Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

Jonathan Komada Eriksen

Applying Twisted Hessian Curves to Supersingular Isogeny Diffie-Hellman

Master's thesis in Communication Technology
Supervisor: Bor de Kock, Colin Boyd
June 2021

NTNU
Norwegian University of Science and Technology

Jonathan Komada Eriksen

Applying Twisted Hessian Curves to Supersingular Isogeny Diffie-Hellman

Master's thesis in Communication Technology
Supervisor: Bor de Kock, Colin Boyd
June 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

Title: Applying Twisted Hessian Curves to Supersingular Isogeny Diffie-Hellman

Student: Jonathan Komada Eriksen

Problem description:

Isogeny-based cryptography is a relatively young field in modern cryptography, which uses the theory of elliptic curves and specifically, special transformations between elliptic curves called isogenies to create cryptographic schemes. Lately, isogeny-based cryptography has gained a lot of attention, because of its resilience against the cryptanalytic power of quantum computers. Compared to other quantum resistant cryptographic schemes, those that arise from isogenies often have certain features which make them promising candidates for replacing traditional Diffie-Hellman based schemes in a world where quantum computers exist. However, a big challenge for isogeny-based cryptography is the high computational costs, which limits the performance of these schemes.

This thesis aims to evaluate the performance of isogeny-based cryptography when using twisted Hessian curves, a specific model of elliptic curves. This will be done to study how well suited twisted Hessian curves are for isogeny-based cryptography. In particular, we will be interested in how we can apply these curves in an efficient way, and to examine what factors limit their performance.

Supervisor: Bor de Kock, IIK

Responsible professor: Colin Boyd, IIK

Abstract

Most public-key cryptography relies on the hardness of one of two mathematical problems. While these problems seem intractable on classical computers, there exists an algorithm for solving both of these problems, which runs on a quantum computer. The result of this is that if a quantum computer large enough to run this algorithm is ever built, it would severely compromise the security of real-world digital communications, such as on the internet. Because of this, new public-key cryptographic standards are currently being developed, which necessarily rely on the hardness of other problems. One example of such a scheme is the Supersingular Isogeny Diffie-Hellman (SIDH) key exchange, which relies on hard problems from the theory of elliptic curves. The hard problems are related to isogenies, which are special, structure-preserving maps between elliptic curves.

In this thesis, we investigate a new way of instantiating the SIDH key-exchange, by using twisted Hessian curves. A twisted Hessian curve is a specific elliptic curve model, which has relatively fast formulae for point doubling and point tripling. Further, direct formulae for isogenies between these curves have recently been studied. We use these formulae to implement the SIDH key exchange. Additionally, we discuss some of the structural properties of twisted Hessian curves, in relation to SIDH. These properties may suggest an alternative way of instantiating the SIDH key exchange, where the requirements for the prime p are significantly relaxed. This gives the possibility of using much smaller field sizes at no loss in security, resulting in key sizes that are about half the size of those used in SIDH today.

Sammendrag

Sikkerheten til nesten all offentlig nøkkelkryptografi er basert på vanskeligheten til et av to matematiske problemer. Selvom disse problemene virker vanskelige på klassiske datamskiner, eksisterer det en algoritme for å løse begge disse problemene, som kun kjører på kvantemaskiner. Resultatet av dette er at dersom det noen gang bygges en kvantemaskin som er stor nok til å kjøre denne algoritmen, vil sikkerheten til digital kommunikasjon, slik som på internett, være ansett som kompromittert. På grunn av dette, utvikles det nå ny offentlig nøkkelkryptografi, som er basert på vanskeligheten av andre matematiske problemer. Et eksempel på et slikt system er Supersingulær Isogeni Diffie-Hellman (SIDH), som baserer seg på vanskelige problemer fra teori om elliptiske kurver. Disse problemene er relatert til isogenier, som er spesielle, struktur-bevarende avbildninger mellom elliptiske kurver.

I denne oppgaven utforsker vi en ny måte å instansiere SIDH på, ved å bruke vridde Hessianske kurver. En vridd Hessiansk kurve er en elliptisk kurve modell, som har relativt raske formler for punkt dobling og tripling. Videre er formler for isogenier mellom slike kurver nylig blitt studert. Vi benytter disse formlene i en implementasjon av SIDH. I tillegg diskuterer vi noen strukturelle egenskaper ved vridde Hessianske kurver, og deres relasjon til SIDH. Disse egenskapene kan gi opphav til en alternativ måte å instansiere SIDH på, hvor kravene til primtallet p er betydelig redusert. Dette gir muligheten til å benytte mye mindre kropper, uten tap av sikker, som igjen resulterer i nøkler som er rundt halvparten så store som de som brukes i SIDH idag.

Preface

This Master's thesis marks the conclusion of my 5 year Master of Science degree in Communications Technology at the Department of Information Security and Communication Technology at the Norwegian University of Science and Technology (NTNU).

I would like to express my sincerest gratitude to my amazing supervisors, Bor de Kock and Colin Boyd, for their invaluable guidance throughout this project. Your expertise in cryptography is truly inspiring, matched only by your brilliant wit. Working with you has been a really enjoyable experience, and I look forward to many more collaborations in the future.

Next, I would like to thank my family. I have, in a way, been lucky enough to have four parents, who have always supported me unconditionally, and for that, I am forever grateful.

Finally, I would like to give many, many thanks to all the fantastic people I have had the pleasure of getting to know here at my stay in Trondheim. These years have been so much fun, and really, I owe it all to you!

Jonathan Komada Eriksen
Trondheim, 2021

Contents

List of Figures

List of Algorithms

Chapter 1
Introduction

Elliptic curves have a long track record in public-key cryptography and currently see widespread usage in both key exchange and digital signatures in the real world. While virtually all public-key schemes being used today would be broken by a sufficiently large quantum computer, elliptic curves may continue to see usage, but in a different form. In 2011, Jao and De Feo constructed the Supersingular Isogeny Diffie-Hellman (SIDH) scheme [JD11], and since then the field of isogeny-based cryptography has gained much attention. Informally, SIDH relies on the hardness of recovering isogenies, which are special transformations between elliptic curves. Currently, there are no known efficient algorithms for solving this problem, even on a quantum computer.

1.1 Motivation

Virtually all public-key cryptosystems which are being used today rely on the hardness of either the discrete logarithm problem or the integer factorization problem. While these problems seem very different, they can both be seen as special cases of another algebraic problem called the abelian hidden subgroup problem. In 1994 Shor created an algorithm, which effectively solved the abelian hidden subgroup problem in polynomial time [Sho94]. The algorithm only runs on a quantum computer, a machine that works fundamentally differently from classical computers. The result of this is that if a quantum computer large enough to run Shor's algorithm is ever built, virtually all modern public-key cryptosystems would be rendered insecure.

To come up with new public-key cryptosystems which can withstand cryptanalytic attacks by both classical and quantum computers, the United States National Institute of Science and Technology (NIST) started the Post-Quantum Cryptography (PQC) Standardization project [NIS16]. The project was started in 2016, and the stated aim is to arrive at a draft of standards for quantum secure key encapsulation mechanisms (KEMs) and digital signatures, two of the most prominent uses of public-key cryptography today. KEMs are used to send and agree upon a shared

secret-key over an insecure channel such as the internet, which can be further used to protect communication. As of spring 2021, the project is currently in its third round and the remaining candidates in the KEM category are all based on one of three classes of problems: lattice problems, problems from coding theory and recovering isogenies.

To arrive at a draft of new standards, a thorough performance analysis of the candidates is required. Performance should be seen as a weighted sum of many metrics. When assessing the performance with respect to real-world applications such as the internet, at least two factors stand out as important:

- The computational cost of the scheme, which dictates how long it takes the communicating parties to compute the necessary calculations.
- The sizes of public keys, which dictates how much memory or bandwidth is needed to store or transmit the keys.

After two rounds of selection, the only candidate in the KEM category that relies on the hardness of the problem of recovering isogenies is SIKE, a key-encapsulation mechanism based on SIDH [JAC+20]. The performance analysis of SIKE is very two-sided, as it has the shortest key-sizes of all remaining candidates while being the computationally most expensive. This is reminiscent of classical elliptic curve cryptography, which also started out as a computationally expensive counterpart to Diffie-Hellman over finite fields that allowed for much smaller key-sizes [Mil85, Kob87]. Since then, classical elliptic curve cryptography has become much faster, and similarly, much work is currently being done on increasing the performance of isogeny-based cryptography. One approach which has already been successful for classical elliptic curve cryptography, and which seems promising for isogeny-based cryptography, is studying different elliptic curve models (see Section 3.2.2) and their applicability to cryptography.

1.2 Research Objective and Questions

The primary research objective in this thesis is to study how suitable twisted Hessian curves are for the SIDH protocol, and for isogeny-based cryptography in general. A twisted Hessian curve is an elliptic curve model, which has relatively fast formulae for point doubling and tripling [BCKL15]. Additionally, direct formulae for isogenies between twisted Hessian curves have recently been studied [DM19, BF19], which suggests that they may be suitable for isogeny-based cryptography.

More specifically, we wish to use properties of twisted Hessian curves to optimize algorithms for computing chains of low degree isogenies in a cryptographic setting and

investigate in general how well twisted Hessian curves synergise with isogeny-based cryptography, and specifically the SIDH key exchange. To achieve the research objective, we aim to answer the following research questions:

(i) What techniques can be applied to optimize the performance of twisted Hessian curves in the setting of isogeny-based cryptography?

(ii) What are the challenges when implementing an isogeny-based protocol with twisted Hessian curves?

(iii) How can the structure of twisted Hessian curves be taken advantage of in the SIDH setting?

We answer these research questions by providing two proof-of-concept implementations. The first implementation applies twisted Hessian curves to SIDH in a straightforward manner, providing some insight to questions (i) and (ii). We highlight one optimization technique, while we also comment on another technique which has been used to speed up implementations on other elliptic curve models, but which currently does not apply to twisted Hessian curves.

The second implementation is much more speculative but uses the twisted Hessian curves in a more active way, providing some insight to questions (i) and (iii). We discuss in detail the particularly simple structure of the 3-torsion group on twisted Hessian curves (the group of points which has order dividing 3) and show how to take advantage of this in the SIDH setting. This can possibly be used to tweak the SIDH protocol, which may result in an altered scheme that has some appealing properties, in exchange for a higher computational cost. Because of the simple structure of the 3-torsion group, the action of isogenies on subgroups of order 3 is predictable (see Proposition 5.1.1 and Proposition 5.1.2), which suggests a way to do all computation in $\mathbb{F}_{p^2}$, even if the whole 3^n-torsion group is only defined over some higher degree extension field (here n is relative to the security level). The main benefit of this is that the tweaked SIDH scheme is much less restrictive than the original SIDH scheme when it comes to selecting field sizes. This results in the possibility of using much smaller primes (around half the bit-length), at no loss in security level. However, as mentioned, this approach is speculative, and more research is required to make the tweaked protocol work.

Both implementations are available on GitHub[1].

[1] https://github.com/Jonathke/SIDH-with-twisted-Hessian-curves

1.3 Limitations

While this thesis focuses on increasing the performance of a cryptographic protocol, the aim is not to implement a cryptographic library suitable for real-world use. Instead, the thesis focuses on applying a novel approach to isogeny-based cryptography and aims to increase the theoretical performance. Specifically, when measuring performance metrics related to computational speed, we focus on the number of field-operations and asymptotic complexities, as opposed to measuring the number of CPU clock cycles or the number of milliseconds, which are other common measures. The rationale for this is that we aim to focus solely on isogeny-based cryptography, ignoring low-level field-arithmetic optimizations.

1.4 Outline of Thesis

This thesis is sectioned into 7 chapters, including this introduction.

Chapter 2 presents the essential background theory, intended to prepare the reader for Chapter 3 and the rest of the thesis. Starting at basic concepts from algebra, the chapter works its way up to introducing elliptic curves. The second part of this chapter provides an introduction to modern cryptography, with a focus on public-key cryptography.

Chapter 3 presents the topic material for this thesis. The chapter goes more in-depth on the theory of elliptic curves, including relevant theory for the SIDH protocol as well as a discussion of different elliptic curve models, before covering the SIDH protocol itself.

Chapter 4 presents an implementation of the SIDH protocol, using twisted Hessian curves, and discusses all aspects and optimizations of this implementation.

Chapter 5 presents a different view on the SIDH protocol, enabled by twisted Hessian curves. We present some novel results and discuss how to apply this to the SIDH protocol, as well as the missing step to create a tweaked SIDH protocol. This discussion includes a proof-of-concept implementation.

Chapter 6 provides examples based on both the protocol discussed in Chapter 4 and the protocol discussed in Chapter 5. After that, it summarizes and discusses both implementations.

Chapter 7 concludes the thesis, before raising two open problems related to our work.

Chapter 2
Background

This chapter contains general concepts necessary for understanding the rest of the thesis. In Section 2.1 and Section 2.2 we focus on providing the necessary mathematical background to understand Section 2.3, which covers the basic properties of elliptic curves over finite fields, in a way that prepares the reader for Chapter 3. Finally, in Section 2.4, we cover general cryptographic concepts, focusing on asymmetric cryptography, and the need for post-quantum cryptography.

2.1 Fundamentals From Algebra

We start by giving an overview of some of the most important definitions and theorems from algebra, which we will rely on throughout this thesis. The proofs of the theorems are omitted but can be found in any introductory book to the subject, for instance, the one by Bhattacharya, Jain and Nagpaul [BJN94].

2.1.1 Groups

Definition 2.1.1. A non-empty set G together with a mapping $* : G \times G \to G$ (called a binary operation) is called **a group** if the following axioms hold:

(i) (associativity) $(a * b) * c = a * (b * c)$ for all $a, b, c \in G$,

(ii) there exists $e \in G$ such that $e * a = a$ for all $a \in G$,

(iii) for every $a \in G$, there exists $a' \in G$ such that $a * a' = e$.

In the previous definition, we call the element $e \in G$ the identity, and we call $a' \in G$ the inverse of a. We may write a group as $(G, *)$ or just G if the binary operation is clear from context. The two main notations used in groups are the multiplicative $(G, *)$ and the additive $(G, +)$. If we are using multiplicative notation, we often abbreviate $a * b$ to ab. Further, we can abbreviate $a * a * \cdots * a$ (n times) to

a^n. If we are using additive notation, we abbreviate $a + a + \cdots + a$ (m times) to ma. The inverse of a is denoted as a^{-1} in a multiplicative group, and $-a$ in an additive group.

Definition 2.1.2. **An abelian group** is a group $(G, *)$ with the added axiom:

(iv) (commutativity) $ab = ba$ for all $a, b \in G$.

The general rule is that $(G, +)$ is always an abelian group, while $(G, *)$ may or may not be abelian.

Definition 2.1.3. Given a group $(G, *)$, a non-empty subset $H \subseteq G$ is **a subgroup** of G if $(H, *)$ is a group. This is written as $H < G$.

We see that for any $a \in G$, we can define the subset $\langle a \rangle = \{a^k \mid k \in \mathbb{Z}\}$ (called the subgroup *generated* by a). Then $\langle a \rangle$ is a subgroup of G. If there exists $a \in G$ such that $\langle a \rangle = G$, then we say that G is cyclic, and further that a *generates* G.

Definition 2.1.4. The **order** of a group G is equal to the cardinality of G as a set, and is written $|G|$. Similarly, the **order** of an element $a \in G$ is equal to $|\langle a \rangle|$.

Equivalently, we can think of the order of an element $a \in G$ to be the smallest non-zero natural number m such that $a^m = e$, where e is the identity in G. If such an m does not exist a has infinite order. We say that a group G is finite if it has finite order. Because of the following theorem, we can say a lot about the order of elements in a finite group just from the order of the group itself.

Theorem 2.1.5. **(Lagrange)** *Let G be a finite group and $H < G$. Then the order of H divides the order of G.*

Lagrange's theorem implies a lot of results that we are used to. For instance, a famous theorem of Fermat says that for p a prime, $a^{p-1} \equiv 1 \pmod{p}$ for all a such that $p \nmid a$. This is simply because the order of the group of integers modulo p under multiplication is $p - 1$. Then the order of a must divide $p - 1$, and subsequently a^{p-1} must equal 1.

Next, for any group G and subgroup $H < G$, and any element $a \in G$, we can look at the set $aH = \{ah \mid h \in H\}$ or similarly $Hb = \{hb \mid h \in H\}$. These sets are called *cosets* of H.

Definition 2.1.6. Let G be a group, and let $H < G$ be a subgroup. H is called **a normal subgroup** of G (written as $H \lhd G$) if $aH = Ha$ for all $a \in G$.

In general, the above distinction is very important. However, it is not hard to see that for abelian groups, every subgroup is normal. When we have a normal subgroup $N \lhd G$ we can look at the *quotient* group, written as G/N, which is the group of all cosets of N in G. This set becomes a group by applying the binary operation in G to the representants of the cosets, e.g. $aH * bH = (ab)H$. An important example of such a quotient group is the quotient group is $\mathbb{Z}/n\mathbb{Z}$, where $n \in \mathbb{N}$ and $n\mathbb{Z} = \{na \mid a \in \mathbb{Z}\}$ (here we consider the group $\mathbb{Z}$ under addition). This is the group of integers quotiened out by the multiples of n, which form a normal subgroup. After defining isomorphisms, we will see that $\mathbb{Z}/n\mathbb{Z}$ is isomorphic to the group of elements modulo n under addition, commonly written as $\mathbb{Z}_n$.

Definition 2.1.7. Let $(G, *_G)$ and $(H, *_H)$ be groups. A mapping $\phi : G \to H$ is **a group homomorphism** if $\phi(a *_G b) = \phi(a) *_H \phi(b)$ for all $a, b \in G$.

Note that the binary operation between a and b is the operation in G, while the binary operation between $\phi(a)$ and $\phi(b)$ is the operation in H. Related to a homomorphism, we have the following two definitions

Definition 2.1.8. Let $\phi : G \to H$ be a homomorphism between groups G and H. We define the **kernel** of ϕ to be the set $\ker \phi = \{a \in G \mid \phi(a) = e_H\}$, where e_H is the identity in H. Further, we define the **image** of ϕ to be the set $\operatorname{Im} \phi = \{b \in H \mid b = \phi(a) \text{ for some } a \in G\}$.

If ϕ is injective, we call ϕ a group *isomorphism*. We know that ϕ is an isomorphism if and only if $\ker \phi = \{e_G\}$, where e_G is the identity in G. If ϕ is a group isomorphism and $\operatorname{Im} \phi = H$, then we say that G is *isomorhpic* to H (as groups), written as $G \cong H$. If two groups are isomorphic, they have the same group structure, meaning that abstractly we can regard them as the same group.

If $H = G$ (implying that ϕ is a homomorphism from G to itself), it is called an *endomorphism*. If ϕ is an isomorphism and $\operatorname{Im} \phi = G$, it is called an *automorphism*.

Further, we know that $\ker \phi$ is always a normal subgroup of G and that $\operatorname{Im} \phi$ is always a (not necessarily normal) subgroup of H. The next theorem connects this in an important way.

Theorem 2.1.9. (Fundamental theorem of group homomorphisms) *Let $\phi : G \to H$ be a group homomorphism. Then $G/\ker \phi \cong \operatorname{Im} \phi$.*

We give the example we mentioned before: define a group homomorphism $\phi : \mathbb{Z} \to \mathbb{Z}_p$ by $\phi(k) = k \bmod n$. It is easily verified that this is a surjective group homomorphism, and further that $\ker \phi = n\mathbb{Z}$. By the fundamental theorem of group homomorphisms, it follows immediately that $\mathbb{Z}/n\mathbb{Z} \cong \mathbb{Z}_n$.

2.1.2 Rings

Next, we move on to discuss rings. Arguably the single most important example of a ring is $\mathbb{Z}$, the ring of integers. Towards the end of this section, we define polynomial rings, which we will be working a lot within this thesis.

Definition 2.1.10. A non-empty set R together with two binary operations $+$ (called addition) and $\cdot$ (called multiplication) is called **a ring** if the following axioms hold:

(i) $(R, +)$ is an abelian group,

(ii) (Associativity) $(a \cdot b) \cdot c = a \cdot (b \cdot c)$ for all $a, b, c \in R$,

(iii) There exists $1 \in R$ such that $1 \cdot a = a$ for all $a \in R$,

(iv) (Distributive property) $a \cdot (b + c) = a \cdot b + a \cdot c$, and $(a + b) \cdot c = a \cdot c + b \cdot c$ for all $a, b, c \in R$.

Note that 1 is not necessarily the integer 1, but symbolizes the multiplicative identity in R (indeed, in the ring of integers $(\mathbb{Z}, +, \cdot)$, the integer 1 is the multiplicative identity). The identity of $(R, +)$ is often denoted as 0 and called the additive identity. If the operation $\cdot$ is commutative, we say that R is a commutative ring. As for multiplicative groups, $a \cdot b$ is often abbreviated to ab.

We see that the definition of rings does not require $a \in R$ to have a multiplicative inverse. However, if it has a multiplicative inverse, i.e. there exists $a^{-1} \in R$ such that $aa^{-1} = 1$, then a is said to be invertible. The invertible elements of a ring are called units, and the units of a ring form a group under multiplication.

We will see that a lot of definitions from Section 2.1.1 have natural analogues in ring-theory.

Definition 2.1.11. Given a ring $(R, +, \cdot)$, a non-empty subset $S \subseteq R$ is **a subring** of R if $(S, +, \cdot)$ is a ring.

We will see that the following definition is somewhat analogous to normal subgroups (see definition 2.1.6).

Definition 2.1.12. A non-empty subset S of a ring R is called **an ideal** of R if the following two conditions hold:

(i) $(S, +)$ is a subgroup of $(R, +)$,

(ii) for any $r \in R, s \in S$, we have that $rs \in S$ and $sr \in S$.

Every ring R has ideals $A = \{0\}$ and $B = R$. These are called trivial ideals. If I is an ideal of R, and $I \neq A, B$, then I is called a non-trivial ideal. If R is commutative[1], then for any element $r \in R$, we define the set $\langle r \rangle = \{rs \mid s \in R\}$, called the ideal generated by r. Sometimes, we will write $\langle r \rangle$ as rR.

Using a ring R and an ideal I of R, we can construct a quotient ring R/I. We define it slightly differently than we did for groups. We let I define an equivalence relation $\equiv$ on the elements of R as follows. For $a, b \in R, a \equiv b \pmod{I}$ if and only if $a - b \in I$. Then we can define R/I to be the set of equivalence classes. We write elements as $a + I$ for some representative $a \in R$, and turn it into a ring by applying the binary operations from R to the representatives, e.g. $(a+I)+(b+I) = (a+b)+I$, and $(a + I)(b + I) = (ab) + I$. An example of this construction is $\mathbb{Z}/n\mathbb{Z}$, equivalent to the ring of integers modulo n (We already discussed the additive structure of this construction, but after defining ring homomorphisms it should be clear that this is actually a ring isomorphism).

Definition 2.1.13. Let R be a ring and I a non-trivial ideal of R. I is called **a maximal ideal** if for any ideal $B \supseteq I$ of R, either $B = I$ or $B = R$.

Maximal ideals will become very useful when constructing fields (see Section 2.1.3). The following definition will be used when defining elliptic curves.

Definition 2.1.14. Let R be a commutative ring and I an ideal of R, with $I \neq R$. I is called **a prime ideal** if $ab \in I$ implies that either $a \in I$ or $b \in I$ for $a, b \in R$.

Note that all maximal ideals are prime ideals, but the converse is in general not true[2]. Next, we move on to define homomorphisms in the setting of rings.

Definition 2.1.15. Let $(R, +_R, \cdot_R)$ and $(S, +_S, \cdot_S)$ be rings. A mapping $\phi : R \to S$ is **a ring homomorphism** if $\phi(a +_R b) = \phi(a) +_S \phi(b)$, and $\phi(a \cdot_R b) = \phi(a) \cdot_S \phi(b)$ for all $a, b \in R$.

Again, it is important to note the differences in binary operations on the left and right-hand side of each equation. The results from group theory still hold. We know that $\ker \phi$ is always an ideal of R, and $\operatorname{Im} \phi$ is always a subring of S. Further,

[1] This requirement is just to simplify things for us since most rings we will be talking about will be commutative. Elements in non-commutative rings also generate so-called left and/or right ideals, but in the commutative setting, these coincide.

[2] For a large class of rings called principal ideal domains, all non-trivial prime ideals are also maximal ideals.

we call ϕ a ring isomorphism if it is injective (which again happens if and only if $\ker \phi = \{0_R\}$, where 0_R is the additive identity in R). If ϕ is an isomorphism and $\operatorname{Im} \phi = S$, we say that R and S are isomorphic (as rings). If R and S are isomorphic as rings, we can regard R and S as the same ring in an abstract sense. We write this as $R \cong S$.

If ϕ is a ring homomorphism from R to itself, it is called a ring endomorphism. If ϕ is an isomorphism and $\operatorname{Im} \phi = R$, it is called a ring automorphism.

We have now defined both group and ring morphisms of different kinds. Often, we will just write homomorphism (or some other kind of morphism), if it is clear from the context whether we are talking about a group homomorphism or ring homomorphism.

Next, we restate Theorem 2.1.9 as a theorem of ring homomorphisms.

Theorem 2.1.16. (Fundamental theorem of ring homomorphisms) *Let $\phi : R \to S$ be a ring homomorphism. Then $R/\ker \phi \cong \operatorname{Im} \phi$.*

Finally, as mentioned, we define polynomial rings, which are generic constructions that we will be working with a lot.

Definition 2.1.17. Let R be a commutative ring. The **polynomial ring** with coefficients in R is a ring, written as $R[x]$, where x is called an *indeterminate*. An element of $R[x]$ is of the form $a_0 + a_1 x + \cdots + a_n x^n, a_i \in R$, for some $n \in \mathbb{N}$. Addition and multiplication works as we are used to with polynomials.

We can see that $R[x]$ contains R in the sense that there exists an isomorphism $\varphi : R \to R[x]$ defined as $\varphi(r) = r$. Keeping this isomorphism in mind, we can say that the units of $R[x]$ are simply the units of R (although really we mean that the units of $R[x]$ are on the form $\varphi(r)$ where r is a unit of R).

Note that since R is any commutative ring, and $R[x]$ is itself a commutative ring, we can construct polynomial rings recursively, e.g. $R[x][y][z]$. These rings are typically specified as *multivariate* polynomial rings and written as $R[x, y, z]$, where elements are rewritten as finite sums of monomials in indeterminates x, y, z.

2.1.3 Fields

Next, we discuss fields, which are a particularly nice type of commutative rings, where all non-zero elements are units.

Definition 2.1.18. Let R be a commutative ring. If the non-zero elements of R (i.e. $R \setminus \{0\}$) form a group under multiplication, then R is called **a field**.

The rational numbers $\mathbb{Q}$, the real numbers $\mathbb{R}$ and the complex numbers $\mathbb{C}$ are all common examples of *infinite* fields. Throughout the thesis be working mostly with *finite* fields, which we discuss a lot in this section.

The following definition applies to all rings and not just fields. However, we will mainly be talking about the *characteristic* of fields, so therefore we state it in this section.

Definition 2.1.19. Let R be a ring. If there exists $n \in \mathbb{N} \setminus \{0\}$ such that $na = 0$ for all $a \in R$ then the smallest such n is called the **characteristic** of R, denoted as $\text{char}(R)$. If no such n exists, then we say that R has characteristic 0.

Of course, all finite rings have non-zero characteristic. Infinite rings however may or may not have characteristic equal to zero. As we will mostly be working with finite fields, the following theorem is important.

Theorem 2.1.20. *Let K be a field. Then either $\text{char}(K) = 0$ or $\text{char}(K) = p$, where p is a prime number.*

The result of the previous theorem and the preceding discussion is that all finite fields have characteristic p, for some prime number p. To construct our finite fields, we will rely on the following theorem.

Theorem 2.1.21. *Let R be a commutative ring, and let $I \subseteq R$ be an ideal. Then the quotient ring R/I is a field if and only if I is maximal.*

The basic example of a finite field is the ring of integers modulo p, where p is a prime number. To see that this is in fact a field, we recall that as discussed, the ring of integers modulo p can be constructed as $\mathbb{Z}/p\mathbb{Z}$. Further, $p\mathbb{Z}$ is a maximal ideal of $\mathbb{Z}$ if and only if p is a prime number. Theorem 2.1.21 then states that the ring of integers modulo p is a field if and only if p is a prime number.

After defining *field extensions*, we will see that the following theorem shows that the field $\mathbb{Z}_p$ and its extensions are in an abstract sense the only finite fields.

Theorem 2.1.22. *Let K be a finite field with $\text{char}(K) = p$. Then K has p^n elements, for some positive integer n. Further, all other fields with p^n elements are isomorphic to K.*

Because of the previous theorem, we will typically write all fields with $q = p^n$ elements as $\mathbb{F}_q$. The rest of this section will have two main goals. To define the *algebraic closure* of a field, and to show how we can construct $\mathbb{F}_q$ from $\mathbb{F}_p$. To do

this, we will rely a lot on the polynomial ring over a field K, denoted as $K[x]$ (see definition 2.1.17). Therefore, we will now state some basic properties of $K[x]$. An element $f \in K[x]$ is called irreducible if $f = gh$ for elements $g, h \in K[x]$ implies that either g or h are units. The ideal generated by f is maximal if and only if f is irreducible. For elements $f \in K[x]$, $f = a_0 + a_1x + a_2\alpha^2 + \cdots + a_nx^n$, we define the degree of f to be n, where n is the highest power of x appearing in f (unless $f = 0$, then we say that f has degree -1). Further we can *evaluate* f in an element α as the sum $f(\alpha) = a_0 + a_1\alpha + a_2\alpha^2 + \cdots + a_n\alpha^n$.

Finally, when working with polynomial rings, we often prefer to use *monic* polynomials, i.e elements of the form $f = a_0 + a_1x + a_2x^2 + \cdots + a_{n-1}x^{n-1} + x^n$. If we are working in a ring of polynomials over a field, then every element of degree $\geqslant 1$ can be transformed to a monic element by multiplying with the inverse of a_n.

Definition 2.1.23. Let K and E be fields. If K is a subring of E, then E is called an extension field of K, or simply an extension of K.

Subrings of fields that are fields themselves are typically called subfields. An example is $\mathbb{Q} \subseteq \mathbb{R} \subseteq \mathbb{C}$, where $\mathbb{Q}$ is a subfield of $\mathbb{R}$, which in turn is a subfield of $\mathbb{C}$.

Definition 2.1.24. Let E be an extension of K. An element $\alpha \in E$ is said to be **algebraic** over K if there exists $f \in K[x]$, $f \neq 0$ such that $f(\alpha) = 0$. If all elements of E are algebraic over K, then E is said to be an **algebraic extension** of K.

If $f(\alpha) = 0$, we call α a root of f. Every element $\beta \in K$ is algebraic over K, since it is a root of $f = x - \beta \in K[x]$. Examples of algebraic extensions are $\mathbb{R} \subseteq \mathbb{C}$. Note that $\mathbb{Q} \subseteq \mathbb{R}$ is not an algebraic extension, since no transcendental number (e.g. $\pi \in \mathbb{R}$) is algebraic.

Definition 2.1.25. We say that a field K is **algebraically closed** if for any algebraic extension E of K, we have $E = K$, or equivalently that any irreducible polynomial in $K[x]$ is linear (i.e. has degree 1).

Clearly, if all (monic) irreducible element of $K[X]$ are of the form $x - \alpha$, then every element that is algebraic over K is also in K. This is again equivalent to saying that any element in $K[X]$ has all its roots in K, as it factors into linear factors. An example of an algebraically closed field is $\mathbb{C}$ (by the fundamental theorem of algebra). Notice that a finite field $\mathbb{F}_q$ is never algebraically closed, as the polynomial $f = 1 + \prod_{\alpha \in \mathbb{F}_q} (x - \alpha) \in \mathbb{F}_q[x]$ has no roots in $\mathbb{F}_q$ (since for all $\alpha \in \mathbb{F}_q, f(\alpha) = 1$).

Definition 2.1.26. The **algebraic closure** of a field K is an algebraic extension of K that is algebraically closed. We will denote the algebraic closure of K as $\bar{K}$.

The reason that we can write and talk about *the* algebraic closure of K is that for any field K, an algebraic closure exists. Further, all algebraic closures of K are isomorphic. For instance, the algebraic closure of $\mathbb{R}$ is $\bar{\mathbb{R}} \cong \mathbb{C}$, since it is an algebraic extension that is algebraically closed. Notice again that the closure of $\mathbb{Q}$ is not $\mathbb{C}$, since $\mathbb{Q} \subseteq \mathbb{C}$ is not an algebraic extension. Throughout the thesis, we will mainly be using the algebraic closure of a finite field $\bar{\mathbb{F}}_q$ when dealing with elliptic curves over $\mathbb{F}_q$ (see Section 2.3). Keep in mind that as discussed, finite fields are never algebraically closed, hence $\bar{\mathbb{F}}_q$ is itself not a finite field.

We move on to describe how to construct $\mathbb{F}_{p^n}$ from $\mathbb{F}_p$. To do this, we first need to recall the definition of vector spaces and translate them to finite fields. A vector space V over K is simply an additive group, which accepts *scalar multiplication*, meaning an operation $* : K \times V \to V$. Using this definition, it is easy to see that any extension field E over K can be turned into a vector space over K by replacing the multiplicative structure of E with scalar multiplication from K.

Definition 2.1.27. Let E be an extension of a field K. Then the **degree** of E over K (written $[E : K]$) is the dimension of E as a vector space over K.

If $\mathbb{F}_q$ is an extension of a finite field $\mathbb{F}_p$, then we have that the order of $\mathbb{F}_q$ must be the order of $\mathbb{F}_p$ raised to its dimension over $\mathbb{F}_p$ as a vector space, showing that $q = p^n$. To construct a field extension of a given degree, we use the following theorem.

