Cryptography on Isogeny Graphs

Cryptography on Isogeny Graphs

Lorenz Panny

Email: lorenz@yx7.cc
Website: https://yx7.cc

First edition January 2021

This research was supported by the Commission of the European Communities through the Horizon 2020 program under project number 643161 (ECRYPT-NET).

A catalogue record is available from the Eindhoven University of Technology Library.
ISBN: 978-90-386-5213-9

Printed by Gildeprint B.V., Enschede, Netherlands.

The cover shows the famous 19th-century woodblock print “The Great Wave off Kanagawa” (神奈川沖 浪裏) by Japanese artist Hokusai (葛飾北斎, c. 1760–1849). It symbolizes mankind’s everlasting struggle against the forces of nature, which bears similarity to the way the laws of mathematics and physics govern cryptography: They mercilessly determine the things we can or cannot do, and no matter how hard we try, there is no overcoming the rule of nature.

Cryptography on Isogeny Graphs

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Technische Universiteit Eindhoven,
op gezag van de rector magnificus prof.dr.ir. F.P.T. Baaijens, voor een commissie
aangewezen door het College voor Promoties, in het openbaar te verdedigen
op donderdag 18 februari 2021 om 16:00 uur

door

Lorenz Stefan Panny

geboren te Eggenfelden, Duitsland

Dit proefschrift is goedgekeurd door de promotoren en de samenstelling van de promotiecommissie is als volgt:

voorzitter:	prof.dr. J.J. Lukkien
1e promotor:	prof.dr. T. Lange
2e promotor:	prof.dr. D.J. Bernstein
leden:	prof. D. Boneh, PhD (Stanford University)
	prof.dr. D.R. Kohel (Université d'Aix-Marseille)
	prof.dr. K.G. Paterson (ETH Zürich)
	dr. F. Vercauteren (KU Leuven)
	dr. B.M.M. de Weger

Het onderzoek of ontwerp dat in dit proefschrift wordt beschreven is uitgevoerd in overeenstemming met de TU/e Gedragscode Wetenschapsbeoefening.

Thanks

I am beyond grateful to my supervisors Tanja and Dan, who are not only excellent scientists, but (little did I know) also incredibly supportive and caring people. Thank you for all the advice and encouragement during these past years.

Thanks to all of my colleagues and friends at TU/e, senior and junior alike, for the lovely working environment and the terrific leisure activities. It's been an absolute pleasure.

Thanks to my coauthors, without whom much of this research would presumably not exist. I'm feeling honored that I got the chance to work together with all of you.

Thanks to Fré and Wouter for inviting me to Leuven, where besides working with them on exciting research problems, I was also warmly welcomed by the other COSIC folks.

Thanks to the members of my doctoral committee, who were free to decline my request, but instead fearlessly took on the daunting task of reading this entire thesis.

Thanks to my fellow ECRYPT-NET fellows for swiftly adopting me in their midst despite my late arrival, and for the fond memories I have of our gatherings all around the world.

Thanks to all the friends I made at various events and conferences, for shallow and profound discussions about anything ranging from isogenies to life in general, and for the fun and games.

Thanks to my CTF team hxp for still going strong after many years, for being co-responsible that I ended up studying cryptography, and for all the awesome hacks.

Thanks to my friends and family everywhere for their continuing affection and support in defiance of geographic distance, and for the good times whenever we do manage to meet up.

Finally, thanks to my parents for life, and everything.

Lorenz Panny
Eindhoven, January 2021

Contents

Chapter 1

Introduction

"We stand today on the brink of a revolution in cryptography." [DH76]

These are the words Diffie and Hellman chose in 1976 to introduce *public-key cryptography* to the world, correctly presaging the unrivaled impact this invention would have on the landscape of information security throughout the following decades. Indeed, many of the technological convenience features current generations take for granted, such as secure online payments and private messaging applications, rely dramatically on tools provided by public-key cryptography. To illustrate why, let us look at the two archetypical examples. The first is public-key encryption: a method of encryption in which anyone may encrypt data for a specific party, but only that intended recipient is capable of deciphering the resulting encrypted text.[1] The second is a digital signature: a tool to certify data in such a way that anyone knowing the signer's identity can verify that the signed message originated at the correct party and has not been tampered with in transit.[2] Evidently, both of these features are indispensable to securing information transmitted over an untrusted network such as the internet: We do not want anyone to spy on us, and we do not want others to put words in our mouths.

Today, we may be facing a revolution of similar scale.

This time, however, the catalyst is a threat rather than an advance. Public-key cryptography must inherently rely on computationally hard problems to give any security guarantees.[3] The most important hard problems have received significant attention from cryptanalysts, and an overnight breakthrough in attacking systems like, for instance, the Diffie–Hellman scheme seems quite unlikely. Instead, one of the biggest dangers comes from an unexpected angle: For the longest time, computer science has been focusing mainly on deterministic machines manipulating strings of zeroes and ones, which we will call *conventional computers*,[4] all the while assuming that this paradigm captures the intuitive notion of "general computation", i.e., all kinds of data processing permitted by the laws of physics, at least up to insignificant differences in efficiency.

"This may not be true when quantum mechanics is taken into consideration." [Sho97a]

Indeed, there are reasons to believe (albeit no proof) that *quantum computers* — machines implementing an alternate model of computation that exploits hidden physical properties of

[1] Intuitively, this may be understood as a technological version of the legal principle "secrecy of correspondence".

[2] Somewhat amusingly, "analog" signatures — writing one's name on paper — provide neither of these guarantees.

[3] Attackers with unlimited resources can, for example, simply try random signatures until the verification algorithm (which they have available) accepts one. More generally, the mathematical relationship between private and public data suffices (information-theoretically) to deduce enough information about the former from the latter to break the scheme. For a concrete example, see [Pan20].

[4] Note that conventional computers do encompass modern microprocessors such as those used in laptops, phones, and high-end supercomputers. As is customary in the field, we use the term "classical computers" interchangeably, although it may be deemed a bit unfortunate due to its obsolescent vibe.

quantum systems — drastically outperform classical computers at certain computational tasks. Due to simplicity and efficiency considerations, it so happened that essentially all public-key cryptosystems widely deployed today are based on problems for which this seems to make a huge difference: They are vulnerable to (variants of) a quantum algorithm discovered in 1994 by Shor [Sho94], which (among other things) breaks the Diffie–Hellman scheme and its descendants very quickly. The only remaining hurdle is the extremely challenging engineering task of building a sufficiently large and reliable quantum computing machine. However, contrary to a relatively widespread misconception in the interested-but-not-expert public, there is hope for cryptography even assuming the presence of large quantum computers.

"There is no justification for the leap [...] to 'quantum computers destroy cryptography'." [Ber09b]

Even though the systems most commonly used nowadays are endangered by the advent of quantum computing, jumping to the conclusion that cryptography in general is doomed would be a fatal mistake:[5] In fact, there are countless constructions known where the availability of a quantum computer does not appear to benefit attackers significantly, or at all. Naturally, these *post-quantum* cryptosystems are based on different, sometimes less versatile hardness assumptions, which often makes the resulting constructions slower, bigger, or harder to use in real systems. This is why determining the best quantum-resistant computational problems to found cryptography on is an important research question, and this thesis contributes to that end by analyzing *isogeny-based cryptography*, a particular class of candidate post-quantum algorithms.

"A mathematical problem, which is hypothetically strong against a quantum computer, [...] consists in searching for an isogeny [...] between elliptic curves over a finite field." [RS06]

In 2006, Stolbunov and his supervisor Rostovtsev were the first to notice that a potentially quantum-resistant key-exchange system could be built from a certain group action on sets of elliptic curves. The action itself is defined in terms of *isogenies*, essentially just a natural notion of morphism between elliptic curves: algebraic maps which are also group homomorphisms. (It was subsequently revealed by Couveignes that he had had a very similar idea in 1997, but his preprint [Cou06] did not appear online until after Rostovtsev–Stolbunov's. Apparently, Couveignes had not thought of the post-quantum properties of his construction, which explains in part why it sunk into oblivion: the scheme simply offered no conceivable benefits at the time.) Even though the best known attack against the Couveignes–Rostovtsev–Stolbunov (CRS) scheme was exponential-time back then, the construction was rather inefficient and seemed unlikely to be very useful. Some four years later, the situation got worse.

