathbb{Z}_{p-1} \times \mathbb{Z}_{p-1}.$$

Over $\mathbb{F}_{p^4}$, the torsion group becomes

$$E(\mathbb{F}_{p^4}) = \mathbb{Z}_{p^2-1} \times \mathbb{Z}_{p^2-1}.$$

The morphism mapping E to its quadratic twist E' is

$$\sigma : E \to E', \tag{6.1}$$

$$(x, y) \mapsto (x, \delta y), \tag{6.2}$$

for a $\delta \in \mathbb{F}_{p^4} - \mathbb{F}_{p^2}$. This is a group morphism, but, as stated before, not an isogeny in $\mathbb{F}_{p^2}$.

Assume P is a point (x_P, y_P) of $E(\mathbb{F}_{p^2})$, and Q is a point (x_Q, y_Q) of $E'(\mathbb{F}_{p^2})$. One could compute the isogenies

$$\phi_P : E \to E/\langle P \rangle$$
$$\phi_Q : E' \to E'/\langle Q \rangle$$

but over $\mathbb{F}_{p^2}$, one cannot compute

$$\phi_P : E \to E/\langle Q \rangle$$
$$\phi_Q : E' \to E'/\langle P \rangle,$$

as these points do not lie on these respective curves. However, it is possible to lift the process to $\mathbb{F}_p^4$, in which Equation 6.1 is an isogeny. Setting $\phi'_P = (\phi_P \circ \sigma)$, $\phi'_Q = (\phi_Q \circ \sigma)$ this gives

$$\phi'_P : E \to E/\langle \sigma(P) \rangle \tag{6.3}$$

$$\phi'_Q : E' \to E'/\langle \sigma^{-1}(Q) \rangle \tag{6.4}$$

which are both well defined isogenies over $\mathbb{F}_{p^4}$.

6.1.3 Kummer line

As described in Section 3.2, for all $x \in \mathbb{F}_{p^2}$, either $(x, y) \in E$, or $(x, y) \in E'$ for a $y \in \mathbb{F}_p^2$, with the exception of points of order 2. This means that like in SIDH, we are still able to represent a point (x, y) using the Kummer line, as a point $(x, y) \in E$ will still be unique as a point $(x, -)$ on the Kummer line.

It is easy to see that when working over the Kummer line, the map σ, as defined in the previous section, acts as the identity map:

$$\sigma : (x, -) \mapsto (x, -).$$

Thus we can lift the implementation to $\mathbb{F}_{p^4}$ without actually having to do any extra calculations. This means that Alice can pick a point of the $(p + 1)$ torsion, Bob can pick a point of the $(p - 1)$ torsion, and without any extra computations they are able to exchange their public keys.

6.1.4 Isogeny graphs

As proven in 3.3.8, the isogeny graphs of two twists E and E' over $\mathbb{F}_{p^2}$ are isomorphic, and both are isomorphic to the isogeny graph of $\mathbb{F}_{p^2}$. This means that to take a walk on an isogeny graph of E is exactly the same as taking a walk on an isogeny graph of E'.

6.2 B-SIDH protocol

This protocol is introduced by Costello in [16]. The main difference compared to normal SIDH is to let Alice and Bob operate on different quadratic twists of a curve E, so that they are working on the $p - 1$ and $p + 1$ torsion groups. This has as benefit that you can take much smaller primes for computations, as normally Alice and Bob operate on torsion groups around size $\sqrt{p}$. The trade off is that we can no longer cherry-pick primes with small degree prime factors for $p+1$, as we now also need to take the prime factors of $p - 1$ into account. In Chapter 5 we describe the search for suitable large primes. As described in Section 6.1.2, to work over two quadratic twists of a curve, the process needs to be lifted to $\mathbb{F}_{p^4}$. But, as explained in Section 6.1.3, this makes no difference for implementation, as we compute over the Kummer line and disregard the y-coordinate.

6.2.1 Protocol

Alice and Bob agree on a prime p, with $p + 1 = p_1^{m_1} \cdot ... \cdot p_k^{m_k}$, and $p - 1 = q_1^{n_1} ....q_l^{m_l}$ as respective prime compositions. Alice and Bob choose two points P_A, Q_A and P_B, Q_B that generate the $p+1$ and $p - 1$ torsion groups. Either $4 \nmid (p - 1)$ or $4 \nmid (p + 1)$. Since both torsion groups need to be coprime (see 4.3.1), the points that generate the torsion group that is not dividable by 4 are multiplied by 2. They pick their secret integers r_A, r_B, and compute their secret kernels $\langle P_A + [r_A]Q_A \rangle, \langle P_B + [r_B]Q_B \rangle$. They compute their secret isogenies ϕ_A, ϕ_B as in the SIDH protocol, and exchange the resulting curves $E_A = E_0/\langle P_A + [r_A]Q_A \rangle$, $E_B = E_0/\langle P_B + [r_B]Q_B \rangle$ and auxiliary points $\phi_A(P_B), \phi_B(Q_A)$ and $\phi_B(P_A), \phi_B(Q_A)$. They compute their new secret kernels $\langle \phi_B(P_A) + [r_A]\phi_B(Q_A) \rangle$ and $\langle \phi_A(P_B) + [r_B]\phi(Q_B) \rangle$. Using these kernels, they both compute their secret isogenies again, and will then end up with curves $E_AB = E_A/\langle \phi_A(P_B) + [r_B]\phi(Q_B) \rangle$ and $E_BA = E_B\langle \phi_B(P_A) + [r_A]\phi_B(Q_A) \rangle$. These curves have the same j-invariant, that will be the shared secret

key. The protocol is explained schematically in figure 4.2, the same as used for the normal SIDH protocol.

6.2.2 Proof of correctness

Theorem 6.2.1. The curves E_{AB} and E_{BA} are isomorphic.

Proof.

$$E_{BA} = E_A/\langle \phi_A(\sigma(P_B)) + [r_B]\phi_A(\sigma(Q_B)) \rangle = E/\langle \sigma(P_B) + [r_B]\sigma(Q_B), \phi_A(\sigma(P_B)) + [r_B]\phi_A(\sigma(Q_B)) \rangle$$

and in the same way

$$E_{AB} = E/\langle \sigma(P_A) + [r_A]\sigma(Q_A), \phi_B(\sigma(P_A)) + [r_A]\phi_B(\sigma(Q_A))$$

Since in $E(\mathbb{F}_{p^4})$ σ is an isomorphism, when working over $\mathbb{F}_{p^4}$ this becomes equal to

$$E_{BA} = E/\langle P_B + [r_B]Q_B, \phi_A(P_B) = [r_B]\phi_A(Q_B)$$
$$E_{AB} = E/\langle P_A + [r_A]Q_A, \phi_B(P_A) = [r_A]\phi_B(Q_A)$$

Then, using a proof analogous of the proof of theorem 4.3.1, it can be shown that $j(E_AB) = j(E_BA)$. The only non-trivial part of the adaptation is that the orders of $R_A = P_A + [r_A]Q_A$ and $R_B = P_B + [r_B]Q_B$ are coprime, but this is the case since we doubled the points that did not not have order dividable by four, making their order an odd number. And since we work on the $p+1$ and $p-1$ torsion, the orders cannot share any other prime divisor. □

The implementation of the BSIDH protocol happens in F_{p^4}, but this is a mere technicality, as we work over the Kummer line, where we do not take the y-coordinate into account. This way the implementation of SIDH can be used, as long as Alice takes her points from the $(p+1)$ torsion of E, and Bob his points from the $(p-1)$ torsion of E^t.

6.3 B-SIDH security analysis

The main security problems, as given in Section 4.5, remain the same for BSIDH compared to SIDH.

- (Supersingular isogeny problem) Given a finite field K and two supersingular elliptic curves E_1, E_2 defined over K such that $\#E_1 = \#E_2$, compute an isogeny $\phi : E_1 \to E_2$.
- (Endomorphism ring computation) Given an elliptic curve E defined over a finite field K, compute its endomorphism ring. There could be made a small remark regarding the field K, as in SIDH this is $\mathbb{F}_{p^2}$, and for BSIDH this is $\mathbb{F}_{p^4}$. At minimum, working over a larger field should not make these problems harder.

In terms of random walks on isogeny graphs, there are two main differences. The first is that the length of the random walk is around length p, meaning that it would be more random than the walks in SIDH. A possible drawback may be the switching between isogeny graphs. But this also happens in CSIDH [12] and CRS [49], and no security problems have come forward so far in these protocols.

As BSIDH also requires the exchange of auxiliary points, it is vulnerable to active attacks, just as SIDH. An adaptation of the attack by Galbraith and Petit [30] is described in Section 7. As is shown there, this attack loses a lot of its strength due to the larger degree isogenies.

6.4 BSIDH usability

Is BSIDH an alternative for SIDH? As discussed in Section 6.3, there are no known security drawbacks to using BSIDH in regards of security so far, and some benefits in regards to the length of the random walks on isogeny graphs. Then there is one other important aspect to consider: the costs. These are split between computation costs and key length. For key length BSIDH is clearly an advantage, as the keys are directly linked to the size of the prime number, which is quadratically larger in SIDH than BSIDH. The other aspect, computation cost, is the main issue. For this we refer back to Chapter 5. For BSIDH to work, we need to have a smooth prime number. The smoother the prime number, the cheaper the computation. Finding such a prime number, however, is not easy, as is demonstrated in Chapter 5. The applicability of BSIDH therefore stands or falls with the option of finding appropriate prime numbers.

One possible application of BSIDH is a server-client trade off. In a case where Alice has a device with strong computational power, and Bob's device's power may be restricted, we can find prime numbers of which one neighbour is a lot more smooth than the other, reducing computation costs for Bob while increasing costs for Alice. This may be the case for instance where Alice is a server, and Bob is a smartphone making a connection to that server. One instance of this will be treated in Section 6.8.

In the rest of the chapter, we will look into one instance of such a trade off. Furtherore we will look in detail in the B-SIDH algorithm, with a special focus on optimal strategies. We will also compute the costs for computing a B-SIDH key exchange for different primes.

6.5 Optimal strategy for BSIDH

As described in Section 4.4, we can optimise large degree isogeny computation by splitting an isogeny of length $m = \ell_1^{k_1} \cdot ... \ell_n^{k_n}$ by sequentially computing isogenies ℓ_i.

$$\phi = \phi_1 \circ ..... \circ \phi_n.$$

The main difference with SIDH is that we now compute the isogeny ϕ while the ϕ_j have different degrees ℓ_i. We compute ϕ by making a list $L = [\ell_1^{k_1}, ..., \ell_m^{k_m}]$, and then computing first k_1 ℓ_1 degree isogenies, up until the end where we compute k_m ℓ_m degree isogenies. We have that $n = \sum_{i=1}^{m} k_i$.

We compute the general isogeny

$$\phi : E_0 \to E_A = E_0/\langle R \rangle,$$

where for each step in between we compute

$$E_{i+1} = E_i/\langle [\prod_{j=r+1}^{m} \ell_j^{e_j} \cdot \ell_r^{k_r - s}] R_i \rangle, \quad i = \sum_{j=1}^{r} e_j + s$$

So for each next isogeny we need to compute the R_i multiplied by the product of all degrees of the isogenies yet to be computed.

In Section 4.4 an optimal strategy for SIDH is described, to make a balance between isogeny computations and multiplication of the kernel point. The question now is how an optimal strategy for BSIDH looks like, so how can we generalise the SIDH optimal strategy to work for a mix of different degree isogenies. We will call a multiplication-based strategy one where we only compute the minimum n isogenies, and in exchange do a lot of extra point multiplications. We call a strategy balanced if we find an optimum between multiplications and isogenies. An example of a really balanced strategy is given in Figure 6.1. For a complete multiplication-based strategy see Figure 6.2.b.