Theorem 2.1.28. *Let E be an extension of a field K. Let $f \in K[X]$ be an irreducible polynomial of degree n, and let $\alpha \in E$ be a root of f. Then $K(\alpha)$ (meaning the subfield of E generated by K and α) is a field extension of K, with $[K(\alpha) : K] = n$. Further, any element of $K(\alpha)$ can be written uniquely as $c_0 + c_1\alpha + \cdots + c_{n-1}\alpha^{n-1}$ for elements $c_i \in K$.*

The last part of the previous theorem says that the set $\{1, \alpha, \alpha^2, \ldots, \alpha^{n-1}\}$ forms a basis of $K(\alpha)$ over K. To construct a field extension of K of degree n, we can therefore start by finding an irreducible $f \in K[X]$ of degree n, and look at the homomorphism $\phi : K[X] \to K(\alpha)$ defined by $\phi(a_0 + a_1x + \cdots + a_mx^m) \to a_0 + a_1\alpha + \cdots + a_m\alpha^m$. Clearly, $\operatorname{Im}\phi = K(\alpha)$, and further $\ker\phi = \langle f \rangle$. Therefore, from Theorem 2.1.16, we see that $K[X]/\langle f \rangle \cong K(\alpha)$, and using Theorem 2.1.21, $K(\alpha)$ is a field because f is irreducible (hence generates a maximal ideal).

It is not guaranteed that there exist irreducible $f \in K[X]$ of degree n for a field K. However, if K is finite, then such an element always exists. Hence, as we will see, when we want to construct a finite field $\mathbb{F}_{p^n}$, we simply start with $\mathbb{F}_p \cong \mathbb{Z}/p\mathbb{Z}$, find an irreducible $f \in \mathbb{F}_p[X]$ of degree n, and construct $\mathbb{F}_{p^n} \cong \mathbb{F}_p[X]/\langle f \rangle$.

2.2 Projective Geometry

In this section, we introduce projective space, which we will use in both the discussion and implementation of elliptic curves. This section is self-contained and discusses informal ideas to give the reader an intuition of the necessary definitions. For more information and a more rigorous approach, we refer the reader to the literature, for instance, the introductory book to computational algebraic geometry by Cox, Little and O'Shea [CLO97].

For the remainder of this section, let K be a field.

2.2.1 Affine Space

Definition 2.2.1. The **affine *n*-space** over K is defined as all n-tuples of the form $\mathbb{A}^n = \{(a_1, \ldots, a_n) \mid a_i \in \bar{K}, 1 \leqslant i \leqslant n\}$.

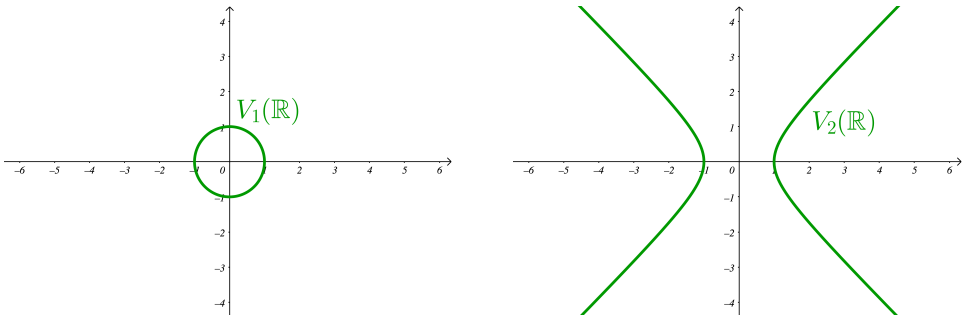
$\mathbb{A}^n$ is in a sense a set of all n-dimensional coordinates over $\bar{K}$. Note that the definition allows the coordinates of $P \in \mathbb{A}^n$ to be in $\bar{K}$. Therefore, we will often restrict ourselves to the following subset.

Definition 2.2.2. The set of ***K*-rational points** of $\mathbb{A}^n$ is the set of all n-tuples of the form $\mathbb{A}^n(K) = \{(a_1, \ldots, a_n) \mid a_i \in K, 1 \leqslant i \leqslant n\}$.

When we talk about an object in $\mathbb{A}^n$, we usually mean a set of points in $\mathbb{A}^n$, i.e. a subset of $\mathbb{A}^n$. We call these types of sets *affine sets*. We will be looking at affine sets defined by a polynomial $f \in \bar{K}[x_1, \ldots, x_n]$. Such a set consists of the points $P \in \mathbb{A}^n$ that are zeroes of f, where P is regarded as a zero of f if $P = (x_{p_1}, \ldots, x_{p_n})$ and $f(x_{p_1}, \ldots, x_{p_n}) = 0$ (we write this as $f(P) = 0$). Equivalently, this definition says that the set are the points whose coordinates are solutions to the equation $f(x_1, \ldots, x_n) = 0$. Similarly, we can define such sets by an equation $g(x_1, \ldots, x_n) = h(x_1, \ldots, x_n)$, where $g, h \in \bar{K}[x_1, \ldots, x_n]$. Note that defining a set by such an equation, is no different than defining it by a single polynomial, as by rewriting this to a new polynomial $f' = g - h$, we can define the same set as the zeroes of f'.

If we have a set $V = \{P \in \mathbb{A}^n \mid f(P) = 0\}$ (i.e. a set as described above), then V contains any solution with coordinates in $\bar{K}$. Therefore, we often restrict ourselves to the subset denoted as $V(K) = \{P \in \mathbb{A}^n(K) \mid f(P) = 0\}$, called the K-rational points of V.

We describe V by its defining polynomial, or by a defining equation. In the latter case, we typically write $V : g(x_1, \ldots, x_n) = h(x_1, \ldots, x_n)$.



(a) $V_1(\mathbb{R})$ forms a circle in $\mathbb{A}^2(\mathbb{R})$ when $V_1 : x^2 + y^2 = 1$.

(b) $V_2(\mathbb{R})$ forms a hyperbola in $\mathbb{A}^2(\mathbb{R})$ when $V_2 : x^2 - y^2 = 1$.

Figure 2.1: Two affine sets over $\mathbb{A}^2(\mathbb{R})$.

Note that the structure of $V(K)$ depends not only on f, but also on the underlying field K. For instance, we can take an $n \in \mathbb{N}, n \geqslant 3$, and look at the affine set $V : x_1^n + x_2^n = 1$ in $\mathbb{A}^2$. If $K = \mathbb{R}$, then $V(K)$ clearly has infinitely many points. However, if $K = \mathbb{Q}$, then Fermat's last theorem, which was famously proven by Andrew Wiles over 300 years after Fermat wrote it down [Wil95], asserts that $V(K) = \{(1, 0), (0, 1)\}$ if n is odd, or $V(K) = \{(\pm 1, 0), (0, \pm 1)\}$ if n is even[3].

Figure 2.1 shows how these affine sets that we have described, form geometric objects. Keep these examples in mind, as in the next section, we will show that these seemingly different affine sets are really just two sides of the same coin.

2.2.2 Projective Space

In this section, we define the projective space $\mathbb{P}^n$. Intuitively, projective spaces are seemingly very different from affine spaces, but we will see that $\mathbb{P}^n$ is in a sense simply an extension of $\mathbb{A}^n$, where we add certain "points at ∞".

We give the following definition.

Definition 2.2.3. The **projective *n*-space** over K is defined as all n-tuples

$$\mathbb{P}^n = \{P \in \mathbb{A}^{n+1} \mid P \neq (0, \ldots, 0)\}/\sim$$

where $\sim$ is the equivalent relation defined by $(a_0, a_1 \ldots, a_n) \sim (b_0, b_1 \ldots, b_n)$ if there exists some $\lambda \in \bar{K}$, such that $b_i = \lambda a_i$ for all i.

[3]Fermat's last theorem states that for any $n \in N, n \geqslant 3$, the equation $x^n + y^n = z^n$ has no non-trivial integer solutions. To see the relation to $V(\mathbb{Q})$, let $P = (\frac{a}{b}, \frac{c}{d}) \in V(\mathbb{Q})$, implying $(\frac{a}{b})^n + (\frac{c}{d})^n = 1$. Assume that P is non-trivial (i.e. a and c are both non-zero). Multiplying on both sides by $(bd)^n$ then gives the non-trivial integer solution to the Fermat equation $(ad)^n + (cb)^n = (bd)^n$.

Informally, we may think of the above definition as saying that $\mathbb{P}^n$ is the set of all *directions* in $\mathbb{A}^{n+1}$. Elements of $\mathbb{P}^n$ are still called points, even if they really are equivalence classes corresponding to lines through the origin of $\mathbb{A}^{n+1}$. It is customary to write a projective point P as $(X_0 : X_1 : \cdots : X_n)$.

As with the affine n-space, we define the following subset of $\mathbb{P}^n$.

Definition 2.2.4. The set of ***K*-rational points** of $\mathbb{P}^n$ is the subset of $\mathbb{P}^n$ denoted as $\mathbb{P}^n(K) = \{(X_0 : X_1 : \cdots : X_n) \in \mathbb{P}^n \mid X_i \in K, 0 \leqslant i \leqslant n\}$.

We will look be looking at subsets of $\mathbb{P}^n$ and $\mathbb{P}^n(K)$, similar to those discussed in Section 2.2.1. These will be called projective sets. However, if we try to simply define $V = \{P \in \mathbb{P}^n \mid f(P) = 0\}$ for any polynomial $f \in \bar{K}[X_0, X_1, \ldots, X_n]$, we quickly run into problems. To illustrate the problem, imagine the projective set $V(\mathbb{R}) = \{(X : Y) \in \mathbb{P}^1(\mathbb{R}) \mid X^2 - Y = 0\}$. Then it seems that the point $P_1 = (1 : 1)$ is in the set, while $P_2 = (2 : 2)$ is clearly not. However, $P_1 \sim P_2$, so they are really the same projective point.

To fix this, we require that f is a *homogenous* polynomial. A homogenous polynomial is a polynomial which, when written as a sum of monomials, has each non-zero term of equal degree. This is enough, since if f is a homogenous polynomial of degree d, then $f(\lambda P) = \lambda^d f(P)$.

To see that $\mathbb{P}^n$ is abstractly equivalent to $\mathbb{A}^n \cup \{\text{points at } \infty\}$ (which was our claim at the start of this subsection), we show that there exists an injective map from $\mathbb{A}^n$ to $\mathbb{P}^n$. We call the complement of this map, the "points at ∞". This map is defined by sending $(x_1, x_2, \ldots, x_n) \in \mathbb{A}^n$ to $(x_1 : x_2 : \cdots : x_n : 1) \in \mathbb{P}^n$. This map is clearly injective, and the complement consists of all points $P \in \mathbb{P}^n$, whose last coordinate is 0. These are the "points at ∞".

However, note that there was no reason for us to select the last coordinate. By selecting coordinate i, we could have created a different map by sending $(x_1, x_2, \ldots, x_n) \in \mathbb{A}^n$ to $(x_1 : \cdots : x_{i-1} : 1 : x_i : \cdots : x_n) \in \mathbb{P}^n$. This map has the same properties, however the "points at ∞" are now the set of all points $P \in \mathbb{P}^n$ whose i'th coordinate is 0. In other words, what set we relate to the "points at ∞" is dependent upon the map we choose.

The inverse of such a map is easily obtained by sending $(x_0 : x_1 : \cdots : x_n) \in \mathbb{P}^n$ to $\left(\frac{x_0}{x_i}, \ldots, \frac{x_{i-1}}{x_i}, \frac{x_{i+1}}{x_i}, \ldots, \frac{x_n}{x_i}\right) \in \mathbb{A}^n$. This map is clearly not defined for any point with $x_i = 0$, corresponding to the "points at ∞". Intuitively, in $\mathbb{P}^2$ we can imagine this as relating every projective point (recall that these projective points are really lines through the origin in $\mathbb{A}^3$) with their intersection with the plane in $\mathbb{A}^3$ defined by $x_i = 1$. It is clear that this map is undefined for lines parallell to the plane.

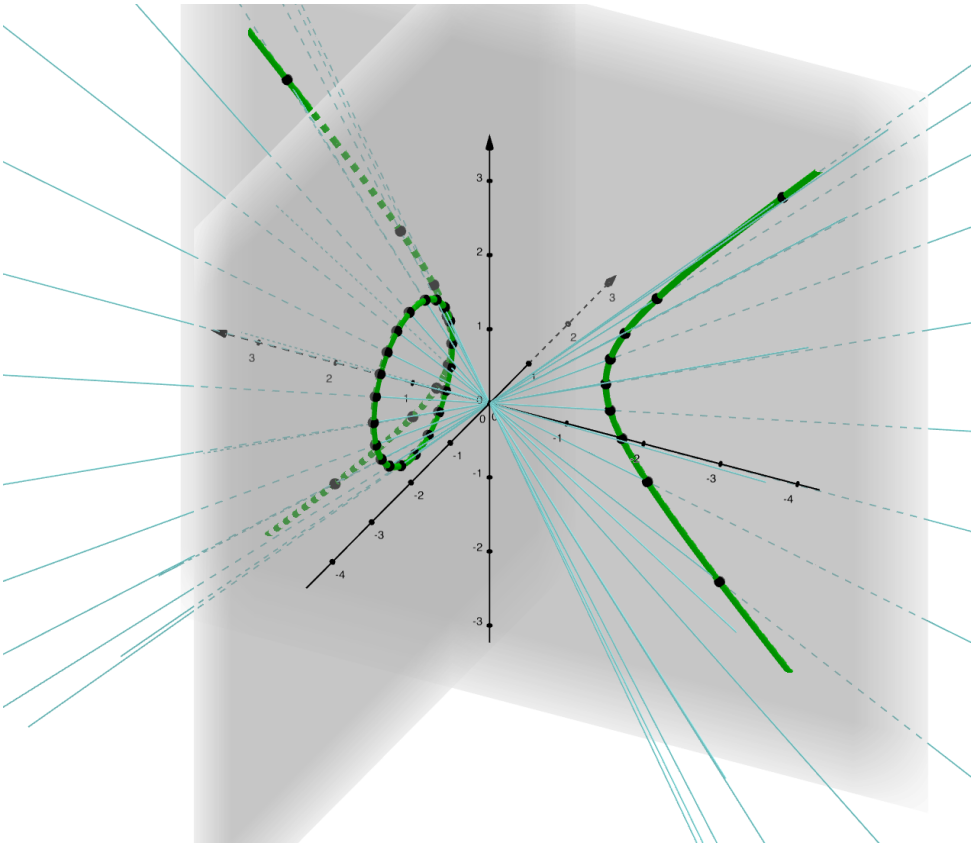


Figure 2.2: Two different affine charts of the projective set $V : X^2 + Y^2 = Z^2$.

We can use this map on a projective set defined as $V = \{P \in \mathbb{P}^n \mid f(P) = 0\}$. This relates the projective set V, to the affine set $V' = \{P \in \mathbb{A}^n \mid \bar{f}(P) = 0\}$, where $\bar{f}$ is the polynomial in n variables, obtained by $\bar{f} = f(X_0, \ldots, X_{i-1}, 1, X_{i+1}, \ldots, X_n)$, and rewriting coefficients. This process of creating $\bar{f}$ from f is called *de-homogenising* f. The opposite of this process is obtained by setting $f = x_i^d \bar{f}\left(\frac{x_0}{x_i}, \frac{x_0}{x_i}, \ldots, \frac{x_n}{x_i}\right)$, where d is the degree of $\bar{f}$. This is called *homogenising* $\bar{f}$.

If V is defined by f, and V' is defined by the de-homogenisation of f, V' is called the *affine chart* of V and written as $V \cap \mathbb{A}^n$. Note that as discussed, many such maps exist, and the affine chart of V will generally vary a lot based on what map we choose.

We end this section with an example which illustrates some of the concepts that we have discussed in this section. Consider the projective set $V : X^2 + Y^2 = Z^2$,

defined over $\mathbb{R}$. The affine chart of V at $Z = 1$ is the affine set $V' = X^2 + Y^2 = 1$. Related to this map, there are no "points at ∞", as for $(X : Y : Z) \in V$, $Z = 0$ implies that $X = Y = 0$ (which is not a valid projective point). Next consider the affine chart at $Y = 1$, which is the affine set $V'' : X^2 + 1 = Z^2$. This map however, gives two "points at ∞", as for $(X : Y : Z) \in V$, $Y = 0$ implies that $X = \pm Z$, so the "points at ∞" for V'' are the projective points $\{(1 : 0 : 1), (-1 : 0 : 1)\}$. This shows that the circle and hyperbola from Figure 2.1 are really just affine charts of the same projective set. This is illustrated in Figure 2.2.

2.3 Elliptic Curves

We now provide a thorough discussion of the basic properties of elliptic curves. Again, we omit proofs, but informally, we try to give some intuition behind the ideas. For a more rigorous approach, we again refer to the literature [Sil09].

Throughout this whole section, unless otherwise stated, K will refer to a field of characteristic not equal to 2 or 3, and $\mathbb{P}^2$ to its 2-dimensional projective space.

Elliptic curves are properly defined as smooth projective varieties of dimension 1 and genus 1, with at least one rational point. To avoid introducing too many mathematical concepts to elaborate on this definition, we instead define elliptic curves as accurately as we can with the concepts we have developed thus far.

Definition 2.3.1. We define an elliptic curve as the following non-singular projective set, with one inflection point $\mathcal{O}$ specified:

$$E = \{(X : Y : Z) \in \mathbb{P}^2 \mid f(X, Y, Z) = 0\}$$

where $f \in \bar{K}[x, y, z]$ is a homogeneous polynomial of degree 3 that generates a prime ideal in $\bar{K}[x, y, z]$, with no multiple root.

We will not worry about proving that f generates a prime ideal when constructing elliptic curves[4]; the reader can assume that any defining polynomial used from this point on satisfies this particular property. If E is defined by an element $f \in \bar{K}[x, y, z]$, that has coefficients in K (meaning that $f \in K[x, y, z]$) and the specified inflection point is in $\mathbb{P}^2(K)$, then we say that E is defined over K and write E as E/K to signify this. This notation should not be confused with quotient structures.

If E contains a singular point, then we say that E is singular, otherwise, we say that E is non-singular. In the geometric sense, this means that E as a curve does

[4]The fact that f has to generate a prime ideal, is to ensure that we can construct its function field (Definition 3.1.1). The function field is a structure related to algebraic varieties. The reader is referred to the literature for more information on general varieties [CLO97].

not contain any cusps or intersections. To ensure that E is non-singular, we must check that the partial derivatives of f never simultaneously vanishes for any point in E, i.e. no $P \in E$ satisfies $\frac{\partial f}{\partial x}(P) = \frac{\partial f}{\partial y}(P) = \frac{\partial f}{\partial z}(P) = 0$. If f contains a multiple root, then that root will be a singular point. When defining curves, we will typically state a sufficient condition on the coefficients of the defining polynomial to satisfy this property (see e.g. Theorem 2.3.2).

The need for a specified inflection point $\mathcal{O}$ will become clear in Section 2.3.1, when we give the set E a group structure. The specified inflection point will serve as the identity.

We now define the standard model of an elliptic curve, called the Weierstrass normal form.

Proposition 2.3.2. Weierstrass normal form: *Let E be a projective set, defined as*

$$E : Y^2Z = X^3 + AXZ^2 + BZ^3$$

with a specified point $\mathcal{O} = (0 : 1 : 0) \in E$. Then E is an elliptic curve if and only if $4A^3 + 27B^2 \neq 0$.

The corresponding polynomial that defines E in the theorem above can be written as $f = Y^2Z - X^3 - AXZ^2 - BZ^3$. This polynomial satisfies all properties of definition 2.3.1. The restriction on A and B is to ensure that E is non-singular.

The following theorem shows why the Weierstrass normal form is the standard model of elliptic curves. See Section 3.1.1 for more information on isomorphisms between elliptic curves.

Theorem 2.3.3. *Any elliptic curve defined over a field K is isomorphic to a curve given by an equation of the form*

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

where $a_1, a_2, a_3, a_4, a_5, a_6 \in K$. Further, if $\mathrm{char}(K) \neq 2, 3$, then the above equation can be further shortened to

$$Y^2Z = X^3 + AXZ^2 + BZ^3 \tag{2.2}$$

where $A, B \in K$.

The above theorem shows that any curve can be written in Weierstrass normal form, as long as the characteristic of K is not 2 or 3. This makes the Weierstrass normal form very well suited for discussing the general theory of elliptic curves,

and we will do so throughout this section, whenever we need a specific curve as an example. Keep in mind that elliptic curves may still be defined by equations different from the Weierstrass normal form; this is precisely the topic of Section 3.2 and a large part of this thesis.

To ease notation, we will often de-homogenise equations, by looking at the affine chart at $Z = 1$, and rewriting the coefficients to $x = X, y = Y$. This is just a notation change, we are still considering E as a projective set, and not as the corresponding affine chart. Specifically, this means that we should not forget potential "points at ∞", which are not defined in the affine chart. For instance, any curve in Weierstrass form has exactly one "point at ∞" (when looking at the affine chart at $Z = 1$), namely the projective point $(0 : 1 : 0)$. When using non-projective notation, we will simply denote this point by $\mathcal{O}$. This will be the specified point for any elliptic curve in Weierstrass normal form.

2.3.1 Arithmetic

We now proceed to give E a group structure. Although it is defined purely algebraically, it is easiest interpreted geometrically. The following is a somewhat non-rigorous description of the addition law.

Let E/K be an elliptic curve, with the specified inflection point $\mathcal{O}$. For a point $\mathcal{O} \in E$ to be an inflection point, the tangent line ℓ of E at $\mathcal{O}$ must intersect E with multiplicity 3 in $\mathcal{O}$. A result of Bezout's theorem, which implies that any line that intersects E, intersects E exactly 3 times [Har77, Corollary 1.7.8], is that this is equivalent to ℓ intersecting E only at $\mathcal{O}$.

Let P and Q be two (not necessarily distinct) points in E. Let L be the line through P and Q (if $P = Q$, L is the tangent line at P). Then, again by Bezout's theorem, L intersects E in a third point, say R. Following, define a line L' through $\mathcal{O}$ and R. Again L' intersects E in a third point. Denote this point by $P + Q$. This construction has the following properties:

(i) $P + \mathcal{O} = P$ for all $P \in E$,

(ii) for all $P \in E$, there exists $-P \in E$ such that $P + (-P) = \mathcal{O}$,

(iii) $P + Q = Q + P$ for all $P, Q \in E$,

(iv) $(P + Q) + R = P + (Q + R)$ for all $P, Q, R \in E$.

All properties except (iv) are immediately clear from the description above. In other words, the $+$ operation turns $(E, +)$ into an abelian group with $\mathcal{O}$ as the identity. We refer the reader to the literature for a proof of (iv) [Sil09, Theorem III.3.4e].

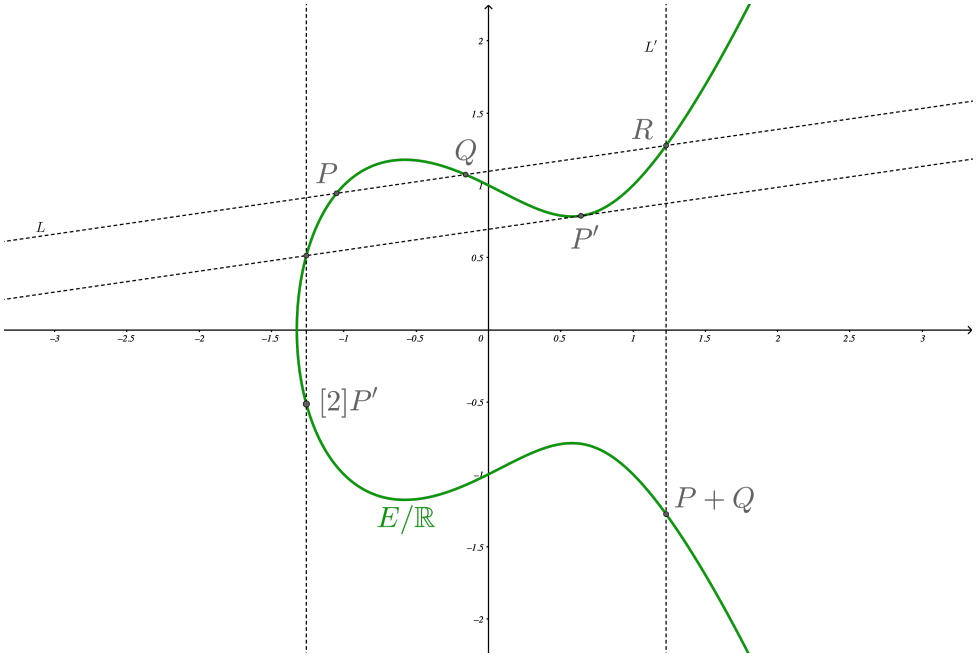


Figure 2.3: An illustration of the + operation on $E : y^2 = x^3 + Ax + B$.

Figure 2.3 shows how the addition law works on $E : y^2 = x^3 + Ax + B$ defined over $\mathbb{R}$, when $E/\mathbb{R}$ is viewed in $\mathbb{A}^2(\mathbb{R})$. Of course, $\mathcal{O}$ is not defined in this affine chart, but we can imagine it being infinitely far up or down in the direction of the y-axis.

Our goal is to work with, and do computations on, elliptic curves defined over finite fields, i.e. $E/\mathbb{F}_q$. Since $E/\mathbb{F}_q$ includes all points of $\mathbb{P}^2(\bar{\mathbb{F}}_q)$ that satisfy the curve equation, we will usually restrict ourselves to the $\mathbb{F}_q$-rational points of E. The K-rational points of a projective set was discussed in Section 2.2.2, but we give a formal definition here for elliptic curves. As we will see, this subset forms a subgroup of E.

Definition 2.3.4. Let E be an elliptic curve defined over K, with specified point $\mathcal{O}$. The group of ***K*-rational points** in E is denoted as

$$E(K) = E \cap \mathbb{P}^2(K) = \{(X : Y : Z) \in E \mid (X, Y, Z) \in K^3\}.$$

Note that since E is defined over K, we have that $\mathcal{O} \in E(K)$.

As $\mathbb{F}_q$ is a finite field, $E(\mathbb{F}_q)$ contains only a finite number of points. Exactly how many points are contained in $E(\mathbb{F}_q)$ is a difficult question, however, we do have the following theorem which bounds $|E(\mathbb{F}_q)|$ from both above and below. Note that $|E(\mathbb{F}_q)|$ is traditionally written as $\#E(\mathbb{F}_q)$.

Theorem 2.3.5. Hasse's theorem: *Let E be an elliptic curve defined over $\mathbb{F}_q$. Then the order of $E(\mathbb{F}_q)$ (i.e. the number of points), denoted $\#E(\mathbb{F}_q)$ is given by*

$$\#E(\mathbb{F}_q) = q + 1 - t$$

where $|t| \leqslant 2\sqrt{q}$.

Further, in order to do computation on $E(\mathbb{F}_q)$, we need to give an algebraic formula for point addition, i.e. a symbolic expression. These expressions are dependent on the nature of the defining equation of E. We will come back to this in Section 3.2. For now, we merely state the explicit addition laws for elliptic curves in Weierstrass normal form for affine coordinates.

Proposition 2.3.6. *Let $E : y^2 = x^3 + Ax + B$ an elliptic curve defined over a field K, and $P, Q \in E(K) \setminus \{\mathcal{O}\}$ with $P = (x_P, y_P)$ and $Q = (x_Q, y_Q)$. Then $-P = (x_P, -y_P)$. Further:*

(i) if $x_P = x_Q$ and $y_P = -y_Q$, then $P + Q = \mathcal{O}$,

(ii) $P + Q = R = (x_R, y_R)$, with $x_R = \lambda^2 - x_P - x_Q$, and $y_R = \lambda(x_P - x_R) - y_P$ where

$$\lambda = \begin{cases} \frac{y_Q - y_P}{x_Q - x_P} & \text{if } x_P \neq x_Q \\ \frac{3x_P^2 + A}{2y_P} & \text{if } x_P = x_Q \end{cases}$$

From the explicit addition laws, it is clear that $E(K)$ is a subgroup of E/K.

2.4 Asymmetric Cryptography

The starting point of cryptography is centered around the following scenario: Two honest parties, Alice and Bob, wish to communicate privately over an insecure channel, where an eavesdropper, Eve, can intercept and read any message. To achieve this, Alice and Bob must scramble (*encrypt*) their messages before sending them, in such a way that only the other party can unscramble (*decrypt*) the message to make sense of it.

2.4.1 Symmetric and Asymmetric Cryptography

The main difference between asymmetric cryptography (also called public-key cryptography) and symmetric cryptography is the presence of a shared secret between the communicating parties. In symmetric cryptography, Alice and Bob rely on their knowledge of some shared secret (called a *key*) to encrypt and decrypt their messages,

and the goal is to ensure that decrypting an encrypted message without the key is impossible, or at least *computationally infeasible*.

We formalize these concepts to define a generic symmetric cryptosystem

Definition 2.4.1. A symmetric cryptosystem consists of the following tuple $(\mathcal{K}, \mathcal{P}, \mathcal{C}, \mathcal{E}, \mathcal{D})$, where

- $\mathcal{K}$ is the key-space, i.e. the set of all possible keys
- $\mathcal{M}$ is the message-space, i.e. the set of all possible messages
- $\mathcal{C}$ is the ciphtertext-space, i.e. the set of all possible ciphertexts
- $\mathcal{E} : \mathcal{M} \times \mathcal{K} \to \mathcal{C}$ is the encryption algorithm, which takes in a key $k \in \mathcal{K}$ and a message $m \in \mathcal{M}$ and outputs a ciphertext $c \in \mathcal{C}$
- $\mathcal{D} : \mathcal{C} \times \mathcal{K} \to \mathcal{M} \cup \{\perp\}$ is the decryption algorithm, which takes in a key $k \in \mathcal{K}$ and a ciphertext $c \in \mathcal{C}$ and outputs either a message $m \in \mathcal{M}$, or a symbol $\perp$, indicating that decryption failed.

Further, the symmetric cryptosystem must satisfy the requirement that $\mathcal{D}(\mathcal{E}(m, k), k) = m$, i.e. that an encryption of $m \in \mathcal{M}$ using a key $k \in \mathcal{K}$ decrypts to itself when using k.

To encrypt a message $m \in \mathcal{M}$, Alice and Bob must first agree on a secret key $k \in \mathcal{K}$. Next, Alice calculates $c = \mathcal{E}(m, k)$, and sends c to Bob. Bob finds $m' = \mathcal{D}(c, k)$, and since they agreed on the same key, $m = m'$.

Of course, as long as $\mathcal{K}$ is a finite set, it cannot be impossible for Eve to recover m from c, as she can simply try to decrypt with all possible keys in $\mathcal{K}$. However, if $\mathcal{K}$ is large, this strategy becomes *computationally infeasible*, i.e. it would simply take too much time for Eve to recover m.

Next, we move on to consider public-key encryption. In many real-world scenarios, the requirement that Alice and Bob must have some shared secret key k is simply not realistic, e.g. if they have never met before. Further, it scales very poorly, as one needs to keep a separate secret key for each person one wishes to communicate privately with.

Therefore, we would like to solve Alice' and Bob's everlasting communication problems, without requiring them to agree on a secret beforehand. The solution to this turns out to be public-key encryption, where Alice instead generates a public key, which can be freely distributed to anyone wishing to send her a message, and a

corresponding private key, which she keeps secret, allowing her to decrypt messages encrypted with her public key.

We formalize this to create a generic public-key encryption scheme.

Definition 2.4.2. A public-key encryption scheme is a tuple of three algorithms $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ where

- $\mathcal{K}$ is the key-generation algorithm, which takes only a security-parameter λ as input, and outputs a public key pk and a secret key sk. Associated with the keys, are a message-space $\mathcal{M}$, and a ciphertext-space $\mathcal{C}$
- $\mathcal{E}$ is the encryption algorithm, which takes in a message $m \in \mathcal{M}$ and a public key pk, and returns a ciphertext $c \in \mathcal{C}$
- $\mathcal{D}$ is the decryption algorithm, which takes in a message c and a private key sk, and returns either a message $m \in \mathcal{M}$, or a symbol $\perp$, indicating that decryption failed.

Further, given any output of the key-generation algorithm $K(\lambda) = (pk, sk)$, the public-key encryption scheme must satisfy satisfy $\mathcal{D}(\mathcal{E}(m, pk), sk) = m$ for all $m \in \mathcal{M}$

Lastly, we should comment that we have not yet formulated any security requirements for either of our generic schemes. At a bare minimum, decrypting without the secret key, and recovering the secret key from knowledge of the public key should be computationally infeasible. However, there exist many more formal security requirements in modern cryptography. Proving cryptosystems secure under these formal security requirements is indeed a large part of modern cryptography, but not the focus of this thesis.

2.4.2 Diffie-Hellman Key Exchange

The first public-key cryptosystem introduced by Diffie and Hellman in 1976 [DH76] was in fact not a public-key encryption scheme, but a key exchange algorithm. The idea was that the communicating parties would use the key exchange algorithm to derive a shared secret, which could then be used to encrypt communication using a symmetric cryptosystem. Still, towards the end of this section, we briefly mention how to turn the Diffie-Hellman key exchange algorithm into a public-key encryption scheme, called the ElGamal encryption scheme.