In 2010, the CRS scheme suffered a potentially devastating blow: Childs, Jao, and Soukharev discovered that the kind of isogeny-finding problem underlying the CRS scheme can be solved with a subexponential-time quantum algorithm invented as early as 2003 by Kuperberg [Kup05]. The attack is based on the same commutative group action that makes the scheme work in the first place, which suggests that the problem is not fixable, and perhaps that "isogeny-based cryptosystems may be uncompetitive [...]" [CJS14]. Luckily, though, this is not the end of the history of isogeny-based post-quantum cryptography.

Having observed that the commutative structure used in CRS introduces weakness against quantum computers, Jao and De Feo set out to build an isogeny-based cryptosystem that does not feature the commutative aspects, and the perfect fit were *supersingular* elliptic curves. On

[5] Trusting *in*secure mechanisms is obviously problematic, but falsely *dis*trusting *secure* mechanisms can be just as harmful: It means that the *perceived* best solution for a particular problem may in fact be much worse than an unfairly dismissed approach, leading to an inferior (or even catastrophic) outcome overall.

the flip side, the absence of the commutative structure makes it harder to even obtain a working cryptosystem, which Jao and De Feo solved with a clever trick that involves sending certain auxiliary information to allow Alice and Bob to complete a key exchange. This scheme became known as *Supersingular-Isogeny Diffie–Hellman*, or SIDH for short.

Despite its name, SIDH actually lacks some features that classical DH offers, the biggest issue being public-key validation: By sending maliciously crafted key-exchange messages, SIDH users can be tricked into revealing information about their secrets, and it is not known how to distinguish manipulated messages from valid messages without breaking the scheme at the same time. This issue can be bypassed in many scenarios, but it greatly reduces the flexibility, and hence usefulness, of the scheme. Thus, commutative group actions like CRS (which has public-key validation) still seemed useful in principle, if only they could be faster.

To this end, De Feo, Kieffer, and Smith [DKS18] took on the quest of accelerating the CRS scheme starting in 2017. Although they went to great lengths, it seems that finding parameters which allow running the most efficient algorithms is simply too hard — hence the performance numbers achieved in their project ended up being rather disappointing. Yet, the ideas developed in their work turned out extremely useful![6] In fact, in addition to De Feo–Kieffer–Smith's results, it took only one more idea to make things fall into place very nicely: the observation that *supersingular elliptic curves defined over a prime field* have all the right properties for the groundwork laid by [DKS18] to work optimally. The resulting cryptosystem is called /ˈsiːˌsaɪd/, spelled CSIDH, which expands to *Commutative Supersingular-Isogeny Diffie–Hellman*.

Nowadays, SIDH and CSIDH are considered the two main paradigms underlying isogeny-based cryptography, and constructions tend to fall cleanly into either one of the two groups. This thesis discusses constructive and destructive[7] aspects of both families.

1.1 — Outline of the thesis

Chapter 1 is what you are reading right now. It introduces the big picture and guides you, the reader, through the remainder of the thesis.

Chapter 2 surveys some of the mathematical background knowledge underlying the technical contributions given in the following chapters. It covers elliptic curves, isogeny graphs and endomorphism rings of elliptic curves, as well as a brief introduction to quantum algorithms.

Chapter 3 introduces *CSIDH*, a cryptographic commutative group action which is relatively efficient and appears to offer decent post-quantum security. The construction is based on complex-multiplication theory for supersingular elliptic curves defined over a prime field.

Chapter 4 gives an improved version of a CSIDH-based signature scheme called *SeaSign*, a combination of a low-soundness identification scheme and the Fiat–Shamir transform with aborts. The key idea is to allow the prover to reject a few identification queries.

Chapter 5 investigates $\overline{\mathbb{F}_p}$-endomorphism rings of supersingular elliptic curves defined over $\mathbb{F}_p$, in particular establishing an explicit connection between properties of the endomorphism ring and the location of a curve in the graph. One key implication is that the open problem of hashing into the supersingular isogeny graph cannot be solved with reduction of CM curves.

[6] This is a perfect example why publishing "negative" results is important.

[7] Fundamentally, this common dichotomy is a misnomer, at least in many cases: As a rule of thumb, one cannot build secure systems without understanding how insecure ones are broken; therefore, cryptographic work tends to be inherently constructive, and "destructive" work is simply enhancing our understanding of how (in)secure systems are.

Chapter 6 proves a quantum polynomial-time reduction from the problem of inverting a one-way group action to the problem of breaking a Diffie–Hellman-style key exchange that uses said group action, similar to famous classical equivalence results for the Diffie–Hellman problem.

Chapter 7 pushes the boundaries of torsion-point attacks, a flavour of isogeny cryptanalysis specific to schemes which reveal restrictions of isogenies to subgroups. Besides improving the method, it shows how to find intentionally weak parameters that may be usable as backdoors.

Chapter 8 gives a survey of some potential, seemingly promising attack avenues that aspiring SIDH cryptanalysts might stumble upon, and explains why these ideas appear to not yield the desired result, i.e., better attacks.

Chapter 9 constructs an efficient quantum circuit to evaluate the CSIDH group action in superposition. The main application is as a subroutine in a subexponential quantum attack, whose overall cost depends rather strongly on the cost of this step.

Chapter 10 demonstrates a reaction attack against HILA5, a lattice-based KEM submitted to NIST's post-quantum standardization project. The construction had failed to protect against chosen-ciphertext attacks, making it vulnerable to an adaption of Fluhrer's attack.

Chapter 11 surveys some of the insights discovered since the papers that form the basis of this thesis were first published. The remarkably short timeframe of these developments underlines how active and exciting isogeny-based cryptography is as an area of research.

Chapter 2

Mathematical preliminaries

This chapter surveys the most important background underlying the remainder of the thesis.

As is common in mathematics, there are several equivalent definitions (or viewpoints) for most of the objects discussed in this chapter. Whenever in doubt, we chose the most concrete and tangible perspective on the matter — thus sometimes sacrificing generality, abstraction, or mathematical beauty in exchange for minimizing the required prior knowledge and hopefully "making things less weird for everyone" [Smi20].

Also note that the reader should not assume that topics are presented in the order they are usually proved in; we have taken the liberty to reorder some results to ease exposition.

2.1 — Cryptographic constructions

As motivation for the mathematics to come, let us first introduce some cryptographic background based on number theory. The goal of all schemes discussed in this section is *key exchange*, that is, to *establish a shared secret over an insecure communication channel* between two parties Alice and Bob. "Insecure" means that all communication between Alice and Bob may be intercepted and (depending on the model) even modified in transit by a malicious party Eve, whose main goal is to extract or influence (some or all properties of) the shared secret.

After the key exchange has been performed, Alice and Bob can use the established common secret as a key to encrypt messages using *secret-key encryption*, which assumes that a secret key has been shared between the communicating parties ahead of time. On the other hand, *public-key encryption* is asymmetric in that a party can single-handedly generate a pair of mathematically related private and public keys, and anyone learning the public key through some means may send encrypted messages to the recipient.

Assuming the availability of secure secret-key encryption, secure key exchange implies secure public-key encryption: the forward direction uses a technique known as *hybrid encryption*, which consists in establishing a shared secret to be used as a symmetric encryption key; see for instance [CS03].

2.1.1 – Diffie–Hellman. One of the most fundamental tools in public-key cryptography is the Diffie–Hellman key agreement scheme already mentioned in the introduction (Chapter 1). Originally described in 1976 as a concrete instantiation [DH76] on multiplicative groups of prime finite fields, it has since been adapted to other finite abelian groups, and more recently to group *actions*. The group-based version works as follows:[1]

[1] One can impose many different requirements upon the group G; the given variant is one of the most restrictive, streamlined choices.

Definition 2.1. *A* group-based Diffie–Hellman scheme *consists of a generator g of a finite cyclic group G of known order q, such that computing multiplications in G is efficient*[2] *and elements of G have efficient unique encodings as bit strings.*

With these data, the Diffie–Hellman key agreement works as follows:
Key generation. *Alice picks an integer $a \in \{0, ..., q-1\}$ at random; this is her private key. Her public key is $A := g^a \in G$. Bob proceeds analogously to generate his key pair $(b, B) \in \{0, ..., q-1\} \times G$.*
Key agreement. *Alice takes Bob's public key B and her private key a and computes $S = B^a$. Similarly, Bob computes $S = A^b$. Hence, $S = g^{ab}$ is shared among Alice and Bob.*

Note that efficient multiplication implies efficient exponentiation using *square-and-multiply*: The main idea is to recursively replace g^n by the equal expression $(g^2)^{\lfloor n/2 \rfloor} \cdot g^{n \bmod 2}$, which reduces the problem to one or two multiplications plus an exponentiation with an integer that is one bit shorter. Therefore, the total cost is $\Theta(\log n)$ multiplications in G.