For CSIDH and B-SIDH determining the strategy is largely the same problem, as in essence it does not matter if we do isogeny computation or class action computation, in both cases we need to find the same balance between multiplications and one other operation, for which that

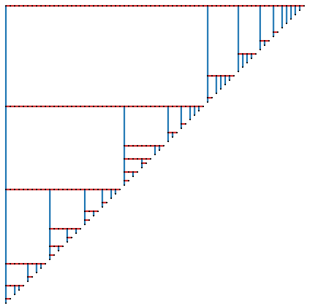


Figure 6.1: Optimal strategy for Alice's torsion for the prime p237.

multiplication is needed. So for B-SIDH we can take the same theory behind optimal strategies that have already been researched for CSIDH, for instance in [13]. An adaptation to BSIDH of their strategy is described below.

The optimal strategy for SIDH as described in Section 4.4 is in the basis really similar. The main difference is that now there

For a fixed list of prime degree isogenies

$$L = [\ell_1, ..., \ell_n]$$

we compute an optimal strategy. Were we to change the order in L, we could get a different strategy.

We will now compute the optimal strategy $P(L)$ analogous to computing a SIDH optimal strategy. Just like in Section 4.4 we use a recursive formula where we compute the cost of a larger triangle by computing optimising over the cost of its smaller triangles. The difference is that the costs of the triangle depends on the degree of the isogenies, so not every edge in our graph has the same costs, where with SIDH we had fixed costs. In line with Equation (4.5) we define the costs as

$$C(L) = \min_{i \in \{1,...,n-1\}} (C(L^i) + C(L^{n-i}) + \sum_{j=1}^{n-i} q_{\ell_j} + \sum_{k=0}^{i-1} p_{\ell_{n-k}}) \tag{6.5}$$

Where q_{l_j} is the cost associated to computing a ℓ-degree isogeny, and $p_{l_{n-j}}$ the costs associated to multiplication for ℓ_i. L^1 and L^2 are disjoint sublists $L^1 = [\ell_1, .., \ell_i]$ and $l^2 = [\ell_{i+1}, ..., \ell_n]$ of L.

We can define the strategy for any list $L_{k,j} = [\ell_{j+1}, .., \ell_{k+j}]$ as follows, where $j \in \{0, .., n-1\}$, $k \in \{1, .., n\}$

$$P(L_{k,j}) = [s] \operatorname{cat} P(L_{k-s,j+s}) \operatorname{cat} P(L_{s,j})$$

Where s is the i such that $C(L_{j,k})$ as defined in Equation 6.5 is minimal.

6.5.1 Order of list of prime numbers

So what is important is the order in which the isogenies are computed. Ideally, we could compute the strategies for all different orders and pick the best one, but given the amount of different primes involved in the computation, it is not feasible to take all orderings of prime numbers into account when computing the optimal strategy, since as discussed in [13] this costs

$$\sum_{i=1}^{n-1} i! \cdot \binom{n}{i} \gg 2^n,$$

where n is the length of the list. This makes it infeasible to compute for large n, so that we will have to make some educated assumptions on the order of the list. In the paper discussing

optimal strategies for CSIDH [13], it is assumed that using a list L with primes ordered from small to big is the best option. But CSIDH uses relatively small primes, where in BSIDH primes are larger and may differ wildly depending on the chosen prime. Therefore it is interesting to discuss a few possible "good" orderings of L that may give the best outcomes. Note that for different primes, different orderings may work better, and don't even have to be the same for Alice and Bob, as the degrees of isogenies they compute is completely different. One important thing to notice is that The last prime in the list should always be the highest, as we do not need to compute xEVAL for the kernel points in the last round (see Section 6.7, and this way we omit the most expensive xEVAL computations. Also it may be that for public key generation and shared secret key computation, there is a inequality in costs for different strategies depending on the ordering, as there are less xEVAL computations in the shared secret key computation, making a more balanced strategy more favourable.

To test our theory we picked the following prime $p237$, that has a lot of large primes in its torsion group for Bob and only few small primes. The difference between a standard CSIDH prime and $p237$ is thus quite large, which makes it a good test prime to see the difference in prime ordering for CSIDH and B-SIDH for optimal strategies.

$$p237 = 188098835761489939757482570291811148273499283258225940944664269318258687 \quad (6.6)$$

Which has as torsion group for Alice

$$T_A = 2^1 9 \cdot 3^2 4 \cdot 17^6 \cdot 19^6 \cdot 31^6 \cdot 37^6 \cdot 53^1 2,$$

and as torsion group for Bob is equal to

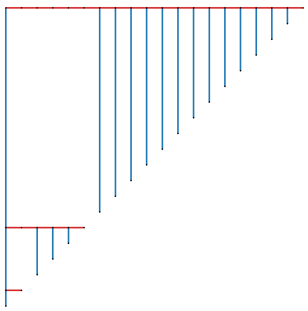
$$T_B = 7{\cdot}13{\cdot}43{\cdot}73{\cdot}103{\cdot}269{\cdot}439{\cdot}881{\cdot}883{\cdot}1321{\cdot}5479{\cdot}9181{\cdot}12541{\cdot}15803{\cdot}20161{\cdot}24043{\cdot}34843{\cdot}48437{\cdot}62753{\cdot}72577.$$

We tried out many possible orderings for Bob's list of primes and computed the strategies and its costs for the key exchange, of which we found four interesting orderings that we want to highlight specifically. They are described below. To demonstrate the importance of having the highest prime last, as ordering five a complete high-to-low ordering is included. Here high-to-low refers to the ordering of the primes based on their size.

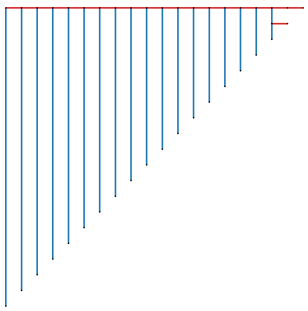
1. From high to low: a complete multiplication based strategy, with the multiplications as cheap as possible. The last prime is the highest.
2. From low to high: a strategy as balanced as possible
3. First the cheapest 5 primes ordered from low to high, then the other primes from high to low. The last prime is the highest. This gives the same optimal strategy as list order 2.
4. From high to low, but the order of the second-to-last 9 primes switched (that are all the primes with size less than 1000). The last prime is the highest. This was an idea to combine a balanced strategy with low multiplicative costs as the costs for the final multiplications are negligible compared to the first ones.
5. For comparison a high-to low strategy where the last prime on the list is the highest prime.

In Figure 6.2 the different strategies depending on the ordering of Bob's primes are shown, with in Table 6.1 the respective costs of public key generation and shared key generation that are computed using the SIBC library [31].

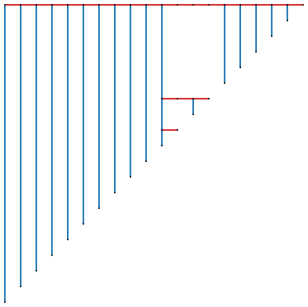
From Table 6.1 we can first conclude that even for torsion groups with large prime factors and few small primes, the differences in costs for different orderings are small. There is virtually no difference between an almost multiplicative ordering such as (2) and a more balanced strategy as (1). However, we see that a combination of a balanced strategy and given as in ordering (4) is



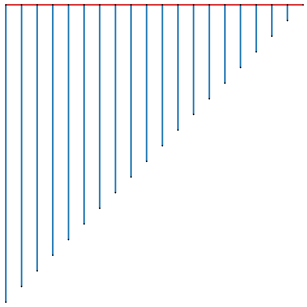
(a) Optimal strategy for order #1 and #3.



(b) Optimal strategy for order #2.



(c) Optimal strategy for order #4.



(d) Optimal strategy for order #5.

Figure 6.2: Plots of different strategies depending on the order of the prime list L for Bob's torsion for the prime $p237$. Different orders are described in this section.

List order	costs PKG	compare PKG	costs SKG	compare SKG
1	2.207	100	1.192	100
2	2.208	100	1.193	100
3	2.201	0.997	1.186	99.5
4	2.207	100	1.192	100
5	2.313	1.298	104.8	108.8

Table 6.1: Different list orderings as described in this section with the costs for public key generation (PKG) and shared secret key generation (SKG), also compared to the costs of the current standard order low-to-high.

cheaper than all other orderings. It is also quite a simple adaptation to make, as one just needs to compute the optimal strategy for a low-to-high ordering and then switch the order of the primes that are in the multiplicative part of that strategy. Even though it is a small improvement, it still decreases costs of isogeny computation and therefore we recommend implementations of B-SIDH to use this ordering over a low-to-high ordering.

In the last part of this chapter we will look concretely into the cost of completing a BSIDH key-exchange for different primes with the most recent knowledge on isogeny computation and optimal strategies.

6.6 Costs of computing large degree isogenies

In this section we will compute the costs of a BSIDH key exchange for three primes, $p237$ 6.6, $p253$ 6.7 and p_{EN} 6.6. The last prime is found using the extending neighbours method as described in Section and 5.5. It is not the highest prime found with this method, as currently the only existing B-SIDH implementation only accepts primes of the form $p \equiv 3 \mod 4$. Therefore the highest 4 primes found were not eligible for this analysis.

$$p_{EN} = 568254508113466749936016007195999,$$

which is 521-smooth and has a bit length of 109. It's neighbours are

$$2 \cdot 7^2 \cdot 11 \cdot 19 \cdot 31^2 \cdot 37 \cdot 41 \cdot 89^3 \cdot 139 \cdot 167 \cdot 211 \cdot 227 \cdot 229^2 \cdot 463 = 568254508113466749936016007195998$$

and

$$2^9 \cdot 3 \cdot 5^3 \cdot 13^4 \cdot 23 \cdot 29 \cdot 127^2 \cdot 131 \cdot 173 \cdot 193 \cdot 271 \cdot 479 \cdot 521^2 = 568254508113466749936016007196000.$$

$$p253 = 11402780996313137804419565692258934141207562497476991733713707020990899136527, \tag{6.7}$$

which is 76667-smooth and has a bit length of 253. It's neighbours are

$$2 \cdot 11^{18} \cdot 19 \cdot 23^{13} \cdot 47 \cdot 79 \cdot 83 \cdot 89 \cdot 151 \cdot 3347 \cdot 17449 \cdot 33461 \cdot 51193$$

and

$$2^9 \cdot 3 \cdot 7^{16} \cdot 17^9 \cdot 31^8 \cdot 311 \cdot 571 \cdot 1321 \cdot 5119 \cdot 6011 \cdot 14207 \cdot 28477 \cdot 76667.$$

We will first describe the algorithm to complete a B-SIDH key exchange in detail and give costs of different elements, then describe the optimal strategies for these primes, and then give an overview of the costs, both expected and calculated using the SIBC-library [31].

6.7 B-SIDH key exchange algorithm

Public parameters For the B-SIDH protocol we have the following public parameters. First there is the starting curve E_0. For primes of the form $p \equiv 3 \mod 4$, this is for now set to be

$$E_0 : y^2 = x^3 + 6x^2 + x.$$

This is always supersingular if $p \equiv 3 \mod 4$. As described in [21], this is a better starting curve than the always supersingular curve $E_0' : y^2 = x^3 + x$, as E_0' is theoretically less secure as it gives information on the first performed 2-isogeny. For B-SIDH, it has not been investigated whether using E_0 as a starting curve has significant benefits over E_0', which is an interesting problem to look at. For this example we only use primes of the form $p \equiv 3 \mod 4$, so we face no problems using E_0, but this may become an issue as from Chapter 5 we see how hard it is to find suitable smooth primes, and we might encounter useful primes of the form $p \equiv 1 \mod 4$.