We formulate the Diffie-Hellman key exchange in a general group-theoretic setting. The public parameters are a cyclic group G generated by g, i.e. $G = \langle g \rangle$, where $|G| = q$. Now, Alice and Bob each pick a secret exponent, a and b respectively, such

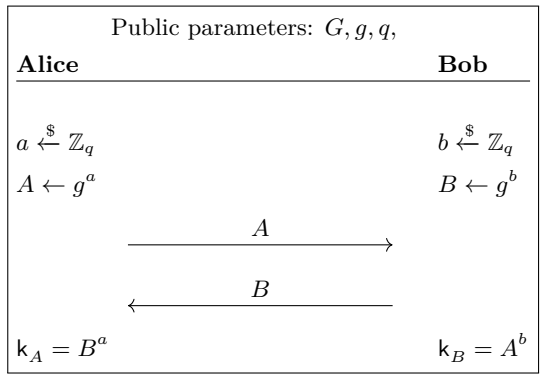


Figure 2.4: The Diffie-Hellman key exchange

that $a, b \in \mathbb{Z}_q$, and compute $A = g^a$ and $B = g^b$. Next, they exchange A and B, and apply their secret exponent to each others value, i.e. Alice calculates B^a and Bob calculates A^b. This is their shared secret, as $B^a = (g^b)^a = g^{ba} = g^{ab} = (g^a)^b = A^b$. This is summarized in Figure 2.4.

Informally, to analyze the security of this scheme, we look at what information Eve has available. Eve has access to g, $A = g^a$ and $B = g^b$, and is trying to find g^{ab}. Trivially, if Eve can reliably recover x from $X(= g^x)$ and g, then she breaks the scheme. This problem of finding x from $X = g^x$ is called the discrete logarithm problem, and because the security of Diffie-Hellman relies on it being hard, we say that the Diffie-Hellman key exchange is based on the discrete logarithm problem. Note that if one were to break the Diffie-Hellman key exchange, it is still an open problem whether this leads to breaking the discrete logarithm problem. In other words, breaking the Diffie-Hellman key exchange might be possible even if the discrete logarithm problem is hard.

Originally, the Diffie-Hellman key exchange used the group $(\mathbb{F}_p)^*$, i.e. the group of units in $\mathbb{F}_p$, but because of the existence of an algorithm which runs in subexponential time called the index calculus algorithm [Adl79], p is required to be rather large for the Diffie-Hellman key exchange to be considered secure when using $G = (\mathbb{F}_p)^*$. Because of this, modern Diffie-Hellman uses a cyclic group generated by an $\mathbb{F}_p$-rational point P on an elliptic curve, i.e. $G = \langle P \rangle < E(\mathbb{F}_p)$ for some $P \in E(\mathbb{F}_p)$. This was first suggested, independently, by Miller [Mil85] and Koblitz [Kob87].

Finally, we turn the Diffie-Hellman key exchange into a public-key encryption scheme $(\mathcal{K}, \mathcal{E}, \mathcal{D})$, to obtain the ElGamal encryption scheme [ElG85]. This is done as follows:

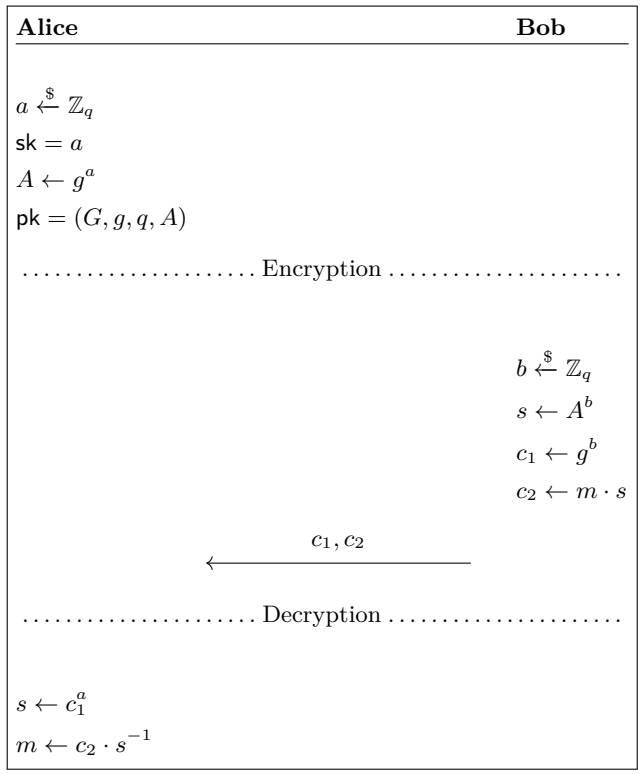


Figure 2.5: The ElGamal encryption scheme

– Alice runs $\mathcal{K}$ which chooses a cyclic group G of order q generated by g. Then it outputs a secret key $\mathsf{sk} = a$ corresponding to Alice's secret in Diffie-Hellman, and computes the group element $A = g^a$, corresponding to the element which Alice would have sent to Bob. $\mathcal{K}$ outputs the public key $\mathsf{pk} = (G, g, q, A)$

– Bob runs $\mathcal{E}$, which take in a message $m \in G$, and the public key pk. It selects some random $b \in \{1, 2, \ldots, n-1\}$, computes a shared secret $s = A^b = g^{ab}$, in addition to $c_1 = B = g^b$. Finally, it calculates $c_2 = m \cdot s$ and outputs $c = (c_1, c_2)$

– To decrypt, Alice runs $\mathcal{D}$ which take in a $c = (c_1, c_2)$, and the private key $\mathsf{sk} = a$. Alice recovers the shared secret by $s = c_1^a = B^a = g^{ba}$, and outputs $m' = c_2 \cdot s^{-1} = m \cdot s \cdot s^{-1} = m$

The public-key encryption scheme $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ is summarized in Figure 2.5.

2.4.3 The RSA Cryptosystem

In 1978 Rivest, Shamir and Adleman introduced the first public-key encryption scheme, later called the RSA cryptosystem [RSA78].

Unlike the Diffie-Hellman key exchange which works for any cyclic group, the RSA cryptosystem is defined for a very specific group, namely $(\mathbb{Z}_n)^*$ (the group of units in $\mathbb{Z}_n$), where $n = pq$ and p and q are prime numbers. It relies on a well known result from elementary number theory, which implies that the order of $(\mathbb{Z}_n)^*$ is $\phi(n)$, where ϕ is the Euler-phi function. For $n = pq$, p and q distinct prime numbers, we have $\phi(n) = (p-1)(q-1)$. We describe the RSA cryptosystem in our public-key encryption scheme framework:

- Alice runs $\mathcal{K}$ which finds two big prime numbers p and q, and calculates $n = pq$ and $\phi(n) = (p-1)(q-1)$. Next, it selects an $e \in (\mathbb{Z}_{\phi(n)})^*$ (i.e. $1 < e < \phi(n)$, with $\gcd(e, \phi(n)) = 1$), and computes $d \equiv e^{-1} \pmod{\phi(n)}$. $\mathcal{K}$ outputs $\mathsf{sk} = d$ and $\mathsf{pk} = (n, e)$

- Bob runs $\mathcal{E}$, which take in a message $m \in \mathbb{Z}_n$, and the public key pk. It outputs $c \equiv m^e \pmod{n}$

- To decrypt, Alice runs $\mathcal{D}$ which take in a c, and the private key $sk = d$. Alice recovers the message by $s \equiv c^d \equiv (m^e)^d \equiv m^{ed} \equiv m^{k\phi(n)+1} \equiv (m^{\phi(n)})^k m \equiv m \pmod{n}$

The last part of the protocol works because the order of $(\mathbb{Z}_n)^*$ is $\phi(n)$. The encryption scheme is summarized in Figure 2.6.

From the protocol description, it is clear that factorizing n is sufficient to break the encryption scheme. To see this, notice that when the factorization of n is known, $\phi(n)$ can be calculated, which leads to recovering d from e (this is in fact how the key-generation algorithm works to begin with). Factorizing integers is known as the integer factorization problem, and since the RSA cryptosystem relies on it being hard, we say that RSA is based on the integer factorization problem. The opposite direction (whether breaking RSA implies the ability to solve the integer factorization problem) is again an open problem.

RSA and Diffie-Hellman together form the basis of most of modern public-key cryptosystems. Because of this, most modern public-key cryptosystems are based on either the integer factorization problem or the discrete logarithm problem.

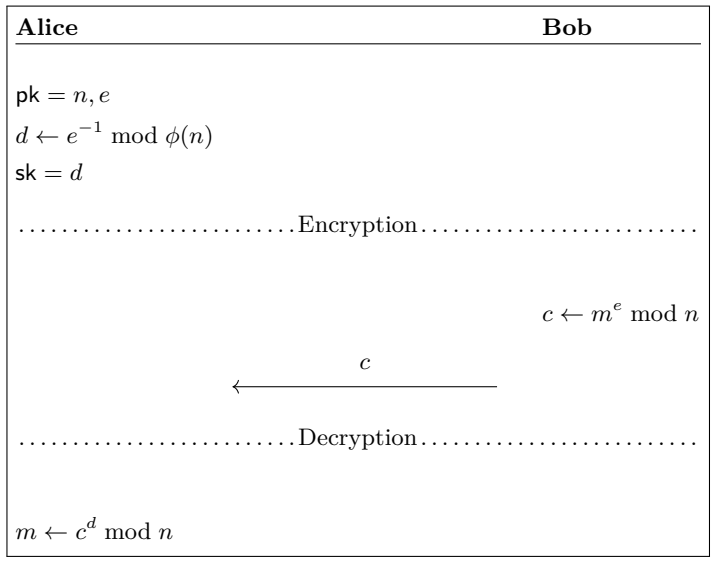


Figure 2.6: The RSA cryptosystem

2.4.4 Shor's Algorithm and The Hidden Subgroup Problem

In 1994 Shor gave a polynomial-time algorithm for solving both the integer factorization problem and the discrete logarithm problem [Sho94]. The algorithm runs on a special type of computer called a quantum computer, which works fundamentally differently from classical computers. As far as we know, no one has yet successfully built a quantum computer, but the implications of Shor's algorithm are, as we have seen in the previous sections, that if a quantum computer is ever built, modern public-key cryptography will be broken.

It turns out that generalizing Shor's algorithm shows that quantum computers can solve the more general *abelian hidden subgroup problem* and that integer factorization and discrete logarithms can be formulated as special cases of this problem [ME98]. In this section, we outline how this can be done. We start by defining the hidden subgroup problem.

Definition 2.4.3. Hidden subgroup problem Given a finitely generated group G, a finite set X and a function $f : G \to X$, where f is constant for all elements in the same cosets of some subgroup $H < G$, and distinct for all elements in different cosets of K (i.e. $a, b \in G$, $f(a) = f(b)$ if and only if $aH = bH$), find a generating set for H.

If the group G in the definition above is abelian, we refer to the problem as

the *abelian* hidden subgroup problem. Next, we outline how to solve the integer factorization problem, given an algorithm that solves the abelian hidden subgroup problem.

To factor N, we rely on a well known result, which states that factoring N is computationally equivalent to calculating $\phi(N)$ [Mil76]. Because of this, we focus our attention to recovering $\phi(N)$. To do this, instantiate the hidden subgroup problem by selecting some $a \in (\mathbb{Z}_n)^*$, and setting $G = (\mathbb{Z}, +)$, $X = \mathbb{Z}_n$ (as a set), and defining $f : (\mathbb{Z}, +) \to \mathbb{Z}_n$ as $f(k) = a^k \bmod n$. Then notice that f is constant on the cosets of $c\mathbb{Z}$, where c is the order of a in $(\mathbb{Z}_n)^*$, so the algorithm that solves the abelian hidden subgroup problem recovers c. From the theorem of Lagrange 2.1.5, we know that $c \mid \phi(n)$, so repeating the proceedure for different $a \in (\mathbb{Z}_n)^*$ eventually recovers $\phi(n)$.

Similarly, we can reduce the discrete logarithm problem to the abelian hidden subgroup problem. Given a cyclic group $K = \langle g \rangle$ of order q, and an element $A \in K$, where $A = g^a$, we are supposed to find a. We instantiate the hidden subgroup problem as $G = ((\mathbb{Z}_q, +) \times (\mathbb{Z}_q, +))$, $X = K$, and define $f : G \to X$ as $f(x, y) = g^x A^{-y}$. Since $A = g^a$, we see that the hidden subgroup is $H = \{(ra, r) \in G \mid r = 0, 1, \ldots, q-1\} = \langle (a, 1) \rangle$, so an algorithm which solves the hidden subgroup problem will recover a.

2.4.5 Key Encapsulation Mechanisms

We end this chapter by briefly describing a class of public-key cryptosystems called key encapsulation mechanisms (KEMs). A KEM is very close to a public-key encryption scheme, but it is specifically designed to transport symmetric key material.

Public-key encryption often has limitations in the message length and is typically much slower than symmetric encryption schemes. Therefore, public-key encryption is often used to send the recipient a secret key which is used to encrypt the message itself using symmetric encryption. Using a KEM instead of plain public-key encryption then eliminates some of the potential security issues which may arise.

Figure 2.7 shows how to turn the RSA cryptosystem into RSA-KEM. The KDF-function is a key-derivation function, which takes an input from some large set, and maps it to a fixed-length output, which can be used as a key. RSA-KEM can also be altered to transport some predefined symmetric key k, where the output of the KDF is not the symmetric key k itself, but a key-encrypting key (KEK) used to encrypt k [RKBT10].

Similar constructions exist for many other public-key encryption schemes.

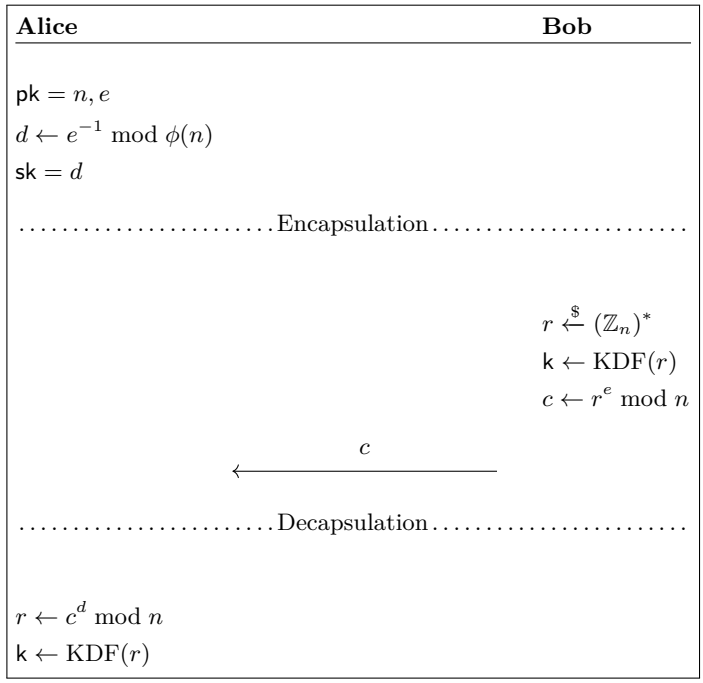


Figure 2.7: RSA-KEM

Chapter 3
Towards Isogeny-Based Cryptography

The goal of this chapter is to cover the specific background material this thesis is based on. Section 3.1 extends on the discussion in Section 2.3, covering isogenies and supersingular curves, two topics necessary for understanding the Supersingular Isogeny Diffie-Hellman protocol. Section 3.2 importantly introduces the main elliptic curve model used in this thesis. Finally, Section 3.3 covers the Supersingular Isogeny Diffie-Hellman protocol.

3.1 More Topics on Elliptic Curves

Elliptic curves have seen many applications within cryptography (with isogeny-based cryptography being one of the newer additions), many of which fall outside of the scope of this thesis. We refer the reader to a few foundational examples from the literature: Miller [Mil85] and Koblitz [Kob87] original papers on elliptic curve cryptography, Lenstra's integer factorization algorithm [LJ87] and Boneh and Franklin's realization of identity-based encryption [BF01].

Proofs of most theorems covered in this section are omitted but can be found in the book by Silverman [Sil09].

3.1.1 Isogenies

In this section, we will be describing special maps between elliptic curves, called isogenies. To do this, we first need the following definitions.

Definition 3.1.1. Let E be an elliptic curve, and let $E' = E \cap \mathbb{A}^2$ be an affine chart of E, with corresponding de-homogenized polynomial f. The **affine coordinate ring of E'**, denoted $K[E']$ is defined as

$$K[E'] = K[x, y]/\langle f \rangle.$$

Further, we can define the **function field of E'**, denoted $K(E')$ as the field of fractions of $K[E']$, i.e.

$$K(E') = \left\{ \frac{f}{g} \;\middle|\; f, g \in K[E'], g \neq 0 \right\}.$$

Recall from Definition 2.3.1, the requirement that f had to generate a prime ideal. Notice that here we are relying on this fact, as field of fractions require an integral domain, and a quotient ring is an integral domain if and only if the the modulus is a prime ideal.

Note also that the definition above is only for some affine chart of E. We extend the definition to $K[E]$ and $K(E)$ by saying that $K[E] = K[E \cap \mathbb{A}^2]$, and similarly $K(E) = K(E \cap \mathbb{A}^2)$, for some affine chart of $\mathbb{A}^2$. Note that in this case, the choice of affine chart does not matter, as all choices lead to isomorphic structures. Although not obvious, by homogenizing elements of $K(E)$, we can create the following equivalent definition of $K(E)$, which may be easier to work with.

Proposition 3.1.2. *Let E be an elliptic curve defined by $h \in K[x, y, z]$. The function field of E can then equivalently be defined as*

$$K(E) = \left\{ \frac{f}{g} \;\middle|\; f, g \in K[x, y, z], g \notin \langle h \rangle, f \text{ and } g \text{ are homogenous}, \deg f = \deg g \right\} \Big/ \sim$$

where $\sim$ is the equaivalence relation on $K(E)$ defined by $\frac{f_1}{g_1} \sim \frac{f_2}{g_2}$ if $f_1 g_2 - f_2 g1 \in \langle h \rangle$.

The fact that we have defined elliptic curves as non-singular curves, makes the following definition easy.

Definition 3.1.3. Let E be an elliptic curve defined over K. A **morphism** is then a map $\phi : E \to \mathbb{P}^n$ defined by

$$\phi(P) \to (f_0(P) : f_1(P) : \cdots : f_n(P))$$

where all $f_i \in \bar{K}(E)$. We write ϕ as $(f_0 : f_1 : \cdots : f_n)$. If all $f_i \in K(E)$, we say that ϕ is defined over K.

Notice that in the above definition, ϕ is well defined regardless of choices of f_i. This is not obvious, but a result of the smoothness of E, and the structure of $\bar{K}(E)$.

If the image of a morphism lies within another curve, we get the following nice theorem.

Theorem 3.1.4. *Let E_1 and E_2 be elliptic curves defined over K. Then any morphism $\phi : E_1 \to E_2$ is either constant or surjective.*

In the above theorem, a constant morphism means a morphism whose image is a single point. From this, we are ready to define isogenies.

Definition 3.1.5. Let E_1 and E_2 be elliptic curves defined over K, with specified points $\mathcal{O}_1$ and $\mathcal{O}_2$ respectively. A morphism $\phi : E_1 \to E_2$ is called **an isogeny** if $\phi(\mathcal{O}_1) = \mathcal{O}_2$.

From Theorem 3.1.4, we see that for any isogeny $\phi : E_1 \to E_2$, we have that either $\operatorname{Im}\phi = \{O_2\}$ or $\operatorname{Im}\phi = E_2$. If $\operatorname{Im}\phi = \{O_2\}$, we call ϕ the zero isogeny.

Next, we will state multiple important theorems regarding isogenies and their structure.

Theorem 3.1.6. *Let $\phi : E_1 \to E_2$ be an isogeny. Then, for all $P, Q \in E_1$, we have*

$$\phi(P + Q) = \phi(P) + \phi(Q).$$

In other words, ϕ induces a group homomorphism between E_1 and E_2.

As isogenies are morphisms, they are rational maps, hence it makes sense to restrict ourselves to K-rational points, i.e to look at $\phi[E_1(K)] = \{\phi(P) \mid P \in E_1(K)\}$ if ϕ is defined over K. Cleary, $\phi[E_1(K)] \subseteq E_2(K)$ is still true. However, in general we no longer have $\phi[E_1(K)] = E_2(K)$, and further, the whole kernel is not guaranteed to lie inside $E_1(K)$.

Since isogenies automatically induce group homomorphisms, it makes sense to look at their kernel. Let $\phi : E_1 \to E_2$ be an isogeny. If ϕ is the zero isogeny, then clearly $\ker\phi = E_1$. On the opposite side, if $\ker\phi = \{O_1\}$, then ϕ is injective (and already surjective by Theorem 3.1.4), and we say that ϕ is an isomorphism, and that E_1 and E_2 are isomorphic. In general, we have the following theorem:

Theorem 3.1.7. *Let $\phi : E_1 \to E_2$ be an non-zero isogeny. Then $\ker\phi$ is a finite subgroup of E_1.*

The previous theorem says something about the kernel, given an isogeny. However, the following theorem turns this around and relates an isogeny to a given kernel. This will be essential in the construction of isogeny-based cryptography.

Theorem 3.1.8. *Given an elliptic curve E and a finite subgroup G, there exists a unique elliptic curve E' (up to isomorphism), and corresponding separable isogeny $\phi : E \to E'$ satisfying $\ker\phi = G$.*

Let ϕ, E, E' and G be as in the previous theorem. To show that an elliptic curve E' is the image of ϕ derived from G, we will write E' as E/G. This shows the group

structure of E'. This should again not be confused with the notation E/K, where K is a field.

In the previous theorem, we used the term separable isogeny. In general, isogenies can be either separable, inseparable and purely inseparable isogenies. However, for our purposes, we will always use isogenies computed from finite subgroups[1], and therefore we will only be dealing with separable isogenies. We refer the interested reader to the literature for more discussion on non-separable isogenies [Sil09, Chapter III §4]. We will in theorems use the word separable isogeny where it is accurate, but as all isogenies used in this thesis will be separable, this should not be the focus of the reader.

Informally, the *degree* of an isogeny is its degree as a rational map. However, in the case of separable isogenies, we have the following proposition, which is very useful in relation to isogeny-based cryptography. Therefore, we give only the proposition and not the exact definition. The reader is welcome to think of the following as a definition.

Proposition 3.1.9. *The* ***degree*** *of a separable isogeny ϕ is equal to the order of its kernel.*

The following subgroups of E are important subgroups for many reasons. We will be relying a lot on them later.

Definition 3.1.10. Let E be an elliptic curve defined over K, and let $m \in \mathbb{N}$. The ***m*-torsion subgroup** of E is the subgroup of E defined as

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

In other words, $E[m]$ are the points in E of order dividing m.

There is an important isogeny, called the *multiplication by m-map* defined as $\phi : E \to E$, where $\phi(P) = [m]P$, which clearly has the m-torsion subgroup as its kernel. If ϕ is the multiplication by m-map, it is usually denoted simply as $[m]$. It can be shown that any isogeny $\psi : E_1 \to E_2$ of degree d (abbreviated as a d-isogeny) has a unique corresponding d-isogeny called the *dual* isogeny $\hat{\psi} : E_2 \to E_1$, such that $\hat{\psi} \circ \psi$ is the multiplication by d-map on E_1. This turns the following definition into an equivalence relation on the set of all elliptic curves over $\bar{K}$.

Definition 3.1.11. Let E_1 and E_2 be elliptic curves defined over $\bar{K}$. If there exists a non-zero isogeny $\phi : E_1 \to E_2$, we say that E_1 and E_2 are **isogenous**.

[1]Theorem 3.1.8 merely states the existence of isogenies corresponding to finite subgroups, but we will in Section 3.2.2 return to constructive theorems on this matter.

If we consider the set of all elliptic curves over a field K, we now actually have two equivalence relations, which partitions this set: curves that are isomorphic, and the coarser partition of curves that are isogenous. We will return to this in Section 3.3.

Finally, we state the following theorem, which shows that the group structure of $E[m]$ is well known.

Theorem 3.1.12. *Let E be an elliptic curve defined over K, and let $m \in \mathbb{N}$, with $m \neq 0$ in K (i.e. if $\mathrm{char}(K) = 0$, then $m \neq 0$, while if $\mathrm{char}(K) = p > 0$, then $p \nmid m$). Then*

$$E[m] \cong \mathbb{Z}/m\mathbb{Z} \times \mathbb{Z}/m\mathbb{Z}.$$

Further, if $\mathrm{char}(K) = p > 0$, then one of the following holds

$$E[p^e] = \{O\}, \text{ for } e = 1, 2, 3, \ldots$$
$$E[p^e] = \mathbb{Z}/p^e\mathbb{Z}, \text{ for } e = 1, 2, 3, \ldots$$

It turns out that in the last part of the previous theorem, $E[p^e] = \mathbb{Z}/p^e\mathbb{Z}$ for almost all curves, and that $E[p^e] = \{O\}$ holds only for *supersingular* curves, which we define in the next section.

3.1.2 Supersingular Curves

Supersingular curves are a certain class of elliptic curves which exist over fields K with $\mathrm{char}(K) = p > 0$. These curves turn out to exhibit a lot of different behaviour than *ordinary* curves. Therefore they can be defined in several different equivalent ways. At the end of the previous section, we already saw one possible definition, based on the structure of the torsion subgroup $E[p^e]$, however, we choose to give another definition, which will be the most relevant in section 3.3, when we discuss some aspects of why supersingular curves are well suited for isogeny-based cryptography.

To give this definition, we must first define the following ring related to an elliptic curve E

Definition 3.1.13. Let E be an elliptic curve defined over K. The **endomorphism ring** of E is defined as

$$\mathrm{End}(E) = \{\phi : E \to E \mid \phi \text{ is an isogeny}\}.$$

Endomorphism rings of abelian groups are common algebraic constructions, which here is extended to elliptic curves as isogenies are the natural analogies of homomorphisms for elliptic curves. To give $\mathrm{End}(E)$ a ring structure, we define

addition of $\phi, \psi \in \text{End}(E)$ by $(\phi + \psi)(P) = \phi(P) + \psi(P)$ and multiplication by composition, i.e. $(\phi \circ \psi)(P) = \phi(\psi(P))$.

Then we simply define supersingular curves as follows:

Definition 3.1.14. Let E be an elliptic curve defined over K. We say that E is **ordinary** if $\text{End}(E)$ is a commutative ring, and that E is **supersingular** if $\text{End}(E)$ is a non-commutative ring.

The above definition hides a lot of information about what we know about the structure of $\text{End}(E)$. The following paragraph briefly mentions this structure using some definitions we have not introduced, however, we will not be using any more than what was mentioned in the definition above.

In general, the structure of $\text{End}(E)$ is restricted to one of three cases, and if we know the characteristic of the field we are working over, it is further restricted to two cases. If $\text{char}(K) = 0$, then $\text{End}(E)$ is usually isomorphic to $\mathbb{Z}$, but it may also be isomorphic to an order in an imaginary quadratic field. In the latter case, E is said to have complex multiplication, which has proven to be a very interesting property. However both of these are commutative, and hence (trivially from our definition) supersingular curves do not exist over fields of characteristic zero. If $\text{char}(K) = p > 0$, $\text{End}(E)$ is never isomorphic to $\mathbb{Z}$, but is instead isomorphic to an order in either an imaginary quadratic field or a quaternion algebra. Of these, curves with $\text{End}(E)$ isomorphic to an order in a quaternion algebra (which is a non-commutative structure) are called supersingular.

As mentioned, the non-commutative nature of $\text{End}(E)$ for supersingular curves is an important feature which some isogeny-based cryptographic schemes benefit from, such as SIDH which we cover in Section 3.3. However, many other properties of supersingular curves are also used in isogeny-based cryptography. These next theorems cover some of these properties. Recall Hasse's theorem (Theorem 2.3.5), which states that $\#E(\mathbb{F}_q) = q + 1 - t$, where $|t| \leqslant 2\sqrt{q}$.

Theorem 3.1.15. *Let E be a supersingular elliptic curve defined over $\mathbb{F}_q$ where $q = p^n$ for some prime $p > 3$, and let $\#E(\mathbb{F}_q) = q + 1 - t$. Then $t = 0 \mod p$.*

From the theorem above, and the theorem of Hasse, we know that if E is defined over $\mathbb{F}_p$ for some prime $p > 3$, then $\#E(\mathbb{F}_p) = p + 1$. This makes it trivial to find the number of points on a supersingular elliptic curve over prime fields, even though it is difficult for ordinary curves[2]. Similarly, we can easily find it over extension fields.

[2]Although even for ordinary curves, there exist polynomial-time algorithms which find the number of points, first discovered by Schoof in 1995 [Sch95].

Theorem 3.1.16. *Let E_1 be a supersingular elliptic curve, and let $\phi : E_1 \to E_2$ be a non-zero isogeny. Then E_2 is supersingular.*

The previous theorem, combined with the fact that all isogenies have a dual, guarantees that an isogeny-class either contains no supersingular elliptic curves or only supersingular elliptic curves. In fact, the next theorem shows that for any field $\mathbb{F}_q$, there exists only one such isogeny-class, i.e. all supersingular elliptic curves belong to the same isogeny-class.

Theorem 3.1.17. *Let $\mathbb{F}_q$ be a finite field of order q. Then all supersingular elliptic curves over $\overline{\mathbb{F}}_q$ are isogenous.*

Finally, we will state two theorems which together state that the number of supersingular curves over $\overline{\mathbb{F}}_p$ is a finite number and that this number is well known. To do so, we must first introduce the j-invariant. The j-invariant will also become relevant in Section 3.3, where we introduce SIDH.

Associated with any elliptic curve E/K is an element in K called the j-invariant[3]. The j-invariant depends on the equation that E is given by. As an example, we give the following proposition for curves in Weierstrass normal form.

Proposition 3.1.18. *Let $E : y^2 = x^3 + Ax + B$ an elliptic curve. Then the j-invariant of E is given by*

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

One of the main reasons we are interested in the j-invariant of an elliptic curve is the following theorem.

Theorem 3.1.19. *Let E_1 and E_2 be elliptic curves over K. Then $j(E_1) = j(E_2)$ if and only if $E_1 \cong E_2$.*

Note that the result above does not state that $E_1(K) \cong E_2(K)$, even if E_1 and E_2 are defined over K. The isomorphism between E_1 and E_2 is considered over $\bar{K}$, i.e. $E_1(\bar{K}) \cong E_2(\bar{K})$.

Now we are ready to state the theorems we wanted. The first theorem states both that the total number of supersingular curves (up to isomorphism) over a field $\overline{\mathbb{F}}_p$ is finite, and further, that they all exist in $\mathbb{F}_{p^2}$.

[3]The theory behind the j-invariant is related to complex elliptic functions and modular forms, topics that are far beyond the material we intend to cover in this thesis. The reader is referred to the literature for details [Kob93, Chapter III]

Theorem 3.1.20. *Let E be a supersingular elliptic curve defined over $\mathbb{F}_q$ for some $q = p^n$. Then $j(E) \in \mathbb{F}_{p^2}$.*

Note that as $\mathbb{F}_p \subseteq \mathbb{F}_{p^2}$, $j(E)$ might also lie within $\mathbb{F}_p$.

Since isomorphic curves have the same j-invariant, a j-invariant is either related to supersingular or ordinary curves. Because of this, if a j-invariant is related to a supersingular curve, we call it a supersingular j-invariant. The number of different supersingular j-invariants in field $\overline{\mathbb{F}}_p$ is equivalent to the number of different isomorphism-classes of supersingular curves over $\overline{\mathbb{F}}_p$, and is given by the following theorem.

Theorem 3.1.21. *Let n be the number of distinct supersingular j-invariants in $\overline{\mathbb{F}}_p$, where $p \neq 2, 3$. Then*

$$n = \left\lfloor \frac{p}{12} \right\rfloor + \begin{cases} 0 & \text{if } p \equiv 1 \pmod{12} \\ 1 & \text{if } p \equiv 5 \pmod{12} \\ 1 & \text{if } p \equiv 7 \pmod{12} \\ 2 & \text{if } p \equiv 11 \pmod{12} \end{cases}$$

As we will see in Section 3.3, the shared secret that the communicating parties end up with in SIDH is a secret j-invariant. Therefore, Theorem 3.1.21 can be used to quickly calculate an upper bound on the size of the (shared) key-space.

3.2 Elliptic Curve Models

In this section, we present the elliptic curve model used in this thesis, the twisted Hessian curve, after briefly mentioning various other elliptic curve models to provide context. For this whole section, we will be working over a finite field $\mathbb{F}_q$, with $\text{char}(\mathbb{F}_q) \neq 2, 3$. Any projective set is then understood to be contained in $\mathbb{P}^n$, where $\mathbb{P}^n$ is the n-dimensional projective space over $\mathbb{F}_q$.

3.2.1 Alternatives to Weierstrass Form

Proposition 2.3.2 introduced the Weierstrass normal form of elliptic curves. While the Weierstrass normal form is commonly used as a standard model of elliptic curves because of their generality (recall Theorem 2.3.3), other models of elliptic curves have been studied extensively because of their applications in implementations of various algorithms in elliptic curve cryptography.

One example of this is the Montgomery form of elliptic curves

Proposition 3.2.1. Montgomery form: *Let E be a projective set, defined as*

$$E : BY^2Z = X^3 + AX^2Z + XZ^2 \tag{3.1}$$

with a specified point $\mathcal{O} = (0 : 1 : 0) \in E$. Then E is an elliptic curve if and only if $B\left(A^2 - 4\right) \neq 0$.

The Montgomery form was introduced by Montgomery in 1987 [Mon87], mainly to speed up algorithms relying heavily on point doubling on elliptic curves, such as Lenstra's factoring algorithm [LJ87]. Despite being introduced early in the history of elliptic curve cryptography, it is still commonly used today. It was used in the suit of algorithms proposed by Costello et al. to speed up SIDH [CLN16], and is still used in the NIST PQC submitted implementation of SIKE [JAC+20].

As an elliptic curve, any Montgomery curve defined over $\mathbb{F}_q$ is isomorphic to some curve in Weierstrass form defined over $\mathbb{F}_q$ by Theorem 2.3.3. However, not all Weierstrass curves over $\mathbb{F}_q$ are isomorphic to a curve in Montgomery form over $\mathbb{F}_q$. One example of a restriction on Montgomery curves is that the $\mathbb{F}_q$-rational points of a curve in Montgomery form always has order divisible by 4.

Another model is the twisted Edwards curve. The Edwards curve model was introduced by Edwards in 2007 [Edw07], and later generalized to the twisted Edwards curve by Bernstein et al. [BBJ+08].