This establishes that the scheme can be run in polynomial time, but thus far we have not made any assumptions about the nature of the group G for the Diffie–Hellman key agreement scheme to be secure; this will be discussed in the following section.

2.1.2 – Hard problems for DH. The strongest conceivable attack against Diffie–Hellman would fully recover the shared secret g^{ab} after observing the public data (g^a, g^b):

Definition 2.2. *The* computational Diffie–Hellman problem *(CDH) in a cyclic group $G = \langle g \rangle$ is to compute the map*

$$\mathrm{dh}_g \colon\ G^3 \to G,\ (g, g^a, g^b) \mapsto g^{ab}.$$

The CDH assumption *is: Given uniformly random inputs in $\{g\} \times G^2$, no efficient algorithm can solve CDH, i.e., compute dh_g, with more than negligible probability.*[3]

On the other hand, one can ask much less from an attacker, such as to learn just a particular function (e.g. one bit of an encoding) of the shared secret g^{ab}. The assumption that this is hard for all "reasonable" properties can be overapproximated by the assumption that no attacker can even tell whether a given element is *likely* to be g^{ab} or not:

Definition 2.3 (DDH)**.** *The* decisional Diffie–Hellman problem *in a cyclic group $G = \langle g \rangle$ is to determine whether a given triplet $t \in G^3$ is of the form (g^a, g^b, g^{ab}).*

The DDH assumption *is: On input triplets sampled uniformly from (with probability $1/2$ each) either the graph of dh_g or the entire set G^3, no efficient algorithm can solve DDH with probability non-negligibly greater than $1/2$.*

The hardness of the decisional Diffie–Hellman problem is an extremely useful assumption for cryptographic applications,[4] but the details would lead us too far astray during this overview.

The obvious way for an attacker Eve to solve CDH (or DDH, for that matter) is to try to recover Alice's or Bob's private key from the public keys, which allows Eve to recompute the same shared secret:

[2] Typically, "efficient" means polynomial in $\log q$, but there may be cases where higher complexities make sense.

[3] We deliberately refrain from instantiating the words "efficient" and "negligible" with concrete semantics in this section, since depending on context they may range from asymptotic statements, where one must consider *families* of groups G, to explicit bounds on the number of arithmetic operations and success probabilities for a fixed group G.

[4] "The Decision Diffie–Hellman assumption (DDH) is a gold mine." [Bon98]

Definition 2.4 (DLP)**.** *The* discrete-logarithm problem *in a cyclic group $G = \langle g \rangle$ of order q is to compute the inverse*

$$\log_g \colon\ G \to \{0, ..., q-1\}$$

of the exponentiation map

$$\exp_g \colon\ \{0, ..., q-1\} \to G,\ x \mapsto g^x\,.$$

The best possible classical attacks on DLP in the *generic-group model*, i.e., using only the interface provided by an abstract group without getting access to the implementation of the group as bit strings and algorithms, are known to take time $\Theta(\sqrt{q})$ when q is prime [Sho97b]. This complexity is achieved by *meet-in-the-middle* techniques, better known as *baby-step giant-step algorithm* in this context following [Sha71]: On input $h \in G$, set $m = \lceil\sqrt{q}\rceil$; build a table of all g^j for $j \in \{0, ..., m-1\}$; look up each $g^{-mi} \cdot h$ for $i \in \{0, ..., m-1\}$; when a match is found, the value $mi + j$ solves the DLP. For composite q, one can do better: The *Pohlig–Hellman algorithm* projects the DLP to successive prime-order subgroups and lifts the individual solutions back to $\mathbb{Z}/q$ using the Chinese remainder theorem and Hensel lifting for prime powers, which implies that the complexity is (up to factors polynomial in $\log q$) dominated by the largest prime factor of q.[5] Of course, sometimes all of these problems are easier than in the generic-group model; for example, let G be the encoding[6] of $\mathbb{Z}/q$ as $\{0, ..., q-1\}$.

We stress that it is *not* generally known whether solving DLP is the best way to break CDH: There could be shortcuts to compute $(g^a, g^b) \mapsto g^{ab}$ that do not first recover a or b. For group-based Diffie–Hellman, a line of work started by den Boer in 1988 for finite fields and generalized by Maurer in 1994 yields equivalence results for DLP and CDH under assumptions that certain "nice" algebraic groups exist; such groups have since been explicitly constructed for many Diffie–Hellman parameters used in practice, thus the resulting bound on the hardness gap between DLP and CDH for these groups mostly depends on the success probability of an assumed CDH solver. This thesis contains a result on the analogous question for group *actions*; see Chapter 6.

2.1.3 – Group-action Diffie–Hellman. As explained in Chapter 1, the biggest problem with DLP is that it can be solved efficiently using Shor's algorithm (see Section 2.6.2) once a sufficiently large quantum computer is available. This has prompted the search for a replacement structure that shares enough of the traits of group exponentiation to be cryptographically useful, while at the same time lacking the aspects that introduce weakness against quantum algorithms. As we will see in Section 2.6.2, a core part of Shor's algorithm to break DLP is the algebraic composition operation $(g^x, g^y) \mapsto g^{x+y}$ on *public* keys, simply given by group multiplication. However, it seems that using a structure which still admits an exponentiation-like function while not having an efficient, algebraically meaningful composition law on public keys might improve the quantum security:

Definition 2.5. *A* group-action Diffie–Hellman scheme *consists of a finite abelian group G acting on a finite set X,*[7] *together with a fixed element $x_0 \in X$ and an efficient sampling algorithm $\mathcal{S}$ returning*

[5] In contrast, the complexity of the best generic *DDH* solver is dominated by the *smallest* prime factor p of q: Project the given DDH triplet to the small subgroup and check there using a generic DDH algorithm. False positives are possible, but the probability of mislabeling a random tuple as a DH tuple is only $1/p$.

[6] As all cyclic groups of order q are isomorphic to $\mathbb{Z}/q$, the difficulty of these problems must really lie in the specific choice of representation for abstract (isomorphism classes of) groups as bit strings and multiplication algorithms.

[7] A group action of G on X is a map $*\colon G \times X \to X$ such that $1 * x = x$ and $(g \cdot h) * x = g * (h * x)$ for all $g, h \in G$ and $x \in X$. (Modulo syntax, this can equivalently be viewed as a group homomorphism $G \to \mathrm{Sym}(X)$.)

elements of G.[8] *We require that computing the action $*\colon G \times X \to X$ is efficient for the elements returned by $\mathcal{S}$, and that elements of X have efficient unique encodings as bit strings.*

With these data, the group-action Diffie–Hellman key agreement works as follows:

Key generation. *Alice picks a group element $a \leftarrow \mathcal{S}()$; this is her private key. Her public key is the element $x_A := a * x_0 \in X$. Bob proceeds analogously to generate his key pair $(b, x_B) \in G \times X$.*

Key agreement. *Alice takes Bob's public key x_B and her private key a and computes $x_S = a * x_B$. Similarly, Bob computes $S = b * x_A$. Hence, $S = ab * x_0$ is shared among Alice and Bob.*

The straightforward group-action generalizations of the discrete-logarithm and computational Diffie–Hellman problems (Definitions 2.4 and 2.2) are known as *vectorizaton* and *parallelization* problems, the terminology being inspired by visualizing a group action as the special case of a vector space acting on an affine space. See also Chapter 6, and [Smi18] for a much more in-depth discussion of the similarities and differences between groups and group actions in (pre- and post-quantum) cryptography.

Note that group-based Diffie–Hellman is very close to being a special case of Definition 2.5: Letting X be the Diffie–Hellman group and x_0 its chosen generator, the multiplicative group $G = (\mathbb{Z}/q)^\times$ acts on X by exponentiation; i.e., given $(\overline{e}, x) \in G \times X$, pick any representative $e \in \mathbb{Z}$ of the residue class $\overline{e}$ and output x^e. Thus, if the private keys are restricted to integers coprime to q, then Diffie–Hellman is indeed an instance of Definition 2.5. In the common case that q is prime, this means excluding only 0 as a private key.

However, this instantiation of course does not achieve the goal of improving the quantum resistance, since the multiplication $X \times X \to X$ still exists and can be used in Shor's algorithm. Chapter 3 of this thesis introduces *CSIDH*, a relatively efficient instance of group-action Diffie–Hellman that appears to withstand Shor's algorithm. Instead, the best known quantum attack is a subexponential-time quantum algorithm for the *abelian hidden shift problem* (see Section 2.6.3).