Also as public parameters we have x-coordinates of the points $P_A, Q_A, PQ_A = P_A - Q_A$ on E_0 of order $N|p+1$. These points can be generated as follows:

Generate a point P_A of order M and a point Q_A of order M on the curve $E/\mathbb{F}_{p^2} : y^2 = x^3 + Ax^2 + x$. These points can be generated by using an iterative algorithm that takes a random point $T = (a + b * i : c + d * i : 1) \in E(\mathbb{F}_p^2)$. Here i is the root of the quadratic formula $x^2 + c$ that creates the extension field F_{p^2} over F_p, with c not a square in $\mathbb{F}_p$ and i not in $\mathbb{F}_p$. Then the algorithm checks whether

$$[\frac{p-1}{q}]T \neq (0 : 1 : 0)$$

for all prime divisors q of $p-1$. If this is the case, the point has order $p-1$. We find two random points S, T this way, and compute their Weil pairing (see Section 3.7.4) to check if they are linearly independent. Continue the search for S until you find a point S such that $e_N(S,T) = 1$ and they are linearly independent. The points P_A and Q_A are then given by

$$P_A = [\frac{p-1}{N}]T, \quad Q_A = [\frac{p-1}{N}]S.$$

Finally the point $PQ_A = P_A - Q_A$ is computed. The x-coordinates of the points are stored.

And finally we have the x-coordinates of the points $P_B, Q_B, PQ_B = P_B - Q_B$ on E_0^t of order $M|p-1$, a quadratic twist of E_0. To find points T that fit our requirements, we compute a $j \in \mathbb{F}_{p^4}$. such that j is not a square in $\mathbb{F}_{p^4}$. The element $k = j^2$ is then a nonsquare element in $\mathbb{F}_p^2$. We can construct our field $\mathbb{F}_{p^4}$ such that $\mathbb{F}_{p^4} = \mathbb{F}_{p^2}[j]$ We find the points by finding elements $x \in \mathbb{F}_{p^2}$ such that $z = x^3 + Ax^2 + x$ is a nonsquare element in $\mathbb{F}_{p^2}$. We then must have that $k \cdot z = y^2$ for a $y \in \mathbb{F}_{p^2}$. The point T is then given by $(x : k(x^3 + Ax^2 + x) : 1)$. Then for finding S the same procedure as described above is followed to make S linearly independent from T. Now the points P_B and Q_B are given by

$$P_A = [\frac{p+1}{M}]T, \quad Q_A = [\frac{p+1}{M}]S$$

Note that we must have $\gcd(N, M) = 1$, so if $p \equiv 3 \mod 4$, so M is usually taken to be $\frac{p-1}{2}$.

Now that we have the public parameters, we need to compute the secret key

Secret key generation For Alice, pick any random integer $s_A \in \{0, .., p+1\}$. For Bob, pick any random integer $s_B \in \{0, .., p-1\}$. Then compute the kernel points

$$R_A = P_A + [s_A]Q_A, \tag{6.8}$$
$$R_B = P_B + [s_B]Q_B. \tag{6.9}$$
$$\tag{6.10}$$

This computation is done using the three-point ladder algorithm 4. For this we need the points PQ_A and PQ_B. Because we asserted that the points P and Q are linearly independent, we know that the order of R_A and R_B will be N and M respectively.

Public key generation The goal of Alice and Bob is to compute the secret isogenies ϕ_A and ϕ_B. This is done by computing $\phi_A(E_0) = E_A = E_0/R_A$ and the extra points $\phi_A(P_B), \phi_A(Q_B), \phi_A(PQ_B)$ (vice versa for Bob). The curve and points are computed using the optimal strategy. The optimal strategy for B-SIDH is treated extensively in Section 6.5. The optimal strategy computation for the curve E_0 results in a list

$$L = [c_1, ..., c_n].$$

With this list we create the outer loop for our public key generation, as described in Algorithm 13.

From our optimal strategy computation we get a strategy $S = [a_1, ..., a_n]$ with instructions on when to compute isogenies and when to compute multiplications.

Algorithm 13 Outer loop of B-SIDH key exchange protocol

Data: A kernel point R_A of order N, a strategy S of length $n-1$, with maximum of isogeny computations s, a list of primes L, a starting curve E and points P_B, Q_B, PQ_B.
Result: A curve $E_A = E_0/E_A$ with curve constant A, the images of P_B, Q_B, PQ_B on E_A.

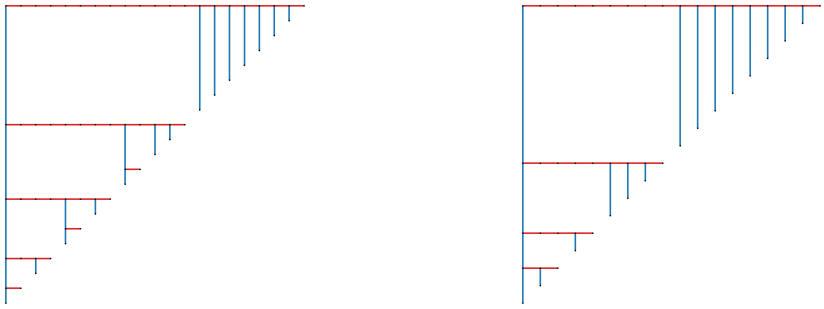
for i *is* 1 *to* $s-1$ *do* **do**
  while $j < s - i$ *do* **do**
    $m = S[s-i-j+1]$
    c = $\prod_1^m \ell_{n-k}$ $T = [c]R$ $j+ = m$
  end
  $I, J, K = KPS_{\ell_i}(T, A, C)$
  $A, C = xISOG_{\ell_i}(T, A, C, I, J, K)$
  $R, P_B, Q_B, PQ_B = xEVAL_{\ell_i}(R, P_B, Q_B, PQ_B, A, C, I, J, K)$
end
$I, J, K = KPS_{\ell_n}(T, A, C)$
$A, C = xISOG_{\ell_n}(T, A, C, I, J, K)$
$P_B, Q_B, PQ_B = xEVAL_{\ell_n}(P_B, Q_B, PQ_B, A, C, I, J, K)$
$A = A/C$

Each point multiplication $[m]R_i$ can be computed using the Montgomery ladder algorithm 3. For isogeny computation we have three different parts, as described in 3.5 and 3.6. To optimise our computation, we use either traditional Velu formula to compute the isogeny, or the sqrtvelu formula. Currently isogenies of degree less than 87 are faster computed using traditional Velu formula. This distinction is also made in Algorithm 13. For each isogeny computation, we start with the kernel points generation as described in Algorithm 3.13. We know from Section 3.4 that the type of elliptic curves we use are always isomorphic to a specific Montgomery curve. So we can use the $xISOG$ algorithm 3.15 to compute the new value A_{i+1} of $E_{i+1} : y^2 = x^3 + A_{i+1}x^2 + x$, where $E_{i+1} = \phi_i(E_i) = E_{i+1}/[m_i]R_i$. Then we need to push points through the isogeny. To be able to compute the next isogeny, we need the image $R_{i+1}\phi_i(R_i)$. Note that we only need the image of this point to compute the next isogeny, so we especially do not need to compute R_{i+1} in the last round of isogeny computation. For the auxiliary points, we need to compute for each round, also the last, the points $PQ_{B,i+1} = \phi_i(P_{B,i}), P_{B,i+1} = \phi_i(Q_{B,i}), Q_{B,i+1} = \phi_i(PQ_{B,i})$. The images of all these points are computed using the $xEVAL$ Algorithm 3.16.

Public key exchange Alice send to Bob the value A_A of her curve $E_A : y^2 = x^3 + A_Ax^2 + x$, together with the images of his public keys on her curve E_A, $\phi_A(P_B), \phi_A(Q_B), \phi_A(PQ_B)$. She receives from Bob the value A_B corresponding to the curve $E_B : y^2 = x^3 + A_Bx^2 + x$, and the images of her public keys on Bob's curve E_B, $\phi_B(P_A), \phi_B(Q_A), \phi_B(PQ_A)$. They compute their new kernel points using again the three point ladder algorithm 4.

$$R_A = \phi_B(P_A) + [s_A]\phi_B(Q_A), \tag{6.11}$$
$$R_B = \phi_A(P_B) + [s_B]\phi_B(Q_B). \tag{6.12}$$
$$\tag{6.13}$$

Shared secret key generation In this step, Alice and Bob compute the curves $E_{AB} = \phi_A(E_B) \cong E_{BA} = \phi_B(E_A)$. This is done in the same way as in Section 6.7, with one difference: We do not need to compute the images of the points P_B, Q_B, PQ_B (or P_A, Q_A, PQ_A) in



(a) Optimal strategy for Alice. (b) Optimal strategy for Bob.

Figure 6.3: Optimal strategies for Alice and Bob for torsion groups of p_{EN}.

the case of Bob. Therefore, we only need to apply the $xEVAL$ Algorithm 3.16 on the point R_A respectively R_B. Again, we use the optimal strategy for this. Note that the optimal strategy may be slightly different since we need to do less $xEVAL$ computations, favouring more isogeny-based strategies as isogeny computations becomes cheaper now. The starting curves for Alice and Bob are now E_B for Alice and E_A for Bob. After computing their isogenies $\phi_A(E_B) = \phi_B/R_A = E_{AB}$ and $\phi_B(E_A) = E_A/R_B = E_{BA}$, using the same method as described in Section 6.7, they arrive at the shared secret curve $E_{AB} \cong E_{BA}$. They compute the j-invariant from the value A_{AB} of $E_{AB} : y^2 + A_{AB}x^2 + x$. The j-invariant is their shared secret key. As follows from Section 7, this key should be hashed before it is used.

This is detail what happens in a B-SIDH key exchange with the most advanced implementation. In the next section we will analyse the costs of this key exchange for two specific prime numbers.

6.8 B-SIDH costs for specific prime numbers

As described in the previous section, we will now look concretely into the costs of a B-SIDH key exchange for different prime numbers. We will first elaborately go into the algorithm and costs of the B-SIDH protocol for the prime P_{EN}, then shortly treat $p237$ and $p253$ B-SIDH key exchanges, and in the end compare the results.

6.8.1 B-SIDH algorithm for p_{EN}

As discussed in section 6.5, we will compute the optimal strategy using the low-to-high ordering, and then see if we can optimise this further. As we can see in Figure 6.3, it is indeed possible to optimise the ordering of the primes a bit further, resulting in around 2000 less multiplications needed in total for the whole key exchange.

For Alice using p_{EN} this will give an optimal strategy

$$y = [8, 5, 4, 2, 1, 1, 2, 1, 1, 3, 1, 2, 1, 7, 6, 5, 4, 3, 2, 1]. \tag{6.14}$$

For Bob using p_{EN} this will give an optimal strategy

$$x = [9, 4, 2, 2, 1, 1, 3, 2, 1, 8, 7, 6, 5, 4, 3, 2, 1]. \tag{6.15}$$

For the prime p_EN we got the following public points, where i denotes the root of the generating polynomial of $\mathbb{F}_{p^2}$ over $\mathbb{F}_p$.

$$P_A = 476885896216129711847003777043170 + 3038035225075470313098663294057 94i,$$
$$Q_A = 952119212325964706565994747372 67 + 535579974553724510749548409096772i,$$
$$PQ_A = 351368631037001848694005622946 57 + 185430798143947645500165812942 58i,$$
$$P_B = 522366699383065120792952767676853 + 448191885270909062726083421604809i,$$
$$Q_B = 203521641971392657745146167565782 + 389175539296221619767167295512964i,$$
$$PQ_B = 102301598585892989427300751395 44 + 176114589698874269173696707456156i.$$

As a secret key for Alice we pick

$$s_A = 114314656912255065538433627282249,$$

and for Bob we get

$$s_B = 10591189053688819735544694357 4000.$$

We then use the optimal strategy as given in Figure 6.3 and Equations 6.14 and 6.15 to compute the public keys of Alice and Bob.

We get

$$E_A : y^2 = x^3 + (171250176368517566676622487461951 + 27746175667139130946930953326218 5i)x^2 + x,$$
$$E_B : y^2 + x^3 + (109359357352596807571045003078604 + 19855316908539366132509381003473 6i)x^2 + x.$$

Then we again compute the secret isogenies ϕ_A and ϕ_B, resulting in a shared secret key

$$j(E_{AB}) = 346730674433960568601827881986809 + 386593403626889625879546386427046i.$$

6.8.2 Costs of p_{EN} key exchange

Here we make an estimation of certain aspects of the costs of a p_{EN} key exchange assuming the public parameters are known, and using the SIBC library [31] we compute the actual costs for the key exchange.