Definition 3.2.2. (Twisted) Edwards curves: Let E be a projective set, defined as

$$E : \left(aX^2 + Y^2\right) Z^2 = Z^4 + dX^2Y^2 \tag{3.2}$$

with a specified point $\mathcal{O} = (0 : 1 : 1) \in E$. Then E is a twisted Edwards curve if and only if $ad(a - d) \neq 0$. If $a = 1$, then E is simply called an Edwards curve.

We give the twisted Edwards curve as a separate definition, as it falls outside our definition of elliptic curves, most importantly because it has singular points at $(1 : 0 : 0)$ and $(0 : 1 : 0)$. Nevertheless, its use has been studied in elliptic curve cryptography for many reasons. Similarly to Montgomery curves, the non-singular group of $\mathbb{F}_q$-rational points on twisted Edwards curves always has order divisible by 4. In addition to speeding up arithmetic, twisted Edwards curves can achieve so-called *complete* addition formula by appropriate choice of parameters, meaning that one formula for addition works for all cases[4]. This has the benefit of simplifying implementation and gives a natural resistance against side-channel attacks. In the context of isogeny-based cryptography, twisted Edwards curves have been studied,

[4]Compare this with the addition formulae for Weierstrass curves (Proposition 2.3.6): Weierstrass curves have different formulae based on whether $P = Q$ or not and whenever $\mathcal{O}$ is involved.

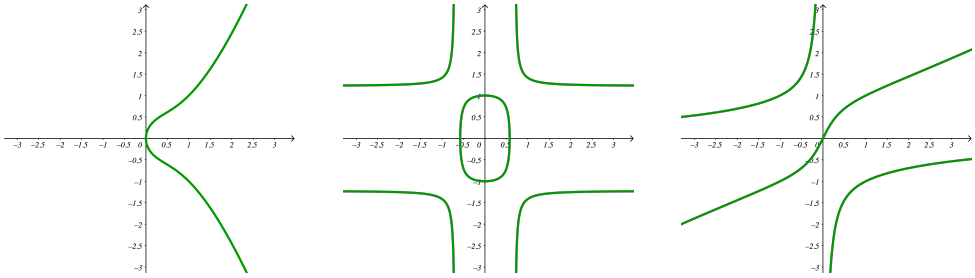


Figure 3.1: From left to right: The Montgomery curve $y^2 = x^3 - x^2 + x$, the twisted Edwards curve $3x^2 + y^2 = 1 + 2x^2y^2$ and the Huff's curve $2x\left(y^2 - 1\right) = y\left(x^2 - 1\right)$.

for instance by Kim et al. who reported that twisted Edwards curves were as fast as Montgomery curves for evaluating low-degree isogenies [KYK+18].

Another elliptic curve model that has been considered for isogeny-based cryptography is Huff's model.

Proposition 3.2.3. Huff's model: *Let E be a projective set, defined as*

$$E : aX\left(Y^2 - Z^2\right) = bY\left(X^2 - Z^2\right) \tag{3.3}$$

with a specified point $\mathcal{O} = (0 : 0 : 1) \in E$. Then E is an elliptic curve if and only if $ab\left(a^2 - b^2\right) \neq 0$.

One property of Huff's model of elliptic curves which makes them interesting for elliptic curve cryptography is that it has an almost complete addition formula[5] which is independent of its parameters a and b [JTV10]. The group of $\mathbb{F}_q$-rational points on an elliptic curve in Huff's model always has order divisible by 8. Huff's model has been studied in the context of isogeny-based cryptography and shown to be competitive with Montgomery and Edwards curves [DKW20].

Figure 3.1 shows the affine plane $A^2(\mathbb{R})$ at $Z = 1$ for examples of all three models introduced in this section, defined over $\mathbb{R}$. Keep in mind that for cryptographic purposes, we consider elliptic curves defined over finite fields. However, plotting curves over finite fields is not very helpful, since the curves look more like randomly scattered points, so we choose to plot the curves over $\mathbb{R}$.

[5]To be more precise, the formula is complete for all cyclic subgroups of odd order, when the curve is defined over a field of odd characteristic.

3.2.2 Twisted Hessian Curves

The elliptic curve model that we study in this thesis is the twisted Hessian curve, which is a generalization of the Hessian curve. Twisted Hessian curves were first introduced by Bernstein et al. [BCKL15].

Definition 3.2.4. (Twisted) Hessian curves: Let E be a projective set, defined as

$$E : aX^3 + Y^3 + Z^3 = dXYZ \tag{3.4}$$

With a specified point $\mathcal{O} = (0 : -1 : 1) \in E$. Then E is an elliptic curve (called a twisted Hessian curve) if and only if $a\left(27a - d^3\right) \neq 0$. If $a = 1$, then E is simply called a Hessian curve.

To simplify notation, we will often write a twisted Hessian curve $E : aX^3 + Y^3 + Z^3 = dXYZ$ as $H(a, d)$. Any twisted Hessian curve $H(a, d)$ is isomorphic to a Hessian curve $H\left(1, \frac{d}{\sqrt[3]{a}}\right)$ through the isomorphism $\varphi(X : Y : Z) = (\sqrt[3]{a}X : Y : Z)$, but keep in mind that the isomorphism may not be defined over $\mathbb{F}_q$ (however, it is always defined over $\mathbb{F}_q\,(\sqrt[3]{a})$).

We present the two formulae for addition on $H(a, d)$. Interestingly, Bernstein et al. showed that if a is not a cube in $\mathbb{F}_q$, then the rotated addition law is complete [BCKL15, Chapter 3 and 4].

Theorem 3.2.5. The standard addition law *Let $H(a, d)$ be a twisted Hessian curve, and let $P = (X_1 : Y_1 : Z_1)$ and $Q = (X_2 : Y_2 : Z_2)$ be points on $H(a, d)$. Define*

$$\begin{aligned} X_3 &= X_1^2 Y_2 Z_2 - X_2^2 Y_1 Z_1 \\ Y_3 &= Z_1^2 X_2 Y_2 - Z_2^2 X_1 Y_1 \\ Z_3 &= Y_1^2 X_2 Z_2 - Y_2^2 X_1 Z_1 \end{aligned}$$

If $(X_3 : Y_3 : Z_3) \neq (0 : 0 : 0)$, then $P + Q = (X_3 : Y_3 : Z_3)$.

Theorem 3.2.6. The rotated addition law *Let $H(a, d)$ be a twisted Hessian curve, and let $P = (X_1 : Y_1 : Z_1)$ and $Q = (X_2 : Y_2 : Z_2)$ be points on $H(a, d)$. Define*

$$\begin{aligned} X_3' &= Z_2^2 X_1 Z_1 - Y_1^2 X_2 Y_2 \\ Y_3' &= Y_2^2 Y_1 Z_1 - a X_1^2 X_2 Z_2 \\ Z_3' &= a X_2^2 X_1 Y_1 - Z_1^2 Y_2 Z_2 \end{aligned}$$

If $(X_3' : Y_3' : Z_3') \neq (0 : 0 : 0)$, then $P + Q = (X_3' : Y_3' : Z_3')$.

The rotated addition only law fails (i.e. it returns the zero coordinate) in cases when $(X_2 : Y_2 : Z_2) = (Z_1 : c^2 X_1 : cY_1)$, where $c^3 = a$. Therefore, if $c \notin \mathbb{F}_q$, then the rotated addition law is complete when working in the $\mathbb{F}_q$ rational part of $H(a, d)$. In general, the standard addition law always works in the cases where the rotated addition law fails[6], so together they always allow for calculating $P + Q$, even if a is a cube.

The negative of a point on a twisted Hessian curve is found as follows.

Proposition 3.2.7. *Let $H(a, d)$ be a twisted Hessian curve, and let $(X : Y : Z) = P \in H(a, d)$. Then $-P = (X : Z : Y)$.*

From Proposition 3.2.7, it is easy to deduce that the points of order 2 are precisely the points $P = (X : Y : Z)$ where $Y = Z$. A similar result about points of order 3 is stated in Theorem 3.2.11, which are particularly simple on a twisted Hessian curve. We return to this in Chapter 5, where it will become very important.

Bernstein et al. also gave very fast tripling formulae for twisted Hessian curves (i.e. a way of calculating $[3]P$ directly as opposed to calculating $P + P + P$). See Section 4.3.3 for more details.

Finally, we restate the isogeny formulae for twisted Hessian curves by Dang and Moody [DM19]. The first formula works for any kernel subgroup G, with $3 \nmid |G|$, while the two other formulae cover the cases for subgroups of size exactly 3. In general, isogenies of arbitrary degree n can be calculated by writing $n = 3^r m$ where $3 \nmid m$, and decomposing the isogeny as r isogenies of degree 3 and an isogeny of degree m. See Section 3.3.4 for more details on decomposition of isogenies.

Theorem 3.2.8. **Case 1:** *Let G be a finite subgroup of $H(a, d)$ of size m, with $3 \nmid m$. Then $\phi : H(a, d) \to H(a^m, D)$ is an isogeny with G as a kernel, given by*

$$\phi(P) = \left(\prod_{R \in G} X(P + R) : \prod_{R \in G} Y(P + R) : \prod_{R \in G} Z(P + R) \right),$$

where $X(P)$ means the X-coordinate of the point P and similarily for $Y(P)$ and $Z(P)$.

In Theorem 3.2.8, the parameter D of the image curve is given by a somewhat convoluted formula. Instead of calculating D directly, it instead possible to recover it by taking the image of a random point and recovering D from the curve equation, which is what we do in our implementation. See Section 4.1.4 for details.

[6]However, it always fails when $(X_2 : Y_2 : Z_2) = (\omega^2 X_1 : \omega Y_1 : Z_1)$, where $\omega^3 = 1$, so for instance it always fails for point doubling.

Next, we give the two formulae for isogenies of degree 3.

Theorem 3.2.9. **Case 2:** *Let* $\mathcal{S}_1 = \{(0,-1,1),(0,-\omega,1),(0,-\omega^2,1)\}$ *be a subgroup of* $H(a,d)$*, where* $\omega^3 = 1, \omega \neq 1$*. Then* $\phi : H(a,d) \to H(d^3 - 27a, 3d)$ *is an isogeny with* $\mathcal{S}_1$ *as a kernel, given by*

$$\phi(X:Y:Z) = (XYZ : aX^3 + \omega^2 Y^3 + \omega Z^3 : aX^3 + \omega Y^3 + \omega^2 Z^3).$$

Theorem 3.2.10. **Case 3:** *Let* $\mathcal{S}_2 = \{(0,-1,1),(1,-c,0),(1,0,-c)\}$ *be a subgroup of* $H(a,d)$*, where* $c^3 = a$*. Then* $\phi : H(a,d) \to H(d^2c + 3dc^2 + 9a, d + 6c)$ *is an isogeny with* $\mathcal{S}_2$ *as a kernel, given by*

$$\phi(X:Y:Z) = (XYZ : c^2X^2Z + cXY^2 + YZ^2 : c^2X^2Y + cXZ^2 + Y^2Z).$$

The fact that $\mathcal{S}_1$ and $\mathcal{S}_2$ are subgroups is easily verified by looking at the addition formula. Furthermore, from Theorem 3.1.12, we know that $E[3]$ has $|\mathbb{Z}/3\mathbb{Z} \times \mathbb{Z}/3\mathbb{Z}| = 9$ elements for any elliptic curve (recall, we assume $\text{char}(\mathbb{F}_q) \neq 3$), so all subgroups of order 3 are on the form of $\mathcal{S}_1$ or $\mathcal{S}_2$ (because there are 3 choices for c, with $c^3 = a$). Therefore, Theorem 3.2.9 and 3.2.10 covers all cases for subgroups of order 3.

It can also be verified that if $(X:Y:Z) = P \in H(a,d)$ and one of X, Y, Z are 0, then $P \in \mathcal{S}_1$ or $P \in \mathcal{S}_2$ [DM19]. This proves the following theorem.

Theorem 3.2.11. *Let* $P = (X:Y:Z) \neq \mathcal{O}$ *be a point on* $H_{a,d}$*. Then* P *has order* 3 *if and only of* $XYZ = 0$*.*

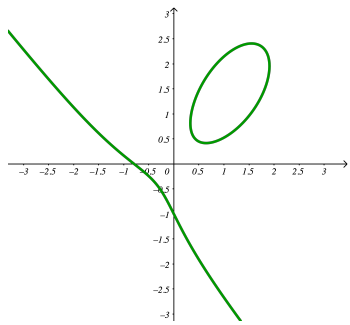


Figure 3.2: The twisted Hessian curve $2x^3 + y^3 + 1 = 6xy$.

Figure 3.2 shows the affine plane $A^2(\mathbb{R})$ at $Z = 1$ for a twisted Hessian curve defined over $\mathbb{R}$. Again, we will only be working with curves over $\mathbb{F}_q$, so the plot is of no particular importance, but is given as a reference to the other affine planes that were shown in Figure 3.1.

3.3 Supersingular Isogeny Diffie-Hellman

Finally, we turn our attention to the Supersingular Isogeny Diffie-Hellman (SIDH) protocol, which we focus on in this thesis.

Isogeny-based cryptography is yet another application of elliptic curves in cryptography. Lately, a lot of work has been published in this area, as SIKE, a key-encapsulation mechanism based on SIDH, was submitted to the National Institute of Science and Technology's Post-Quantum Cryptography (NIST PQC) standardization contest [NIS16]. It is by spring 2021 among the reserves in the third round [AASA+20].

There exist other isogeny-based cryptosystems, but we restrict our attention to SIDH. We refer the reader to the literature for details on other isogeny-based cryptosystems, e.g. the first published cryptosystem based on isogenies by Rostovtsev and Stolbunov's [RS06][7], or the more recently proposed Commutative SIDH (CSIDH) [CLM+18], which works very similar to Rostovtsev and Stolbunov's protocol, but which uses supersingular elliptic curves to achieve a massive speed-up.

3.3.1 The Protocol

From a high-level perspective, SIDH looks very similar to regular Diffie-Hellman. The protocol works because the diagram in Figure 3.3 commutes, at least up to isomorphism. We recall Theorem 3.1.8, which states that for every finite subgroup G of an elliptic curve E, all isogenies having G as a kernel has the same codomain (up to isomorphism). This suggests the following key exchange scheme:

- The public parameter is some starting curve E,
- Alice and Bob both pick a secret subgroup (generated by R_A and R_B respectively), apply the corresponding isogeny (ϕ_A and ϕ_B), and send the new curve ($E/\langle R_A \rangle$ and $E/\langle R_B \rangle$) over to the other party,
- Alice and Bob next apply the isogeny corresponding to the image of their secret subgroups (generated by $\phi_B(R_A)$ and $\phi_A(R_B)$ respectively), and since isogenies can be composed together, the resulting curve corresponds to $E/\langle R_A, R_B \rangle$, which means they have a shared secret.

There is one central issue with the simplified explanation above. The difficulty here is that Alice cannot give Bob R_A to evaluate in ϕ_B, because R_A is supposed to be secret, and vice versa. We use the next paragraphs to show how Alice and Bob

[7] Unpublished work by Couveignes has since shown that he already had some of the key ideas behind Rostovtsev and Stolbunov's cryptosystem in 1997 [Cou06].

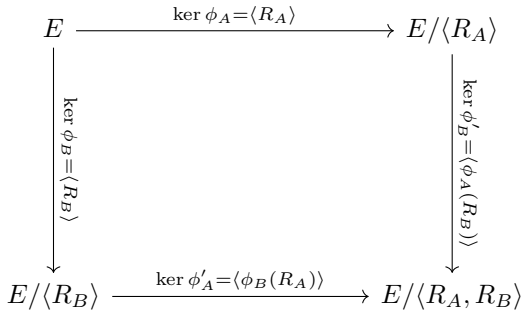


Figure 3.3: High level view of SIDH. Arrows are isogenies corresponding to the given kernel.

can recover their generators without revealing them to the other party. The main idea is that Alice and Bob work in separate torsion subgroups of E and that Alice and Bob evaluate generators of each other's torsion subgroups instead of the secrets R_A and R_B itself.

One key idea is to select some prime $p = \ell_A^{e_a} \ell_B^{e_b} f \pm 1$, where ℓ_A and ℓ_B are two small, distinct primes (typically 2 and 3) and f is some small cofactor to ensure that p is a prime. We fix the field that we are working over to be $\mathbb{F}_{p^2}$. Then, because of Theorem 3.1.15, any supersingular curve E over $\mathbb{F}_{p^2}$ has order $p^2 + kp + 1$, where $k \in \{-2, -1, 0, 1, 2\}$. We select a curve with $k = \pm 2$, such that $\#E(\mathbb{F}_{p^2}) = (\ell_A^{e_a} \ell_B^{e_b} f)^2$. The structure of a curve of that order is given as $E(\mathbb{F}_{p^2}) \cong \mathbb{Z}/(\ell^{e_A} \ell^{e_B} f)\mathbb{Z} \times \mathbb{Z}/(\ell^{e_A} \ell^{e_B} f)\mathbb{Z}$. How to select k, and get the correct curve was shown by Bröker [Brö09]. Further, recall Theorem 3.1.12, which states that the structure of $E[\ell^e] \cong \mathbb{Z}/\ell^e\mathbb{Z} \times \mathbb{Z}/\ell^e\mathbb{Z}$. From this, it is straight forward to derive the following proposition:

Proposition 3.3.1.

Let $E(\mathbb{F}_{p^2}) \cong \mathbb{Z}/(\ell^{e_A} \ell^{e_B} f)\mathbb{Z} \times \mathbb{Z}/(\ell^{e_A} \ell^{e_B} f)\mathbb{Z}$. Then every point in $E[\ell_A^{e_A}]$ and $E[\ell_B^{e_B}]$ is $\mathbb{F}_{p^2}$-rational.

The proposition is a direct result of the structure of $E(\mathbb{F}_{p^2})$, which contains $(\ell_A^{e_A})^2$ points of order dividing $\ell_A^{e_A}$. Since this is the same number of elements as there are in $E[\ell_A^{e_A}]$, every point in $E[\ell_A^{e_A}]$ is also in $E(\mathbb{F}_{p^2})$. The same argument works for $E[\ell_B^{e_B}]$.

We are now ready to accurately describe the protocol. The protocol starts by fixing p, $\mathbb{F}_{p^2}$ and a starting supersingular curve E as described above, in addition to a basis $\{P_A, Q_A\}$ for $E[\ell_A^{e_A}]$ and a basis $\{P_B, Q_B\}$ for $E[\ell_B^{e_B}]$. Alice samples random

Public parameters:
$E, \{P_A, Q_A\}, \{P_B, Q_B\}$

Alice		**Bob**
$n_A, m_A \xleftarrow{\$} \mathbb{Z}_{\ell_A^{e_A}}$		$n_B, m_B \xleftarrow{\$} \mathbb{Z}_{\ell_B^{e_B}}$
$R_A \leftarrow [n_A]P_A + [m_A]Q_A$		$R_B \leftarrow [n_B]P_B + [m_B]Q_B$
$E_A \leftarrow \phi_A(E)$		$E_B \leftarrow \phi_B(E)$
	$E_A,$ $\phi_A(P_B),$ $\phi_A(Q_B)$ $\longrightarrow$	
	$E_B,$ $\phi_B(P_A),$ $\phi_B(Q_A)$ $\longleftarrow$	
$R'_A \leftarrow [n_A]\phi_B(P_A) + [m_A]\phi_B(Q_A)$		$R'_B \leftarrow [n_B]\phi_A(P_B) + [m_B]\phi_A(Q_B)$
$E_{AB} \leftarrow \phi'_A(E_B)$		$E_{AB} \leftarrow \phi'_B(E_A)$
$\mathsf{k} \leftarrow j(E_{AB})$		$\mathsf{k} \leftarrow j(E_{AB})$

Figure 3.4: The SIDH protocol.

$m, n \in \mathbb{Z}_{\ell_A^{e_A}}$, and finds her secret $R_A = [m]P_A + [n]Q_A$. To ensure that R_A has order $\ell_A^{e_A}$, she requires that at least one of m and n are not divisible by ℓ_A. She then computes her corresponding secret isogeny ϕ_A, and computes $\phi_A(E) = E/\langle R_A \rangle$, $\phi_A(P_B)$ and $\phi_A(Q_B)$, and sends Bob $(E/\langle R_A \rangle, \phi_A(P_B), \phi_A(Q_B))$. Bob does the same, but for his parameters. To arrive at a shared secret, Alice finds $\phi_B(R_A)$ by computing $[m]\phi_B(P_A) + [n]\phi_B(Q_A)$, which works because $[m]\phi_B(P_A) + [n]\phi_B(Q_A) = \phi_B([m]P_A + [n]Q_A) = \phi_B(R_A)$. She then uses $\phi_B(R_A)$ to compute the corresponding isogeny ϕ'_A, and finds $\phi'_A(E_B) = E/\langle R_A, R_B \rangle$. Again Bob does the same for his parameters, arriving at $E/\langle R_B, R_A \rangle \cong E/\langle R_A, R_B \rangle$. Finally, since these curves are the same up to only up to isomorphism, Alice and Bob both take the j-invariant of their final curve to use as their shared secret. The protocol is summarized in Figure 3.4.

3.3.2 Random Walks in Isogeny-Graphs

We give a brief mention of another view of isogeny-based cryptography. The main motivation for introducing this view is that in Chapter 5, we present a tweaked SIDH scheme, where the random isogeny-walks are much closer to the diameter of the

supersingular isogeny-graph. Because of this, it can achieve the so-called mixing property that regular SIDH fails to achieve.

In a general sense, all isogeny-based cryptography is based on random walks in isogeny-graphs. We give an informal introduction to this view. For a more thorough description, we refer to the literature [Feo17, Part III]. Since we focus on SIDH, we restrict our attention to supersingular isogeny graphs.

For a fixed prime p, and another prime ℓ, we can create a graph from an equivalence class of isogenous curves over $\bar{\mathbb{F}}_p$. Denote this graph $\mathcal{G}_\ell(\bar{\mathbb{F}}_p)$. In this graph, the vertices are isomorphism classes of elliptic curves, while edges between vertices are isogenies of degree ℓ. Since all isogenies have a dual (see the discussion succeeding Theorem 3.1.10), this graph can be constructed as an undirected graph[8]. For SIDH, and many other examples in isogeny-based cryptography, we can now think of the secret keys as a secret path between two vertices in this graph.

Notice that in SIDH, the communicating parties do not work in the same isogeny-graph. Both parties graphs have the same set of vertices, but the edges are different. In particular, Alice works in $\mathcal{G}_{\ell_A}(\bar{\mathbb{F}}_p)$, while Bob works in $\mathcal{G}_{\ell_B}(\bar{\mathbb{F}}_p)$.

Supersingular isogeny graphs are what is called Ramanujan, which gives them a lot of interesting properties. We avoid the definition of Ramanujan graphs but instead focus on the interesting properties that are a result of this. For any primes p and ℓ, the supersingular isogeny graph $\mathcal{G}_\ell(\bar{\mathbb{F}}_p)$ has the following properties:

An exponential number of vertices in $\log p$. By Theorem 3.1.21, the number of vertices is $\approx \frac{p}{12}$.

It is $\ell + 1$-regular. This simply means that all vertices have degree $\ell + 1$.

A short diameter. The diameter of a graph is the longest distance between two vertices, where the distance is calculated as the length of the shortest path between two vertices. In $\mathcal{G}_\ell(\bar{\mathbb{F}}_p)$, the diameter is of size $\Theta(\log p)$.

The graph is an expander graph. Informally, this leads to the fact that random walks of length close to the diameter are almost the same as uniform samples of the vertices.

The last point on expander graphs is called the mixing property. Notice that in SIDH, the isogeny-walks are not long enough to achieve the mixing property in general, because the isogeny-walks are of length $\approx \frac{1}{2}\log p$, while the diameter is of

[8]This is only partially true. There are some complicating details here regarding curves with supersingular j-invariants 0 and 1728, which we have omitted.

length $\approx \log p$. Therefore, informally, the public keys in SIDH cannot be seen as uniform samples of the vertices in a supersingular isogeny graph. To fix this, we could simply have opted to select a smaller prime p. The caveat then is that the torsion groups that Alice and Bob are working in, are no longer necessarily $\mathbb{F}_{p^2}$-rational. We will return to this in Chapter 5, where we show some ideas on how to do the isogeny-calculation in SIDH without a $\mathbb{F}_{p^2}$-rational torsion group.

3.3.3 Security and Cryptanalysis

In this subsection, we briefly discuss aspects related to the security of the SIDH protocol. A more thorough discussion, as well as security proofs, can be found in [JD11, Chapter 5 and 6]. We start by defining a problem that recovers the secret from the information the adversary has available. This problem is analogous to the discrete logarithm problem for regular Diffie Hellman.

Throughout this section, let the public parameters $p, E, \{P_A, Q_A\}, \{P_B, Q_B\}$ be as before, i.e. $p = \ell_A^{e_A}\ell_B^{e_B} f \pm 1$, and E a supersingular curve with $E(\mathbb{F}_{p^2}) \cong (\mathbb{Z}/(\ell^{e_A}\ell^{e_B} f)\mathbb{Z})^2$ and $\{P_A, Q_A\}, \{P_B, Q_B\}$ generators for $E[\ell_A^{e_A}]$ and $E[\ell_B^{e_B}]$ respectively.

Definition 3.3.2. Computational Supersingular Isogeny (CSSI) problem. Let $\phi_A : E \to E_A$ be an isogeny with $\ker \phi_A = \langle [n]P_A + [m]Q_A \rangle$, where $m, n \in \mathbb{Z}_{\ell_A^{e_A}}$, and not both divisible by ℓ_A. Given $E_A, \phi_A(P_A), \phi_A(Q_A)$, find a generator R_A of $\ker \phi_A$.

A more general version of the CSSI problem is simply the following: Given E and $E_A = E/\langle R_A \rangle$, find a generator of $\langle R_A \rangle$. However, this does problem does not consider the additional potential advantage an adversary may gain from seeing the images of the torsion points. Currently, the best known attacks against the CSSI problem do not use the additional information at all. However, Petit has shown that some related problems may benefit from the extra information [Pet17]. This suggests that at least some information is leaked and that the CSSI problem may be strictly easier than the more general version mentioned at the start of this paragraph.

There exists an algorithm which runs in quantum subexponential time (meaning there is an algorithm which runs on a quantum computer that solves the problem in subexponential time), which solves the analogue of the generic CSSI problem for ordinary curves [CJS14]. This algorithm affects both the protocol by Rostovtsev and Stoblunov, and CSIDH. However adapting the algorithm to solving CSSI in quantum subexponential time does not seem possible, as the algorithm relies on the commutative nature of the endomorphism ring of the curve, while as we have seen

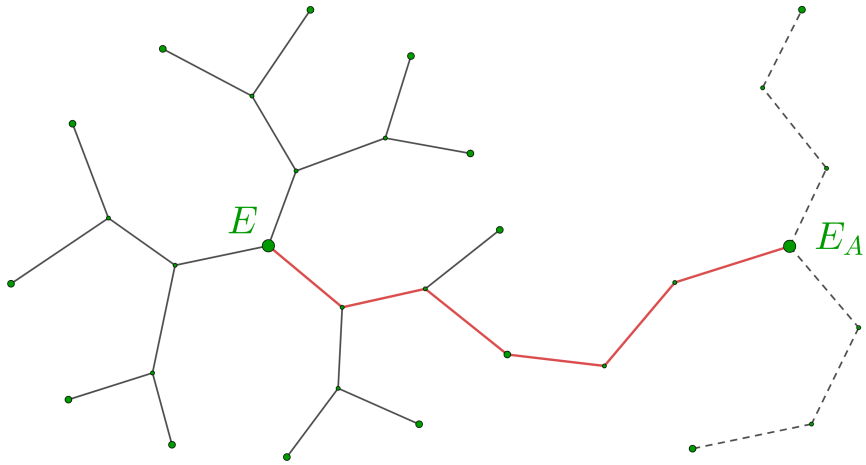


Figure 3.5: Visual example of the claw-finding algorithm for solving the CSSI problem, with $\ell_A = 2$ and $e_A = 6$. The graph is a subgraph of an isogeny graph where the edges are isogenies of degree 2

the endomorphism ring of supersingular curves are non-commutative (by definition 3.1.14)[9].

To assess the difficulty of bruteforcing a solution, one can ask about many choices of secrets there are, which corresponds to asking about the size of the key-space. To answer this, we must recall that $E[\ell_A^{e_A}] \cong (\mathbb{Z}/\ell_A^{e_A}\mathbb{Z})^2$, and that $E[\ell_A^{e_A}] \subseteq E(\mathbb{F}_{p^2})$. It can be shown using elementary group theory that $(\mathbb{Z}/\ell_A^{e_A}\mathbb{Z})^2$ contains $\ell_A^{e_A-1}(\ell_A+1)$ different cyclic subgroups of order $\ell_A^{e_A}$. Since $E[\ell_A^{e_A}] \subseteq E(\mathbb{F}_{p^2})$, all of these are possible secret keys, so the key space has size $\ell_A^{e_A-1}(\ell_A+1)$. In practice, what is often done is to simply select a single secret number n and compute $R_A = P_A + [n]Q_A$, which reduces the number of secrets to $\ell_A^{e_A}$. In any case, the key space is approximately of size $\sqrt{p}$ when $\ell_A^{e_A} \approx \ell_B^{e_B}$, which quickly makes brute forcing computationally infeasible. Still, notice that this is still much smaller than the total number of supersingular j-invariants in SIDH, as given in Theorem 3.1.21.

To improve on the brute force strategy, we use the fact that to recover a secret isogeny $\phi : E \to E_A$, it is sufficient to recover the dual of the secret $\hat{\phi} : E_A \to E$. This suggests the following improvement: Build a tree of all possible isogenies from

[9]It might be surprising that the algorithm still affects CSIDH, even when CSIDH uses supersingular curves. This comes from the fact that CSIDH uses the subring of $\mathrm{End}(E)$ of isogenies defined over $\mathbb{F}_p$, which is again commutative (hence the name).

E of degree $\ell_A^{e_A/2}$. Next, try isogenies from E_A of degree $\ell_A^{e_A/2}$ until the image curve collides with one of the leaf nodes in the tree from E. This then forms a path from E to E_A in the isogeny-graph corresponding to the secret isogeny of degree $\ell_A^{e_A}$. See Figure 3.5.

This strategy suggests that the problem CSSI problem can be viewed as a special case of the so-called claw finding problem. On a classical computer, this can be solved (using the strategy discussed in the previous paragraph) in $\mathcal{O}\big(\sqrt[4]{p}\big)$ time and $\mathcal{O}\big(\sqrt[4]{p}\big)$ memory. On a quantum computer, this can be improved to give a $\mathcal{O}\big(\sqrt[6]{p}\big)$ attack against SIDH [Tan09].

3.3.4 Smooth-Degree Isogeny-Computation

Mathematically, given a curve E and a subgroup G, one can compute the corresponding isogeny ϕ, with $\ker\phi = G$, by using Vélu's formula [Vél71]. The problem is that the complexity of Vélu's formula grows too quickly with the degree of the isogeny. To speed up computation, Alice and Bob can compute their high-degree isogeny as a composition of many low-degree isogenies instead. In this section, we focus on how to do this in an efficient way. The goal of the section is to show the ideas which lead to the optimal strategies, covered in Section 4.2.2. The strategies discussed in this section were introduced by Jao and De Feo [JD11, Chapter 4.2].

Let E be a curve as in the previous sections, and let R be a point of order ℓ^e. To compute the isogeny ϕ with $\ker\phi = \langle R\rangle$, we decompose ϕ as e isogenies of degree ℓ, such that $\phi = \phi_{e-1}\circ\phi_{e-2}\circ\cdots\circ\phi_0$. To do this, we can use the natural strategy in Algorithm 3.1. Here, `IsoFromKer`(P) returns the isogeny with kernel $\langle P\rangle$ (which can for instance be computed with Vélu's formula, or in our case, one of Theorem 3.2.8, Theorem 3.2.9 or Theorem 3.2.10).

Algorithm 3.1 Basic strategy for computing isogeny of degree ℓ^e

Input: An elliptic curve E, a point $R \in E$ of order ℓ^e
Output: The curve $E/\langle R\rangle$

1: $R_0 \leftarrow R, E_0 \leftarrow E$
2: **for** $i \leftarrow 0, e-1$ **do**
3: $\quad\phi_i \leftarrow$ `IsoFromKer`$([\ell^{e-i-1}]R_i)$
4: $\quad E_{i+1} \leftarrow \phi_i(E_i)$
5: $\quad R_{i+1} \leftarrow \phi_i(R_i)$
6: **end for**
7: **return** E_e

Notice that for each i, the point $[\ell^{e-i-1}]R_i$ is a point of order ℓ, which implies that ϕ_i is a degree ℓ isogeny by proposition 3.1.9. From this, we see that the strategy works. Clearly, since R is in $\ker\phi$, we have $\langle R\rangle \subseteq \ker\phi$. Further, because ϕ has

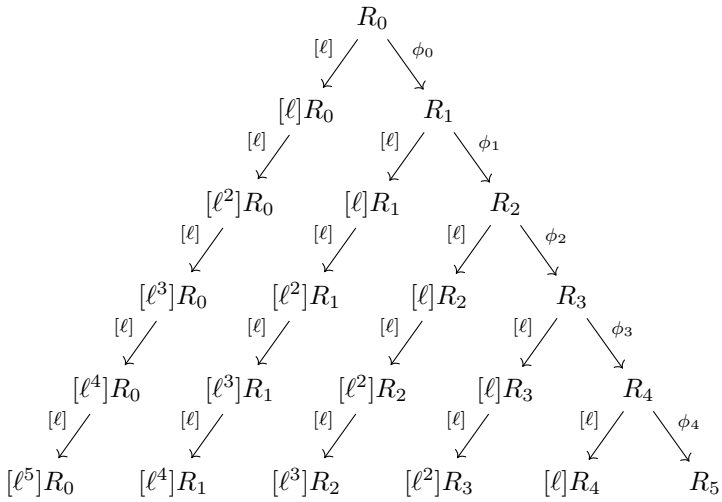


Figure 3.6: A multiplication-focused strategy for computing an isogeny of degree ℓ^e.

degree ℓ^e, and R is a point of order ℓ^e, we have that $|\ker\phi| = |\langle R\rangle|$, from which it follows that $\ker\phi = \langle R\rangle$. We visualize this calculation as shown in Figure 3.6 for a small example of $e = 6$. Every ϕ_i has kernel $[\ell^{5-i}]R_i$, so in order to evaluate get ϕ_i, we must first compute the corresponding leaf-node $[\ell^{5-i}]R_i$. To get the whole isogeny ϕ, corresponds to calculating every leaf-node in the tree in Figure 3.6.