2.1.4 – Groups vs. actions. Note that DLP-based systems seem to support different applications than group-action-based systems: For instance, verification of *Schnorr signatures* relies on computing the expression $g^s \cdot h^m$, which does not translate to group actions in any obvious way since the operation $\cdot$ is lacking. On the other hand, anything relying on DLP is immediately unsuitable for post-quantum applications due to Shor's algorithm (Section 2.6.2). Note that it is the very same operation $(g^x, g^y) \mapsto g^{x+y}$ that makes the signature scheme as well as the quantum attack work, which suggests that a straightforward adaption of Schnorr signatures and many other applications to the post-quantum setting may be difficult. Hence, signature schemes based on group actions are currently much slower than pre-quantum, DLP-based signatures. Chapter 4 contributes to changing this by speeding up the *SeaSign* signature scheme based on the CSIDH group action (Chapter 3).

While some features of isogeny-based cryptography remain unattractive compared to other cryptosystems, most importantly speed, there are applications where isogenies are the best available option: One such case is *non-interactive key exchange*, a piece of functionality instantiated *pre-quantumly* with Diffie–Hellman, but for which only few post-quantum candidates exist.

2.1.5 – Non-interactive key exchange. The qualifier *non-interactive* refers to the property that two parties can each compute their public key entirely on their own, and two public keys

[8] For the scheme to stand any chance at being secure, the distribution of group elements returned by $\mathcal{S}$ must have sufficient min-entropy, the ideal case being that $\mathcal{S}$ returns uniformly random elements of G. As this is sometimes tricky to do in instantiations (see for example Section 3.4), we allow a bit more flexibility in how the sampling is done.

together determine a unique shared secret that the parties can compute separately, without any need to exchange additional (often randomized) data in real time:

Definition 2.6. *A* non-interactive key exchange *(NIKE) scheme consists of sets K, X, Y, an efficient sampling algorithm $\mathcal{S}$ returning elements of K, and efficient (potentially randomized) algorithms*

$$\mathcal{G}\colon K \to X \quad \text{and} \quad \mathcal{F}\colon K \times X \to Y\,,$$

such that for all $a, b \leftarrow \mathcal{S}()$ and $A \leftarrow \mathcal{G}(a)$, $B \leftarrow \mathcal{G}(b)$, we have $\mathcal{F}(a, B) = \mathcal{F}(b, A)$.

For security, we require that recovering the secret a given A and oracle access to $\mathcal{F}(a, -)$ is hard. One particular requirement for a NIKE to be secure is that participants can reuse public keys many times without changing them; in particular, Alice and Bob must not reveal partial information about their secrets when processing a key exchange.

The description of Diffie–Hellman schemes above does not consider the possibility that one of the participants (or an attacker manipulating data on the wire) might deliberately try to trick the other into revealing information about their long-term private key. In particular, it is silently assumed that inputs to the group algorithms are in fact encodings of valid group elements; in actual implementations, one must thus consider what happens when data outside the expected set is received: Attackers may send malformed inputs to exploit potential faulty behaviour of algorithms on these bad inputs, and thereby leak information about secrets.

For example, *small-subgroup attacks* pose a threat to most group-based DH instantiations: they are based on sending fake public-key elements on which the group algorithms will "work" even though they lie outside the actual DH group, and interpreting the reaction of the recipient allows deducing information about their private key modulo the size of that subgroup. Combining this kind of leakage from different subgroups can lead to recovery of the entire secret. Another example is described in Chapter 10 of this thesis: Lattice-based Diffie–Hellman-like schemes with noise may be vulnerable to tinkering with certain parts of the noise until the key exchange fails, and the threshold where this occurs is correlated with the recipient's private key.

If possible, the easiest way to avoid invalid inputs is to simply test each input for the desired properties: For instance, checking that a group element has some prescribed large prime order can be done using a single group exponentiation and two identity tests. Unfortunately, testing validity of public keys is not always as easy: Key-exchange schemes with more complex mathematical structure usually require some relationship between different components of the public data, and tampering with some parts individually is often impossible to detect without breaking the scheme in the first place.

On the bright side, the CSIDH scheme (and its predecessors) presented in Chapter 3 fits into the framework of Definition 2.6 and offers very easy and cheap public-key validation (see Section 3.5), which makes it a viable candidate NIKE, and the current best incarnations drastically outperform what appears to be the only other post-quantum NIKE proposal [AJL17].

Most lattice-based key exchanges, on the other hand, are hardened against active attackers using the *Fujisaki–Okamoto transform* (FO), which essentially consists of sending an encryption of the sender's own private key as the very first message post-key-exchange. The recipient then recomputes the sender's public key according to the alleged private key, and aborts if the result does not match the key used in the actual key exchange. However, this procedure merely leads to a *key-encapsulation mechanism* (KEM), in which one party uses an ephemeral (one-time) key pair, and hence does not give a NIKE. See Chapter 10 for more details on the FO transform, as well as a concrete attack on a scheme that lacks it.

2.2 — Elliptic curves

This thesis is about the use of isogenies of elliptic curves in cryptography. As one literally cannot spell *isogenies of elliptic curves* without *elliptic curves*, let us discuss those first.

Standard references for the contents of this section and much more in-depth background are: Washington's *Elliptic Curves: Number Theory and Cryptography* [Was08], a gentle introduction with focus on algorithms and applications; Silverman's *The Arithmetic of Elliptic Curves* [Sil09], a comprehensive summary from a more mathematical perspective; and Hartshorne's *Algebraic Geometry* [Har77], which starts from scheme theory and treats elliptic curves as a special case.

Definition 2.7. *Let k be a field. An* elliptic curve *over k is a pair (E, O), where E is a smooth projective genus-one curve over k and O is a k-rational point on E, the* base point.

Throughout this chapter, k refers to a field and E, E' are elliptic curves over k unless noted otherwise. The base point O is usually omitted. We write E/k for "E is defined over k".

2.2.1 – Weierstraß curves. In practice, we can immediately replace this general, but rather abstract definition by concrete equations, giving a more elementary and tangible perspective of elliptic curves. Note that there are various choices for such equations (often referred to as curve "models"), each having different computational benefits depending on context. Traditionally, most of the theory is developed on *Weierstraß curves*, but note that modern cryptographic applications typically work with *Montgomery* or *Edwards* curves (see Section 2.2.6).

Definition 2.8. *A* short Weierstraß curve *over a field k is a projective curve defined by an equation*

$$Y^2Z = X^3 + aXZ^2 + bZ^3 \tag{$*$}$$

with $a, b \in k$, such that the discriminant $\Delta := -16(4a^3 + 27b^2)$ *is non-zero.*[9] *The unique point $[0:1:0]$ with $Z = 0$ is called the* point at infinity *and denoted by ∞.*

For brevity, Weierstraß curves are instead often written as affine curves

$$y^2 = x^3 + ax + b \tag{2.1}$$

with the implicit understanding that the projective closure ($*$) is meant. In particular, the point at infinity ∞ is retained as a "point" on the curve even though it does not correspond to a solution of (2.1); see Definition 2.10 below.

Proposition 2.9. *Let $\mathrm{char}(k) \notin \{2, 3\}$ and let (E, O) be an elliptic curve over k. Then E is isomorphic (as an algebraic curve) over k to a short Weierstraß curve defined over k, such that O corresponds to ∞ under the isomorphism. In particular, if E is a short Weierstraß curve, then (E, ∞) is an elliptic curve.*

Note that short Weierstraß curves do not exist in characteristic 2, and in characteristic 3 they fail to cover all isomorphism classes of elliptic curves. In arbitrary characteristic, results similar to Proposition 2.9 hold with slightly more complicated formulas — *long* Weierstraß equations. We work mainly in large characteristic and will hence often omit the attribute "short" when talking about short Weierstraß curves. Inspired by Proposition 2.9, we write ∞ for the the base point on any elliptic curve.

Definition 2.10. *The set of* points *on a short Weierstraß curve $E\colon y^2 = x^3 + ax + b$ is the set of pairs $(x, y) \in \overline{k} \times \overline{k}$ satisfying the Weierstraß equation, together with a single extra point denoted ∞.*

For any field extension K/k, we write $E(K)$ for the subset of points on E defined over K, i.e., with coordinates in K. The point at infinity ∞ is defined over k. Referring to E as a set is shorthand for $E(\overline{k})$.

[9] For $\Delta = 0$, the resulting curve is not smooth.

2.2.2 – The j-invariant. How many isomorphism classes of elliptic curves are there?