In Equation (3.4) it is given that computing a multiplication $P = [m]P$ has a cost of maximum $16\lceil \log(m) \rceil$. If we take Bob's torsion group and follow the optimal strategy, we can compute the order of the first multiplication in the key-exchange. This is given by

$$16log(\frac{M}{\ell_1}) = 16 * \lceil \log(5682545081134667499360160071959 98/7) \rceil = 1702$$

The costs for the most costly isogeny computation can be estimated by Equation (3.17). We then have the costs for $xEVAL$, $xISOG$ and KPS. This is given by, where $b = \frac{\sqrt{(521)-1}}{2}$,

$$cost(b) = 4(9b^{\log_2(3)}(1 - 2\frac{2}{3}^{\log_2(b)+1}) + 2b\log_2(b)) +$$
$$3((1 - \frac{1}{3^{\log_2 b+1}}b^{\log_2(3)}) + 37b + 3\log_2(b) + 16 = 3489.$$

In total the costs for Alice and Bob to compute their p_{EN} B-SIDH key-exchange are 43000 M and 41000 M respectively.

Torsion group	Costs PKG	Costs SKG	Total costs
p237A	0.029	0.22	0.051
p237B	0.743	0.389	1.132
p253A	0.313	0.157	0.470
p253B	0.256	0.131	0.387
p_{EN}A	0.028	0.016	0.043
p_{EN}B	0.026	0.015	0.041

Table 6.2: Costs in $\mathbb{F}_{p^2}$ multiplications for computing the costs for public key generation, shared key computation and the costs of a B-SIDH key exchange for Alice, Bob and for primes $p237$ and $p253$.

6.8.3 Key exchanges for $p237$ and $p253$

To make a comparison in the costs of the key exchange for p_{EN}, here we also give the costs for two other primes. $p237$ is a prime that has a very smooth $p+1$ torsion group and significantly less smooth $p-1$ torsion group. This is, as described in Section 6.4, a possible prime for applying key-exchanges between two devices where one device has significantly less computing power than the other. $p253$ is a more balanced prime, with both torsion groups being almost equally smooth.

The total costs of computing a B-SIDH key exchange with $p253$ and $p237$ are described in Table 6.2. These costs are computed using the SIBC-library [31]. Costs are given in millions of $\mathbb{F}_p$ multiplications, ignoring the cost of additions.

It is from Table 6.2 that completing a $B-SIDH$ key exchange for $p237$ we have that Alices key exchange is around 20 times cheaper than Bob's key exchange. In the case of $p253$, the total costs are lower for computing the whole key exchange compared to $p237$, but both Alice and Bob use around eight times more computation then Alice does in her $p237$ key exchange.

As comparison we also added the costs for a p_{EN} key exchange. Since the key size of p_{EN} is around half of that of $p237$ and $p253$, we can make a comparison of costs by doubling the costs for the p_{EN} key exchange, as the costs of a B-SIDH key exchange scale linearly when the key size grows exponentially (assuming the size of the primes dividing $p-1$ and $p+1$ stays the same, and knowing that point multiplication using the Montgomery ladder scales logarithmically). We would then get "costs" of $86000M$ for Alice, and 84000M for Bob. We then see that the costs for Alice and Bob to compute a p_{EN} key exchange are in comparison still higher than the costs for Alice to compute a $p237$ key exchange, which is to be expected as her largest primes in $p237$ are 10 times smaller than the keys of Alice and Bob in p_{EN}. But compared to $p253$, the costs of p_{EN} are still around 5 times cheaper. This indicates how much costs savings can be gained when we find more optimal primes for B-SIDH.

Now we look a bit closer into costs of specific isogenies. If we take the estimation from Chapter 5 that we would need at least $B = 1750$ as a smoothness bound for 230-bit smooth primes, we use Equation (3.17) to compute an estimation for a 1750-degree isogeny, in which we compute the costs given in millions of $\mathbb{F}_{p^2}$ multiplications. The costs are for performing KPS, xISOG and one xEVAL computation using the $\sqrt{}$élu formula of Section 3.6.

$$cost(1750) = 8834M.$$

If we compare that to the costs of just one 76667-degree isogeny as used in Bob's torsion of $p237$, that has a computed cost of

$$cost(76667) = 171929M.$$

To compare we note that one 76667-isogeny equals the costs of one 1750 and one 43 degree isogeny, which has a cost of 270 multiplications, we see that it is over 10 times cheaper to use the 1750-degree isogeny. In conclusion we see that our found prime p_{EN} would give a cheap B-SIDH key exchange had it been of the correct length. We hope in the future the extending neighbours method may lead to a viable smooth prime for B-SIDH key exchange.

7. Analysis of B-SIDH security

In this chapter we will introduce an encryption scheme for BSIDH, and adapt a known attack on SIDH to the B-SIDH encryption scheme, by applying it to larger degree isogenies. As far as we know, this scheme has not been adapted to large degree isogenies before. This chapter can therefore be seen as one of the original contributions of the thesis.

We can adapt the SIDH and B-SIDH protocols from a key-exchange protocol to an encryption scheme. Key exchange protocols of SIDH and B-SIDH are described in Sections 4.2.1 and 6.2 respectively. In this case, both parties simultaneously generate private and public keys, resulting in a shared secret key, which can then be used as a key for a different cryptographic system to encode a message, for instance using AES. In the encryption protocol however, Bob uses Alice's public keys to encrypt a message, and Alice uses Bob's public keys to decrypt this message. So the SIDH or B-SIDH key would be used as a one-time pad to encrypt a message, without the addition of any other cryptographic protocol. The same encryption protocol can be described for both SIDH and B-SIDH.

In this chapter we will first describe the encryption protocol for B-SIDH, then give a short summary of an attack on SIDH as described in [30]. We then adapt this protocol to work for general isogenies with large prime factors, such as the ones used in B-SIDH, and give a short overview of its costs. In the final section we will describe specific countermeasures for B-SIDH against the attack of section 7.1.1.

7.1 The encryption protocol

The use of encryption protocol is to encode a message m using a shared secret key established by a B-SIDH protocol. Below an encryption protocol for B-SIDH is given. Assume Bob wants to send Alice a secret message using B-SIDH as encryption method. They agree on a curve E over a field K, with Alice working in $E[t_A]$ and Bob working in $E[t_B]$. The procedure uses the same secret key $j(E_AB)$, the j-invariant of their shared curve E_AB, as the key exchange protocols. To protect the secret key j_{AB}, it will be hashed before being used. For this they agree on a hash function $H_k : \mathbb{F}_{p^2} \to \{0,1\}^n$, where $k \in \mathbb{F}_{p^2}$ is a random key chosen by Alice. Alice picks random integers $0 \leq a_1, a_2 \leq t_A$, with either a_1 or a_2 coprime to t_a and computes $E_A, \phi_A(Q_B), \phi_A(P_B)$ as described in 6, with $\phi_A : E \to E/\langle [a_1]P_a, [a_2]Q_A \rangle$. Bob receives Alice's public key. He has his message $m \in \{0,1\}^w$. He picks random integers $0 \leq b_1, b_2 \leq t_B$, with at least b_1 or b_2 coprime to t_B and computes $E_B, \phi_B(Q_A), \phi_B(P_A)$. He then computes E_{AB} using his secret keys. Now he can encode the message m, $c = m \oplus H_k(j(E_{AB})$. He sends the tuple $(E_B, \phi_B(Q_A), \phi_B(P_A), c)$ to Alice. To decode the message, Alice computes E_{AB} using $\phi_B(Q_A), \phi_B(P_A)$ and E_B. She recovers m by $m = c \oplus H_k(j(E_{AB}))$. In the table below the protocol is explained schematically.

B-SIDH encryption protocol

Public parameters :

$E,\ K,\ P_A,\ P_B,\ Q_A,\ Q_B, f_k$

Alice		**Bob**
$R_A = [a_1]P_A + [a_2]Q_A$		
$\phi_A : E \to E_A = E/\langle R_A \rangle$		
$k \leftarrow \mathbb{F}_{p^2}$		
	$\xrightarrow{E_A, \phi_A(P_B), \phi_A(Q_B), k}$	
		$R_B = [b_2]P_B + [b_1]Q_B$
		$\phi_B : E \to E_B = E/\langle R_B \rangle$
		$E_{AB} = E_A/\langle [b_1]\phi_A(P_B), [b_2]\phi_A(Q_B) \rangle$
		m is Bob's secret message
		$c = m \oplus H_k(j(E_{AB}))$
	$\xleftarrow{E_B, \phi_B(P_A), \phi_B(Q_A), c}$	
$E_{AB} = E_B/\langle [a_1]\phi_B(P_A), [a_2]\phi_B(Q_A) \rangle$		
$m = c \oplus H_k(j(E_{AB}))$		
Output: m		

In the method described above we used a secret key for Alice (a_1, a_2). Now we show that instead of a_1, a_2, Alice could also pick only one secret key a.

Theorem 7.1.1. Let $P, Q \in E[t_A]$ be linearly independent generators of $E[t_A]$. Then for some $(a_1, a_2) \in \mathbb{Z}^2$ with at least one coprime to t_A, we have that $(a_1, a_2) \sim (1, \alpha)$ or $(a_1, a_2) \sim (\alpha, 1)$ for some $\alpha \in \mathbb{Z}$.

Proof. We define $(a_1, a_2) \sim (a_1', a_2')$ if $\langle [a_1]P + [a_2]Q \rangle = \langle [a_1']P + [a_2']Q \rangle$ for all $P, Q \in E[p-1]$. This relation is satisfied if $(a_1, a_2) = (\theta a_1', \theta a_2')$ for any $\theta \in \mathbb{Z}_{2^n}^*$. Now if a_1 is coprime to t_A, we have that a_1 is invertible modulo the order of the group. So let $\theta \equiv a_1^{-1} \mod p-1$, then θ is also not divisible by any of the q_k, and thus we will have

$$\langle [a_1]P + [a_2]Q \rangle = \langle [\theta a_1]P + [\theta a_2]Q \rangle = \langle P + [\alpha]Q \rangle.$$

The first equality holds since θ is coprime to t_A. The second equality follows if we define $\alpha = \theta a_2$. If a_1 is dividable by at least one of the q_k, a_2 cannot be divided by any of them, and we can show using the same procedure that $(a_1, a_2) \sim (1, \alpha)$ □

So there is no loss of generality for Alice to have her key of the form $(1, \alpha)$ or $(\alpha, 1)$ instead of (a_1, a_2) But even if Alice does not use such a simplification, an attacker can assume the secret key is one of these forms without any loss of generality.

7.2 An attack on encryption protocol

In this section we will describe an attack on encryption protocols where Alice's torsion group has size ℓ^n, for a prime number ℓ. This is for instance the case in the SIDH protocol. The section below is an adaptation of the attack described in [30]. In this paper only the case $\ell = 2$ is treated

explicitly, but in Remark 2 it is explained shortly how to extend the attack to $\ell = p$ for any prime p. In the next section, we will look at the generic case where the torsion group equals $\prod_{i=1}^{m} p_i^{k_i}$.

Assume we have access to either one of these oracles:

1. $O(E, R, S) = E/\langle [a_1]R + [a_2]S \rangle$.
   In the case of the key exchange protocol, this corresponds to Alice taking Bob's public keys, completing her side of the protocol and giving the shared j-invariant as output. In the encryption protocol, this corresponds to an encryption without the hash function $c = m \oplus j(E_{AB})$, and Alice decrypting Bob's ciphertext and returning the plaintext m.

2. $O(E, R, S, E')$ which returns 1 if $j(E') = j(E/\langle [a_1]R + [a_2]S \rangle)$ and 0 otherwise.
   In the setting of the encryption protocol, this corresponds to Bob having access to a decryption oracle for Alice. By choosing a random ciphertext c, Bob could ask for a decryption of (E_B, R, S, c) and get an m such that $c = m \oplus H_k(j(E_{AB}))$. Bob can then check whether or not $c \oplus m = H_k(j(E'))$

First we will explain the attack for Oracle 2, which is a more complicated attack but also is not as easy to counter . Due to the fact that for the encryption protocol this attack corresponds to an attack on a hashed secret key $j(E_{AB}$, we will call this attack the "attack with hashed key", and the attack using Oracle 1 the "attack with unhashed key."