However, Jao and De Feo showed that this is far from the optimal strategy. The improved strategies rely on Theorem 3.1.6, which states that isogenies are also group homomorphisms. This gives a lot of flexibility, since calculating $[\ell^k]\phi_i(R_i)$ is the same as calculating $\phi_i([\ell^k]R_i)$. So, for instance, we can do the previous calculation in a similarily trivial (although, slightly less intuitive way), as seen in Figure 3.7.

This strategy might give a slight speed up if evaluating isogenies is faster than multiplying by $[\ell]$, but in terms of operations, it is similar to the first strategy. However, in general, we can "mix" these strategies, and select a subset of all dashed arrows in Figure 3.8 to reach all leaf nodes in as few arrows (which corresponds to calculations) as possible.

By selecting the lowest number of arrows needed to reach all leaf nodes, we improve the calculations needed from $\mathcal{O}(e^2)$ to $\mathcal{O}(e\log(e))$. Interestingly, because the cost of multiplying by $[\ell]$ and evaluating l-isogenies may be different, a slightly unbalanced strategy will sometimes be faster than just calculating the fewest number of arrows needed. Jao and De Feo have already shown how to create an algorithm, which given weights p, q corresponding to costs of multiplying by $[\ell]$ and evaluating

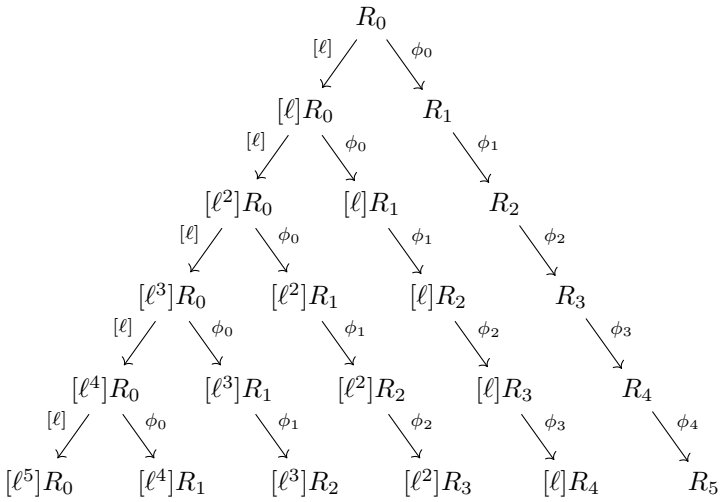


Figure 3.7: An isogeny-focused strategy for computing an isogeny of degree ℓ^e.

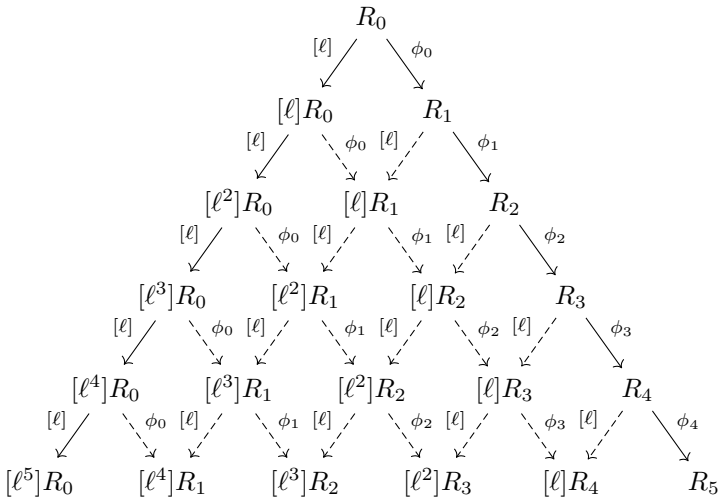


Figure 3.8: In general, one can choose any subset of the dashed arrows, such that there is a path from the root node to all leaf nodes.

l-isogenies, returns the subset of arrows, which corresponds to the lowest total cost when p and q is regarded as weights of left and right arrows respectively. See Section 4.2.2 for more details.

3.3.5 Supersingular Isogeny Key Encapsulation

We end this chapter with a brief mention of the Supersingular Isogeny Key Encapsulation (SIKE) protocol. Since this thesis focuses on the more generic SIDH key exchange, we avoid the details, but refer to the reference specification for details [JAC+20].

The goal of the NIST Post-Quantum Cryptography contest is to standardize KEMs and digital signatures which can withstand attacks by quantum computers as well as classical computers. SIKE is a key encapsulation protocol based on SIDH, submitted to the standardization contest.

The basic idea behind the construction of SIKE is fairly straightforward. Since SIDH works very similar to the classic Diffie Hellman key exchange, one can turn SIDH into a public-key encryption scheme, almost exactly the same way that one turns Diffie Hellman into ElGamal (see Section 2.4.2), and from there turn this construction into a key encapsulation mechanism. However, one has to be a bit careful in doing this. Galbraith et al. showed that without countermeasures, SIDH is not secure when using static private keys [GPST16]. Because of this, directly translating ElGamal to the SIDH setting results in an insecure cryptosystem. Therefore, SIKE employs a variant of a generic construction called the HHK transform [HHK17], which turns SIDH into a secure KEM.

Chapter 4
SIDH with Twisted Hessian Curves

This chapter outlines how to directly apply twisted Hessian curves to the SIDH protocol. First, we provide some necessary results about twisted Hessian curves. Section 4.2 covers two algorithms required for an efficient implementation, while Section 4.3 covers all essential aspects of an implementation of SIDH using twisted Hessian curves.

4.1 Mathematical Aspects

In this section, we will go through some results, useful for working with twisted Hessian curves in a cryptographic setting.

4.1.1 Computing in $\mathbb{F}_{p^2}$

Arithmetic in the field we are working in lies at the heart of the implementation, and the run-time will be dominated by these operations. Still, the goal of the thesis is not to provide a fast implementation in this sense; we base our calculations on the number of field operations, so the efficiency of the field operations is not important.

Nevertheless, in addition to providing a generic framework, which allows instantiating a finite field as a quotient of any irreducible polynomial, we also provide a simpler and much faster implementation of arithmetic specialized for $\mathbb{F}_{p^2} \cong \mathbb{F}_p[X]/\langle X^2+1\rangle$. The polynomial X^2+1 is irreducible if and only if $p \equiv 3 \pmod 4$. This is a direct result of a well known observation from number theory that -1 is not a square mod p if and only if $p \equiv 3 \pmod 4$. This is then a requirement for any p that we choose. For obvious reasons, we denote the roots of X^2+1 as $\pm i$, and have a basis for our finite field as $\{1, i\}$, which means that our field is really instantiated as

$\mathbb{F}_{p^2} = \mathbb{F}_p(i) = \{a + bi \mid a, b \in \mathbb{F}_p, i^2 = -1\}$. Now, we can define field operations as

$$\begin{aligned}(a+bi) + (c+di) &= (a+c) + (b+d)i,\\ (a+bi) - (c+di) &= (a-c) + (b-d)i,\\ (a+bi)(c+di) &= (ac) + (ad+bc)i + (bd)i^2 = (ac-bd) + (ad+bc)i,\\ (a+bi)^{-1} &= a(a^2+b^2)^{-1} - b(a^2+b^2)^{-1}i,\end{aligned}$$

where of course operations between elements a, b, c, d happens in $\mathbb{F}_p$ (i.e. modulo p). For exponentiation, see 4.2.1.

We also provide a much more flexible and mathematically rigorous framework, which allows for instantiating a finite field as a quotient of $\mathbb{F}_p[X]$ over any irreducible polynomial. While in general, this framework is easier to test and to work with, it is also much slower, as addition, subtraction and multiplication are simply implemented as operations in $\mathbb{F}_p[X]$ followed by taking the remainder modulo the quotient polynomial by using the euclidean algorithm, and inversions are done by the extended euclidean algorithm. As it is easier to follow the optimized implementation, we focus on this in this chapter.

Finally, there is also a case to be made for considering instantiating the field as $\mathbb{F}_{p^2} = \mathbb{F}_p(\omega)$, where ω is a root of the irreducible polynomial $X^2 + X + 1$, or equivalently $\omega \neq 1, \omega^3 = 1$. While $\mathbb{F}_p(i)$ is recommended in the SIKE specification [JAC+20], Bernstein et al. showed that tripling points on twisted Hessian curves is faster in $\mathbb{F}_p(\omega)$ [BCKL15]. We look more at this field in Section 5.3.1.

4.1.2 The j-Invariant of a Twisted Hessian Curve

From Theorem 2.3.2, we know that any elliptic curve can be written in Weierstrass normal form over any field of characteristic not equal to 2 or 3. To rely on theory for curves on Weierstrass normal form, we wish to find a curve on Weierstrass normal form which corresponds to a given curve on twisted Hessian form. Bernstein et al. already showed that any curve given by an equation of the form $y^2 + a_1xy + a_2y = x^3$ (called a triangular curve) is isomorphic to a twisted Hessian curve on form $(a_1^3 - 3^3a_2)x^3 + y^3 + 1 = 3a_1xy$ [BCKL15, Theorem 5.3]. We use the inverse of this map to make the following observation.

Lemma 4.1.1. *Let K be a field with $char(K) \neq 2, 3$. Let $E : ax^3 + y^3 + 1 = dxy$ be a twisted Hessian curve defined over K. Then there exists an isomorphism from E to an elliptic curve in Weierstrass normal form $E' : y^2 = x^3 + Ax + B$ where $A = -d\left(8a + \left(\frac{d}{3}\right)^3\right)$ and $B = 16a^2 + 40a\left(\frac{d}{3}\right)^3 - 2\left(\frac{d}{3}\right)^6$.*

Proof. Notice that the triangular curve, is just a curve in Weierstrass long form with certain coefficients set to zero. Therefore, we can compose the inverse of the map

from a triangular curve to a twisted Hessian curves with the map from Weierstrass long form to Weierstrass normal form, and get the result above. To find the inverse map, we solve the set of equations

$$\begin{aligned} a &= a_1^3 - 3^3 a_2, \\ d &= 3a_1, \end{aligned}$$

which gives

$$\begin{aligned} a_1 &= \frac{d}{3}, \\ a_2 &= a - \frac{1}{3^3}\left(\frac{d}{3}\right)^3. \end{aligned}$$

This means that the curve $E : ax^3 + y^3 + 1 = dxy$ is isomorphic to the curve

$$y^2 + \left(\frac{d}{3}\right)xy + \frac{1}{3^3}\left(\frac{d^3}{3^3} - a\right)y = x^3$$

which is a curve in Weierstrass long form with $a_1 = \frac{d}{3}, a_2 = \frac{1}{3^3}\left(\frac{d^3}{3^3} - a\right)$ and $a_2 = a_4 = a_6 = 0$. We use the isomorphism from Weierstrass long form to Weierstrass normal form [Sil09, p.42-43] to calculate

$$\begin{aligned} A &= -3^3\left(\left(a_1^2 + 4a_2\right)^2 - 24\left(2a_4 + a_1a_3\right)\right) \\ &= -3^3\left(\left(a_1^2\right)^2 - 24(a_1a_3)\right) \\ &= -3^3\left(\left(\frac{d}{3}\right)^4 - 24\left(\frac{d}{3}\right)\left(\frac{1}{3^3}\left(\frac{d^3}{3^3} - a\right)\right)\right) \\ &= -3^3\left(\frac{d^4}{3^4} - \frac{8}{3^3}d\left(\frac{d^3}{3^3} - a\right)\right) \\ &= -\frac{d^4}{3} + 8\frac{d^4}{3^3} - 8ad \\ &= -\frac{9d^4 - 8d^4}{3^3} - 8ad \\ &= -d\left(8a + \left(\frac{d}{3}\right)^3\right). \end{aligned}$$

and similarly, we find

$$\begin{aligned}
B &= -54\left(-\left(a_1^2+4a_2\right)^3+36\left((a_1^2+4a_2)(2a_4+a_1a_3)\right)-216\left(a_3^2+4a_6\right)\right)\\
&= -54\left(-\left(a_1^2\right)^3+36\left((a_1^3a_3)\right)-216\left(a_3^2\right)\right)\\
&= -54\left(-\left(\frac{d}{3}\right)^6+36\left(\left(\frac{d}{3}\right)^3\left(\frac{1}{3^3}\left(\frac{d^3}{3^3}-a\right)\right)\right)-216\left(\frac{1}{3^3}\left(\frac{d^3}{3^3}-a\right)\right)^2\right)\\
&= -54\left(-\frac{d^6}{3^6}+\frac{4}{81}\left(d^3\left(\frac{d^3}{3^3}-a\right)\right)-\frac{8}{3^3}\left(\frac{d^3}{3^3}-a\right)^2\right)\\
&= \frac{2d^6}{27}-\frac{8\left(d^6-3^3ad^3\right)}{3^4}+16\left(\frac{d^6}{3^6}-\frac{2ad^3}{3^3}+a^2\right)\\
&= 16a^2+\frac{40ad^3}{27}-\frac{2d^6}{3^6}\\
&= 16a^2+40a\left(\frac{d}{3}\right)^3-2\left(\frac{d}{3}\right)^6
\end{aligned}$$

which completes the proof. □

Immediately from this, we get the result we need.

Proposition 4.1.2. *Let K be a field with $char(K) \neq 2, 3$. Let $E : ax^3+y^3+1 = dxy$ be a twisted Hessian curve defined over K. The j-invariant is of E is then given by:*

$$j(E) = \frac{\left(216ad+d^4\right)^3}{a\left(d^3-27a\right)^3}.$$

Proof. This is simply the j-invariant of the isomorphic Weierstrass curve from Lemma 4.1.1 after cancelling out terms. The j-invariant of a curve in Weierstrass normal form was given in Proposition 3.1.18. □

Another generalization of the Hessian curve of form $x^3 + y^3 + c = dxy$ was studied by Farashahi and Joye [FJ10]. They presented an equivalent formula for the j-invariant their curve. The fact that these formulas are the same is not surprising, as there exists an easy isomorphism between these curves and the twisted Hessian curves leaving the parameters fixed; simply permute the projective coordinates $(X : Y : Z) \to (Z : Y : X)$. Such an isomorphism, given by simply relabeling the coordinates, actually means that the curves are equal, as projective sets (see Section 2.2.2). To see this, notice that the generalization studied by Farashahi and Joye is really just the affine plane taken at $X = 1$ of a projective twisted Hessian curve, followed by rewriting coefficients as given by the isomorphism.

4.1.3 Degree 2 isogenies

We base the calculation of isogenies of degree 2 on the generic formula for isogenies of degree not divisible by 3 given by Dang and Moody [DM19], the standard addition law (Theorem 3.2.5) and the fact that for points of order 2 on a twisted Hessian curve, we have $Y = Z$ (see Proposition 3.2.7).

Let $E : aX^3 + Y^3 + Z^3 = dXYZ$ be a twisted Hessian curve. Combining the mentioned theory, we find that for a given point $(a : b : b) \in E$, the isogeny $\phi : E \to E'$ having $\{(0 : -1 : 1), (a : b : b)\}$ as kernel is given by

$$\phi(X : Y : Z) = (X(X^2b^2 - a^2YZ) : Y(Z^2ab - b^2XY) : Z(Y^2ab - b^2XZ)), \quad (4.1)$$

where $E' = a^2X^3 + Y^3 + Z^3 = DXYZ$. As mentioned in Section 3.2.2, the curve parameter D is in general given by the slightly convoluted formula

$$D = \frac{(1-2n)d + 6\sum_{i=1}^{n} \frac{1}{s_i t_i}}{\prod_{i=1}^{n} s_i}$$

for a kernel $\{(0 : -1 : 1) \cup (s_i : t_i : 1)\}_{i=1}^{n}$. In the degree 2-case, this turns into

$$D = \frac{-d + \frac{6}{\frac{a}{b}}}{\frac{a}{b}} = \frac{b(6b - ad)}{a^2}. \quad (4.2)$$

While this might not seem too bad, it still involves an inversion, and in the next section, we show how to avoid this when calculating 2^n-isogenies, for $2 \leqslant n \in \mathbb{N}$.

4.1.4 Recovering Curve Parameter From Arbitrary Point

In general, we would like to avoid inversions, as it is by far the slowest field-operation. Similar to how we avoid inversions in elliptic curve addition laws by using projective coordinates, one can use projective curve parameters as well. This was for instance done by Costello et al. [CLN16], in an optimization of SIDH using Montgomery curves. However, we choose a different approach which has the added benefit of ignoring the calculation of the d-coordinate, at the cost of a single inversion at the end of the smooth degree isogeny computation.

When looking at the image curve under an isogeny discussed in the previous section, the curve parameter a^2 is independent of d. Further, both the standard addition law and rotated addition law are also independent of the parameter d. The result of this is that we can do the whole 2^e-isogeny calculation while simply ignoring the parameter d throughout the whole calculation, and only recover it right at the end.

Given a twisted Hessian curve $E : aX^3 + Y^3 + Z^3 = dXYZ$, where d is unknown, we can easily recover d if we have a and a point $(U, V, W) = P \in E$, where $[3]P \neq \mathcal{O}$, i.e. P is not a point of order dividing 3. We can do this, since as long as $UVW \neq 0$ (Theorem 3.2.11), d is can easily be recovered by

$$\begin{aligned} dUVW &= aU^3 + V^3 + W^3, \\ d &= \frac{aU^3 + V^3 + W^3}{UVW}, \end{aligned} \tag{4.3}$$

which of course requires one inversion.

To apply this to SIDH, let Alice be the one working in the 2^{e_A}-torsion. For the first part of the key exchange, she can calculate $\phi_A = \phi_{e_A-1} \circ \cdots \circ \phi_0$, with $\phi_A : E \to E_A$, always ignoring the d parameter. At the end, Alice knows only the parameter a on the curve E_A, but she also has $\phi_A(P_B)$ and $\phi_A(Q_B)$. Of course, none of these points have order dividing 3 (they are the basis for Bob's 3^{e_B}-torsion), so she can recover d with equation 4.3. This way, she manages the whole calculation with a single inversion.

Alice is slightly worse off in the second part of the key exchange, as she no longer has any arbitrary points at the very end to recover d from. Therefore, she instead only ignores d when calculating $\phi'_{e_A-2} \circ \cdots \circ \phi'_0$, because on the second to last curve, she still has a kernel point she can use to recover d (this is what she uses to calculate ϕ'_{e_A-1}), which by construction is of order 2. So on the second to last curve, she recovers the parameter d of that curve, finds ϕ'_{e_A-1} as usual, and calculates the parameter D on the image curve as given in equation 4.2. In other words, for part II of the key exchange, she requires 2 inversions, which is still much better than the e_A inversions that would be required when calculating the parameter d every step.

4.2 Algorithmic Aspects

In this section, we briefly present two algorithms which our implementation relies a lot on.

4.2.1 Square and Multiply

This first algorithm is a simple but effective algorithm which does exponentiation by n in any in a multiplicative group (or multiplication by n in an additive group) in $\mathcal{O}(\log n)$ time[1]. We present it here using multiplicative notation, but in our implementation, it is also used for scalar-multiplication on elliptic curves. In the case of additive groups, the algorithm is typically called *double and add*.

[1]It is actually even more general as it works for any semi-group.

To compute x^n, start by writing n in binary form as $(b_k b_{k-1} \dots b_0)$. Rewriting using elementary exponent laws gives

$$x^n = x^{b_k 2^k + b_{k-1} 2^{k-1} + \cdots + b_0} = \left(x^{2^k}\right)^{b_k} \left(x^{2^{k-1}}\right)^{b_{k-1}} \dots \left(x^{2^1}\right)^{b_1} \left(x^{2^0}\right)^{b_0},$$

where the right-hand side above clearly requires at k squarings and at most k multiplications. There exist multiple ways of calculating this in an efficient manner. A simple and intuitive algorithm is shown in Algorithm 4.1, which starts with the rightmost binary digit of n. To avoid division and calculating the logarithm, we use a slightly different approach in our implementation, starting from the left-most digit of n, but it is based on the same principle.

Algorithm 4.1 Square and multiply

Input: An element $x \in (G, \cdot)$, an exponent $n \in \mathbb{N}^+$
Output: x^n

```
k ← ⌊log_2 n⌋
y ← 1
for i ← 0, k do
    if n ≡ 1 (mod 2) then
        y ← y · x
    end if
    x ← x^2
    n ← ⌊n/2⌋
end for
return y
```

4.2.2 Jao and De Feo's Optimal Strategy

In Section 3.3.4, we looked at the basic idea behind calculating smooth-degree isogenies in an efficient way. In this section, we show how we find an optimal strategy for computing ℓ^e-degree isogenies, given weights p and q, corresponding to cost of multiplication and isogeny-evaluation respectively. This was originally done by Jao and De Feo in their original supersingular isogeny Diffie-Hellman paper [JD11].

Let the size n of a strategy be the number of leaf nodes in the tree corresponding to the isogeny-calculation (see Section 3.3.4). We call a connected subset of the edges in such a tree that reaches all leaf nodes a *valid strategy*. Any valid strategy can be used to compute the isogeny, but we are after a strategy with the lowest total weight, where p is the weight of edges going down-left, and q is the weight of edges going down-right. Such a strategy is called an *optimal strategy*.

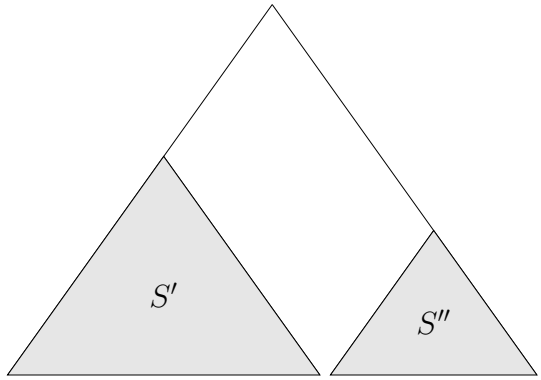


Figure 4.1: An optimal strategy S of size $n \geqslant 2$ always consists of two strictly smaller optimal strategies S' and S''.

There exists only one strategy of size $n = 1$ (since that corresponds to a single node and no edges). For all other strategies, we have the following lemma.

Lemma 4.2.1. *Let S be an optimal strategy of size $n \geqslant 2$. Then S consists of an optimal strategy S' of size i, and an optimal strategy S'' of size $n - i$ for some $1 \leqslant i \leqslant n - 1$, along with straight paths from the root of S to the roots of S' and S''.*

Intuitively, the lemma makes sense, but for a rigorous proof see [JD11, Section 4.2.2]. The lemma is illustrated in Figure 4.1. A result of Lemma 4.2.1 is that given p, q, the cost $C_{p,q}$ of all optimal strategies of size n is given by

$$C_{p,q}(n) = \min_{1 \leqslant i \leqslant n-1} \left(C_{p,q}(n-i) + C_{p,q}(i) + ip + (n-i)q \right)$$

which gives an easy $\mathcal{O}(n^2)$ algorithm for calculating the optimal strategy of size n. Note that the run time of this algorithm is not important as the output can be pre-computed or even hard-coded into the implementation. The algorithm is given in Algorithm 4.2

Using the output of an optimal strategy calculation, we upgrade the simple $\mathcal{O}(n^2)$ algorithm given in Algorithm 3.1 to a $\mathcal{O}(n \log n)$ algorithm. See Section 4.3.3 for details.

4.3 Implementation

In this section, we cover all essential parts of the implementation provided in the package `optimizedImplementation`. All arithmetic happens in $\mathbb{F}_{p^2}$, which is implemented in `fpsquare.jl` as explained in Section 4.1.1.

Algorithm 4.2 Compute optimal strategy of size n

Input: Costs p, q, a size $n \in \mathbb{N}^+$ with $n \geqslant 2$
Output: Optimal strategy S of size n

```
S ← [1 : ϵ]
C = [1 : 0]
for k ← 2, n do
    b = arg min_{1⩽i⩽k−1} (C_{p,q}(k − i) + C_{p,q}(i) + ip + (k − i)q)
    S[k] ← [b] || S[k − b] || S[b]          ▷ The sign || concatenates the arrays
    C[k] ← C_{p,q}(k − b) + C_{p,q}(b) + bp + (k − b)q
end for
return S[n]
```

4.3.1 Generating Parameters

In total, we provide 5 different sets of parameters. The first set, given in `p132.jl` is not considered to be cryptographically secure, but is rather meant as an easy-to-handle example set of parameters. Further, we provide the parameter sets `p434.jl`, `p503.jl`, `p610.jl` and `p751.jl`, corresponding to security estimates of 128, 152, 189 and 256 bits respectively, as argued in [JAC+20]. For more details on the security levels of these parameters, we refer to the literature [ACC+19, CLN+20, JS19]. Note that while the referenced literature is all on the security of SIKE, the underlying hard problem of SIKE is the same as for SIDH.

Each parameter set contains the following parameters.

The prime p, the field $\mathbb{F}_{p^2}$: Defined parameters $\ell_A = 2$, e_A, $\ell_B = 3$, e_B, f, such that $p = \ell_A^{e_A}\ell_B^{e_B}f - 1$ is a prime number with $p \equiv 3 \pmod 4$ Additionally, it instantiates the field $\mathbb{F}_{p^2}$.

A non-trivial cuberoot of one ω: The parameter ω, with $\omega \neq 1$, $\omega^3 = 1$.

Curve parameters a and d Curve parameters $a = 1$ and d such that the twisted Hessian curve $E : aX^3+Y^3+Z^3 = dXYZ$ is supersingular with j-invariant $\neq 0$.

The points $\{P_A, Q_A\}$ and $\{P_B, Q_B\}$: The torsion basis $\{P_A, Q_A\}$, defined such that $\langle P_A, Q_A\rangle = E[\ell_A^{e_A}]$, and the torsion basis $\{P_A, Q_A\}$, defined such that $\langle P_B, Q_B\rangle = E[\ell_B^{e_B}]$.

Optimal strategies S_A and S_B: Optimal strategies for both $\ell_A^{e_A}$-isogenies and $\ell_B^{e_B}$-isogenies computed by Algorithm 4.2.

All parameter sets start by creating a prime of bit-length as stated in the name of the file, on the form $2^{e_A}3^{e_B}f - 1$, where $2^{e_A} \approx 3^{e_B}$ and f is as small as possible.

We fix our field to be $\mathbb{F}_{p^2} \cong \mathbb{F}_p[X]/\langle X^2+1\rangle$. Further, it precomputes a non-trivial cube root of one, i.e. $\omega \neq 1$ with $\omega^3 = 1$. The reason for this will become clear in Section 4.3.3

Next, we find the starting curve. We select some random supersingular j-invariant not equal to 0. This is obtained by selecting the supersinular twisted Hessian curve $E_0 : X^3 + Y^3 + Z^3 = 0$, which clearly has j-invariant 0 and further has $\#E_0(\mathbb{F}_{p^2}) = (2^{e_A}3^{e_B}f)^2$. From E_0, we do a random isogeny-walk (By simply taking random isogenies). The resulting curve is by Theorem 3.1.16 supersingular, so we use its j-invariant (In the miniscule chance that j-invariant still is equal to 0, simply take a new random walk). From this, we simply use Proposition 4.1.2, with the selected j-invariant and $a = 1$ and solve for d. If the equation has no solutions in $\mathbb{F}_{p^2}$, we select a new j-invariant, until a d is found, and a nice starting curve $E : X^3 + Y^3 + Z^3 = dXYZ$ is selected.

Once we have a curve, we generate a torsion basis for $E[2^{e_A}]$ and $E[3^{e_B}]$. Start by generating random points P and Q on E, and calculate $P_A = [3^{e_B}f]P$ and $Q_A = [3^{e_B}f]Q$. By Lagrange's theorem (2.1.5), both P_A and Q_A has order dividing 2^{e_A}, so by checking that $[2^{e_A-1}]P_A \neq \mathcal{O}$, $[2^{e_A-1}]Q_A \neq \mathcal{O}$, we ensure that their orders are equal to 2^{e_A} (if they are not, then select new P, Q). Finally, we must ensure that the points are linearly independent, so that they form a basis for $E[2^{e_A}] \cong \mathbb{Z}/2^{e_A}\mathbb{Z} \times \mathbb{Z}/2^{e_A}\mathbb{Z}$. This can be done by using a bilinear pairing, for instance the Weil-pairing [Sil09, Chapter III, §8], as described in the original SIDH paper [JD11, Chapter 4.1]. Another easy check, not involving any pairings, is simply to check that $[2^{e_A-1}]P_A$ and $[2^{e_A-1}]Q_A$ are linearly independent[2]. Since the points in this case have order 2, this consists simply of checking that they are not equal (for the points of order 3 case, one must simply check that the points are not equal, and that $[2][3^{e_3-1}]P_B \neq [3^{e_B-1}]Q_B$). Once a basis for $E[2^{e_A}]$ is found, we repeat the proceedure for $E[3^{e_B}]$ (of course instead calculating $P_B = [2^{e_A}f]P$ and $Q_B = [2^{e_A}f]Q$).

Finally, the last part of the parameter generation is to select some weights p, q corresponding to the cost of doubling a point and evaluating a 2-isogeny respectively, and run Algorithm 4.2 with $n = e_A$, to find the optimal strategy to be used in the 2^{e_A}-isogeny calculation. Then repeat the procedure with $n = e_B$, where the weights p, q correspond to the cost of tripling a point and evaluating a 3-isogeny respectively, to find the optimal strategy to be used in the 3^{e_B}-isogeny calculation

[2]This approach was actually pointed out to us by De Feo at a public forum [29].

4.3.2 Computing Secret Generator

We describe the procedure for Alice who works in the group $E[2^{e_A}]$. Of course, Bob does the same but replacing the group with $E[3^{e_B}]$.

Generating the secret generator amounts to sampling random $n_1, n_2 \in \mathbb{Z}_{2^{e_A}}$, with either $n_1 \nmid 3$ or $n_2 \nmid 3$, and then computing $R_A = [n_1]P_A + [n_2]Q_A$. Of course, the secret is really $\langle R_A \rangle$, and not just R_A itself, so we should note that any generator of $\langle R_A \rangle$ is the same secret. Then, assuming that $n_1 \nmid 3$, we could have just as well[3] have calculated the secret $\langle P_A + [n_2 \cdot n_1^{-1}]Q_A \rangle$, so this justifies simply sampling a single $n \in \mathbb{Z}_{2^{e_A}}$, and calculating $R_A = P_A + [n]Q_A$.

Therefore, we need to implement a generic addition procedure and a procedure for scalar multiplication. We implement addition based on the rotated addition law (see Theorem 3.2.6). Note that it is only almost complete since $a = 1$ is of course a cube, but because of the cryptographic size of the parameters, the algorithm will fail with only negligible probability.

We use the procedure for addition given by Bernstein et al. [BCKL15], which requires 11 multiplications (note that here we take advantage of the fact that $a = 1$ when calculating the cost). The algorithm is given in Algorithm 4.3.

Algorithm 4.3 Add Points

Input: Points $P, Q \in E$ for some twisted Hessian curve E with parameters a and d. $P = (X_1 : Y_1 : Z_1), Q = (X_2 : Y_2 : Z_2)$
Output: $P + Q$

1: $A \leftarrow X_1 Z_2$
2: $B \leftarrow Z_1 Z_2$
3: $C \leftarrow Y_1 X_2$
4: $D \leftarrow Y_1 Y_2$
5: $E \leftarrow Z_1 Y_2$
6: $F \leftarrow a X_1 X_2$
7: $G \leftarrow (D + B)(A - C)$
8: $H \leftarrow (D - B)(A + C)$
9: $J \leftarrow (D + F)(A - E)$
10: $K \leftarrow (D - F)(A + E)$
11: $X \leftarrow G - H$
12: $Y \leftarrow K - J$
13: $L \leftarrow B - F$
14: $Z \leftarrow J + K - G - H - (L + L)(C + E)$
15: **return** $(X : Y : Z)$

[3] Although note that now the number of choices of secrets is now reduced from $3 \cdot 2^{e_A - 1}$ to 2^{e_A}.

The same algorithm can be used for doubling a point (note that the rotated addition law never fails when doubling a point, regardless of a), but again Bernstein et al. showed that it was possible to do better with a dedicated doubling formula, requiring only 7 multiplications (or 6 when $a = 1$) and 3 squarings[4]. We implement this procedure, as shown in Algorithm 4.4.

Algorithm 4.4 Double Point

Input: A point $P \in E$ for some twisted Hessian curve E with parameters a and d. $P = (X_1 : Y_1 : Z_1)$
Output: $[2]P$
1: $A \leftarrow X_1^2 X_1$
2: $B \leftarrow Y_1^2 Y_1$
3: $C \leftarrow Z_1^2 Z_1$
4: $D \leftarrow aA$
5: $X \leftarrow (C - B)X_1$
6: $Y \leftarrow (B - D)Z_1$
7: $Z \leftarrow (D - C)Y_1$
8: **return** $(X : Y : Z)$

Scalar multiplication is now easily done by using the additive version of Algorithm 4.1, where point addition is calculated using Algorithm 4.3, and doubling is calculated using Algorithm 4.4.

4.3.3 Computing Isogeny

As this computation is slightly different for the 2^{e_A}-degree isogenies and 3^{e_B}-degree isogenies, we describe them both in this section, starting with the 2^{e_A}-degree isogenies.