Proposition 2.11. *Two elliptic curves are isomorphic over $\bar{k}$ if and only if they have the same j-*invariant*, which for a Weierstraß curve $E\colon y^2 = x^3 + ax + b$ is given by the formula*

$$j(E) = 1728 \cdot 4a^3/(4a^3 + 27b^2)\,.$$

Conversely, for any j-invariant in k, we can recover an explicit curve equation over the same field: For $\operatorname{char}(k) \notin \{2, 3\}$, we may use the short Weierstraß equation

$$E\colon y^2 = x^3 - 3j(j - 1728)x - 2j(j - 1728)^2\,.$$

We also often need the notion of a *function* on an (elliptic) curve: These are simply rational expressions in some coordinates on the curve, modulo the defining equation(s):

Definition 2.12. *Let E be an elliptic curve defined over k. The* function field *of E, denoted $k(E)$, is the set of rational functions $E \to k$. In particular, if E is a Weierstraß curve $y^2 = x^3 + ax + b$, then*

$$k(E) \;\cong\; k(x, y)/(y^2 - x^3 - ax - b)\,.$$

Note that a function on E need not necessarily be defined at all points of E; in fact, every non-constant function on a projective curve has at least one pole. Thus, the evaluation of a function at points of E is sometimes better viewed as a map to $\mathbb{P}^1(k)$, rather than $\mathbb{A}^1(k) = k$. The poles are mapped to the point "at infinity", i.e., the unique point in $\mathbb{P}^1(k)\backslash\mathbb{A}^1(k)$.

2.2.3 – The group structure. Perhaps the main reason elliptic curves have been studied in depth for centuries is that they carry a geometric group structure, a fact that makes them stand out among most other algebraic curves and certainly helps explain why the (a priori) relatively arbitrary-looking equation (2.1) is interesting.

Proposition 2.13. *Let (E, ∞) be an elliptic curve. There is a unique abelian group structure on the set of points of E such that the neutral element is ∞ and the composition law is given by rational maps.*[10]

On a Weierstraß curve, the group operation admits a nice geometric interpretation: The sum of three points on the curve equals ∞ if and only if there exists a straight line intersecting the curve in these points with the correct multiplicities.[11]

Definition 2.14. *For any $\ell \in \mathbb{Z}$, let $[\ell]\colon E \to E$ be the (scalar-)multiplication-by-ℓ homomorphism on E, defined by adding together ℓ copies of a point. The kernel of $[\ell]$ is denoted by $E[\ell]$ and called the* ℓ-torsion subgroup *of E.*

As is customary when finding a group somewhere, we immediately feel a burning desire to learn about its structure, which in this case is quite easy to describe:

Proposition 2.15. *Let E/k be an elliptic curve and ℓ a non-zero integer. If $\operatorname{char}(k) = p > 0$, factor ℓ as $m \cdot p^r$ with $m \notin p\mathbb{Z}$; otherwise, let $m = \ell$. Then as groups*

$$E[\ell] \;\cong\; \mathbb{Z}/m \times \mathbb{Z}/\ell \quad \textit{or} \quad E[\ell] \;\cong\; \mathbb{Z}/m \times \mathbb{Z}/m\,.$$

In particular, either $E[p] \cong \mathbb{Z}/p$ or $E[p] \cong \{0\}$, and if $\operatorname{char}(k) \nmid \ell$ then $E[\ell] \cong \mathbb{Z}/\ell \times \mathbb{Z}/\ell$.

[10] Thus, E is an *abelian variety* of dimension one — another common definition of elliptic curves.

[11] The point ∞ lies on every vertical line. When the points in question are not distinct, the condition on multiplicities implies that the straight line must be a tangent of the curve.

Therefore, the ℓ-torsion of an elliptic curve is almost always a (2-dimensional) *torus* over $\mathbb{Z}/\ell$. This is not a coincidence: In fact, it is a classical result that elliptic curves defined over $\mathbb{C}$ are analytically isomorphic to complex tori, which is one way to arrive at the above structure result.

The p-torsion has a large impact on further structural properties of an elliptic curve, hence some terminology is in order:

Definition 2.16. *An elliptic curve E/k is called* supersingular *if $p = \mathrm{char}(k) > 0$ and $E[p] \cong \{0\}$; all other elliptic curves (in particular those in characteristic zero) are called* ordinary.[12]

For computations on elliptic curves, the field of definition of the points we are working with is crucial, hence let us discuss the sizes of rational subgroups.

2.2.4 – Point counting. The first fundamental result is that an elliptic curve defined over $\mathbb{F}_q$ has approximately as many rational points as a one-dimensional set "morally" ought to: about q. In fact, this estimate is (almost) correct with a square-root error bound:

Hasse's Theorem 2.17 [Has36]**.** *Let $E/\mathbb{F}_q$ be an elliptic curve. Then*

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

with

$$|t| \leq 2\sqrt{q}\,.$$

Determining the exact number of rational points is significantly more difficult, and all naïve strategies have exponential (in $\log q$) cost. Fortunately, Schoof discovered a clever algorithm that is much more efficient:

Theorem 2.18 [Sch85]**.** *There is an explicit[13] algorithm which, given the coordinates of a (long) Weierstraß curve E over $\mathbb{F}_q$, computes the number of $\mathbb{F}_q$-rational points on E in time polynomial in* $\log q$.

Sometimes, it is easier to determine the number of points, such as when E is defined over a subfield — it then suffices to compute the number of points over the smallest field of definition:

Proposition 2.19. *Let $E/\mathbb{F}_q$ be an elliptic curve and $\#E(\mathbb{F}_q) = q + 1 - t$. If $\alpha, \beta \in \mathbb{C}$ are the two (not necessarily distinct) complex roots of the polynomial $X^2 - tX + q \in \mathbb{Z}[X]$, then for any $n \geq 1$,*

$$\#E(\mathbb{F}_{q^n}) = q^n + 1 - (\alpha^n + \beta^n)\,.$$

Moreover, supersingular curves can only have a few possible group orders, and conversely supersingularity may be detected by point counting:

Proposition 2.20. *Let $E/\mathbb{F}_q$ be an elliptic curve, where $q = p^r$. Then E is supersingular if and only if p divides $t = q + 1 - \#E(\mathbb{F}_q)$. Furthermore, if $q = p \geq 5$, this is equivalent to $\#E(\mathbb{F}_p) = p + 1$.*

The upshot of Proposition 2.20 is that the group order of supersingular elliptic curves can easily be controlled by choosing an appropriate base-field prime — one of the main motivations for their use in isogeny-based cryptography. Another fact with useful implications is that all supersingular elliptic curves can (up to isomorphism) be defined over a small extension of the prime field, and moreover we have very explicit and tight bounds on the number of such curves:

[12] Note the potentially confusing terminology: Supersingular elliptic curves are *not* singular; rather, the word should be interpreted to mean "very rare", referring to the fact that supersingular curves are fairly sparse.

[13] The (sometimes muddled) distinction between *having* an algorithm and merely knowing abstractly that it *exists* is important even in practice: Ignoring arbitrarily costly precomputation to *find* the algorithm is formalized in the so-called *non-uniform* model of computation, whose relevance to reality (and cryptography in particular) is debatable. See [BL13].

Proposition 2.21. *Let E be a supersingular elliptic curve defined over a field k of characteristic $p > 0$. Then $j(E) \in \mathbb{F}_{p^2}$. In particular, E is isomorphic (over $\bar{k}$) to a curve defined over $\mathbb{F}_{p^2}$.*

In characteristic p, there are exactly $\lfloor p/12 \rfloor + \varepsilon$ supersingular j-invariants, where $\varepsilon \in \{0, 1, 2\}$.