7.2.1 Attack with hashed key

First step of the attack As described in theorem 7.1.1 above, we can assume that Alice's key is either $(\alpha, 1)$ or $(1, \alpha)$.

The attacker generates $E_b, S = \phi_B(Q_A), R = \phi_B(P_A)$ and E_{AB} as in the protocol. We can write Alice's secret key α in ℓ-expansion: $\alpha = \alpha_0 + \alpha_1 \cdot \ell + ... + \alpha_n \cdot \ell^n$. If we want to recover α_i, we can write the key as $\alpha = K_i + \alpha_i \cdot \ell^i + \alpha' \cdot \ell^{i+1}$, with K_i known, α_i and α' unknown, and $\alpha_i \in \{0, ..., \ell - 1\}$.

To find the first ℓ-ary bit, query $(E_B, R, S + [m][\ell^n - 1]R, E_{AB})$ to the oracle for $m \in \{0, .., \ell - 1\}$.

If the oracle returns 1 then $E_B/\langle [a_1]R + [a_2](S + [\ell^{n-1}]R) \rangle$ is isomorphic to $E_B/\langle [a_1]R + [a_2]S) \rangle$. In order to find the first ℓ-ary bit, query different values for m to the oracle until it returns 1.

Continuation of the attack After the first ℓ-ary bit α_0 has been recovered, we now continue to find the whole key $\alpha = \alpha_0 + \alpha_1 \cdot \ell + ... + \alpha_n \cdot \ell^n$. We will do this iteratively by finding each α_i after recovering a_0 to a_{i-1} for $i \in \{1...n\}$.

To recover α_i, the attacker has to find integers a, b, c, d that will used in a query to the oracle as follows

$$(E_B, [a]R + [b]S, [c]R + [d]S, E_{AB})$$

For each part of the key α_i, these integers have to satisfy four conditions that prevent the attack from being discovered and recover the bit α_1.

1. if $\alpha_i = m$ then $\langle [a + \alpha c]R + [b + \alpha d]S \rangle = \langle R + [\alpha]S \rangle$
2. if $\alpha_i \neq m$ then $\langle [a + \alpha c]R + [b + \alpha d]S \rangle \neq \langle R + [\alpha]S \rangle$
3. $[a]R + [b]S$ and $[c]R + d[S]$ both have order ℓ^n.
4. The Weil pairing $e_{\ell^n}(\phi_B(P_A), \phi_B(Q_A)) = e_{\ell^n}(P_A, Q_A)^{\deg \phi_B} = e_{\ell^n}(P_A, Q_A)^{p+1}$.

The first two conditions help to distinguish the bit a_i, the third condition prevents the attack from being detected via order checking, and the fourth condition prevents the attack from being detected via Weil pairing validation checks. Taking into account these conditions, these are the proposed values for the integers a_i, b_i, c_i and d_i, to recover a key bit α_i:

$$a_i = 1, \quad b_i = -x \cdot \ell^{n-i-1} \tag{7.1}$$
$$c_i = 0, \quad d_i = 1 + \ell^{n-i-1}. \tag{7.2}$$

Here $x = K_i + m$ for an $m \in \{0, ..., \ell - 1\}$. Now we will verify that these integers indeed satisfy all four conditions as stated above.

First we check the first and second condition. We have that with the integers as described in Equation 7.1 we get for the subgroup computation

$$\begin{aligned}
&\langle R - [x \cdot \ell^{n-i-1}]S + [\alpha][1 + \ell^{n-i-1}]S\rangle \\
&= \langle R + [\alpha]S + [\ell^{n-i-1}][\alpha - x]S\rangle \\
&= \langle R + [\alpha]S + [\ell^{n-i-1}][K_i + \ell^i \cdot \alpha_i + \ell^{i+1} \cdot \alpha' - (K_i + m)]S\rangle \\
&= \langle R + [\alpha]S + [\ell^{n-1}(\alpha_i - m)]S\rangle.
\end{aligned}$$

This is equal to $\langle R + [\alpha]S\rangle$ if and only if

$$(\ell^{n-i-1})(\alpha_i - m) \equiv 0 \mod \ell^n. \tag{7.3}$$

So we would need at most $\ell - 1$ queries to recover α_i, where we change $m \in \langle 0, ..., \ell - 1\rangle$ until Equation 7.3 is satisfied.

For the satisfaction of the third condition, we note that

$$[a]R + [b]S = \text{ord}(R + [x \cdot \ell^{n-i-1}]S) = \ell^n,$$

since R and S are linearly independent.Since $1 + \ell^{n-i-1}$ is coprime to ℓ^n we also have

$$\text{ord}([c]R + [d]S) = \text{ord}([1 + \ell^{n-i-1}]S) = \ell^n.$$

To make sure the fourth condition is also met, we make a slight adaptation. When we compute the Weil pairings of the two points, we get

$$\begin{aligned}
&e_{\ell^n}(R', S') = e_{\ell^n}(R - [x\ell^{n-i-1}]S, [1 + \ell^{n-i-1}]S) \\
&= e_{\ell^n}(R, [1 + \ell^{n-i-1}]S) \cdot e_{\ell^n}(R, [x\ell^{n-i-1}]S) = e_{\ell^n}(R, S)^{1+\ell^{n-i-1}},
\end{aligned}$$

which is not the correct value. So we need to choose a θ such that

$$e_{\ell^n}(\theta R', \theta S') = e_{\ell^n}(R, S)^{\theta^2(1+\ell^{n-i-1})} = e_{\ell^n}(R, S) = e_{\ell^n}(P_A, Q_A)^{p+1}$$

Note that

$$\langle [\theta]R' + [\alpha][\theta]S'\rangle = \langle [\theta](R' + [\alpha]S'\rangle = \langle R' + [\alpha]S'\rangle$$

as long as θ is coprime to the order ℓ^n. This means that θ has to be the square root of the inverse of $1 + \ell^{n-i-1}$ modulo ℓ^n,

$$\theta = \sqrt{(1 + \ell^{n-i-1})^{-1}} \mod \ell^n.$$

Does such a square root exist for all primes ℓ?

The following lemmas proof that this is indeed the case. The lemmas are stated here without proof as they are basic number theoretic results.

Lemma 7.2.1. A number x is a quadratic residue modulo p^m for a prime $p > 2$ if and only if x is a quadratic residue modulo p

Lemma 7.2.2. If $x \equiv 1 \mod n$, then $x^{-1} \equiv 1 \mod n$.

For $1 + \ell^{n-i-1}$ modulo ℓ^n we now have that

$$1 + \ell^{n-i-1} \equiv 1 \mod \ell \quad \text{if} \quad n - i - 1 \geq 1,$$

so

$$(\ell^{n-i-1})^{-1} \equiv 1 \mod \ell$$

by lemma 7.2.2. And 1 is always a quadratic residue modulo ℓ, so $(\ell^{n-i-1})^{-1}$ is a quadratic residue modulo ℓ, and therefore a quadratic residue modulo ℓ^n. This concludes that there does indeed exist a θ such that

$$\theta = \sqrt{1 + \ell^{n-1-i}} \mod \ell^n,$$

and therefore an attacker can create a key that disguises as Alice's key, and will not be noticed under Weil pairing verification.

To execute such an attack would require on average $n\frac{\ell-1}{2}$ queries and is bounded by $n(\ell - 1)$ queries. This grows rapidly when ℓ grows in size.

7.2.2 Attack with unhashed key

In this section we will give the attack as given in Appendix B of [30], corresponding to having access to Oracle 1. In the case of the encryption protocol, this would give us access to an unhashed version of the secret key. This attack is more powerful especially when ℓ is large, but is also easy to detect. We again have Alices secret key α we are trying to recover. We assume we have recovered the first i ℓ-ary bits of α, so we know

$$\alpha = K_i + \ell^i \alpha_i + \ell^{i+1} \alpha',$$

with K_i known and a_i, a' unknown. The attacker computes E_B, $R = \phi_B(P_A)$ and $\phi_B(Q_A)$, and also the final curve E_{AB}. He then queries the oracle on $(E_B, R, [\ell^{n-1-i}]S)$. The oracle computes the j-invariant of the elliptic curve

$$E_B/\langle R + [\alpha][\ell^{n-i-1}]S\rangle = E_B/\langle R + [K_i + \ell^i + \ell i + 1\alpha'][\ell^{n-i-1}]S\rangle \tag{7.4}$$

$$= E_B/\langle R + [\ell^{n-i-1}][K_i]S + [\ell^{n-1}\alpha_i]S\rangle. \tag{7.5}$$

Since we know K_i, we can compute $R + [\ell^{n-i-1}][K_i]S$. By trying all $\ell - 1$ different options for a_i the attacker can recover a_i. Since the attacker knows $j(E_{AB})$, he only needs to query the oracle once for each α_i, as he can then compute

$$j(E_B/\langle R + [K_i]S + [x]S\rangle \tag{7.6}$$

for all x in $\{0, .., \ell - 1$. The moment the j-invariant of 7.6 matches the j-invariant of 7.4, we know α_i. This attack then only uses n queries to the oracle in total, independent of the size of ℓ. This is especially an improvement for large ℓ.

7.3 Generalisation to arbitrary numbers

In this section we will generalise the attack to any number $n = \prod p_i^{k_i}$, with p_i prime. This gives the attack that can be used for attacking the B-SIDH protocol, and has to the best of our knowledge not been published before.

Take any integer

$$t_A = \prod_{i=1}^{m} p_i^{k_i}.$$

In this section we will describe the active attack on the torsion group $E(t_A)$. Similarly to the previous section, we need integers a, b, c, d such that we can query the oracle with

$$(E_B, [a]R + [b]S, [c]R + [d]S, E_{AB}) \tag{7.7}$$

and retrieving the secret key of Alice. For this, equation 7.7 must satisfy the same four conditions as in the previous section. The difference here is that we are working over a random number t_A that is not prime. For this, we split the process into different prime divisors p_k of t_A.

$$a_{k,i} = 1, \quad b_{k,i} = -x \cdot p_k^{n_k - i - 1} \cdot \prod_{j=1, j\neq k}^{m} p_j^{n_j}$$
$$c_{k,i} = 0, \quad d_{k,i} = 1 + p_k^{n_k - i - 1} \cdot \prod_{j=1, j\neq k}^{m} p_j^{n_j}.$$

Alice's secret key α is defined modulo $\prod_{j=1}^{m} p_j^{n_j}$. We can find $\alpha_j \mod p_j^{n_j}$ for all primes p_j as will be described below, and since the p_j are all coprime, we can use the Chinese remainder theorem to find

$$\alpha \mod \prod_{j=1}^{m} p_j^{n_j}.$$

We will check the same conditions (1) to (4) as given in Section 7.2.1. The first two conditions remain the same, and they still hold in the general case:

$$\langle R - [x \cdot p_k^{n_k - i - 1} \cdot \prod_{j=1, j\neq i}^{m} p_j^{n_j}]S + [\alpha][1 + p_k^{n_k - i - 1} \cdot \prod_j = i^m p_j^{n_j}]S\rangle$$
$$= \langle R + [\alpha]S - [x - \alpha][p_k^{n_k - i - 1} \cdot \prod_{j=1, j\neq i}^{m} p_j^{n_j}]S\rangle$$

We have

$$\langle R + [\alpha]S - [x - \alpha][p_k^{n_k - i - 1} \cdot \prod_{j=1, j\neq k}^{m} p_j^{n_j}]S\rangle = \langle R + [\alpha]s\rangle$$

when

$$(x - \alpha)(p_k^{n_k - i - 1} \cdot \prod_{j=1, j\neq k}^{m} p_j^{n_j}) = 0 \mod \prod_{j=1}^{m} p_j^{n_j}.$$