The implementation is based on the discussion in Section 4.1.3 and Section 4.1.4, in addition to Section 4.2.2. When we evaluate a point P in an isogeny of degree 2 with kernel given by $R = (U : V : V)$, the values U^2, UV, V^2 stay the same regardless of P. Therefore, we can pre-computing these values to save 3 multiplications when evaluating a point. We give the algorithm in Algorithm 4.5, which uses 15 multiplications for evaluating a point.

Next, we combine Algorithm 4.4 and 4.5, together with an optimal strategy to give a fast calculation of 2^{e_A}-degree isogenies. Given a curve E, a kernel point $R_A \in E$ of order 2^{e_A}, and a set of points $\mathcal{P} = \{P_1, P_2, \ldots, P_k \mid P_i \in E, \text{for } 1 \leqslant i \leqslant k\}$ where at least one point does not have order dividing 3, algorithm finds $E_A = E/\langle R_A \rangle$ and the set $\{\phi(P_1), \phi(P_2), \ldots, \phi(P_k) \mid \phi : E \to E_A\}$. The algorithm uses a datastructure

[4]Bernstein et al. also gave an even faster doubling formula, requiring 7 multiplications and only 2 squarings, however, this formula is not independent of d, which is a feature we require as discussed in Section 4.1.3.

Algorithm 4.5 Evaluate isogeny of degree 2

Input: A point $P \in E$, and precomputed values U^2, UV, V^2 where U, V are coordinates of the kernel point $R = (U : V : V)$. $P = (X_1, Y_1, Z_1)$
Output: $\phi(P)$, where $\phi : E \to E/\langle R \rangle$

$A \leftarrow X_1 X_1$
$B \leftarrow Y_1 Y_1$
$C \leftarrow Z_1 Z_1$
$D \leftarrow X_1 Y_1$
$E \leftarrow X_1 Z_1$
$F \leftarrow Y_1 Z_1$
$X \leftarrow X_1 \left(A \left(V^2\right) - \left(U^2\right) E\right)$
$Y \leftarrow Y_1 \left(C(UV) - \left(V^2\right) D\right)$
$Z \leftarrow Z_1 \left(B(UV) - \left(V^2\right) A\right)$
return $(X : Y : Z)$

called a `dequeue` (double-ended queue), which is a stack with an additional function, `PopFirst`, which returns and removes the bottom of the stack. The logic of the algorithm is simply to traverse the isogeny-calculation tree as given by the optimal strategy. It also uses `precompute2`, which given a point $K = (a : b : b)$, returns the values U^2, UV, V^2. The pseudo-code is given in Algorithm 4.6.

Notice that as argued, we need an arbitrary point P, with $[3]P \neq \mathcal{O}$ to recover the curve parameter d given an a. In the case that the input list $\mathcal{P} = \{P_1, P_2, \ldots, P_k \mid P_i \in E, \text{for } 1 \leqslant i \leqslant k\}$ does not contain such a point, Algorithm 4.6 fails. However, this is easy to fix. We can, as argued in Section 4.1.4, use the kernel point on the second last curve, to recover d one step earlier. This slightly altered algorithm is also provided in the implementation.

Next, we move on to describing how the implementation works for calculating isogenies of degree 3^{e_B}. Logically the algorithms for calculating isogenies of degree 3^{e_B} is the same as Algorithm 4.6, except relying on triplings, and degree 3-isogenies. We now go through procedures for tripling points and evaluating degree 3-isogenies, but leave it to the reader to alter Algorithm 4.6 to fit the degree 3^{e_B} calculation. Alternatively, it is provided in the implementation. The main reason that we choose not to discuss it in detail, is that we discuss a completely different approach to the 3^{e_B}-isogeny calculation in Chapter 5.

Tripling a point P can be done in $11+6 = 17$ multiplications and 3 squarings using Algorithm 4.4 to find $[2]P$, followed by Algorithm 4.3 which gives $[2]P + P = [3]P$. But again, Bernstein et al. showed that triplings could be done more efficiently, assuming that $d \neq 0$. While the requirement $d \neq 0$ is not ideal in the isogeny-case as we jump from curve to curve, we argue that this is not much of a problem. The

Algorithm 4.6 Evaluate isogeny of degree 2^{e_B}

Input: the parameter a of a twisted Hessian curve E, A point $R \in E$ of order 2^{e_A}, a set of points $\mathcal{P} = \{P_1, P_2, \ldots, P_k \mid P_i \in E, \text{for } 1 \leqslant i \leqslant k\}$ containing at least one point not in $E[3]$, and an optimal strategy S of size e_A

Output: parameters a, d of the curve $E_A = E/\langle R \rangle$ and the set $\{\phi(P_1), \phi(P_2), \ldots, \phi(P_k) \mid \phi : E \to E_A\}$

$D \leftarrow$ empty `dequeue`
`Push`$(D, (e_A, R))$
$i \leftarrow 1$
while D not empty **do**
    $(h, K_i) \leftarrow$ `Pop`(D)
    if h $= 1$ **then**
        $a \leftarrow a^2$
        $U^2, UV, V^2 \leftarrow$ `precompute2`(K_i)
        $D' \leftarrow$ empty `dequeue`
        while D not empty **do**
            $(h, K_i) \leftarrow$ `Popfirst`(D)
            $L_i \leftarrow$ `iso2eval`$(K_i, U^2, UV, V^2))$
            `Push`$(D', (h - i, L_i)$ ▷ alg. 4.5
        end while
        $D \leftarrow D'$
        $\mathcal{P} \leftarrow \{$`iso2eval`$(P, U^2, UV, V^2)) \mid P \in \mathcal{P}\}$
    else
        $s_i \leftarrow S[i]$
        `Push`$(D, (h, K_i))$
        `Push`$(D, (h - s_i, [2^{s_i}]K_i))$ ▷ Use alg. 4.4 s_i times
    end if
end while
$P \xleftarrow{\$} \mathcal{P}$
while $[3]P = \mathcal{O}$ **do**
    $P \xleftarrow{\$} \mathcal{P}$
end while
$d \leftarrow$ `recover_d`(a, P) ▷ See Section 4.1.4
return $a, d, \mathcal{P}$

reader need only note that the only curves with $d = 0$ are the ones with j-invariant 0. This is precisely why we did not want to start with j-invariant 0. By starting with a random supersingular j-invariant, hitting a curve with j-invariant 0 during an isogeny-calculation happens only with negligible probability. The tripling algorithm is given in Algorithm 4.7. It triples a point P on a curve E with $d \neq 0$ in 11 multiplications (10 if $a = 1$) and 6 squarings.

Algorithm 4.7 Triple Point

Input: A point $P \in E$ for some twisted Hessian curve E with parameters a and $d \neq 0$. $P = (X_1 : Y_1 : Z_1)$
Output: $[3]P$
1: $W \leftarrow X_1^2 X_1$
2: $V \leftarrow Y_1^2 Y_1$
3: $S \leftarrow Z_1^2 Z_1$
4: $R \leftarrow aA$
5: $A \leftarrow (R - V)^2$
6: $B \leftarrow (R - S)^2$
7: $C \leftarrow (V - S)^2$
8: $D \leftarrow (A + C)$
9: $E \leftarrow (A + B)$
10: $F \leftarrow RC$
11: $G \leftarrow VB$
12: $X \leftarrow (R + V + S)(B + D)$
13: $Y \leftarrow d(F + F - V(C - E))$
14: $Z \leftarrow d(G + G - R(B - D))$
15: **return** $(X : Y : Z)$

Recall, Theorem 3.2.9 and Theorem 3.2.10, which together cover all cases for isogenies of degree 3. Notice that given a kernel point $R = (X : Y : Z)$ of order 3, we can easily check which theorem we should use by checking whether $X = 0$. An algorithm for case 2 is given in Algorithm 4.8. Using a precomputed non-trivial cuberoot of one ω, it requires 6 multiplications (5 if $a = 1$), 3 squarings, and 4 multiplications by ω.

Next, an algorithm for case 3 is given in Algorithm 4.9. Recall that these isogenies are generated by a point $R = (1 : -c : 0)$ or $R = (1 : 0 : -c)$, with $c^3 = a$. However, the point is using projective coordinates, so we might not know what c is (the point is only on the form $(b : -bc : 0)$ or $(b : 0 : -bc)$ for some $b \in \mathbb{F}_{p^2}$). Therefore, to avoid inversions, we change the formula in Theorem 3.2.10, by scaling it with b^2. We precompute the values b^2, cb^2, c^2b^2 and give them as input to the algorithm. The algorithm uses 11 multiplications and 3 squarings after precomputation.

Finally, we have all elements required for the 3^{e_B}-degree isogeny calculation. As

Algorithm 4.8 Evaluate isogeny of degree 3, case 2

Input: A point $P \in E$, the curve parameter a of E, and an element $\omega \in \mathbb{F}_{p^2}$ with $\omega^3 = 1, \omega \neq 1$. $P = (X_1 : Y_1 : Z_1)$
Output: $\phi(P)$, where $\phi : E \to E/\langle(0 : -\omega : 1)\rangle$

$A \leftarrow X_1^2 X_1$
$B \leftarrow Y_1^2 Y_1$
$C \leftarrow Z_1^2 Z_1$
$D \leftarrow aA$
$X \leftarrow X_1 Y_1 Z_1$
$Y \leftarrow D + \omega(\omega B + C)$
$Z \leftarrow D + \omega(B + \omega C)$
return $(X : Y : Z)$

Algorithm 4.9 Evaluate isogeny of degree 3, case 3

Input: A point $P \in E$, where E is some curve with parameters a, d and precomputed elements c^2b^2, cb^2, b^2, cb, b, where $c^3 = a$ and $b \in \mathbb{F}_{p^2}$
Output: $\phi(P)$, where $\phi : E \to E/\langle(1 : -c : 0)\rangle$

$A \leftarrow X_1^2$
$B \leftarrow Y_1^2$
$C \leftarrow Z_1^2$
$D \leftarrow Y_1 Z_1$
$E \leftarrow \left(b^2\right) D$
$F \leftarrow \left(c^2b^2\right) A$
$G \leftarrow \left(cb^2\right) X_1$
$X \leftarrow EX_1$
$Y \leftarrow FZ_1 + GB + EZ_1$
$Z \leftarrow FY_1 + GC + EY_1$
return $(X : Y : Z)$

mentioned, the algorithm runs like Algorithm 4.6, with a few minor modifications. The modifications include using the tripling formula and degree 3 isogeny formulae instead of the doubling and degree 2 isogeny formulae, and the fact that we calculate a and d in every step as given in Theorem 3.2.9 and Theorem 3.2.10 (in other words, there is no need to recover d at the end).

4.3.4 Key Exchange

With all the hard work in Sections 4.3.1 - 4.3.3, the SIDH key exchange itself becomes easy to summarize, which we do in this final section of this chapter. We describe Alice's calculation (who does degree 2^{e_A}-isogenies). Of course, Bob's calculation looks exactly the same, only using corresponding $E[3^{e_B}]$-algorithms. An example can be found in Section 6.1.1.

Setup: All parameters are set up as described in Section 4.3.1. These parameters are persistent and are generated before the key exchange takes place. Typically, they will be hard-coded into the implementation.

Computing Stage I: Once Alice and Bob wish to initialize the key exchange, Alice samples a random $n \in \mathbb{Z}_2^{e_A}$, calculates her secret $R_A = P_A + [n]Q_A$ using addition and scalar multiplication algorithms described in Section 4.3.2. Next, she runs Algorithm 4.6 with R_A as generator, her optimal strategy S_A and the set of points $\mathcal{P} = \{P_B, Q_B\}$ to be consisting of Bob's basis points evaluated in the isogeny. She then sends Bob the output consisting of curve-parameters a_A, d_A and Bob's new torsion basis $\phi_A(P_B), \phi_A(Q_B)$.

Computing Stage II: Once Alice receives the output of Bob's stage I calculation, say a_B, d_B, P'_A, Q'_A, she runs Algorithm 4.6 again, but this time with the secret as $R'_A = P'_A + [n]Q'_A$, and $\mathcal{P} = \varnothing$. She uses the output parameters a, d to calculate the j-invariant using the formula given in Proposition 4.1.2, which is the shared secret.

Chapter 5

Computing the 3^n-Isogeny Without an Explicit Generator

In this section, we discuss a new approach to calculating in the 3^{e_B}-torsion using the particularly simple structure of points of order 3 on a twisted Hessian curve. The new approach greatly simplifies computing 3^{e_B}-isogenies without a $\mathbb{F}_{p^2}$-rational 3^{e_B}-torsion group. Although ultimately, we fail to make the key exchange work by using this approach, we show that there is only one step missing. This is discussed more thoroughly in Section 5.2.2. Empirically, the implementation also supports this claim.

For the whole chapter, we assume that $\ell_A = 2$, $\ell_B = 3$. Further, when referring to Alice, we mean the participant working in the 2^{e_A}-torsion group and when referring to Bob, we mean the participant working in the 3^{e_B}-torsion group.

5.1 Foundations

The alternate approach to computing 3^{e_B}-degree isogenies we present in this section heavily relies on the structure of points of order 3 on twisted Hessian curves. We start by describing an easy way to calculate cube roots in multiplicative groups of a certain order, before discussing points of order 3. Proposition 5.1.1 and Proposition 5.1.2 are essential to this chapter, as they provide the basis for the alternative calculation.

5.1.1 Cube Roots in Finite Fields

The difficulty of taking cube roots in a finite field $\mathbb{F}_q$ is dependent on the order of the field. This is analogous to the situation for square roots. Recall that if $q \equiv 3 \pmod 4$, we can easily calculate the square root of $a \in \mathbb{F}_q$. If a is a square, then generalizing Eulers criterion to cyclic groups says that $a^{\frac{q-1}{2}} = 1$. If this is the case, we can find a square root of a by calculating $b = a^{\frac{q+1}{4}}$. To see that this works, notice that

$$b^2 = \left(a^{\frac{q+1}{4}}\right)^2 = a^{\frac{q+1}{2}} = a^{\frac{q-1}{2}} a = a.$$

This exponentiation takes $\mathcal{O}(\log q)$ time by using Algorithm 4.1. However, this approach fails for $q \not\equiv 3 \pmod 4$, in which case one must use more complicated algorithms, such as the Tonelli-Shanks algorithm [Sha73].

We show similar techniques for cube roots. While e.g. the Tonelli-Shanks algorithm generalizes to n-th roots [AMM77], we would like a simpler way. By generalizing Eulers criterion, we find that a is a cube if and only if $a^{\frac{q-1}{3}} = 1$ (any cube root of a must have order dividing $q-1$, and further $(\mathbb{F}_q)^*$ is cyclic, so this is clear). Taking a similar approach as we did for square roots, it is easy to see that if $q \equiv 7 \pmod 9$, then a cube root of a can be obtained by calculating $c = a^{\frac{q+2}{9}}$, because

$$c^3 = \left(a^{\frac{q+2}{9}}\right)^3 = a^{\frac{q+2}{3}} = a^{\frac{q-1}{3}} a = a.$$

Of course, keep in mind that this approach requires $q + 2 \mid 9$, so it only works for $q \equiv 7 \pmod 9$.

5.1.2 Subgroups of Order 3

From this point forward, we assume that $\mathbb{F}_q$ is a field with $q \equiv 7 \pmod 9$, and for any cube $a \in \mathbb{F}_q$, we set $c = a^{\frac{p+2}{9}}$. Recall the fact that for a twisted Hessian curve $E/K : aX^3 + Y^3 + Z^3 = dXYZ$, there exist four different subgroups of order 3 (Section 3.2.2). If we now fix an $\omega \neq 1, \omega^3 = 1$, we can index the subgroups as:

- $\mathcal{S}_1^E = \{(0 : -\omega : 1), (0 : -\omega^2 : 1), (0 : -1 : 1)\}$,
- $\mathcal{S}_2^E = \{(1 : -c : 0), (1 : 0 : -c), (0 : -1 : 1)\}$,
- $\mathcal{S}_3^E = \{(1 : -\omega c : 0), (1 : 0 : -\omega c), (0 : -1 : 1)\}$,
- $\mathcal{S}_4^E = \{(1 : -\omega^2 c : 0), (1 : 0 : -\omega^2 c), (0 : -1 : 1)\}$.

We prove two propositions regarding these subgroups and degree 2 and 3 isogenies. We start by saying something about these subgroups under degree 3 isogenies.

Proposition 5.1.1. *Let $\phi : E_1 \to E_2$ be one of the two isogenies from Theorem 3.2.9 and 3.2.10, and let $P \in E_1$. Then $\phi(P) \in \mathcal{S}_1^{E_2}$ if and only if P is a point of order 3.*

Proof. If $P \in \ker \phi$, we are done. So assume $P \notin \ker \phi$, and assume $\phi(P) \in \mathcal{S}_1^{E_2}$. Then, since $P \notin \ker \phi$, we have $\phi(P) \in \mathcal{S}_1^{E_2} \setminus \{\mathcal{O}\}$, which means that $X(\phi(P))$ (the X-coordinate of $\phi(P)$) is 0. By looking at the formulas in Theorem 3.2.9 and 3.2.10, we see that this implies that $XYZ = 0$, which again implies that P has order 3 by Theorem 3.2.11. Conversely, if P has order 3, then $X(\phi(P)) = 0$, which from the classification of points of order 3 shows that $\phi(P) \in \mathcal{S}_1^{E_2}$. □

Next, we show a proposition regarding the images of the subgroups of order 3 under degree 2 isogenies.

Proposition 5.1.2. *Let $\phi : E_1 \to E_2$ be one an isogeny of degree 2 on the form given in Equation 4.1. Then:*

(i) $\phi\left[\mathcal{S}_1^{E_1}\right] = \mathcal{S}_1^{E_2}$,

(ii) $\phi\left[\mathcal{S}_2^{E_1}\right] = \mathcal{S}_2^{E_2}$,

(iii) $\phi\left[\mathcal{S}_3^{E_1}\right] = \mathcal{S}_4^{E_2}$,

(iv) $\phi\left[\mathcal{S}_4^{E_1}\right] = \mathcal{S}_3^{E_2}$.

Proof. Since isogenies are group homomorphisms, they preserve subgroups. Further, since the isogeny is of degree 2, no element of order 3 can be in the kernel, and subgroups of order 3 go to subgroups of order 3. Therefore, it is enough to see what happens to a single point $\mathcal{O} \neq P \in \mathcal{S}_i^{E_1}$ for $1 \leqslant i \leqslant 4$ under the isogeny. We simply do this case by case:

(i) $(0 : -w : 1) = P \in \mathcal{S}_1^{E_1}$. Then

$$\phi(P) = \big(0 : -w(ab) : (-w)^2(ab)\big) = \big(0 : -w^2 : 1\big) \in \mathcal{S}_1^{E_2},$$

(ii) $\left(1 : -a^{\frac{q+2}{9}} : 0\right) = P \in \mathcal{S}_2^{E_1}$. Then

$$\phi(P) = \left(b^2 : -\left(a^{\frac{q+2}{9}}\right)^2 b^2 : 0\right) = \left(1 : -\left(a^2\right)^{\frac{q+2}{9}} : 0\right) \in \mathcal{S}_2^{E_2},$$

(iii) $\left(1 : -\omega a^{\frac{q+2}{9}} : 0\right) = P \in \mathcal{S}_3^{E_1}$. Then

$$\phi(P) = \left(b^2 : -\left(\omega a^{\frac{q+2}{9}}\right)^2 b^2 : 0\right) = \left(1 : -\omega^2\left(a^2\right)^{\frac{q+2}{9}} : 0\right) \in \mathcal{S}_4^{E_2},$$

(iv) $\left(1 : -\omega^2 a^{\frac{q+2}{9}} : 0\right) = P \in \mathcal{S}_4^{E_1}$. Then

$$\phi(P) = \left(b^2 : -\left(\omega^2 a^{\frac{q+2}{9}}\right)^2 b^2 : 0\right) = \left(1 : -\omega\left(a^2\right)^{\frac{q+2}{9}} : 0\right) \in \mathcal{S}_4^{E_2},$$

which completes the proof. □

5.2 An Alternative Way of Computing

We start by recalling the 3^{e_B}-torsion calculation as it is usually done. Bob selects a random point R_B of order 3^{e_B}, and sets $R_0 \leftarrow R_B$. First, he calculates the kernel of ϕ_0, which is $\langle [3^{e_B-1}]R_0 \rangle = \mathcal{S}_i$ for some $1 \leqslant i \leqslant 4$. The next step is to calculate the kernel of ϕ_1, which is equal to $\langle \phi_0 \left([3^{e_B-2}]R_0 \right) \rangle$ (even though Bob may calculate it in a different way). This shows that the preimage of the kernel generator is a point of order 9. From Proposition 5.1.1, we know that the kernel of ϕ_1 corresponds to one of the subgroups $\mathcal{S}_i$, $2 \leqslant i \leqslant 4$. Continuing like this, we see that when Bob is done, he has really computed a series of subsets

$$\mathcal{S}_{i_0}^{E_0} \to \mathcal{S}_{i_1}^{E_1} \to \cdots \to \mathcal{S}_{i_{e_B-1}}^{E_{e_B-1}} \tag{5.1}$$

where $i_0 \in \{1, 2, 3, 4\}$, and $i_j \in \{2, 3, 4\}$ for $j \geqslant 1$.

Combinatorically, it is not difficult to see that there exist $4 \cdot 3^{e_B-1}$ different such chains, and since there are exactly $4 \cdot 3^{e_B-1}$ different choices of $\langle R_B \rangle$ for Bob (see Section 3.3.3) which each correspond to a unique chain, we see that there is a nice one-to-one correspondence between chains of the type given in 5.1 and the choices of secret for Bob.

This gives rise to the alternate idea for computing the degree 3^{e_B} isogeny, which uses less elliptic curve arithmetic in exchange for many cube root calculations. Bob can simply pick a random chain of indexes $i_0 \to i_1 \to \cdots \to i_{e_B-1}$, with $i_0 \in \{1, 2, 3, 4\}$, and $i_j \in \{2, 3, 4\}$ for $j \geqslant 1$. In practice, we do not lose much security[1] from just picking all $i_k \in \{2, 3, 4\}$ for $0 \leqslant k \leqslant e_B - 1$. This has the advantage of only needing to rely on Theorem 3.2.10. Bob computes his isogeny as the chain of isogenies corresponding to the chain of subgroups $\mathcal{S}_{i_k}^{E_k}$ for $0 \leqslant k \leqslant e_B - 1$. From the discussion above, we have shown that this is completely equivalent to selecting a random generator point R_B of order 3^{e_B} and doing the usual calculations as explained in Chapter 4.

Relying on Proposition 5.1.2, we use the next section to show some steps towards how Bob can re-do the same isogeny calculation from Alice's curve E_A to complete the key exchange. However, we also discuss the step that is missing in order to complete the scheme. Chapter 6 provides more insight into the potential advantages of this way of calculating, but notice already that Bob no longer does any computation with the 3^{e_B}-torsion basis, and hence does no longer require $E[3^{e_B}]$ to be $\mathbb{F}_{p^2}$-rational. More discussion on this can be found in Section 6.3.

[1] The reduced security comes from a reduction in the size of the key-space by $\frac{1}{4}$, corresponding to a security loss of 2 bits against exhaustive search and the claw-finding algorithm from Section 3.3.3. To see that it also affects the claw-search, notice that the attacker can ignore 1 out of the 4 branches from E_0.

5.2.1 Recovering the Secret

Set $m = e_B - 1$. Assume that Bob has done his calculation as described in the previous section with the chain $i_0 \to \cdots \to i_m$, while Alice has done her 2^{e_A}-degree isogeny calculation of ψ_A the usual way, using the secret $R_A \in E_0$. Set $\psi_A^{(0)} = \psi_A$. Then we say that $\psi_A^{(n)}$ is the isogeny with $[2^{e_A-n-1}](\phi_{n-1} \circ \cdots \circ \phi_0)(R_A)$ as kernel. For Bob to recalculate his part of the key exchange from Alice's curve E_A, he has to find all subgroups of the form $\psi_A^{(k)}\left[\mathcal{S}_{i_k}^{E_k}\right]$ for $0 \leqslant k \leqslant m$. This is illustrated in Figure 5.1. Keep in mind that Alice only actually calculates $\psi_A^{(0)}$ and $\psi_A^{(e_B)}$.

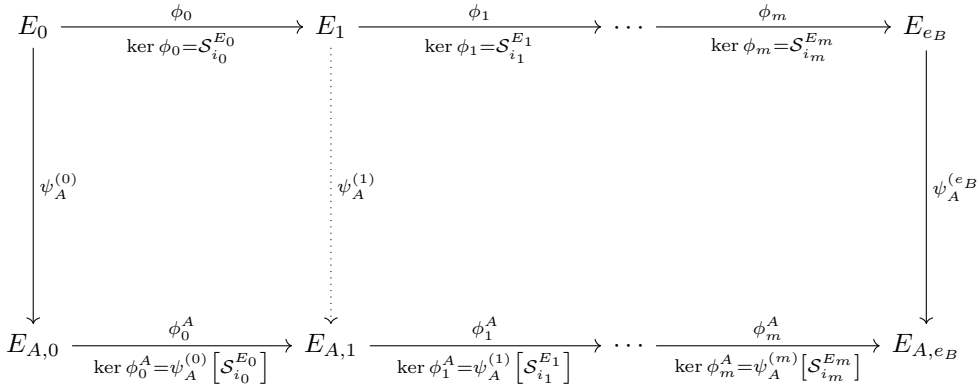


Figure 5.1: The structure of the required computation.

By Proposition 5.1.2, Bob can easily find the first step, $\psi_A^{(0)}\left[\mathcal{S}_{i_0}^{E_0}\right]$. By repeatedly using Proposition 5.1.2 e_A times, we see that if $e_A \equiv 0 \pmod 2$, then we simply have $\mathcal{S}_{i_0}^{E_A} = \psi_A^{(0)}\left[\mathcal{S}_{i_0}^{E_0}\right]$. On the other hand, if $e_A \equiv 1 \pmod 2$, we must simply swap index 3 and 4, while leaving 1 and 2. This show that the key exchange actually works in the trivial case $e_B = 1$. However, the caveat is that the final curve Alice and Bob each arrive at need not be the exact same curve, even though they are isomorphic and have the same j-invariant.

We now show why this is important. Assume that in the trivial case $e_B = 1$, Bob and Alice can reach the exact same final curve by using the above approach. Then the same approach also works for any $e_B \in \mathbb{N}$. This can easily be proven by induction.

Base case: By assumption this holds for $e_B = 1$, i.e. $\phi_0^A \circ \psi_A^{(0)} = \psi_A^{(1)} \circ \phi_0$.

Induction step: Assume this for $e_B = n$, i.e.

$$(\phi_{n-1}^A \circ \cdots \circ \phi_0^A) \circ \psi_A^{(0)} = \psi_A^{(n)} \circ (\phi_{n-1} \circ \cdots \circ \phi_0).$$

Then for $e_B = n+1$ we can show that

$$\begin{aligned}
(\phi_n^A \circ \phi_{n-1}^A \circ \cdots \circ \phi_0^A) \circ \psi_A^{(0)} &= \psi_A^{(n+1)} \circ (\phi_n \circ \phi_{n-1} \circ \cdots \circ \phi_0),\\
\phi_n^A \circ ((\phi_{n-1}^A \circ \cdots \circ \phi_0^A) \circ \psi_A^{(0)}) &= \psi_A^{(n+1)} \circ (\phi_n \circ \phi_{n-1} \circ \cdots \circ \phi_0),\\
\phi_n^A \circ (\psi_A^{(n)} \circ (\phi_{n-1} \circ \cdots \circ \phi_0)) &= \psi_A^{(n+1)} \circ (\phi_n \circ \phi_{n-1} \circ \cdots \circ \phi_0),\\
\phi_n^A \circ \psi_A^{(n)} \circ (\phi_{n-1} \circ \cdots \circ \phi_0) &= \psi_A^{(n+1)} \circ \phi_n \circ (\phi_{n-1} \circ \cdots \circ \phi_0),\\
\phi_n^A \circ \psi_A^{(n)} &= \psi_A^{(n+1)} \circ \phi_n
\end{aligned}$$

which follow from the base case.

We argue that achieving this equality in the base case should not be unrealistic; by using curves in Weierstrass form together with Vélu's formula, one already gets it for free [Leo20]. However, as we discuss in the next section, the closest we get using twisted Hessian curves is within a factor of ω.

5.2.2 The Missing Piece

Recall that all twisted Hessian curves $H(a,d)$ are isomorphic to a Hessian curve of the form $H\left(1, \frac{d}{\sqrt[3]{a}}\right)$. If $\phi : H_1 \to H_2$ is an isogeny of degree 3 as given in Theorem 3.2.9 or Theorem 3.2.10, it is not entirely clear that even if all four $\mathcal{S}_i^{H_1}$ are $\mathbb{F}_{p^2}$-rational, then so should all four $\mathcal{S}_i^{H_2}$ be. But note that this is the same situation as for the usual SIDH calculation. A consequence of this is that the parameter a on the curve H_2 must be a cube, and subsequently, the isomorphism taking H_2 to the corresponding Hessian curve is defined over $\mathbb{F}_{p^2}$. Denote this isomorphism by φ.

Note that φ is not unique, as there exist three different choices for $\sqrt[3]{a}$. However, we conjecture that if E_0 is a Hessian curve, and further by always composing the 3-isogenies with the *correct* isomorphism φ, we get the situation discussed in Section 5.2.1 (i.e. Alice and Bob end up with the exact same curve, and not just isomorphic curves, which is what we need to make the key exchange work). Note that this is not trivial, as there exist up to 12 different values for d for a given j-invariant (and a given value a), and the conjecture says that we can get within 3 different values of d. Finding the right value d is, in other words, a matter of choosing the correct cube root of a.

To show that the key exchange works, up to a matter of choosing the correct cube root of a, we implement the calculation with an oracle. Note that if $c = a^{\frac{q+2}{9}}$, then the all cube roots of a are given by $c\omega^i$ for $i \in \{0,1,2\}$. To restrict the power of the oracle, we only allow it to return the correct number $i \in \{0,1,2\}$ as given above, in order to show that this is the only missing step. We stress that we implement the oracle by using Alice's secret, so this renders the key exchange completely insecure,

but as explained this is only meant to show that if one can somehow select the correct cube root of a in each step, the calculation works.

5.3 Implementation

In this section, we cover all parts of the implementation of the alternative strategy for computation. Because the implementation uses many multiplications by a non-trivial cube root of one, we would like this to be as simple as possible. By implementing the field $\mathbb{F}_{p^2}$ as $\mathbb{F}_p[X]/\langle X^2 + X + 1\rangle$, multiplications by ω become almost trivial.

5.3.1 Computing in $\mathbb{F}_p(\omega)$

We implement the field $\mathbb{F}_p(\omega) \cong \mathbb{F}_p[X]/\langle X^2 + X + 1\rangle$ in addition to $\mathbb{F}_p(i) \cong \mathbb{F}_p[X]/\langle X^2 + 1\rangle$ (which was discussed in Section 4.1.1). The roots of the polynomial $X^2 + X + 1 \in \mathbb{F}_p[X]$ are non-trivial cube roots of 1. It is irreducible if and only if $p \equiv 2 \pmod 3$. To see this, notice that if $p \equiv 2 \pmod 3$, then $|(\mathbb{F}_p)^*| \equiv 1 \pmod 3$, so by Lagrange's theorem (Theorem 2.1.5), no element of order 3 can exist in $\mathbb{F}_p$, and subsequently $\omega \notin \mathbb{F}_p$. Conversely, if $p \equiv 1 \pmod 3$, then for any generator α of $(\mathbb{F}_p)^*$, we have $\left(\alpha^k\right)^3 = \left(\alpha^{2k}\right)^3 = 1$ for $k = \frac{p-1}{3}$, so α^k and α^{2k} are roots of $X^2 + X + 1$. Finally, if $p = 3$, then it is obviously reducible. We fix a root ω of $X^2 + X + 1$, and set $\{1, \omega\}$ as a basis for $\mathbb{F}_p(\omega)$. Notice that $\omega^2 = -1 - \omega$. Now, the field operations become

$$
\begin{aligned}
(a + b\omega) + (c + d\omega) &= (a + c) + (b + d)\omega, \\
(a + b\omega) - (c + d\omega) &= (a - c) + (b - d)\omega, \\
(a + b\omega)(c + d\omega) &= (ac) + (ad + bc)\omega + (bd)\omega^2 = (ac - bd) + (ad + b(c - d))i, \\
(a + bi)^{-1} &= (a^{-1} - b^2(a^3 + a^2b + ab^2)^{-1}) - (ab(a^3 + a^2b + ab^2)^{-1})\omega,
\end{aligned}
$$

where again, operations between elements a, b, d, c happens in $\mathbb{F}_p$.

The main reason we implement $\mathbb{F}_p(\omega)$ for this alternative calculation, is that multiplication by ω happens frequently. The advantage here is that in $\mathbb{F}_p(\omega)$, the multiplication $(a + b\omega) \cdot \omega = -b + (a - b)\omega$ is very fast.

5.3.2 Generating Parameters

The parameter generation is done similarly as in Section 4.3.1, except from a few notable differences which we summarize here. Each parameter set contains the following:

Parameters for the prime p, the field $\mathbb{F}_{p^2}$: Defines parameters e_A and f with $3 \mid f$, such that $p = 2^{e_A} f - 1$ is a prime number with $p \equiv 5 \pmod 9$. Additionally, it defines a parameter e_B, and instantiates the field $\mathbb{F}_{p^2}$.

A non-trivial cube root of one ω: Precomputes $\omega \neq 1$ with $\omega^3 = 1$.

Curve parameter d: Curve parameter d such that the Hessian curve $E : X^3 + Y^3 + Z^3 = dXYZ$ is supersingular with j-invariant $\neq 0$.