2.2.5 – Elliptic-curve cryptography. With all the tools we have available now, it is trivial to instantiate the Diffie–Hellman key agreement with an elliptic-curve group: Everyone simply agrees on an elliptic curve E over a finite field $\mathbb{F}_q$ and a point $P \in E(\mathbb{F}_q)$ of large prime order and then proceeds exactly as described in Section 2.1.1.[14]

2.2.6 – Alternate curve models. For computations, the Weierstraß form of elliptic curves is often not the best choice, as evaluating the addition formulas involves inconvenient case distinctions (which in turn lead to side-channel risks in cryptographic implementations). The two most commonly used alternatives are the *Montgomery* and *Edwards* forms, which are available (over the same field) for sizeable subsets of all elliptic curves:

Definition 2.22. *A* Montgomery curve *over a field k of characteristic $\neq 2$ is a projective curve defined by the affine equation*

$$By^2 = x^3 + Ax^2 + x$$

with $A, B \in k$ and $B(A^2 - 4) \neq 0$. In some cases (such as in Chapter 3), one can always choose $B = 1$, and we then refer to A as the Montgomery coefficient *of the curve.*

Montgomery curves do not solve the problem of exceptional cases in the addition law known from Weierstraß curves. However, they do offer extremely clean and efficient formulas for computations on the *x-line* or *Kummer line* of a Montgomery curve, which as a variety simply equals the quotient $X = E/\{\pm 1\}$. Note that while we do lose the addition law when quotienting by $[\pm 1]$, the scalar-multiplication operation which lies at the heart of elliptic-curve Diffie–Hellman commutes with $[\pm 1]$, hence is well-defined on X. It can be implemented very efficiently using the *ladder step*

$$\texttt{DBLADD}\colon (P, Q, P - Q) \mapsto ([2]P, P + Q)$$

which is well-defined on the Kummer line X and can be used as a building block inside scalar-multiplication algorithms such as the *Montgomery ladder*. See [BL17] or [CS18] for more details on Montgomery curves.

Definition 2.23. *A* (twisted) Edwards curve *over a field k of characteristic $\neq 2$ is an affine curve defined by the equation*

$$ax^2 + y^2 = 1 + dx^2y^2$$

with $a, d \in k$ and $ad(a - d) \neq 0$.

Edwards curves, just like Montgomery curves, offer very efficient arithmetic on the Kummer line, but in addition they admit *complete addition formulas*: That is, any two points can be added by evaluating the very same rational functions, with no exceptional points and no case distinction. Hence, they are very useful to build efficient side-channel resistant implementations of elliptic-curve cryptosystems that require individual point additions, rather than just scalar multiplications. For more details about Edwards curves, see [BL17] or [BL07].

[14] Note that elliptic curves are usually written additively, whereas Section 2.1.1 uses multiplicative notation.

2.3 — Isogenies of elliptic curves

Having shown the basics of elliptic curves — an eminent building block in cryptography in its own right — in the previous section, it is now time to introduce the central object in this thesis:

Definition 2.24. *An* isogeny *between two elliptic curves $E, E'/k$ is a non-zero rational map*

$$\varphi\colon E \longrightarrow E'$$

which is also a group homomorphism. An isogeny is defined over k *if it can be written using rational functions in $k(E)$, i.e., as fractions of polynomials with coefficients in k. Two curves E, E' are called* isogenous *whenever there exists an isogeny $E \to E'$.*[15]

Let $\mathrm{Hom}_k(E, E')$ denote the set of all isogenies $E \to E'$ defined over k, together with the constant morphism $0\colon E \to E',\ P \mapsto \infty$. For brevity, write $\mathrm{Hom}(E, E') := \mathrm{Hom}_{\overline{k}}(E, E')$. These sets carry an abelian group structure given by point-wise addition $(\varphi + \psi)(P) = \varphi(P) + \psi(P)$.

It is worth mentioning that Definition 2.24 is not what some algebraic geometers prefer, as it generalizes relatively poorly to abelian varieties of higher dimension. However, it appears that this definition is the most natural for the purposes of this thesis, as we will be working exclusively with elliptic curves in any case — and the fact that isogenies are group homomorphisms (often derived as a theorem in other literature) is crucial from an applied perspective.

For concreteness, we specialize the definition to short Weierstraß curves:

Proposition 2.25. *Let E, E' be short Weierstraß curves over k and $\varphi\colon E \to E'$ an isogeny defined over k. Then there exist polynomials $f, g, h \in k[x]$ such that for all $P = (x_P, y_P) \in E$,*

$$\varphi(P) = \left(\frac{f}{h^2}(x_P),\ y_P \cdot \frac{g}{h^3}(x_P)\right)$$

where defined, and $\varphi(P) = \infty$ if P is a pole of f/h^2 or g/h^3.

To measure the algebraic (and, as we shall see, computational) complexity of an isogeny, we introduce its *degree*: essentially the lowest possible degree of a polynomial expression (as shown in Proposition 2.25) to write down the isogeny.

Fact 2.26. *Let $\varphi\colon E \to E'$ be an isogeny of elliptic curves over k. The* pullback

$$\varphi^*\colon k(E') \hookrightarrow k(E),\ f \mapsto f \circ \varphi$$

embeds the function field $k(E')$ as a subfield of the function field $k(E)$.[16] *The* degree *of the isogeny φ is defined to be the degree $[k(E) : k(E')]$ of this extension. Note that degrees are multiplicative since degrees of field extensions are: $\deg(\varphi \circ \psi) = \deg(\varphi) \cdot \deg(\psi)$.*

Definition 2.27. *An* isomorphism *of elliptic curves is an isogeny of degree one. An* automorphism *is an isomorphism from a curve to itself.*

A core fact about isogenies (often used as a defining property) is that their kernels are finite: only finitely many points are mapped to ∞. Isogenies are classified according to the relationship between the degree and the cardinality of the kernel:

Definition 2.28. *An isogeny φ is called* separable *if $\#\ker(\varphi) = \deg(\varphi)$, else it is* inseparable. *Moreover, φ is* purely inseparable *if $\ker(\varphi)$ is trivial.*[17]

[15] See Proposition 2.34 as to why being isogenous is an equivalence relation.

[16] The connection from isogenies to function field extensions goes much further; in fact, the pullback construction is a functor inducing an contravariant equivalence of categories to its image.

[17] Fun fact: Isomorphisms — isogenies of degree 1 — are both separable and purely inseparable, but not inseparable.

Inseparable isogenies are the exception rather than the norm: In a sense, there is really only one source of inseparability, and "most" isogenies are separable.

Proposition 2.29. *In characteristic zero, all isogenies are separable. In positive characteristic $p > 0$, every isogeny φ admits a unique decomposition*

$$\varphi = \psi \circ \pi_p^r$$

with ψ separable and $\pi_p \colon (x, y) \mapsto (x^p, y^p)$ the (p-power) Frobenius isogeny. Thus, powers of π_p are (up to composition with isomorphisms) the only purely inseparable isogenies.

As mentioned above, every isogeny has a finite kernel subgroup. It is therefore a natural question how much the correspondence works in the other direction: Is every finite subgroup of an elliptic curve the kernel of a (unique) isogeny?[18] Existence will follow from Proposition 2.31, but inseparable isogenies immediately show that we cannot hope for uniqueness. Fortunately, this is the only failure, and so we get a one-to-one correspondence between finite subgroups and *separable* isogenies (up to isomorphism):

Proposition 2.30. *Let E be an elliptic curve over a perfect field k and $H \leq E$ a finite subgroup defined over k.*[19] *Then there exists an elliptic curve E' and a separable isogeny $\varphi_H \colon E \to E'$, both defined over k, such that $\ker(\varphi_H) = H$. The pair (E', φ_H) is unique up to k-isomorphism of E'.*

By analogy with the homomorphism theorem for groups, the curve E' is denoted by E/H.

Note that φ_H is often referred to as "*the* isogeny with kernel H", since the technically correct phrasing "a separable isogeny with kernel H, up to post-composition with k-isomorphisms" is quite a mouthful.

2.3.1 – Computing isogenies. We have seen that abstractly, isogenies are (more or less) determined by their kernels, but of course this implies nothing about *computing* an isogeny from its kernel. This is known, too: By now, there is a variety of algorithms to compute isogenies, each with its own constraints and applications. All of them are based in one way or another on the following result from the seventies:

Proposition 2.31 (Vélu's formulas [Vél71])**.** *Let $E \colon y^2 = x^3 + ax + b$ be a Weierstraß curve over a field k and $H \leq E$ a finite subgroup. For any function $\pi \in k(E)$ and point $P \in E$, define*

$$f_\pi(P) := \pi(P) + \sum_{\substack{Q \in H \\ Q \neq \infty}} \big(\pi(P+Q) - \pi(Q)\big).$$

Let $x, y \in k(E)$ be the Weierstraß coordinate functions on E. Then the map

$$\varphi \colon E \to E/H,\ P \mapsto \big(f_x(P), f_y(P)\big),$$

where poles of f_x, f_y get mapped to the point at infinity, is a separable isogeny with kernel H. The codomain is a Weierstraß curve, whose equation can easily be recovered as well using a few more operations.

Conceptually, these formulas first represent the quotient group E/H simply as cosets $P + H$ with $P \in E$, then exploit the coordinate projections x, y of the domain curve E to construct

[18] This is an analogue in the isogeny setting of the classical homomorphism theorem $G/\ker(f) \cong \operatorname{im}(f)$ for groups.