Rewriting this gives

$$(x - \alpha)(p_k^{n_k - i - 1}) = 0 \mod p_k^{n_k}. \tag{7.8}$$

Using the same tactics and notation as in Section 7.3, if we write α as

$$\alpha = K_i + \alpha_i \cdot p_k^{n_k} + \alpha'$$
$$x = K_i + m,$$

then Equation 7.8 only holds if $\alpha_{k,i} = m$. Applying this method , we can find

$$\alpha_k = \alpha \mod p_k^{n_k} \quad \text{for all} \quad p_k.$$

To find out α after retrieving all α_k, we use the Chinese remainder theorem as described in Theorem 2.5: For simplicity assume we have two primes p_1 and p_2, with associated keys α_1 and α_2. In the general case, repeat the process with each extra prime number. Then the key α can be recovered as follows

$$\alpha \equiv \alpha_{p_1} \mod p_1^{n_1}$$
$$\alpha \equiv \alpha_{p_2} \mod p_2^{n_2}.$$

There exist integers m_1 and m_2 such that $m_1 n_1 + m_2 n_2 = 1$, which can be found by the Euclidean algorithm. A solution for α is then given by

$$\alpha = \alpha_{p_1} m_2 n_2 + \alpha_{p_2} m_1 n_1.$$

The third condition is satisfied, as we have for the order of the subgroups:

$$\mathrm{ord}([a]R + b[S]) = \mathrm{ord}([c]R + [d]S) = \prod_{j=1}^{m} p_j^{n_j}.$$

This equality holds since

$$\mathrm{ord}([a]R + b[S]) = \mathrm{ord}(R - [x \cdot p_k^{n_k - i - 1}]S)$$

as S and R are linearly independent, and

$$\mathrm{ord}([c]R + [d]S) = \mathrm{ord}(1 + p_k^{n_k - i - 1} \cdot \prod_{j=1, j \neq k}^{m} p_j^{n_j})$$

since

$$(1 + p_k^{n_k - i - 1}) \cdot \prod_{j=1, j \neq k}^{m} p_j^{n_j}$$

is linearly independent to

$$\prod_{j=1}^{m} p_j^{n_j}.$$

For the fourth condition we need to find, just like in the previous section, a θ such that for

$$D = \prod_{j=1}^{m} p_j^{n_j}$$

we have

$$e_D(\theta R', \theta S') = e_D(R, S)^{\theta^2 (1 + D \cdot p_k^{n_k - i - 1})} = e_D(R, S)$$

Taking into account that

$$\langle [\theta]R' + [\alpha][\theta]S' \rangle = \langle [\theta](R' + [\alpha]S' \rangle = \langle R' + [\alpha]S' \rangle$$

as long as θ is coprime to the order $\prod_{j=1}^{m} p_j^{n_j}$. So, we need θ to be

$$\theta = \sqrt{(1 + p_k^{n_k - i - 1} \cdot \prod_{j=i}^{m} p_j^{n_j})^{-1}} \mod \prod_{j=1}^{m} p_j^{n_j}$$

To find this θ, we again need a lemma from number theory:

Lemma 7.3.1. For a composite number

$$N = \prod_{j=1}^{m} p_j^{n_j},$$

x is a square modulo N if and only if N is a square modulo all $p_j^{n_j}$.

Now we can apply this lemma together with lemmas 7.2.1 and 7.2.2 to get that such a θ exists, in the case all prime numbers are odd. First, it is clear that

$$1 + p_k^{n_k - i - 1} \cdot \prod_{j=1, j \neq n}^{m} p_j^{n_j} \mod \prod_{j=1}^{m} p_j^{n_j} \equiv 1 \mod p_j^{n_j} \quad \text{for all} \quad j \neq k.$$

Using lemma 7.2.1 we can show that this is also in the case $j = k$. Then using lemma 7.2.2 we now get that

$$(1 + p^{n_k - i - 1} \cdot \prod_{j=1, j \neq i}^{m} p_j^{n_j})^{-1} \mod p_j^{n_j} \equiv 1 \mod p_j^{n_j},$$

thus the inverse is square for all prime numbers. and then using lemma 7.3.1 it can be shown that

$$(1 + p^{n_k - i - 1} \cdot \prod_{j=1, j \neq i}^{m} p_j^{n_j})^{-1}$$

is a square number modulo $\prod_{j=1}^{m} p_j^{n_j}$, thus proving such a θ exists.

In conclusion, the four conditions given in section 7.2.1 hold in the general case over all numbers N, not just powers of prime numbers.

The algorithm for computing the keyα for arbitrary N is given below, with

$$D = \prod_{j=1}^{m} p_j^{n_j}.$$

Note that this protocol works for all $p \geq 3$. For $p = 2$, as described in [30], θ does not always exist if $n - i - 1 \leq 3$. This means that the last 2 bits of α_2 can only be found using brute force. This means for finding the key α, first

$$\alpha' = \alpha \mod \prod_{j=1, p_j \neq 2}^{m} p_j^{n_j}$$

has to be found using the Chinese Remainder Theorem as described above, and the final key α has to be found by computing all possible values for α_2 and applying the CRT to α' and all possible values of α_2.

Algorithm 14 Adaptive attack on BSIDH

Data: $D, P_A, Q_A, P_B, Q_B, \phi_A(Q_B), \phi_A(P_B)$
Result: Alice's secret key α
Set $\alpha = 0$.
for $k = 1$ *to* m **do**
  Set $\alpha_k = 0$
  for $i = 0$ *to* n **do**
    Set $\alpha_{k,i} = 0$
    Choose random (b_1, b_2)
    Set $G_B = \langle [b_1]P_B + [b_2]Q_B \rangle$
    Set $\phi_B : E \to E_B = E/G_B$
    Set $(R, S) = (\phi_B(P_A), \phi_B(Q_A))$
    Set $E_{AB} = E_A/\langle [b_1]\phi_A(P_B) + [b_2]\phi_A(Q_B) \rangle$
    Set $\theta = \sqrt{(1 + p^{n_k - i - 1} \cdot \prod_j = i^m p_j^{n_j})^{-1}} \mod \prod_{j=1}^{m} p_j^{n_j}$
    for $x = 1$ *to* $p_k - 1$ **do**
      Query the oracle on $(E, [\theta](R - [K_{k,i}x.]S), [\theta][1 + .]S, E_{AB})$
      If *Response is true,* $\alpha_{k,i} = x$
    end
    Set $\alpha_k = K_{k,i} + p_k^i + \alpha_{k,i}$
  end
  Set $\alpha = \mathrm{CRT}(\alpha, \alpha_k)$
end
return α

7.3.1 Costs of active attack

As remarked already in [30], an adversary needs to query a lot more to the oracle when the prime numbers get larger. In Algorithm 14, the number of queries to the oracle $\#Q$ is at maximum

$$\#Q \leq \sum_{j=1}^{m} n_j \cdot (p_j - 1) \tag{7.9}$$

if the order of the torsion group is $\prod_{j=1}^{m} p_j^{n_j}$. For SIDH, a torsion group of Alice would be around the size 2^{250}, which would give the same security as using the prime p_{PTE} as given in Chapter 5. For the SIDH case, an attacker would need 250 queries to retrieve α. In the case of B-SIDH, there is not one simple calculation. We give an example of the costs with prime p_{PTE} that is used in the last chapter, and calculate the costs for Alice's torsion group of order N, where N equals

$$N = 3^2 \cdot 23^2 \cdot 41^2 \cdot 71^2 \cdot 83^2 \cdot 919^2 \cdot 1117^2 \cdot 1163^2 \cdot 1237^2 \cdot 6571^2 \cdot 11927^2 \cdot 18637^2 \cdot 32029^2. \tag{7.10}$$

We for now assume the attacker wants to find out the whole key a by using queries, and does not at some point switch to using brute force. As the maximum amount of queries Q is given by

$$Q = \sum (p_i - 1) \cdot e_i,$$

where the p_i are the primes in equation 7.10, each occuring e_i times. This yields a total of 147616 maximum queries to find a, and an expected value of 73808 queries to find a. When we compare the results for SIDH and our example of B-SIDH, we see that it would take around an expected 300 times more queries to retrieve the B-SIDH key then the SIDH key.

But there is a large caveat here. There is a trade-off between queries and computing the key by brute force. For in the case of SIDH, when an attacker has made half of its queries, around 150, he would still need to try an expected 2^{124} solutions before he finds the complete key. For B-SIDH, this is not the case. He could first query to the oracle to find the keys a_i modulo the smallest

prime factors p_i. Then, using the Euclidian algorithm, brute force the key. So in our example with p_{PTE}, the attacker could for instance decide not to query to find the keys a_i modulo 32029 and 18637. When we assume the maximum amount of queries needed, this saves the attacker 101328 out of 147616 queries. He computes a', the key modulo all primes except 32029 and 18637, and then tables all possible values x of $32029^2 * 18637^2$, around 10^{18} values, and computes with the Euclidean algorithm the 10^{18} possible values for the key a. This way we can reduce the amount of queries needed by around 66%. Another note that is to be made is that when B-SIDH primes get smoother, the amount of queries required will decrease significantly. But then this caveat will also be less exploitable.

Practically, what does this increased costs of the attack mean? While on paper, it looks like a 400 time increase in queries is quite significant. This could imply that in some cases we would not need to apply the costly countermeasures as described in Section 7.4, where for SIDH we would need them. If we only use the key once, both SIDH and B-SIDH would not need extra security measures. If we want to use our keys more than once, this is where we might find an application where SIDH would need extra protective measures, but B-SIDH does not. As described in Section 1.1.2, for many instances we use static keys - secret keys are hardcoded into the application, so one cannot change the secret key after a certain amount of queries have been made. But against side channel attacks, it may have some applications. Also, there are some instances where we could use ephemeral private keys, for instance with website security certificates, where private keys are reestablished after a certain amount of time or uses. An example of this would be Microsofts SChannel TLS, which used for some time a private key that was reset after one hour [10]. If for instance we would reuse a private key 250 times, an attacker can know a complete SIDH key, but for a B-SIDH key there would still be around 2^{220} options for the key left - so it would still be infeasible for the attacker to retrieve the key. We expect that in most cases where one would take extra measures to protect an SIDH key against this active attack, one would also take those measures for a B-SIDH key. These measures are described in Section 7.4. However, we highly encourage more research into this topic, as both for academic reasons and possible practical reasons, it is interesting to further investigate this difference between the two protocols.

For completeness we will now give the attack with unhashed key for general numbers.

7.3.2 Attack with unhashed key

In this section we modify the attack given in Appendix B of [30] to an attack on the BSIDH protocol. For this attack we use Oracle 1 as given in this chapter.

Assume again Alice has a public key α, and Alice works in $E[\prod_{j=1}^{m} p_j^{n_j}]$. For any given p_k, we can write α as follows,

$$\alpha = (K_i + p_k^i \alpha_{k,i} + p^{i+1} \alpha'_k + \beta).$$

Here K_i is the part of the key modulo $p_k^{n_k}$ that is already known, $\alpha_{k,i} \in \{0, ..., p_k - 1\}$ unknown, α'_k unknown and β the part of the key that is $0 \mod p_k^{n_k}$.