The points $\{P_A, Q_A\}$: The single torsion basis $\{P_A, Q_A\}$, with $\langle P_A, Q_A \rangle = E[\ell_A^{e_A}]$.

Optimal strategy S_A: Optimal strategy for computing 2^{e_A}-isogenies obtained by Algorithm 4.2.

We need $3 \mid f$, because we need the whole 3-torsion group to be rational for some supersingular curve $E \cong \mathbb{Z}/\ell_A^{e_A} f\mathbb{Z} \times \mathbb{Z}/\ell_A^{e_A} f\mathbb{Z}$. This is equivalent to saying that we want all $\mathcal{S}_i^E$ groups to be $\mathbb{F}_{p^2}$-rational. Note that this is much less strict than the usual requirement $\ell_B^{e_B} \mid p+1$. Further, we need $\ell_A = 2$ (although typically, we would want to do this in the usual case for Section 4.3.1 as well), because we rely on Alice using Theorem 3.2.8 to calculate her isogenies. Separately, we define the security parameter e_B corresponding to the length of Bob's isogeny-walk. Finally, we require $p \equiv \pm 5 \pmod 9$, to ensure that $q = p^2 \equiv 7 \pmod 9$.

Another difference with Section 4.3.1, is that if the field $\mathbb{F}_{p^2}$ is implemented as $\mathbb{F}_p[X]/\langle X^2 + X + 1\rangle$, then as discussed in Section 5.3.1 the parameter ω is simply $X + \langle X^2 + X + 1\rangle$. Further, notice that this time we must start with a Hessian curve (but again, this has advantages in the usual setting as well). Finally, only Alice needs a torsion basis and an optimal strategy.

5.3.3 Computing Secret Isogeny

We only consider Bob (the participant working in the 3^{e_B}-torsion) for this alternative way of computing, as Alice's part of the key exchange works exactly as in Chapter 4, i.e. she uses Algorithm 4.6, together with an optimal strategy.

Implementing Bob's alternate way of computing actually results in a much easier algorithm than Algorithm 4.6 (or the 3^{e_B} version of it). However, for the second part of the key exchange, we need the oracle as we have argued. Therefore, to be precise we now describe two different algorithms.

To make the first algorithm easily readable, we use two high level functions. The function `IsoFromKer` takes in a kernel of the form $\mathcal{S}_i^E, 2 \geqslant i \geqslant 4$ returns another function ϕ representing to the isogeny with kernel $\mathcal{S}_i^E$. If ϕ takes in a point, it returns the point evaluated in the isogeny, for instance using Algorithm 4.9, while if

it takes in a curve, it returns the image curve with the parameters given in Theorem 3.2.10. We also use the function `Hessian`, which takes in a curve, and returns another function φ representing the isomorphism taking the curve $H(a,d)$ to $H(1,d/(a^{\frac{q+2}{9}}))$. Again, φ can take in a point, and return the point evaluated in the isomorphism (recall that the isomorphism is defined as $\varphi(X:Y:Z) = ((a^{\frac{q+2}{9}})X:Y:Z)$), or it can take in a curve $H(a,d)$, and return the image curve $H(1,d/(a^{\frac{q+2}{9}}))$.

Algorithm 5.1 Alternate computation of 3^{e_B} isogeny - stage I

Input: Alice torsion points $P_A, Q_A \in H_{1,d}$, and precomputed values $\omega \neq 1$, $\omega^3 = 1$, Bob's secret $s_B = (s_0, s_1, \ldots, s_{e_B-1}) \in \{2,3,4\}^{e_B-1}$
Output: Bobs curve E_B, $\phi(P_A), \phi(Q_A)$ where $\phi : E \to E_B$.

$P_0 \leftarrow P_A$
$Q_0 \leftarrow Q_A$
for $i \leftarrow 0, e_B - 1$ **do**
    $\phi_i \leftarrow \texttt{IsoFromKer}(\mathcal{S}_{s_i}^{E_i})$
    $E'_{i+1} \leftarrow \phi(E_i)$
    $\varphi_i \leftarrow \texttt{Hessian}(E'_{i+1})$
    $E_{i+1} \leftarrow \varphi(E'_{i+1})$
    $P_{i+1} \leftarrow \varphi(\phi(P_i))$
    $Q_{i+1} \leftarrow \varphi(\phi(Q_i))$
end for
return $E_{e_B}, P_{e_B}, Q_{e_B}$

For part two of the key exchange, the algorithm becomes even easier, as Bob has no points he needs to evaluate. This time we write it without using the high-level functions. We assume that if Alice's parameter $e_A \equiv 1 \pmod 2$, Bob has changed his secret according to the discussion in 5.2.1. The only special function we use is `S`, which takes in an integer $k \in \{2,3,4\}$, and returns the parameter ω^{k-2} (Recall that all curves E we are dealing with are Hessian curves. Then the output represents the parameter c of the non-identity points in $\mathcal{S}_i^E$). Further, we use the Oracle $\mathcal{O}(d,i)$, which takes in the current step i and current curve parameter d, and returns an integer $k \in \{0,1,2\}$ such that $a^{\frac{q+2}{9}}\omega^k$ is the *correct* cube root (as discussed in Section 5.2.2).

Note that Algorithm 5.2 does not need to query the oracle in the final iteration, since we are only after the j-invariant of the final curve.

While both Algorithm 5.1 and Algorithm 5.2 are easy in the sense that they clearly have linear run time the parameter e_B, and that we completely avoid any elliptic curve additions, it should be noted that they do require an inversion and an exponentiation by $\frac{q+2}{9}$ in each step to find the isomorphic curve on Hessian form. This is further discussed in Section 6.3.3

Algorithm 5.2 Alternate computation of 3^{e_B} isogeny - stage II

Input: Parameter d of Alice's curve $E_A = H(1, d)$, a precomputed value $\omega \neq 1$, $\omega^3 = 1$, Bob's secret $s_B = (s_0, s_1, \ldots, s_{e_B-1}) \in \{2, 3, 4\}^{e_B}$
Output: The shared secret $j(E_{AB})$.

for $i \leftarrow 0, e_B - 2$ **do**
  $c \leftarrow \mathtt{S}(s_i)$
  $a \leftarrow d^2c + 3dc^2 + 9$ ▷ See Theorem 3.2.10
  $d \leftarrow d + 6c$
  $d \leftarrow d\left(a^{\frac{q+2}{9}}\right)^{-1}$ ▷ Isomorphism to Hessian form
  $d \leftarrow d\omega^{\mathcal{O}(d,i)}$ ▷ Needs Oracle to return *correct* cube root
end for
$c \leftarrow \mathtt{S}(s_{e_B-1})$
$a \leftarrow d^2c + 3dc^2 + 9$
$d \leftarrow d + 6c$
$ss \leftarrow \frac{\left(216ad+d^4\right)^3}{a(d^3-27a)^3}$ ▷ See Proposition 4.1.2
return ss

5.3.4 The Oracle

Finally, we explain how we have implemented the oracle. As mentioned, this runs during Bob's calculation while requiring Alice's secret, which of course compromises security. The oracle is simply implemented as part of a proof-of-concept to show that if one can find the correct cube root in each step, the key exchange works.

As discussed in Section 5.2.1, Bob needs to make his degree 3 isogenies commute with Alice's degree 2^{e_A} isogeny. Therefore, we give the oracle the knowledge of which value d Alice would have reached to help Bob succeed in selecting the correct cube root. Note though that as explained, the oracle is limited in power, and does not give Bob the parameter d, but only a $k \in \{0, 1, 2\}$ corresponding to the factor of ω^k that Bob is off. To do this, during stage I, Bob gives the oracle his curve E_i in each step, and the oracle calculates the isogeny $\psi_A^{(i)}$, and then saves the parameter d of the image curve $E_{A,i}$ As mentioned, the oracle does not need to be queried for the final curve E_{A,e_B}, so it does not need to do this calculation either. The oracle's precalculation is illustrated in Figure 5.2.

Then, for stage II, once the oracle has this list of d values, it is trivial to let Bob query the oracle and return the correct $k \in \{0, 1, 2\}$ for the current step, assuming the conjecture from Section 5.2.2 is true.

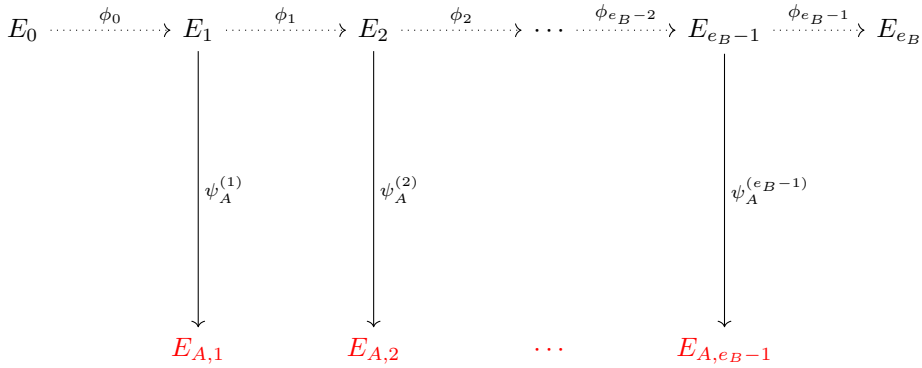


Figure 5.2: The oracle stores all values in red. Note that the oracle clearly needs the knowledge of Alice's secret and can easily recover Bob's secret.

5.3.5 Key Exchange

Finally, we summarize the key exchange for Bob. Alice does exactly the same calculation as in Section 4.3.4, except from the fact that Bob no longer has any torsion basis that Alice needs to evaluate in stage I.

Setup: The parameters are set up as described in Section 4.3.1. Again, these parameters are persistent, and will typically be hard-coded into the implementation.

Computing Stage I: Bob no longer needs to calculate a secret generator point. Instead, he samples a random secret $\boldsymbol{s_B} = \{s_0, \ldots, s_{e_B-1}\} \in \{2,3,4\}^{e_B}$, and runs Algorithm 5.1, with the starting curve $H(1,d)$, Alice's torsion points P_A, Q_A, a precomputed ω and his secret $\boldsymbol{s_B}$. He then sends Alice the output consisting of curve-parameters a_B, d_B and Alice's new torsion basis $\phi_B(P_A), \phi_B(Q_A)$.

Computing Stage II: If $e_A \equiv 1 \pmod 2$, Bob calculates his new secret as

$$\boldsymbol{s'_B} = \{s'_i \mid s'_i = \rho(s'_i)\}$$

where

$$\rho(s_i) = \begin{cases} 2, & \text{if } s_i = 2 \\ 4, & \text{if } s_i = 3 \\ 3, & \text{if } s_i = 4 \end{cases}$$

while if $e_A \equiv 0 \pmod 2$, he simply sets $\boldsymbol{s'_B} = \boldsymbol{s_B}$. Once Bob receives the curve $H(1,d_A)$ from Alice, he runs Algorithm 5.2, with the curve $H(1,d_A)$, ω and his secret $\boldsymbol{s'_B}$, which outputs the shared secret.

Chapter 6
Examples and Discussion

Our work shows that twisted Hessian curves may have certain use cases in isogeny-based cryptography, but that more work is required. This chapter will start by provide examples from the implementations in Chapters 4 and 5. The rest of this chapter will summarize and discuss the advantages and disadvantages of our implementations.

6.1 Examples

We provide three examples calculated by our implementations, one example corresponding to Chapter 4, and two examples from Chapter 5.

6.1.1 Example from Chapter 4

This section contains an example from of the implementation `sidh.jl`, from the folder `optimizedImplementation`, using real-world parameter sizes. The example uses a 434-bit prime, which corresponds to a NIST-security level of 128-bit quantum security. The prime is the same as the corresponding prime in the SIKE specification [JAC+20].

Setup: We set $\ell_A = 2, \ell_B = 3, e_A = 216, e_B = 137$ and $f = 1$. This gives

$$p = 24439423661345221551909145011457493619085780243761596511325807336\\ 20522123933197672597021667182861844589871902669288493934231473356 7.$$

We find a non-trivial cube root of 1 in $\mathbb{F}_p(i) \cong \mathbb{F}_p[X]/\langle X^2 + 1\rangle$ as

$$\omega = 12219711830672610775954572505728746809542890121880798255662903668 1\\ 02610619665988362985108335914309222949359513346442469671157366783+\\ 13538673401042258750337839019119680082373927747576152192914450022 4i.$$

For a starting curve, we start at the supersingular curve $X^3+Y^3+Z^3=0$, and do a random isogeny-walk and end up with a Hessian curve $E: X^3+Y^3+Z^3 = d_0XYZ$ with

$$\begin{aligned}d_0=&219385922741314142691951261586269668302223402600131893711006618643854082018527224312489320097711699701534967947124705531888751377+\\&224554752264866562421933351177644545140296865788315782851609350792\\&779153303820149178052194510470373497511640633640028034483846144471i.\end{aligned}$$

For torsion bases, we use the points

$$\begin{aligned}P_A=(&181207408233620968593030222052887554276296428713272278824059665758\\&479845583875399280522903604318660485707071229291740945171371275721+\\&136762723551604131775126461548117364316956443400400420940500100671\\&764096533880871281201023239972891461297116828047092344226318091191i:\\&123478600471055028680842749773143654697022672527740951724140751861\\&010548010038474279217655164696202927925120988814385880950435500931+\\&207160613791258635181993161078041490632235106712523508148992103074\\&770939177846807974519566806425003408597685104188878930255014815 6i:\\&1+0i),\end{aligned}$$

$$\begin{aligned}Q_A=(&232029288806673682927056418563186300111590954264753734697722404384644974063366001264646120311077938414578368933654163493575384282 2+\\&104691521398866550099463624248283818358983982790576512111173860641\\&007743425947934958618826281630916374869103755067463634360390711091i:\\&709181353567272911256862210952807946696763903242648890611048620 3\\&118822284575301556214707120650315493693128524298007477335563767 66+\\&182003712934905759287093535298471175593731420520631720849517953729\\&825518495809480840017070637188106342851570843320586604857605117781i:\\&1+0i),\end{aligned}$$

$$\begin{aligned}P_B=(&890446169287068080369192387021925192298911211076087534358180974486050544378817634427203876124623598036845702805884072432747621985 6+\\&215777275068529710522576406898606150588129043598312668531758619502\\&673591264565353811516498296216296270693851146557520833058504641031:\\&237187303387687212359319987113091578513521867904498747151814273641\\&717103023469909617300887493875882277177529545552942001510367755 64+\\&243287215067351808871804947676207895309197120677853566404937160063\\&535913830903008850434420964371952451543404848100967380765592817021:\\&1+0i),\end{aligned}$$

$$\begin{aligned}Q_B=(&974610913146544746358186754466353545649158018433472886628273578776\\&067908903888261520608035826780878589010296370114544007532370133 4+\\&893517947633210701220863528795709266686296635762915473523052459472\\&797786640967062395028012998819223764201064022460497136237822330 3i:\\&213671306307462877992580638781033157725450006925180533659006396 84\\&481665937005251343971817809497997946960016096566799444455337338 04+\\&214483602749462757400456514317741798963128061475233166333371378839\\&542651491500759965445967473450819078036592093286223463477255692321:\\&1+0i).\end{aligned}$$

Finally, optimal strategies are precomputed.

Computing Stage 1: Alice and Bob now select their secrets, respectively s_A and s_B, as

$$s_A=120231223570276940140560322631130823525829820896026719501650603 94$$

$$s_B=475825548982873932422545236651188955935624876708823574263933508 53$$

Alice then calculates her secret generator as $R_A = P_A + [s_A]Q_A$, and calculates the isogeny $\phi_A : E \to E_A$ having $\langle R_A \rangle$ as a kernel. This way she finds the curve $E_A : a_A X^3 + Y^3 + Z^3 = d_A XYZ$, with

$$a_A = 1+0i$$

$$\begin{aligned} d_A = {} & 21677449413174358211475694290765981741990790630081883537768562 86 \\ & 77487247332917809222555153604166233123583704701936318982189628180+ \\ & 95949546982634345795971973662264485350406479360339392905470523213 \\ & 92871200461639558024298263162109645974538017134948822298516273927i \end{aligned}$$

Additionally, Alice calculates the image of Bob's torsion basis, which become

$$\begin{aligned} \phi_A(P_B) = {} & (15591743284946458628137282444633719008494321953843071728398580050 6 \\ & 53960010124734245630228005928197961738923959727409646490854268030+ \\ & 11077864045320816338345647293466904468213746361203936029649016007 7 \\ & 21781657630948095524721334682102801646015755713275064230255393790i: \\ & 45404981531611258135438329045534589602048747004497400128260881817 \\ & 23161897333806392490460412514973159921078389165183906416164919027+ \\ & 18843891359190882759141613317274286656432749875945302770022486900 4 \\ & 24286780110756403605663314735856633495717793363678597744149370621i: \\ & 24551545822597449188796641512191162048931922350863750624427119482 \\ & 99200100008751059367296157304841470158856313298136551092889812100+ \\ & 39426679119270019095896791898520060149233587874557773392724999504 \\ & 90655872887563229259581538765208529547073931227091572486819632209i) \end{aligned}$$

$$\begin{aligned} \phi_A(Q_B) = {} & (13826011445955156390571869653129282877542744305581742061253232744 5 \\ & 00847434549393297616705455910757221588567803910422834714835239107+ \\ & 54931354650703174386094391619472066533916742480687095901561820998 \\ & 16771107299814619140929605366429711822949781443762794591760251 57i: \\ & 19399035362983200283866957581583608349655516297791174461522259260 7 \\ & 76230210129304384072547976623839700851711143346424779351606119152+ \\ & 13686969076281678038946173123128315812634497654855697962956854584 6 \\ & 08936935669586808535012559237490196941750801956896471833099083781: \\ & 19089814351358606491581416836027514056140947111397235336502664514 6 \\ & 68773396935456155595807206738179342267959575634350548710281086649+ \\ & 13639236499400571492740173442638018503443695562499120535419196594 2 \\ & 15821014670934218296555473249013327759955863854576872855379564498i) \end{aligned}$$

In a similar fashion, Bob calculates his secret $R_B = P_B + [s_B]Q_B$, and finds $\phi_B : E \to E_B$, $\phi_B(P_A)$ and $\phi_B(Q_A)$. His public curve, $E_B : a_B X^3 + Y^3 + Z^3 = d_B XYZ$, has parameters

$$\begin{aligned} a_B = {} & 45898057719423745658836698742390396836029037692301728878303984781 \\ & 37532797177936931217478298352960196187478613372060195477425211947+ \\ & 71042515039875438207444487828749687464683562097921351054497779218 \\ & 75854403475070182171506673853183473039699478825147082613762290788i \end{aligned}$$

$$\begin{aligned} d_B = {} & 11245758355461968949495643007297501905422932689257956273473797265 4 \\ & 59243565921876092755339423128440199400621759275877525647858880690+ \\ & 19192273145459537806440277098997967579829489020537485868001342193 \\ & 09802788848009081809665659940574973986196433987206522464452697335i \end{aligned}$$

while the image of Alice's torsion basis is

$$\begin{aligned}\phi_B(P_A)=(&11521640146029317006931658756757980511598616342343805399496089401824256800998328624400943933737513097318980391418866457605906927982+\\&21032260251961973399351908852484835462527877688130370495698464056871919190092076660100083898989874649093214688946864318024221578755i:\\&15166269718181857443782979831381052383984854954259948797186130836283225844048676401486150047827433582023303445367786408151699012126+\\&13147956477365648324994510434082950113037181000658427515570615105145971081294138265966601636850104279643059898237976154839296891064i:\\&80746068552703593024426486912826140693124843415109481484953923706623953263362997357206701948284169120395344628098944070614346780344074207139389087843086381767493637440218987555611304971957206495145451728736573309197643443292790592136964685538145402229036664982i)\end{aligned}$$

$$\begin{aligned}\phi_B(Q_A)=(&46528781554807707803681500221758816925744040423135604602085345198\\&41520240500791124308834193182766932628959353924855707032805763897+\\&14898060276256846272710247440334110811261592623756019562168763755614426407493002993142685668716793351301927705843120914675600590680 6i:\\&22454320755137964272367127234451336828259606502392892738660029889155643187872260057843112019389622838066110628434325818464953921325+\\&15369569451783948705909851250663728023138815689597164192464308506678866136998971306376261300441961957090081646464876656921428682783i:\\&35907918011416943988301013025806941387281982053134308944942931303408331214057254577638921341104991367465477150369923463913084029812023296029322756939621302597083482597915791903649702544735715004902987586259041081371068648898050661258158368347832763177119720130i)\end{aligned}$$

Computing stage II: Once Alice receives $E_B, \phi_B(P_A)$ and $\phi_B(Q_A)$, she recovers her kernel as $\phi_B(R_A) = \phi_B(P_A) + [s_A]\phi_B(Q_A)$, and calculates the isogeny $\phi'_A : E_B \to E_{BA}$ as the isogeny having $\phi_B(R_A)$ as kernel. Bob proceeds similarily, and ends up with the curve E_{AB}. Finally, the j-invariant of $E_{AB} \cong E_{BA}$ is

$$\begin{aligned}j(E_{AB})=j(E_{BA})=&50354154695947312077142804149239395066948787462807726978261811683\\&99043208109136611457853596051991205272167857877700544284555708594+\\&12431867558099556481396787863449270848964179362196738339025826899043859091844610425405795351292443426978425410601331479674528451521i\end{aligned}$$

which becomes their shared secret.

This concludes the example from Chapter 4. The example used parameters from `p434.jl`, corresponding to a NIST security level of 128-bit quantum security. We note that the ephemeral parameters, $a_A, a_B, d_A, d_B, \phi_A(P_B), \phi_A(Q_B), \phi_B(P_A)$ and $\phi_B(Q_B)$, may depend on the implementation. However, using the same starting parameters and selecting the same secrets will result in the same shared secret regardless.

6.1.2 Examples from Chapter 5

To help understand the computation from Chapter 5, we first provide a detailed example of the calculation, using tiny parameters, before showing a less detailed example with cryptographically secure parameters.

Setup: We start by selecting a suitable prime. We see that $\ell_A = 2, e_A = 11, f = 3$ gives the prime $p = 2^{11} \cdot 3 - 1 = 6143$, which satisfies the requirements of the prime. Further, we set $e_B = 7$, which gives $3^{e_B} = 2187 \approx 2048 = 2^{e_A}$. Then we instantiate the field as $\mathbb{F}_p(\omega) \cong F[X]/\langle X^2 + X + 1\rangle$. The element $\omega = X + \langle X^2 + X + 1\rangle \in F[X]/\langle X^2 + X + 1\rangle$, satisfies $\omega^3 = 1$. From this point on, all arithmetic happens in $\mathbb{F}_p(\omega)$.

Next, we do a random isogeny-walk from the curve $H(1, 6 + 6\omega)$ (which has j-invariant 0), and end up with the starting curve $H(1, 535 + 1621\omega)$ (which has j-invariant $5736 + 2534\omega$). At this point, Bob has all the parameters he needs, but we must still generate a torsion basis and optimal strategy for Alice.

We simply select two random points on this curve, multiply them by f and assert that the resulting points provide a valid torsion basis for Alice. This gives

$$P_A = (3862 + 22\omega : 394 + 5264\omega : 1),$$
$$Q_A = (1196 + 1524\omega : 3245 + 3484\omega : 1).$$

Finally, we generate the optimal strategy for Alice, with estimated weights $p = 23358$ and $q = 32132$. The optimal strategy is computed by using Algorithm 4.2.

Computing Stage I: Bob samples the random secret $\boldsymbol{s_B} = (2, 3, 2, 4, 3, 2, 4)$. He then does his isogeny-walk, starting at $E_0 = H(1, 535 + 1621\omega)$, and with $P_0 = P_A, Q_0 = Q_A$. Step by step, he finds the isogeny with the kernel given by his secret, composed with the isomorphism taking the curve to Hessian form.

(i) $\phi_0 : E_0 \to E_0/\mathcal{S}_2^{E_0} = E_1$, where $E_1 = H(1, 1053 + 2208\omega)$,
$P_1 = \phi_0(P_0) = (4400 + 4252\omega : 3501 + 698\omega : 3125 + 736\omega)$,
$Q_1 = \phi_0(Q_0) = (4668 + 1685\omega : 4606 + 5754\omega : 46 + 4390\omega)$.

(ii) $\phi_1 : E_1 \to E_1/\mathcal{S}_3^{E_1} = E_2$, where $E_2 = H(1, 943 + 4623\omega)$,
$P_2 = \phi_1(P_1) = (4685 + 1448\omega : 4816 + 47\omega : 1344 + 639\omega)$,
$Q_2 = \phi_1(Q_1) = (2924 + 3718\omega : 1147 + 2533\omega : 3765 + 4179\omega)$.
...

(vii) $\phi_6 : E_6 \to E_6/\mathcal{S}_4^{E_6} = E_7$, where $E_7 = H(1, 4689 + 4720\omega)$,
$P_7 = \phi_6(P_6) = (4668 + 3272\omega : 2527 + 3371\omega : 1083 + 1648\omega)$,
$Q_7 = \phi_6(Q_6) = (742 + 3622\omega : 5414 + 5037\omega : 3586 + 3275\omega)$.

Bob is now done with his calculation, and sends Alice the parameter $d_B = 4689 + 4720\omega$ of his curve, in addition to the image of Alice's torsion basis, P_7, Q_7.

Alice has done her calculation as described in Chapter 4 (except that Bob does not have any torsion basis), with her secret $n = 1859$. She simply sends Bob the parameter of her curve $d_A = 969 + 1389\omega$.

Computing Stage II: Alice does the exact same computation as in stage I, except starting at the curve $H(1, d_A)$, and with torsion basis P_7, Q_7, and ends up with the curve $H(1, 2216 + 364\omega)$. Alice then finds her secret key as the j-invariant of this curve, which is $2643 + 314\omega$. This becomes her shared key.

Bob, however, must change his secret before starting the stage II computation, since $e_A = 11 \equiv 1 \pmod 2$. Bob stores his new secret $\boldsymbol{s'_B} = (2, 4, 2, 3, 4, 2, 3)$. This time, in each step he must query the oracle to figure out which cube root to choose when sending the curve to Hessian form.

(i) Bob finds the isogeny with kernel equal to $\mathcal{S}_2^{E_{A,0}} = \langle (1 : -1 : 0) \rangle$. The image of this isogeny is $H(1579 + 4996\omega, 975 + 1389\omega)$. He queries the oracle and finds that the "correct" cube root is $(1579 + 4996\omega)^k \omega^{\mathcal{O}(\cdot)} = (1579 + 4996\omega)^k \omega^2 = 2715 + 5503\omega$. This gives him the first step
$\phi_0^A : E_{A,0} \to E_{A,1}$, where $E_1 = H(1, (975 + 1389\omega)/(2715 + 5503\omega)) = H(1, 2890 + 3723\omega)$.

(ii) In the next step, the kernel is $\mathcal{S}_4^{E_{A,1}} = \langle (1 : -\omega^2 : 0) \rangle$. The image of this isogeny is the curve $H(3517 + 2002\omega, 2884 + 3717\omega)$. Bob queries the oracle again, and gets $a^k \omega^{\mathcal{O}(\cdot)} = a^k \omega^0 = 3381 + 2025\omega$. This completes step 2 as
$\phi_1^A : E_{A,1} \to E_{A,2}$, where $E_{A,2} = H(1, (2884 + 3717\omega)/(3381 + 2025\omega)) = H(1, 771 + 4641\omega)$

...

(vii) The final isogeny gives $E_{A,7} = H(99 + 4409, 5893 + 4643\omega)$. Unlike in the previous steps, since this is the last curve, Bob does not need to query the oracle to find the corresponding Hessian curve.

Finally, Bob calculates the j-invariant of the curve $H(99 + 4409, 5893 + 4643\omega)$, which gives the shared key $2643 + 314\omega$, same as Alice ended up with.

This concludes the detailed example. In this example, Alice worked in the $E[2^{11}]$-torsion, which was $\mathbb{F}_{p^2}$-rational, while Bob implicitly worked in the $E[3^7]$-torsion, even though it was not $\mathbb{F}_{p^2}$-rational, by using the alternative calculation. From the choices of public-keys, it seems like both Bob and Alice had ≈ 2000 different choices of secrets. However, notice that since $p = 6143$, there only existed $\lfloor \frac{p}{12} \rfloor + 2 = 513$ different supersingular j-invariants. This means that there only existed 513 different possible shared keys for Alice and Bob to end up with. In other words, the security level here was bounded by the number of j-invariants, and not the length of their

isogeny-walks. This is in contrast to traditional SIDH. This will be further discussed in Section 6.3.

Next, we provide an example, using a 256-bit prime. The security level of this prime corresponds to the security level of the 434-bit prime used in Section 6.1.1.

Setup: We set $\ell_A = 2, e_A = 216, e_B = 137$ and $f = 879$. This gives

$$p=92569504376661767107469946333946310008308788857082668792539349254143.$$

Since we use the field $\mathbb{F}_p(\omega) \cong \mathbb{F}_p[X]/\langle X^2 + X + 1\rangle$, precomputing ω is trivial. Next, we find our starting curve. Again, we start at the supersingular curve $X^3 + Y^3 + Z^3 = (6 + 6\omega)XYZ$, and do a random isogeny-walk to end up with a supersingular curve $E : X^3 + Y^3 + Z^3 = d_0XYZ$ with

$$\begin{aligned}d_0=&56651119911200501235685304671022582616310475250193662095773797569 34+\\ &82574505981473371002562942499396087965969775358884069554980223902745\omega.\end{aligned}$$

For Alice's torsion basis, we use the points

$$\begin{aligned}P_A=\big(&80688760787762445901542266177232284214254309177320915381419714336548+\\ &32337205480749456754565353563076240595776808766763190715201279433291\omega:\\ &65352292466146515189292147346535155416026979696475993340802784014906+\\ &14274849928215998628952249590512577131889812320389947503924977641 23\omega:\\ &1+0\omega),\end{aligned}$$

$$\begin{aligned}Q_A=\big(&26933582349313129346992030939025106346144152421552560337036631338648+\\ &78801036083640836099057856713777644273475452420778728247146237412696\omega:\\ &71787261617380092825103444820374042162460157337969907529928505703032+\\ &82198952277648735151608585947204407871020711737536308660905582902 98\omega:\\ &1+0\omega).\end{aligned}$$

Additionally, Alice's optimal strategy is precomputed.

Computing Stage 1: Alice and Bob now select their secrets. Alice selects the secret

$$s_A=24977314604014664086101644993156626987090949695850726634173073983343,$$

while bob selects the secret chain of indexes

$$\begin{aligned}\boldsymbol{s_B}=&3,2,3,3,2,3,2,4,2,3,2,3,3,2,3,3,3,3,2,3,2,3,2,2,2,4,2,3,2,4,3,4,3,3,2,2,2,2,3,3,4,4,2,4,2,3,\\ &4,2,3,2,3,2,4,4,4,2,4,2,3,4,2,2,2,2,4,4,2,2,3,3,3,4,2,4,4,2,4,4,4,3,3,2,4,2,2,2,3,3,3,3,2,4,\\ &2,3,3,4,2,4,4,3,3,4,4,2,4,2,4,3,4,2,3,3,2,2,2,4,3,4,3,4,4,3,2,4,3,3,4,3,4,2,4,3,3,4,2,2,4.\end{aligned}$$

Alice then does as before. She calculates her secret generator as $R_A = P_A + [s_A]Q_A$, and uses it to calculate the isogeny $\phi_A : E \to E_A$ having $\langle R_A\rangle$ as a kernel. This way she finds the curve $E_A : X^3 + Y^3 + Z^3 = d_AXYZ$, with

$$\begin{aligned}d_A=&43508942752908707946530506875684541257899722934479343657879843172 75+\\ &50513365856571436286331209026668740142583402064906686702202007708240\omega.\end{aligned}$$

Since Bob has no torsion points, this is all Alice needs to send to Bob. Bob, however, does his isogeny-walk by picking a chain of subgroups of order 3 corresponding to his secret, as described in Chapter 5. In the end, he has calculated the isogeny $\phi_B : E \to E_B$, where $E_B : X^3 + Y^3 + Z^3 = d_B XYZ$ has parameter

$$\begin{aligned} d_B = &627644490103113986123585158579214100435602006335769950373600362673 79+ \\ &508460269312595952691942712867414887967436207949502813646903136128 36\omega. \end{aligned}$$

Since Alice still does the calculation as usual, Bob also need to calculate the image of Alice's torsion basis. This becomes

$$\begin{aligned} P_A = \big(&665227727807437885609446868290154108461397682722235051340218512 8619+ \\ &228982159766708715057677436890167574567910673046989722318295869611 49\omega: \\ &114094786092955781475526318015361914104004931626745160721593084428 53+ \\ &576666437013339534432460635731117736420890530849951711296179266977 74\omega: \\ &391520502182915694174631914684836618210133977033105493145889305435 28+ \\ &465220855842945218589484220767295930531430840538379454891002102769 73\omega), \end{aligned}$$

$$\begin{aligned} Q_A = \big(&886291021844285042512203579508282311091110806346933226384971408286 25+ \\ &726040144061520046295738987189435709591455801216086068583069394297 67\omega: \\ &244182754497288699658826135650778434944988023741330636695785043354 9+ \\ &184468240385590623660510011269714246189576933317639429082853327587 5\omega: \\ &658101475027260166580830502802686697289222353842730516742112344571 1+ \\ &419934240363524301877594402686130079977542106236487949584785773006 10\omega). \end{aligned}$$

Computing stage II: Alice does her calculation exactly as in Section 6.1.1 and ends up with E_{BA}.