[19] That is, field automorphisms σ of $\overline{k}$ which fix k map H to itself. Note that *not* necessarily $H \subseteq E(k)$; rather, every σ acts as a permutation on H.

functions in $k(E)$ invariant under translations by the desired kernel subgroup H. In other words, this yields well-defined functions $f_x, f_y \in k(E/H)$, which are subsequently used as coordinate functions on the codomain E/H to set up an embedding into the plane.

2.3.2 – Smooth degrees. It is not hard to see from Proposition 2.31 that naïvely evaluating Vélu's formulas takes $\Theta(|H|)$ operations, i.e., the cost is exponential in $\log(\deg(\varphi))$. Can we do better? In many cases, the answer is yes. Observe that we can immediately reduce the problem to prime-degree isogenies:

Lemma 2.32. *Let $H \leq E$ be a finite subgroup of an elliptic curve E. For any subgroup $H' \leq H$, the isogeny $\varphi\colon E \to E/H$ with kernel H can be decomposed as*

$$\varphi\colon\ E \xrightarrow{\psi'} E/H' \xrightarrow{\psi} E/H$$

where $\ker(\psi') = H'$ and $\ker(\psi) = \psi'(H) \leq E/H'$.

Corollary 2.33. *Let k be a field and suppose that an isogeny of prime degree ℓ can be computed in $T(\ell)$ arithmetic operations in k. Let E/k be an elliptic curve and $H \leq E(k)$ a finite subgroup, and suppose the prime factorization $|H| = \prod_{i=1}^{r} \ell_i^{e_i}$ is given. Then the isogeny $\varphi\colon E \to E/H$ can be computed in*

$$O\left(\sum_{i=1}^{r} e_i \cdot T(\ell_i)\right) \cdot \operatorname{polylog}(|H|)$$

arithmetic operations in k.

Note that naïvely evaluating Proposition 2.31 shows the upper bound $T(\ell) \in O(\ell)$. Using elliptic resultants, the "$\sqrt{}$élu" algorithm [BDLS20] achieves a (very close to) square-root speedup over this naïve strategy, yielding the best known result $T(\ell) \in \tilde{O}(\sqrt{\ell})$. See also Section 11.6.

2.3.3 – Defined kernels. Since cryptosystems rely on the hardness of finding isogenies, we have to compute isogenies of extremely large degrees. To do so efficiently, we exploit the shortcut provided by Corollary 2.33: In practice, all isogenies we compute from their kernel have smooth degree.[20]

However, there is another problem: The complexities above are given in terms of operations *in k*, but in general, a subgroup of E/k of large size ℓ is only defined over an extension field of exponential (in $\log \ell$) degree over k. Thus, for isogenies of sufficiently large (smooth or not) degree, even *writing down the kernel* is impossible in practice! Two tricks are employed to avoid this issue. First, one may use special curves whose group orders over a small enough field are amenable to nice, smooth-degree isogenies: In (current) practice, this usually means supersingular curves defined over (at most) $\mathbb{F}_{p^2}$ with a prime of the form $p = s - 1$ where s is smooth. The second technique in principle applies to any base field and curve, but is much less efficient in practice: Using isogenies of *powersmooth* degree $\prod_{i=1}^{r} \ell_i^{e_i}$ allows decomposing the kernel H as an internal direct product $H_1 \times \cdots \times H_r \subseteq H$, where each $|H_i| = \ell_i^{e_i}$ is small, hence each subgroup H_i is defined over only a small extension. Therefore, the isogeny can be computed as a chain $E \to E/H_1 \to (E/H_1)/H_2 \to \cdots \to E/H$, where the kernel of the i^{th} step is computed by pushing H_i through all previous isogeny steps. What this gains is that all computations can now be performed in fields big enough to contain each pair of subgroups $H_i \times H_j \subseteq H$, rather than (previously, without decomposing) the entire group H at once.

[20] An integer is *B-smooth* if it has no prime factor larger than B. Similarly, an integer is *B-powersmooth* if it is not divisible by a prime *power* larger than B.

2.3.4 – Isogeny graphs. An extremely useful (and mathematically pleasing) property of isogenies is that every isogeny comes with a natural complementary isogeny in the opposite direction, the *dual isogeny*. The dual behaves somewhat like an inverse, modulo a scalar multiplication by the degree:

Proposition 2.34. *Every isogeny φ has a unique* dual isogeny $\widehat{\varphi}\colon E' \to E$ *with the property that $\widehat{\varphi} \circ \varphi = [\deg(\varphi)]\colon E \to E$ and $\varphi \circ \widehat{\varphi} = [\deg(\varphi)]\colon E' \to E'$. The dual isogeny abides by the rules*

$$\begin{aligned} \widehat{\widehat{\varphi}} &= \varphi; \\ \widehat{\varphi + \psi} &= \widehat{\varphi} + \widehat{\psi} \quad \text{(for } \psi \neq -\varphi\text{)}; \\ \widehat{\varphi \circ \psi} &= \widehat{\psi} \circ \widehat{\varphi}. \end{aligned}$$

The kernel of the dual $\widehat{\varphi}$ equals $\varphi(E[\deg(\varphi)]) \leq E'$.

Among (many) other things, the existence of the dual isogeny implies that being isogenous is an equivalence relation. How hard it is to decide whether two curves are isogenous?

Tate's Isogeny Theorem 2.35 [Tat66]**.** *Two elliptic curves E, E' defined over a finite field $\mathbb{F}_q$ are isogenous over $\mathbb{F}_q$ if and only if $\#E(\mathbb{F}_q) = \#E'(\mathbb{F}_q)$.*

While Tate's theorem equips us with a neat, easily checkable criterion to determine whether two curves are isogenous, it reveals no information at all about the isogeny whose existence is established. To learn more about the nature of the connecting isogenies between different curves known to be isogenous, the structure of *isogeny graphs* is studied:

Definition 2.36. *Let k be a field and $\mathcal{S}$ a set of positive integers (often a single prime) not divisible by* $\mathrm{char}(k)$*. Define the* $\mathcal{S}$-isogeny graph $G_{k,\mathcal{S}}$ over k *as follows:*

- *Nodes: Elliptic curves defined over k, up to k-isomorphism.*
- *Edges: Isogenies of degree $\ell \in \mathcal{S}$ defined over k, up to post-composition with k-isomorphisms.*

When $\mathcal{S} = \{\ell\}$, we write $G_{k,\ell}$ for the ℓ-isogeny graph.

Note that definitions of isogeny graphs vary wildly between different authors with regard to the set of curves considered (sometimes only a single connected component) and the class of isomorphisms under which identifications are made. We prefer the version above as it seems to be one of the most versatile choices. Typically, we will restrict our attention to subgraphs of $G_{k,\ell}$ comprised of all curves with a given number of k-rational points, since (by Tate's theorem) such subgraphs are always disconnected from the rest of the graph anyway.

The existence of the dual isogeny implies that isogeny graphs can be viewed as undirected graphs almost everywhere: Exceptions can only occur at nodes with j-invariant 1728 or 0, due to their potential for having extra automorphisms, which may collapse the duals of multiple outgoing isogenies into just one incoming edge. Note that this affects only the multiplicities of directed edges, but not the existence of edges between two nodes.

Away from these special nodes, connected components of ℓ-isogeny graphs admit only two possible shapes: On one hand, very regular structures known as *volcanoes*, and on the other hand the much more random-looking *Pizer graphs*.[21] See Figure 2.1 for representative examples. An

[21] Pizer graphs are frequently referred to as *supersingular isogeny graphs* — a slight misnomer, as isogeny graphs of supersingular curves *can* be volcanoes (cf. Chapter 3). However, we will occasionally refer to "the" supersingular $\mathcal{S}$-isogeny graph in characteristic p, which means the unique supersingular component of $G_{\overline{\mathbb{F}_p},\mathcal{S}}$, though see Proposition 2.74.

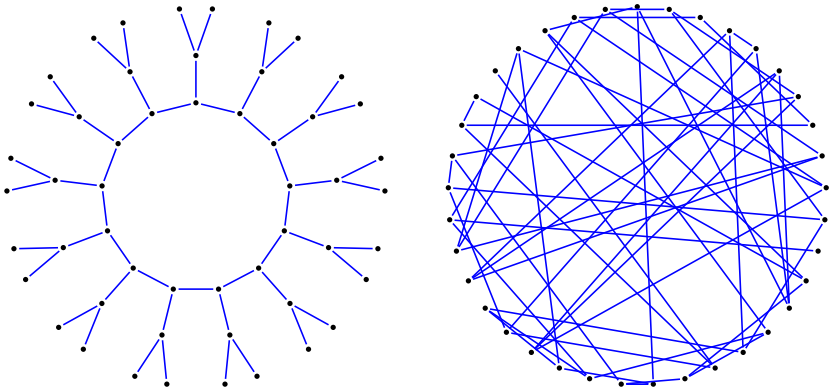


Figure 2.1: Typical components of isogeny graphs. Left: volcano graph. Right: Pizer graph. In both cases $\ell = 2$.