As in the previous attack, the adversary computes $E_B, R = \phi_B(P_A), S = \phi_B(Q_A)$ and he queries the oracle on

$$(E_B, R, p_k^{n_k - i - 1} \cdot \prod_{j=1, j \neq k}^{m} p_j^{n_j}).$$

This elliptic curve that the oracle returns is

$$E_B/\langle R + [\alpha][p_k^{n_k-i-1} \cdot \prod_{j=1,j\neq k}^{m} p_j^{n_j}]S\rangle$$

$$= E_B/\langle R + [p_k^{n_k-i-1} \cdot \prod_{j=1,j\neq k}^{m} p_j^{n_j}][(K_i + p_k^i\alpha_{k,i} + p^{i+1}\alpha'_k + \beta]S\rangle$$

$$= E_B/\langle E + [p_k^{n_k-i-1} \cdot \prod_{j=1,j\neq k}^{m} p_j^{n_j}][K_i]S + [p_k^{n_k-1} \cdot \prod_{j=1,j\neq k}^{m} p_j^{n_j}]S\rangle.$$

The part

$$R + [p_k^{n_k-i-1} \cdot \prod_{j=1,j\neq k}^{m} p_j^{n_j}][K_i]S$$

is known, so the attacker can recover $\alpha_{k,i}$ by trying all p_k different values for $\alpha_{k,i}$. This means that to recover one p_k-ary bit, we only need to query to the oracle once. We can repeat this process for all primes p_k. The total amount of queries needed would be maximum

$$\sum_{j=1}^{m} \lceil \log_{p_j}(n_j) \rceil \tag{7.11}$$

It is clear that this attack can easily be detected using some countermeasures. In the next section we describe the countermeasures against both this attack and the attack using Oracle 2.

7.4 Possible countermeasures against the attacks

To secure the SIDH protocol, there are some validation steps Alice can take to detect an malintendant adversary. Suggestions to this are made in [18], [30] and [35]. First we will discuss the less costly validation steps and show that they protect against the attack given in Section 7.3.2, but not against the attack of Section 7.3. Lastly, we give the Fujisaki-Okamoto protocol for BSIDH and demonstrate that this does in fact protect BSIDH from both attacks.

The first, and most easy protection measure is to hash the j-invariant

This action is the difference between Oracle 1 and Oracle 2. As can be seen in the section above, an oracle that returns a non-hashed j-invariant is vulnerable for simple attacks that retrieve the secret key in a logarithmic scale amount of queries.

A second method is checking if the points have the required order. The order of the points is $D = \prod_{j=1}^{m} p_j^{n_j}$, and one can verify that R and S have the correct order by calculating

$$\left[p_k^{n_k-1} \cdot \prod_{j=1,j\neq k}^{m} p_j^{n_j}\right]R$$

for all p_k. But, in the first attack, the points are specifically designed to have the correct order, and will bypass this validation check.

A third method is computing the Weil pairing of the points R and S. As described in Section 3.7.5, two points are generating the torsion subgroup if and only if the order of the weil pairing is the same as the order of the torsion. But even more than that, one can check, proven in Section 3.7.7, that the points $\phi_B(P_A), \phi_B(Q_A)$ are indeed consistent with being images of the correct points under an isogeny of the correct degree. The attack as given in Algorithm 14 bypasses this validation by multiplying the constructed points with a θ such that the points generate the correct order for the Weil pairing.

The Kirkwood et al. Validation Method The methods described above all fail to protect from the attack of Section 7.3, as the points are specifically constructed to bypass these checks. A more rigorous protection measure is to apply the Fujisaki-Okamoto transform [28] in the context of supersingular isogenies. This was first proposed by Kirkwood et al. in [35], although a specific algorithm was not given. A formal analysis of this protocol is given by [30].

The Fujisaki-Okamoto protocol for BSIDH is identical to the one for SIDH, and is recited below. Bob first picks a random seed r_B and using a pseudo random function he computes $(b_1, b_2) = PRF(r_B)$. He receives the public keys $(E_A, \phi_A(P_B), \phi_A(Q_B))$ from Alice. He computes his public keys $(E_B, \phi_B(P_A), \phi_B(Q_A))$, with $E_B = E_0/\langle [b_1]P_B + [b_2]Q_B \rangle$. He then calculates the shared curve $E_{AB} = E_A\langle [b_1](\phi_A(P_B)) + [b_2]\phi_A(Q_B)\rangle$. Using a key derivation function he computes the session key and the validation key: $SK|VK = KDF(j(E_{AB}))$. Bob then sends his public keys to Alice, together with the encoded secret random seed $c_B = \mathrm{Enc}_{VK}(r_B \oplus SK)$. Alice on her turn computes E'_{AB} from the public keys of Bob, and computes $SK'|VK' = KDF(j(E'_{AB}))$. She tries to retrieve Bob's random seed $r'_B = DEC_{VK'}(c_B) \oplus SK'$. To test if she got the correct random seed, she computes $(b'_1, b'_2) = PRF(r'_B)$, and recalculates Bob's public keys. If these keys correspond to the ones she received from Bob, she will use $SK = SK'$ for future communication. Else, she rejects SK and the protocol terminates.

Proposition 7.4.1. The Kirkwood validation method secures the BSIDH protocol against the attack with hashed key.

Proof. Assume Bob sends a query to the oracle, together with the encoded random seed c_B, as follows

$$\left(E_B, [\theta]\left(R - \left[x \cdot p_k^{n_k - i - 1} \cdot \prod_{j=1, j\neq k}^{m} p_j^{n_j}\right] S\right), [\theta]\left[1 + p_k^{n_k - i - 1} \cdot \prod_{j=1, j\neq k}^{m} p_j^{n_j}\right] S, E_{AB}\right)$$

Alice first computes E'_{AB}, using the public keys. This may match E_{AB} as described in Section 7.3. Now she obtains the correct random seed $r'_B = r_B$. She computes $(b_1, b_2) = PRF(r_B)$. She then calculates $E_B, R = \phi_B(P_A)$ and $S = \phi_B(Q_A)$. But the points Bob sent her are not R, S, but instead

$$R - [x \cdot p_k^{n_k - i - 1} \cdot \prod_{j=1, j\neq k}^{m} p_j^{n_j}]$$

and

$$[\theta][1 + p_k^{n_k - i - 1} \cdot \prod_{j=1, j\neq k}^{m} p_j^{n_j}]S.$$

So Alice knows Bob sent the wrong public keys to her, and terminates the communication. □

In summary we can see that the active attacks as described in [30] also work on B-SIDH, be it in a much less effective way. The same countermeasures against the attack that work for SIDH also work to protect B-SIDH.

Bibliography

[1] Gora Adj, Omran Ahmadi Darvishvand, and Alfred Menezes. "On isogeny graphs of supersingular elliptic curves over finite fields". In: *Finite Fields and Their Applications* 55 (Jan. 2019), pp. 268–283. DOI: 10.1016/j.ffa.2018.10.002.

[2] Gora Adj, Jesús-Javier Chi-Domınguez, and Francisco Rodrıguez-Henrıquez. "On the velu's formulae and its applications to CSIDH and B-SIDH constant-time implementations". In: (2020).

[3] Elaine Barker et al. *Recommendation for Pair-WiseKey-Establishment Schemes Using Discrete Logarithm Cryptography*. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Ar3.pdf. 2018.

[4] Jao et al. *SIKE: supersingular isogeny key encapsulation*. URL: sike.org.

[5] Frank Arute et al. "Quantum supremacy using a programmable superconducting processor". In: *Nature* 574.7779 (2019), pp. 505–510.

[6] Michael Francis Atiyah and Ian Grant Macdonald. *Introduction to commutative algebra*. CRC Press, 2018.

[7] Daniel J Bernstein et al. "Twisted edwards curves". In: *International Conference on Cryptology in Africa*. Springer. 2008, pp. 389–405.

[8] Daniel Bernstein et al. "Faster computation of isogenies of large prime degree". In: *arXiv preprint arXiv:2003.10118* (2020).

[9] Xavier Bonnetain and André Schrottenloher. "Quantum security analysis of CSIDH". In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2020, pp. 493–522.

[10] Colin Boyd, Anish Mathuria, and Douglas Stebila. *Protocols for authentication and key establishment*. Vol. 1. Springer, 2003.

[11] G Bruno. *Finding B-smooth primes*. https://github.com/GiacomoBruno/BSmooth. 2021.

[12] Wouter Castryck et al. "CSIDH: an efficient post-quantum commutative group action". In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2018, pp. 395–427.

[13] Jesús-Javier Chi-Domınguez and Francisco Rodrıguez-Henrıquez. "Optimal strategies for CSIDH". In: *Advances in Mathematics of Communications* (2020).

[14] Andrew Childs, David Jao, and Vladimir Soukharev. "Constructing elliptic curve isogenies in quantum subexponential time". In: *Journal of Mathematical Cryptology* 8.1 (2014), pp. 1–29.

[15] J. B. Conrey, M. A. Holmstrom, and T. L. McLaughlin. "Smooth Neighbors". In: *Experimental Mathematics* 22.2 (2013), pp. 195–202. DOI: 10.1080/10586458.2013.768483. eprint: https://doi.org/10.1080/10586458.2013.768483. URL: https://doi.org/10.1080/10586458.2013.768483.

[16] Craig Costello. *B-SIDH: supersingular isogeny Diffie-Hellman using twisted torsion*. Cryptology ePrint Archive, Report 2019/1145. https://eprint.iacr.org/2019/1145. 2019.

[17] Craig Costello and Huseyin Hisil. "A simple and compact algorithm for SIDH with arbitrary degree isogenies". In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2017, pp. 303–329.

[18] Craig Costello, Patrick Longa, and Michael Naehrig. "Efficient algorithms for supersingular isogeny Diffie-Hellman". In: *Annual International Cryptology Conference*. Springer. 2016, pp. 572–601.

[19] Craig Costello, Michael Meyer, and Michael Naehrig. "Sieving for twin smooth integers with solutions to the Prouhet-Tarry-Escott problem". In: ().

[20] Craig Costello and Benjamin Smith. "Montgomery curves and their arithmetic". In: *Journal of Cryptographic Engineering* 8.3 (2018), pp. 227–240.

[21] Craig Costello et al. "Improved classical cryptanalysis of SIKE in practice". In: *IACR International Conference on Public-Key Cryptography*. Springer. 2020, pp. 505–534.

[22] Jean-Marc Couveignes. *Hard Homogeneous Spaces*. Cryptology ePrint Archive, Report 2006/291. `https://eprint.iacr.org/2006/291`. 2006.

[23] Richard Crandall and Carl B Pomerance. *Prime numbers: a computational perspective*. Vol. 182. Springer Science & Business Media, 2006.

[24] Joan Daemen and Vincent Rijmen. "AES proposal: Rijndael". In: (1999).

[25] Luca De Feo, David Jao, and Jérôme Plût. "Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies". In: *Journal of Mathematical Cryptology* 8.3 (2014), pp. 209–247.

[26] Luca De Feo et al. "SQISign: compact post-quantum signatures from quaternions and isogenies". In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2020, pp. 64–93.

[27] Whitfield Diffie and Martin Hellman. "New directions in cryptography". In: *IEEE transactions on Information Theory* 22.6 (1976), pp. 644–654.

[28] Eiichiro Fujisaki and Tatsuaki Okamoto. "Secure integration of asymmetric and symmetric encryption schemes". In: *Annual International Cryptology Conference*. Springer. 1999, pp. 537–554.

[29] Steven D. Galbraith. "Constructing Isogenies between Elliptic Curves Over Finite Fields". In: *LMS Journal of Computation and Mathematics* 2 (1999), pp. 118–138. DOI: `10.1112/S1461157000000097`.

[30] Steven D. Galbraith et al. "On the Security of Supersingular Isogeny Cryptosystems". In: *Advances in Cryptology – ASIACRYPT 2016*. Ed. by Jung Hee Cheon and Tsuyoshi Takagi. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 63–91.

[31] Francisco Rodríguez-Henríquez Gora Adj Jesús-Javier Chi-Domínguez. *SIBC Python library*. `https://github.com/JJChiDguez/sibc/`. 2021.

[32] Lov K Grover. "A fast quantum mechanical algorithm for database search". In: *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*. 1996, pp. 212–219.

[33] Robin Hartshorne. *Algebraic geometry*. Vol. 52. Springer Science & Business Media, 2013.

[34] David Jao, Stephen D Miller, and Ramarathnam Venkatesan. "Expander graphs based on GRH with an application to elliptic curve cryptography". In: *Journal of Number Theory* 129.6 (2009), pp. 1491–1504.

[35] Daniel Kirkwood et al. "Failure is not an option: Standardization issues for post-quantum key agreement". In: *Workshop on Cybersecurity in a Post-Quantum World*. 2015, p. 21.