Unlike the previous example, Bob does not need to change his secret since Since $e_A \equiv 0 \pmod 2$. Once Bob receives E_A, he calculates a new isogeny, similar to how he did in stage I, except that this time he uses the oracle to select the correct cube root in each step. He ends up with the curve E_{AB}. Again, since $E_{AB} \cong E_{BA}$, they both take the j-invariant of their final curve, to arrive at the shared secret.

$$\begin{aligned} j(E_{AB}) = j(E_{BA}) = &251578552858651116764539893246621843288574800849040572721995536150 35+ \\ &489126368126295980518797072672271172135776347263582087069627895649 80\omega. \end{aligned}$$

This concludes the example of the alternative calculation, using real-world parameter sizes. The main takeaway should be that while this example had the same security level as the example in Section 6.1.1, all parameters are only $\approx \frac{1}{2}$ as long (i.e. half the bit-length). Additionally, the size of Alice's public key is greatly reduced, as she no longer needs to send the image of a torsion basis for Bob.

6.1.3 Performance Benchmarks

Table 6.1 shows some benchmarks for the calculation of stage II of the key exchange for both Alice and Bob. As mentioned in the introduction, the focus of the thesis

Implementation					
Chapter	NIST sec. lvl	Field	Size of prime	Alice	Bob
Chp 4	N/A	$\mathbb{F}_p(i)$	132-bit	8.572 ms	11.443 ms
	128-bit	$\mathbb{F}_p(i)$	434-bit	53.019 ms	80.583 ms
	152-bit	$\mathbb{F}_p(i)$	503-bit	72.666 ms	90.220 ms
	189-bit	$\mathbb{F}_p(i)$	610-bit	84.599 ms	107.982 ms
	256-bit	$\mathbb{F}_p(i)$	751-bit	103.350 ms	156.870 ms
Chp 5	N/A	$\mathbb{F}_p(i)$	71-bit	8.071 ms	11.605 ms
	N/A	$\mathbb{F}_p(\omega)$	71-bit	8.699 ms	12.919 ms
	128-bit	$\mathbb{F}_p(i)$	226-bit	70.389 ms	259.152 ms
	128-bit	$\mathbb{F}_p(\omega)$	226-bit	95.981 ms	283.145 ms
	152-bit	$\mathbb{F}_p(i)$	259-bit	95.465 ms	360.223 ms
	152-bit	$\mathbb{F}_p(\omega)$	259-bit	98.583 ms	382.018 ms
	189-bit	$\mathbb{F}_p(i)$	315-bit	106.838 ms	556.323 ms
	189-bit	$\mathbb{F}_p(\omega)$	315-bit	112.162 ms	633.556 ms
	256-bit	$\mathbb{F}_p(i)$	382-bit	129.416 ms	832.607 ms
	256-bit	$\mathbb{F}_p(\omega)$	382-bit	134.379 ms	899.399 ms

Table 6.1: Performance benchmarks of the implementations for different parameters.

has not been the absolute speed of the implementation, however, we provide a summary here as an example. The run times were calculated as the median of multiple evaluations using the package BenchmarkTools.jl[1]. The implementations were run on a computer with a 2.0 GHz Intel Core i5-1038NG7 processor, with four cores.

As expected, Bob's computation is significantly slower in the implementation from Chapter 5 than in the implementation from Chapter 4. More surprisingly, Alice's computation is measured as slower in the implementation from Chapter 5 as well, despite being logically the same as in Chapter 4, but with smaller parameters. We return to the theoretical discussion of these costs in Section 6.3.3. However, possible sources of errors for this estimate might be that the heavy oracle computation causes some performance leak, or simply because of differences in the implementations.

6.2 Calculating the 2^n-Isogeny

In Chapter 4, we described a working implementation of SIDH using twisted Hessian curves. The work is mainly done by combining three different earlier works in the

[1] https://juliaci.github.io/BenchmarkTools.jl/dev/manual/

literature: Jao and De Feo's original SIDH (which used Montgomery curves) [JD11], a paper by Bernstein et al. introducing twisted Hessian curves, and addition, doubling and tripling formulae on these curves [BCKL15], and a paper by Dang and Moody providing isogeny-formulae for twisted Hessian curves [DM19].

In addition to combining these papers, the section presented novel observations. In particular, Proposition 4.1.2 presented a closed formula for the j-invariant of a twisted Hessian curve, and Section 4.1.4 discussed ideas which showed that one can ignore one of the curve coefficients when calculating 2^n-isogenies for $n \in \mathbb{N}$.

6.2.1 Efficiency

Since we simply applied the formulae for isogenies from Section 3.2.2, the costs remain the same as can be found in the literature. A comparison of the formulae with isogeny-formulae for other curve types can be found in the paper by Dang and Moody [DM19]. The same goes for doubling and tripling formulae which can be found in the original paper on twisted Hessian curves by Bernstein et al. [BCKL15].

While both isogeny formulae, and especially tripling formulae on twisted Hessian curves, are nearly competitive (i.e. only slightly worse) compared to other elliptic curve models, so-called differential addition formula for twisted Hessian curves does not yet exist in the literature to our knowledge. This allows for computation using only two projective coordinates, which significantly decreases the computational cost of SIDH in schemes that enjoy differential addition formulae (e.g. Montgomery curves [CLN16] and general Huff's curves [DKW20]).

Although a differential addition formula was not found in this work, we showed that calculating in 2^n-isogenies was possible while ignoring one of the curve parameters. Compared to directly applying Theorem 3.2.8, this saves 1 inversion and 4 multiplications by Equation 4.2 (recall that we have to square a anyway, unless $a = 1$), at the cost of 9 multiplications and 1 inversion at the end of the calculation. This is a significant performance gain, as we typically require $n \approx \frac{1}{2}\log_2 p$ steps. While this is a step in the right direction, it is highly unlikely to be enough to bridge the gap to other curve types with differential addition formulae.

6.3 Calculating the 3^n-Isogeny

In Chapter 5, we presented a new approach to calculating 3^n-isogenies in SIDH, when using twisted Hessian curves. The approach was mainly enabled by two propositions. Proposition 5.1.1 made it possible to ignore generator points, and simply compute a chain of length e_B, consisting of 3-isogenies, while Proposition 5.1.2 gives a trivial way to recover the subgroups of order 3, between two curves that are connected by

a degree 2^n isogeny[2] for some $n \in \mathbb{N}$. However, composition of isogenies based on Proposition 5.1.2, only allows degree 3 and degree 2^n isogenies to commute up to the same j-invariant, meaning that image curves may not be exactly the same, even though they are isomorphic. To provide motivation, we showed that if they were exactly the same, this would lead to degree 3^n and degree 2^m isogenies commuting for arbitrary $n, m \in \mathbb{N}$, which would be enough to make the tweaked SIDH scheme work. However, the requirement that they commute exactly is necessary, since isomorphisms may permute the subgroups of order 3.

6.3.1 Motivation for Alternative Computation

The main motivation for our tweaked SIDH scheme comes from the fact that in traditional SIDH, there is a mismatch between the security gained from the size of the prime, and the security gained from the length of the isogeny-walk. Recall that in SIDH, the prime p has the form $\ell_A^{e_A}\ell_B^{e_B}f \pm 1$, where $\ell_A^{e_A} \approx \ell_B^{e_B}$, and that Alice does an ℓ_A-isogeny walk of length e_A, while Bob does an ℓ_B-isogeny walk of length e_B. A consequence of this is that the CSSI-problem (discussed in Section 3.3.3) instantiated with the starting curve and a public-key curve in the graph $\mathcal{G}_{\ell_A}(\bar{\mathbb{F}}_p)$ (see Section 3.3.2) is much easier than it is for two arbitrary curves (the best known attack in the general case is an algorithm by Delfs and Galbraith [DG16] which uses $\mathcal{O}\big(\sqrt[2]{p}\big)$, while the claw-finding algorithm discussed in Section 3.3.3 uses $\mathcal{O}\big(\sqrt[4]{p}\big)$). This means that potentially, one can use another prime p', with $p' \ll p$, without affecting the security level, as long as the isogeny walks are of the same length, and that the number of supersingular curves over $\bar{\mathbb{F}}_{p'}$ is greater than $(\ell_A+1)\ell_A^{e_A-1}$ and $(\ell_B+1)\ell_B^{e_B-1}$. A prime p' as low as $p' \approx \sqrt{p}$ can be selected.

However, in the usual case, reducing the size of the prime comes at a great cost. Recall that the prime has the specific restriction, to ensure that both the $\ell_A^{e_A}$-torsion group and $\ell_B^{e_B}$-torsion group are $\mathbb{F}_{p^2}$-rational (see Section 3.3.1). Therefore, if we select $p' \approx \sqrt{p}$, this will generally no longer be the case, which means that we need to go to a larger field-extension $\mathbb{F}_{p^n}$, $n > 2$, to find a torsion-basis, and arithmetic has to happen in $\mathbb{F}_{p^n}$. Not only is this much slower than arithmetic in $\mathbb{F}_{p^2}$, but a potential reduction in the size of the public keys is also forfeit as the images of the torsion points (recall that in SIDH, Alice's public key consists of $E_A, \phi_A(P_B), \phi_A(Q_B)$) are generally only defined over the $\mathbb{F}_{p^n}$, where elements require an n-dimensional vector to be represented.

Another approach to solving this problem was given in a recent paper by Costello [Cos19]. While the techniques used by Costello are completely different from what we discussed in Chapter 5, some of the advantages of the resulting scheme are similar to what we potentially could achieve.

[2]When the 2^n-isogenies are computed as a chain of 2 isogenies as given in Theorem 3.2.8.

6.3.2 No Need for an $\mathbb{F}_{p^2}$-Rational 3^{e_B}-Torsion Group

We now summarize how our calculation from Chapter 5 approaches the problem in 6.3.1. Recall Section 5.2, which showed that picking a point R of order 3^{e_B} to generate the kernel of a 3^{e_B}-isogeny was equivalent to simply picking a chain of indexes, and using the corresponding subgroups of order 3 as classified in Section 5.1.2. Using the equivalent approach, it should be clear that, in practice, we do not do any computation with the torsion points, we simply pick subgroups of order 3. Therefore, whether the whole 3^e-torsion is $\mathbb{F}_{p^2}$-rational is no longer relevant.

Alice still requires an $\mathbb{F}_{p^2}$-rational 2^{e_A}-torsion group, since her calculation is the same as explained in Chapter 4. Further, Bob requires that all subgroups of order 3 are $\mathbb{F}_{p^2}$-rational, i.e. that the 3-torsion group is $\mathbb{F}_{p^2}$-rational (as was discussed in Section 5.3.2). And finally, to allow easy cube roots, we require $p^2 \equiv 7 \pmod 9$, as discussed in Section 5.1.1. Using a prime of the form $p = 2^{e_A} f \pm 1$, with $3 \mid f$, and $p \equiv \pm 5 \pmod 9$, we can find supersingular curves E with $E(\mathbb{F}_{p^2}) \cong \mathbb{Z}/(2^{e_A} f)\mathbb{Z} \times \mathbb{Z}/(2^{e_A} f)\mathbb{Z}$, which satisfies all these properties (by the same method as mentioned in Section 3.3.1).

6.3.3 Computational Cost

As in Chapter 5, let Alice be the one working in the 2^{e_A}-torsion, and let Bob be the one working in the 3^{e_B}-torsion. Alice uses the same methods described in Chapter 4. In other words, Alice' computational cost in terms of the number of field-operations is exactly the same as discussed in the previous section. However, the size of the field we are working over is much smaller (we return to this in Section 6.3.4), which means that the field-operations themselves are cheaper.

Bob's computation, however, is likely more expensive, even without counting the cost of the oracle. In each step, Bob has to calculate a cube root and an inversion, which dominates the computational cost. These operations are to get the curve on Hessian form, which may or may not be necessary, depending on if and how the oracle is replaced. However, even if it turns out to be unnecessary to get the curve on Hessian form, Bob must always do a cube root calculation to find the subgroups of order 3. Even though we have set up the field in a way to make cube roots easier, it still requires an exponentiation by $k = \frac{q+2}{9} = \frac{p^2+2}{9}$, which takes $\log_2 k \approx 2 \log_2 p$ squarings by the square and multiply algorithm (Algorithm 4.1). The number of multiplications varies based on the Hamming weight of k, but on average, it is around $\log_2 p$ multiplications. In total, counting squarings as multiplications, the cost of the alternative computation is about $3 \log_2 p$ multiplications in each step. It is difficult to compare it with the cost of the usual SIDH calculation, but if we compare it with the most basic strategy, the purely multiplication focused isogeny calculation (Algorithm 3.1), we see that it has about the same cost. The basic strategy uses one average

$\approx \frac{1}{4}\log_3 p$ curve triplings in each step, which each takes 17 multiplications and 3 squarings by Algorithm 4.7. Counting squarings as multiplications, this gives a total of

$$20 \cdot \frac{1}{4} \cdot \frac{\log_2 p}{\log_2 3} \approx 3.155 \log_2 p$$

multiplications per step. Therefore, we can compare the cost of the tweaked SIDH scheme to the cost of using the purely multiplication focused strategy. This is much slower than regular SIDH, which has access to the speedup gained from using optimal strategies, so the speed of the alternative calculation has to be regarded as quite slow.

6.3.4 The Field Size

Most of the performance gain in the tweaked SIDH scheme is a result of the fact that we avoid the need for a $\mathbb{F}_{p^2}$-rational 3^{e_B}-torsion. Recall that regular SIDH requires a prime p of the form $p = 2^{e_A}3^{e_B}f \pm 1$, where f is as small as possible (in practice, often $f = 1$) and $2^{e_A} \approx 3^{e_B}$. For isogeny-walks of equal length, the alternative calculation only requires the prime to be on the form $p' = 2^{e_A}f' \pm 1$, where $3 \mid f'$ to ensure that the 3-torsion group is $\mathbb{F}_{p^2}$-rational. This gives that $p' \approx 3 \cdot \sqrt{p}$, assuming $f = 1$ and $f' = 3$. However, to avoid lowering the security level, one should choose $f \geqslant 12$, because the total number of supersingular isomorphism classes is "only" $\approx \frac{p}{12}$ (Theorem 3.1.21), but either way, the bit-length of the field is only slightly larger than half of what is required in regular SIDH.

Furthermore, recall the example in Section 6.1.2. Alice's public key only consists of the curve parameter of her public curve, compared to regular SIDH, where her public key also consists of images of Bob's torsion points. This has the advantage of further reducing the key-sizes in the alternative computation. Additionally, it is still unknown whether the torsion points affect the security of SIDH, so avoiding the need for them is an advantage in itself. We return to this in Section 6.3.6

6.3.5 Mixing Property

Another interesting benefit from the reduced field sizes is that the isogeny-walks that Bob and Alice do are now of length $\approx \log p$ in their respective supersingular graphs. This is much closer to the diameter of the graph, and in many cases, it may be possible to select parameters, so that the isogeny-walks indeed are longer than the diameter of the graph, which means that the random-walks actually achieve the mixing property mentioned in Section 3.3.2.

6.3.6 Petit's Attacks on Unbalanced SIDH

As mentioned in Section 3.3.3, Petit has shown that revealing the images of torsion points under the secret isogeny may impact security. In the paper by Petit, it is stressed that the attacks do not impact regular SIDH. However, without going into the details of the attacks, it affects so-called unbalanced variants, where $\ell_A^{e_A} \gg \ell_B^{e_B}$. While at first glance, this may seem to affect our tweaked scheme, we argue that it does not. The significance of $\ell_A^{e_A} \gg \ell_B^{e_B}$ is that Petit's attacks rely on one isogeny-walk being much longer than the other[3] (among other things), which is not the case in our tweaked SIDH scheme. In a sense, we regard the torsion groups that Alice and Bob are working over as the roughly same size in both the original SIDH scheme and our tweaked variant, the difference being that in the original SIDH scheme, they are both $\mathbb{F}_{p^2}$-rational, while in our tweaked variant, we have to go to some larger field-extension of $\mathbb{F}_p$ to see Bob's whole torsion group (but, as explained, we ignore this when doing the calculation).

Although this is quite far from what we have done in this thesis, if one could generalize our tweaked variant in such a way that neither Alice nor Bob require an explicit torsion basis, then of course this would be a benefit of the tweaked SIDH variant, as we would not have to worry about potential security problems such as Petit's attack related to images of torsion points.

[3]Let N_1 and N_2 denote the length of the isogeny walks. Then one attack requires $N_1 > N_2^4$ [Pet17, Section 4.4], while another requires $\log N_2 \in \mathcal{O}(\log \log N_1)$ [Pet17, Section 4.5].

Chapter 7
Conclusion

In this thesis, we have studied how suitable twisted Hessian curves are in the context of the supersingular isogeny Diffie-Hellman (SIDH) key exchange. Formulae for isogenies between twisted Hessian curves enable them to be used in the SIDH protocol, and additionally, they have fast doubling and tripling formulae, which may potentially lead to efficient implementations, given more work. What is more, subgroups of order 3 have a particularly simple structure on twisted Hessian curves. This simple structure relates nicely to 3-isogenies (recall that separable 3-isogenies have subgroups of order 3 as kernels), which may be possible to exploit to create new SIDH-like schemes with certain appealing properties.

7.1 Summary

The first part of the thesis outlined how one can use twisted Hessian curves when implementing SIDH. We showed that one advantage of using twisted Hessian curves when calculating a composition of n degree 2-isogenies is that calculating the image curves of 2-isogenies is very cheap. We showed that this only requires a single squaring, since one can ignore one of the curve coefficients, at the cost of recovering the curve coefficient at the end of the chain. However, this optimization is not enough to close the gap in computing cost, when comparing with curves that have a differential-addition formula.

The second part of the thesis presented a novel view on the 3^n-isogeny calculation in SIDH. We showed that when using twisted Hessian curves, there is a one-to-one correspondence between 3^n-isogenies, and chains of specific subgroups of length n. This one-to-one correspondence was a consequence of a result we showed, which roughly stated that the dual-isogeny of a 3-isogeny is always generated by a specific subgroup of order 3 when using twisted Hessian curves. This, combined with another result which showed how the subgroups of order 3 permute under 2-isogenies, suggests a different way to calculate 3^n-isogenies in SIDH which does not require an $\mathbb{F}_{p^2}$-

rational 3^n-torsion group. Although we did not manage to make it fully functional, we showed that there is only a single step missing.

Although our tweaked SIDH scheme seems to be slower than regular SIDH, it has a lot of interesting properties. The relaxed requirement on the prime p means that it is possible to select primes that are only slightly larger than half the bit-length of the primes used in regular SIDH, without a loss in security level. Additionally, even though we showed that the tweaked SIDH is likely quite slow, there is a lot of room for improvement. The run-time is dominated by repeated exponentiation by a large k. The significance here is that k is always fixed, which allows for much better algorithms than the simple square-and-multiply which we used. Still, we consider the primary performance gain of the tweaked SIDH scheme to be the reduced key-sizes.

Our work shows that twisted Hessian curves may have some usen cases in isogeny-based cryptography and that more work is warranted. As it currently stands, twisted Hessian curves offer no particular benefits over implementations using e.g. Montgomery curves. However, with more research, twisted Hessian curves could be used in both standard implementations of SIDH and other SIDH-like schemes with relaxed requirements on the prime used to define the field. We have given an example of such a scheme, which potentially uses primes which are around half the bit length of those that are used in SIDH today.

7.2 Research Questions

In the introduction, we stated that our primary research objective was *to study how suitable twisted Hessian curves are for the SIDH protocol, and for isogeny-based cryptography in general.* To do this, we presented three research questions, which we aimed to answer throughout the thesis. Here, we briefly summarize these answers.

(i) *What techniques can be applied to optimize the performance of twisted Hessian curves in the setting of isogeny-based cryptography?*

For SIDH-based schemes, we have primarily presented two optimization techniques. The computational cost of SIDH is dominated by the cost of calculating long chains of low-degree isogenies. Chains of 2-isogenies can be computed faster by ignoring one of the curve-coefficients, as both the point doubling formula and the isogeny formula is independent of this parameter. Further, key-sizes can be significantly reduced by closing the gap in security gained by the length of the isogeny-walks and the size of the prime. This gap can be reduced since the 3^{e_B}-isogeny computation can be done without an $\mathbb{F}_{p^2}$-rational 3^{e_B}-torsion group as shown in Chapter 5.

(ii) *What are the challenges when implementing an isogeny-based protocol with twisted Hessian curves?*

One challenge, which limits the computational speed, when applying twisted Hessian curves to isogeny-based cryptography is the lack of a formula for differential addition. Other challenges were discussed in Chapter 4. One notable example is the fact that the tripling formula (Algorithm 4.7) does not work for twisted Hessian curves of the form $H(a, 0)$, which are supersingular whenever 0 is a supersingular j-invariant.

(iii) *How can the structure of twisted Hessian curves be taken advantage of in the SIDH setting?*

One interesting structural aspect of these curves are their particularly simple 3-torsion group. How to take advantage of this in the SIDH setting was discussed thoroughly in Chapter 5, and could lead to the optimization mentioned in the answer to research question (i) regarding the 3^{e_B}-isogeny computation.

7.3 Future Work

Our work leaves room for more research on twisted Hessian curves in the setting of isogeny-based cryptography. Specifically, related to the work we have done, we give two open problems which, if solved, could make twisted Hessian curves an interesting candidate for implementations of SIDH. The first open problem is the most general, while the second is related to our tweaked SIDH-scheme.

Open problem 1: *Find a differential addition formula for twisted Hessian curves.*

Differential addition formulae have been used to speed up SIDH and elliptic curve cryptography in general on curve models such as the Montgomery curve, and would likely benefit twisted Hessian curves as well.

Open problem 2: *Make isogenies computed from Theorem 3.2.8 and Theorem 3.2.10 commute exactly.*

As we have shown, solving this problem would lead to our tweaked SIDH scheme working, which enables SIDH without a $\mathbb{F}_{p^2}$-rational 3^{e_B}-torsion group. There may be other approaches as well, but we believe this is the best approach, as empirical evidence shows that they already commute exactly up to a factor of ω, where ω is a non-trivial cube root of one.

References

[AASA+20] Gorjan Alagic, Jacob Alperin-Sheriff, Daniel Apon, David Cooper, Quynh Dang, John Kelsey, Yi-Kai Liu, Carl Miller, Dustin Moody, Rene Peralta, et al. Status report on the second round of the NIST post-quantum cryptography standardization process. *US Department of Commerce, NIST*, 2020.

[ACC+19] Gora Adj, Daniel Cervantes-Vázquez, Jesús-Javier Chi-Domínguez, Alfred Menezes, and Francisco Rodríguez-Henríquez. On the cost of computing isogenies between supersingular elliptic curves. In Carlos Cid and Michael J. Jacobson Jr:, editors, *SAC 2018: 25th Annual International Workshop on Selected Areas in Cryptography*, volume 11349 of *Lecture Notes in Computer Science*, pages 322–343, Calgary, AB, Canada, August 15–17, 2019. Springer, Heidelberg, Germany.

[Adl79] Leonard M. Adleman. A subexponential algorithm for the discrete logarithm problem with applications to cryptography (abstract). In *20th Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 29-31 October 1979*, pages 55–60. IEEE Computer Society, 1979.

[AMM77] Leonard M. Adleman, Kenneth L. Manders, and Gary L. Miller. On taking roots in finite fields. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pages 175–178. IEEE Computer Society, 1977.

[BBJ+08] Daniel J. Bernstein, Peter Birkner, Marc Joye, Tanja Lange, and Christiane Peters. Twisted edwards curves. In Serge Vaudenay, editor, *Progress in Cryptology - AFRICACRYPT 2008, First International Conference on Cryptology in Africa, Casablanca, Morocco, June 11-14, 2008. Proceedings*, volume 5023 of *Lecture Notes in Computer Science*, pages 389–405. Springer, 2008.

[BCKL15] Daniel J. Bernstein, Chitchanok Chuengsatiansup, David Kohel, and Tanja Lange. Twisted Hessian curves. In Kristin E. Lauter and Francisco Rodríguez-Henríquez, editors, *Progress in Cryptology - LATINCRYPT 2015: 4th International Conference on Cryptology and Information Security in Latin America*, volume 9230 of *Lecture Notes in Computer Science*, pages 269–294, Guadalajara, Mexico, August 23–26, 2015. Springer, Heidelberg, Germany.

[BF01] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 213–229, Santa Barbara, CA, USA, August 19–23, 2001. Springer, Heidelberg, Germany.

[BF19] Fouazou Lontouo Perez Broon and Emmanuel Fouotsa. Analogue of Vélu's formulas for computing isogenies over Hessian model of elliptic curves. *IACR Cryptol. ePrint Arch.*, 2019:1480, 2019.

[BJN94] Phani Bhushan Bhattacharya, Surender Kumar Jain, and SR Nagpaul. *Basic Abstract Algebra*. Cambridge University Press, 1994.

[Brö09] Reinier Bröker. Constructing supersingular elliptic curves. *J. Comb. Number Theory*, 1(3):269–273, 2009.

[CJS14] Andrew M. Childs, David Jao, and Vladimir Soukharev. Constructing elliptic curve isogenies in quantum subexponential time. *J. Math. Cryptol.*, 8(1):1–29, 2014.

[CLM+18] Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. CSIDH: An efficient post-quantum commutative group action. In Thomas Peyrin and Steven Galbraith, editors, *Advances in Cryptology – ASIACRYPT 2018, Part III*, volume 11274 of *Lecture Notes in Computer Science*, pages 395–427, Brisbane, Queensland, Australia, December 2–6, 2018. Springer, Heidelberg, Germany.

[CLN16] Craig Costello, Patrick Longa, and Michael Naehrig. Efficient algorithms for supersingular isogeny Diffie-Hellman. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part I*, volume 9814 of *Lecture Notes in Computer Science*, pages 572–601, Santa Barbara, CA, USA, August 14–18, 2016. Springer, Heidelberg, Germany.

[CLN+20] Craig Costello, Patrick Longa, Michael Naehrig, Joost Renes, and Fernando Virdia. Improved classical cryptanalysis of SIKE in practice. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020: 23rd International Conference on Theory and Practice of Public Key Cryptography, Part II*, volume 12111 of *Lecture Notes in Computer Science*, pages 505–534, Edinburgh, UK, May 4–7, 2020. Springer, Heidelberg, Germany.

[CLO97] David A. Cox, John Little, and Donal O'Shea. *Ideals, Varieties, and Algorithms - an Introduction to Computational Algebraic Geometry and Commutative Algebra*. Undergraduate Texts in Mathematics. Springer, 1997.

[Cos19] Craig Costello. B-SIDH: supersingular isogeny Diffie-Hellman using twisted torsion. Cryptology ePrint Archive, Report 2019/1145, 2019. https://eprint.iacr.org/2019/1145.

[Cou06] Jean-Marc Couveignes. Hard homogeneous spaces. Cryptology ePrint Archive, Report 2006/291, 2006. http://eprint.iacr.org/2006/291.

[DG16] Christina Delfs and Steven D. Galbraith. Computing isogenies between supersingular elliptic curves over $\mathrm{F_p}$. *Des. Codes Cryptogr.*, 78(2):425–440, 2016.

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

[DKW20] Robert Dryło, Tomasz Kijko, and Michał Wroński. Efficient montgomery-like formulas for general huff's and huff's elliptic curves and their applications to the isogeny-based cryptography. Cryptology ePrint Archive, Report 2020/526, 2020. https://eprint.iacr.org/2020/526.

[DM19] Thinh Dang and Dustin Moody. Twisted Hessian isogenies. Cryptology ePrint Archive, Report 2019/1003, 2019. https://eprint.iacr.org/2019/1003.

[Edw07] Harold Edwards. A normal form for elliptic curves. *Bulletin of the American mathematical society*, 44(3):393–422, 2007.

[ElG85] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31:469–472, 1985.

[Feo17] Luca De Feo. Mathematics of isogeny based cryptography. *CoRR*, abs/1711.04062, 2017.

[FJ10] Reza Rezaeian Farashahi and Marc Joye. Efficient arithmetic on Hessian curves. In Phong Q. Nguyen and David Pointcheval, editors, *PKC 2010: 13th International Conference on Theory and Practice of Public Key Cryptography*, volume 6056 of *Lecture Notes in Computer Science*, pages 243–260, Paris, France, May 26–28, 2010. Springer, Heidelberg, Germany.

[GPST16] Steven D. Galbraith, Christophe Petit, Barak Shani, and Yan Bo Ti. On the security of supersingular isogeny cryptosystems. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology – ASIACRYPT 2016, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 63–91, Hanoi, Vietnam, December 4–8, 2016. Springer, Heidelberg, Germany.

[Har77] Robin Hartshorne. *Algebraic Geometry*, volume 52 of *Graduate Texts in Mathematics*. Springer, 1977.

[HHK17] Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. A modular analysis of the Fujisaki-Okamoto transformation. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017: 15th Theory of Cryptography Conference, Part I*, volume 10677 of *Lecture Notes in Computer Science*, pages 341–371, Baltimore, MD, USA, November 12–15, 2017. Springer, Heidelberg, Germany.

[29] Luca De Feo (https://crypto.stackexchange.com/users/40730/luca-de-feo). Independent parameters basis for torsion-groups in SIDH: Is the Weil-pairing necessary? Cryptography Stack Exchange. URL:https://crypto.stackexchange.com/questions/89671 (version: 2021-05-06).

[JAC+20] D. Jao, R. Azarderakhsh, M. Campagna, C. Costello, L. De Feo, B. Hess, A. Hutchinson, A. Jalali, K. Karabina, B. Koziel, B. LaMacchia, P. Longa, M. Naehrig, G. Pereira, J. Renes, V. Soukharev, and D. Urbanik. Supersingular Isogeny Key Encapsulation. Technical report, October 2020. Available at https://sike.org/files/SIDH-spec.pdf.

[JD11] David Jao and Luca De Feo. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In Bo-Yin Yang, editor, *Post-Quantum Cryptography - 4th International Workshop, PQCrypto 2011*, pages 19–34, Tapei, Taiwan, November 29 – December 2 2011. Springer, Heidelberg, Germany.

[JS19] Samuel Jaques and John M. Schanck. Quantum cryptanalysis in the RAM model: Claw-finding attacks on SIKE. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part I*, volume 11692 of *Lecture Notes in Computer Science*, pages 32–61, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Heidelberg, Germany.

[JTV10] Marc Joye, Mehdi Tibouchi, and Damien Vergnaud. Huff's model for elliptic curves. Cryptology ePrint Archive, Report 2010/383, 2010. http://eprint.iacr.org/2010/383.

[Kob87] Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of computation*, 48(177):203–209, 1987.

[Kob93] Neal I. Koblitz. *Introduction to Elliptic Curves and Modular Forms*, volume 97 of *Graduate Texts in Mathematics*. Springer, 1993.

[KYK+18] Suhri Kim, Kisoon Yoon, Jihoon Kwon, Seokhie Hong, and Young-Ho Park. Efficient isogeny computations on twisted edwards curves. *Secur. Commun. Networks*, 2018:5747642:1–5747642:11, 2018.

[Leo20] Christopher Leonardi. A note on the ending elliptic curve in SIDH. Cryptology ePrint Archive, Report 2020/262, 2020. https://eprint.iacr.org/2020/262.

[LJ87] Hendrik W Lenstra Jr. Factoring integers with elliptic curves. *Annals of mathematics*, pages 649–673, 1987.

[ME98] Michele Mosca and Artur Ekert. The hidden subgroup problem and eigenvalue estimation on a quantum computer. In Colin P. Williams, editor, *Quantum Computing and Quantum Communications, First NASA International Conference, QCQC'98, Palm Springs, California, USA, February 17-20, 1998, Selected Papers*, volume 1509 of *Lecture Notes in Computer Science*, pages 174–188. Springer, 1998.

[Mil76] Gary L. Miller. Riemann's hypothesis and tests for primality. *J. Comput. Syst. Sci.*, 13(3):300–317, 1976.

[Mil85] Victor S. Miller. Use of elliptic curves in cryptography. In *CRYPTO*, volume 218 of *Lecture Notes in Computer Science*, pages 417–426. Springer, 1985.

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

[NIS16] NIST. Announcing request for nominations for public-key post-quantum cryptographic algorithms. https://csrc.nist.gov/news/2016/public-key-post-quantum-cryptographic-algorithms, 2016. Accessed: 2020-09-24.

[Pet17] Christophe Petit. Faster algorithms for isogeny problems using torsion point images. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017, Part II*, volume 10625 of *Lecture Notes in Computer Science*, pages 330–353, Hong Kong, China, December 3–7, 2017. Springer, Heidelberg, Germany.

[RKBT10] J. Randall, B. Kaliski, J. Brainard, and S. Turner. Use of the RSA-KEM Key Transport Algorithm in the Cryptographic Message Syntax (CMS). RFC 5990, RFC Editor, September 2010.

[RS06] Alexander Rostovtsev and Anton Stolbunov. Public-Key Cryptosystem Based On Isogenies. Cryptology ePrint Archive, Report 2006/145, 2006. http://eprint.iacr.org/2006/145.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the Association for Computing Machinery*, 21(2):120–126, 1978.

[Sch95] René Schoof. Counting points on elliptic curves over finite fields. *Journal de théorie des nombres de Bordeaux*, 7(1):219–254, 1995.

[Sha73] Daniel Shanks. Five number-theoretic algorithms. In *Proceedings of the Second Manitoba Conference on Numerical Mathematics (Winnipeg), 1973*, 1973.

[Sho94] Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *35th Annual Symposium on Foundations of Computer Science*, pages 124–134, Santa Fe, NM, USA, November 20–22, 1994. IEEE Computer Society Press.

[Sil09] Joseph H. Silverman. *The Arithmetic of Elliptic Curves*, volume 106 of *Graduate Texts in Mathematics*. Springer, 2009.

[Tan09] Seiichiro Tani. Claw finding algorithms using quantum walk. *Theor. Comput. Sci.*, 410(50):5285–5297, 2009.

[Vél71] Jacques Vélu. Isogénies entre courbes elliptiques. *CR Acad. Sci. Paris, Séries A*, 273:305–347, 1971.

[Wil95] Andrew Wiles. Modular elliptic curves and Fermat's last theorem. *Annals of mathematics*, pages 443–551, 1995.

Jonathan Komada Eriksen

Applying Twisted Hessian Curves to SIDH

NTNU
Norwegian University of
Science and Technology