[36] David Russell Kohel. "Endomorphism rings of elliptic curves over finite fields". PhD thesis. University of California, Berkeley, 1996.

[37] Hendrik W Lenstra Jr. "Solving the Pell equation". In: *Notices of the AMS* 49.2 (2002), pp. 182–192.

[38] Florian Luca and Filip Najman. "On the largest prime factor of ^{2}-1". In: *Mathematics of computation* 80.273 (2011), pp. 429–435.

[39] Victor S Miller. "Use of elliptic curves in cryptography". In: *Conference on the theory and application of cryptographic techniques*. Springer. 1985, pp. 417–426.

[40] Peter L Montgomery. "Speeding the Pollard and elliptic curve methods of factorization". In: *Mathematics of computation* 48.177 (1987), pp. 243–264.

[41] Arnold K Pizer. "Ramanujan graphs and Hecke operators". In: *Bulletin of the American Mathematical Society* 23.1 (1990), pp. 127–137.

[42] Arnold K Pizer. "Ramanujan graphs and Hecke operators". In: *Bulletin of the American Mathematical Society* 23.1 (1990), pp. 127–137.

[43] Krissie Pladson. *German industry could win big with new quantum computer*. 2021. URL: https://www.dw.com/en/german-industry-could-win-big-with-new-quantum-computer/a-57920916 (visited on 06/16/2021).

[44] J. M. Pollard. "Theorems on factorization and primality testing". In: *Mathematical Proceedings of the Cambridge Philosophical Society* 76.3 (1974), pp. 521–528. DOI: 10.1017/S0305004100049252.

[45] John Proos and Christof Zalka. "Shor's discrete logarithm quantum algorithm for elliptic curves". In: *arXiv preprint quant-ph/0301141* (2003).

[46] Joost Renes. "Computing isogenies between Montgomery curves using the action of (0, 0)". In: *International Conference on Post-Quantum Cryptography*. Springer. 2018, pp. 229–247.

[47] Ronald L Rivest, Adi Shamir, and Leonard Adleman. "A method for obtaining digital signatures and public-key cryptosystems". In: *Communications of the ACM* 21.2 (1978), pp. 120–126.

[48] Martin Roetteler et al. "Quantum Resource Estimates for Computing Elliptic Curve Discrete Logarithms". In: *Lecture Notes in Computer Science* (2017), pp. 241–270. ISSN: 1611-3349. DOI: 10.1007/978-3-319-70697-9_9. URL: http://dx.doi.org/10.1007/978-3-319-70697-9_9.

[49] Alexander Rostovtsev and Anton Stolbunov. *PUBLIC-KEY CRYPTOSYSTEM BASED ON ISOGENIES*. Cryptology ePrint Archive, Report 2006/145. https://eprint.iacr.org/2006/145. 2006.

[50] P. W. Shor. "Algorithms for Quantum Computation: Discrete Logarithms and Factoring". In: *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*. SFCS '94. USA: IEEE Computer Society, 1994, pp. 124–134. ISBN: 0818665807. DOI: 10.1109/SFCS.1994.365700. URL: https://doi.org/10.1109/SFCS.1994.365700.

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

[52] Carl Størmer. "Quelques théorèmes sur l'équation de Pell x2- Dy2=±1 et leurs applications". In: *Christiania Videnskabens Selskabs Skrifter, Math. Nat. Kl* 2 (1897), p. 48.

[53] Jacques Vélu. "Isogénies entre courbes elliptiques". In: *CR Acad. Sci. Paris, Séries A* 273 (1971), pp. 305–347.

[54] William C Waterhouse. "Abelian varieties over finite fields". In: *Annales scientifiques de l'École normale supérieure*. Vol. 2. 4. 1969, pp. 521–560.

[55] Noson S. Yanofsky and Mirco A. Mannucci. *Quantum Computing for Computer Scientists*. Cambridge University Press, 2008. DOI: 10.1017/CBO9780511813887.

[56] Gustavo HM Zanon et al. "Faster key compression for isogeny-based cryptosystems". In: *IEEE Transactions on Computers* 68.5 (2018), pp. 688–701.

Appendix A

In this appendix we give the histogram plots for the distribution of smooth neighbour pairs and primes when running the extending neighbours algorithm for different smoothness bounds B.

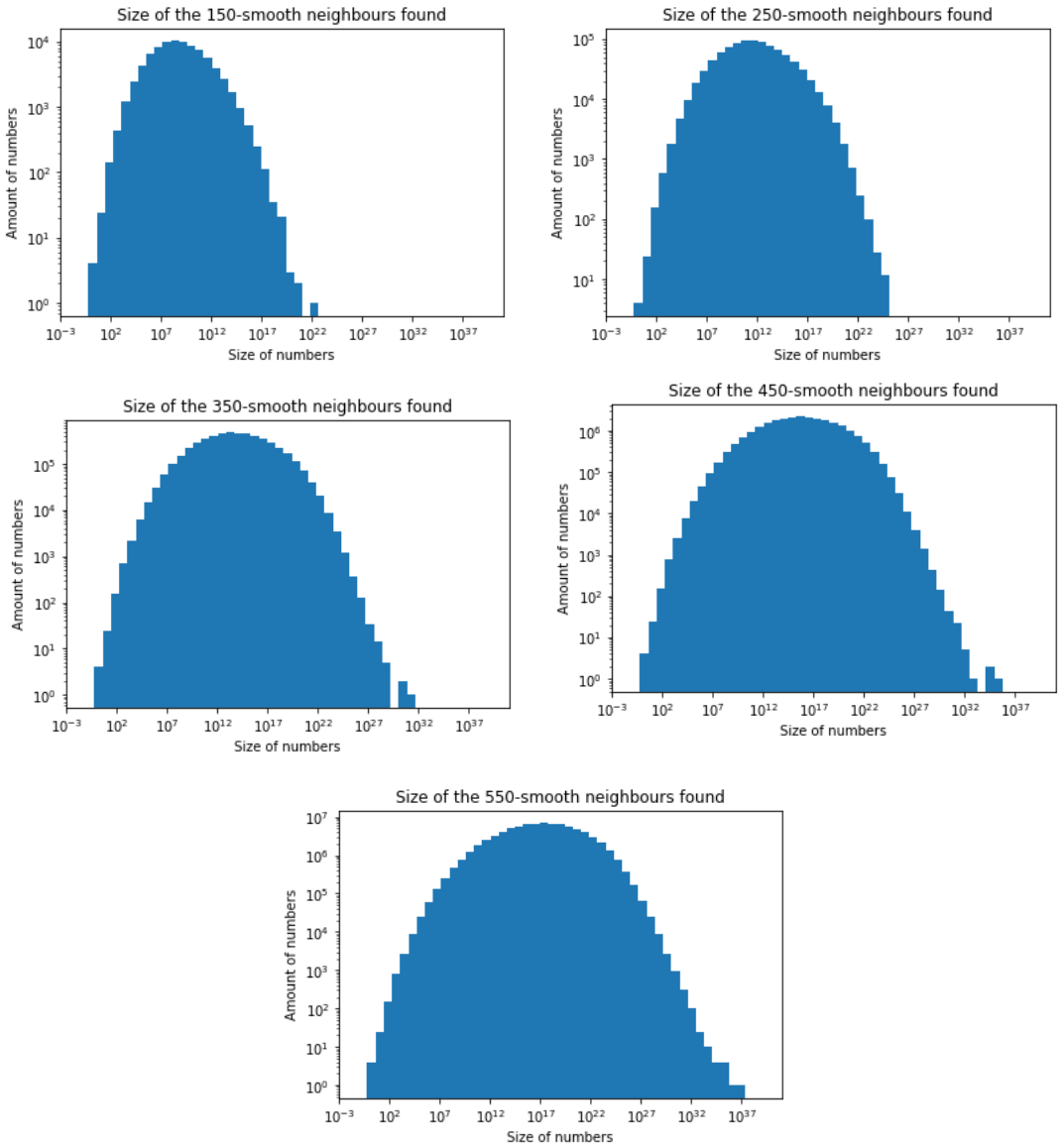


Figure 7.1: Density distribution for smooth neighbour pairs found for different smoothness bounds

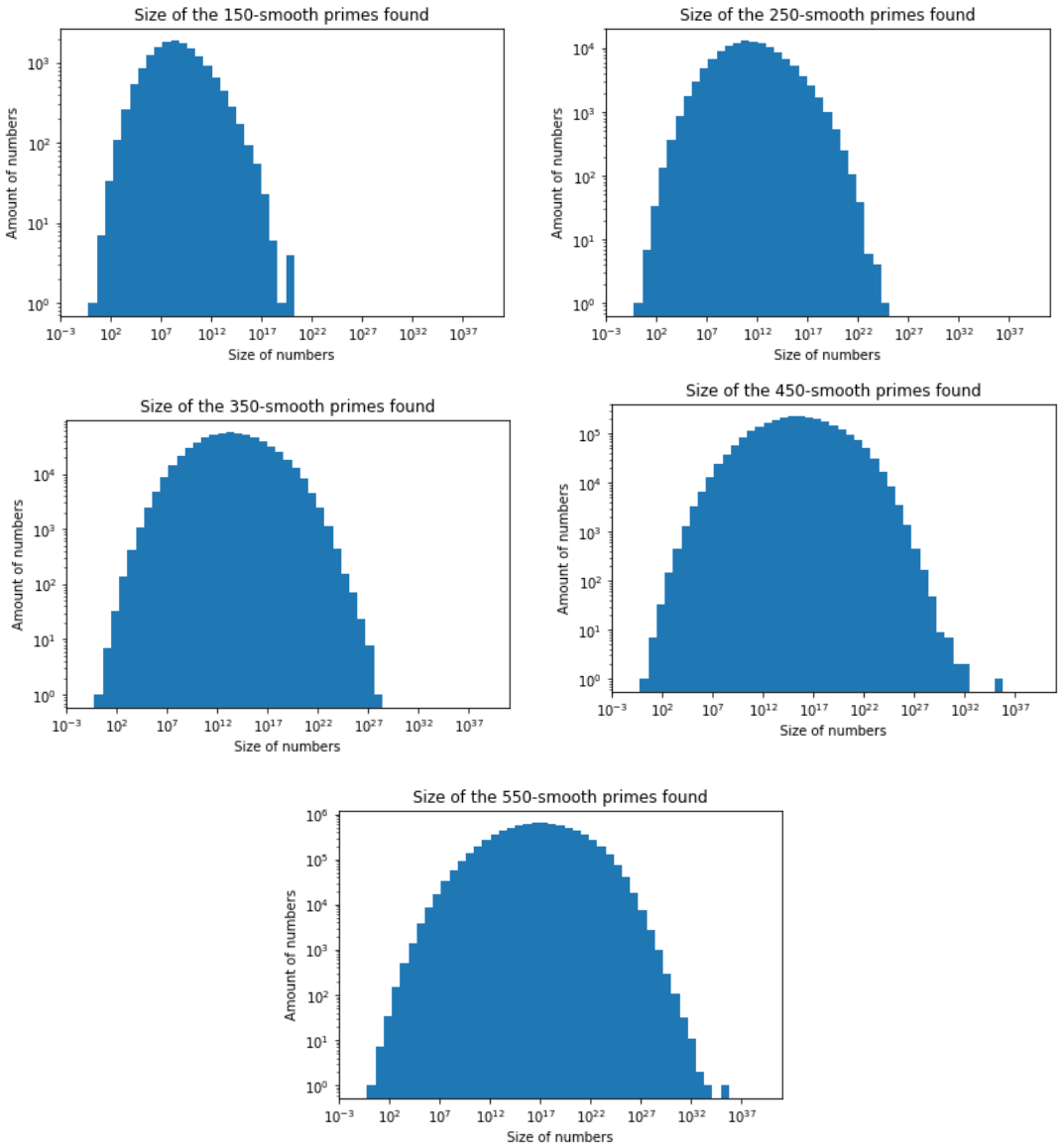


Figure 7.2: Density distribution for smooth neighbour pairs found for different smoothness bounds