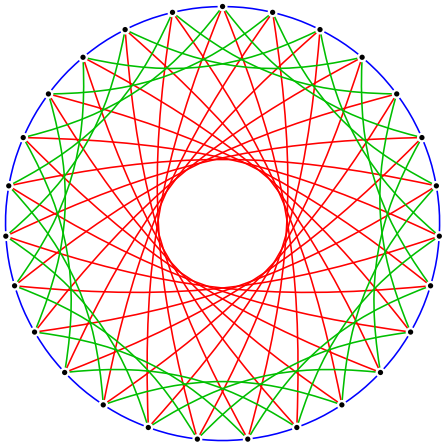


Figure 2.2: An $\mathcal{S}$-isogeny graph for the set $\mathcal{S} = \{3, 5, 7\}$ in the volcanic case. Here, each ℓ-isogeny volcano ($\ell \in \mathcal{S}$) has depth zero (see Definition 2.67).

ℓ-isogeny volcano consists of a single cycle,[22] the *crater*, and each node on the cycle is the root of a complete tree which is ℓ-ary below the root, occasionally referred to as *lava*, which however almost always has depth zero, i.e., is empty — so most volcanoes (in a certain sense) are actually made up of just a crater. In the volcano setting, the regular structure of a single ℓ-isogeny graph is highly compatible with other ℓ'-isogeny graphs, and thus a beautiful picture unfolds when considering the $\mathcal{S}$-isogeny graph for a set $\mathcal{S}$ of more than one prime; see Figure 2.2 for an example. Pizer graphs by contrast have much less easily comprehensible structure and look quite random.

To decide which kind of graph to create, the laws of mathematics simply sneak a peek at the structure of the *endomorphism ring* of the curves in question: The isogeny graph is a Pizer graph if and only if the ring of *k-rational* endomorphisms is non-commutative. Thus, we shall now digress to analyze endomorphism rings of elliptic curves. Section 2.5 will continue the discussion of isogeny graphs including the new tools developed in the following.

[22] In some cases, the "cycle" is degenerate and consists of just one or two nodes.

2.4 — Endomorphisms, quadratic fields, and quaternion algebras

As hinted in the previous section, the structure of isogeny graphs is intimately connected to the study of *endomorphism rings*.

Definition 2.37. *Let E/k be an elliptic curve. An* endomorphism *of E is an isogeny or the zero map from E to itself; in other words, it is an element of*

$$\mathrm{End}(E) := \mathrm{Hom}(E, E)\,.$$

With the point-wise addition inherited from $\mathrm{Hom}(E,E)$ and multiplication given by composition of endomorphisms, this set forms the endomorphism ring *of E.*

*Similarly, the k-*rational endomorphism ring *(or k-*endomorphism ring *for short) of E is the subring $\mathrm{End}_k(E) := \mathrm{Hom}_k(E,E)$. When $k = \mathbb{F}_q$, we sometimes write End_q instead of $\mathrm{End}_{\mathbb{F}_q}$.*

Due to the existence of scalar-multiplication morphisms, we can clearly always embed $\mathbb{Z}$ as a subring of $\mathrm{End}(E)$, and in fact the typical case in characteristic zero is $\mathrm{End}(E) = \mathbb{Z}$. This is far from true over finite fields, where elliptic curves *always* have non-scalar endomorphisms. To analyze the endomorphism ring further, it is helpful to adopt a more algebraic stance:

Fact 2.38. *Taking the dual isogeny of a non-zero endomorphism and mapping zero to itself defines an involution on $\mathrm{End}(E)$, usually referred to as* conjugation *and written $\overline{}$ in this context.*[23]

The existence of the involution $\vartheta \mapsto \overline{\vartheta}$ is critical in that it determines many other important properties of the endomorphism ring. Recalling that $\overline{\vartheta}\vartheta = \deg(\vartheta) \in \mathbb{Z}$ and using the fact that $\deg(\vartheta+1) - \deg(\vartheta) - 1 = \vartheta + \overline{\vartheta}$ is an integer, we can immediately make the crucial observation that every $\vartheta \in \mathrm{End}(E)$ satisfies a quadratic equation with coefficients in $\mathbb{Z}$:

$$\vartheta^2 = (\vartheta + \overline{\vartheta})\vartheta - \overline{\vartheta}\vartheta\,.$$

Prompted by this, the usual definitions for algebraic numbers carry over:

Definition 2.39. *Let $\vartheta \in \mathrm{End}(E)$. Its* norm *is $\mathrm{N}(\vartheta) = \vartheta\overline{\vartheta}$, and its* trace *is $\mathrm{tr}(\vartheta) = \vartheta + \overline{\vartheta}$.*

Notice that the norm is the same as the degree of an endomorphism.

Proposition 2.40. *Every endomorphism $\vartheta \in \mathrm{End}(E)$ is an algebraic integer of degree at most two. In particular, the norm $\mathrm{N}(\vartheta)$ and trace $\mathrm{tr}(\vartheta)$ are integers, and ϑ satisfies the characteristic equation*

$$\vartheta^2 - \mathrm{tr}(\vartheta)\cdot\vartheta + \mathrm{N}(\vartheta) = 0\,.$$

2.4.1 – Frobenius. Probably the most well-known example of Proposition 2.40 comes from elliptic curves over a finite field $\mathbb{F}_q$: There, one can immediately exhibit an endomorphism that is typically (but not always) non-scalar, namely the *(q-power) Frobenius endomorphism*

$$\pi\colon E \to E,\ (x,y) \mapsto (x^q, y^q)\,.$$

When $\#E(\mathbb{F}_q) = q + 1 - t$, the characteristic equation of π is $\pi^2 - t\pi + q = 0$.[24] Suppose $\pi \notin \mathbb{Z}$. Thus, the *Frobenius order* $\mathbb{Z}[\pi] \subseteq \mathrm{End}_k(E)$ is isomorphic to the ring $\mathbb{Z}[\sqrt{t^2 - 4q}]$. Sometimes, this is already the whole story and we have $\mathrm{End}_k(E) = \mathbb{Z}[\pi]$. However, the endomorphism ring can

[23] Some sources use $\overline{}$ and $\widehat{}$ interchangeably.

[24] Hence, the key to point counting lies in computing the *trace of Frobenius*, which is indeed the core idea underlying Schoof's algorithm 2.18.

be bigger in several ways. First, some of the endomorphisms $\pi - a \in \mathbb{Z}[\pi]$ may be divisible by an integer d,[25] which implies $\mathbb{Z}[\pi] \subsetneq \mathbb{Z}\left[\frac{\pi-a}{d}\right] \subseteq \mathrm{End}_k(E)$. Second — and much more severely — the set $\{1, \pi\}$ may fail to span the entirety of the endomorphism ring even when denominators are allowed.

2.4.2 – Endomorphism algebras. To separate the issues concerning denominators from more fundamental questions about the general structure of $\mathrm{End}_k(E)$ — such as the rank — it is convenient to introduce a coarser object:

Definition 2.41. *The* endomorphism algebra *of an elliptic curve E is the $\mathbb{Q}$-algebra*

$$\mathrm{End}_k^\circ(E) := \mathrm{End}_k(E) \otimes_{\mathbb{Z}} \mathbb{Q}.$$

In simpler terms, $\mathrm{End}_k^\circ(E)$ consists of elements of the form α/d with $\alpha \in \mathrm{End}_k(E)$ and $d \in \mathbb{Z}\backslash\{0\}$.

An explicit embedding shows that the isomorphism class of the endomorphism algebra is an isogeny invariant:

Proposition 2.42. *Let $\varphi\colon E \to E'$ be an isogeny defined over k. The map*

$$\mathrm{End}_k(E) \to \mathrm{End}_k(E'),\ \alpha \mapsto \varphi\alpha\widehat{\varphi}$$

is a homomorphism of additive groups with image $\deg(\varphi) \cdot \mathrm{End}_k(E')$. Correcting the failure to preserve the ring structure by dividing out the degree, the map

$$\iota_\varphi\colon\ \mathrm{End}_k^\circ(E) \to \mathrm{End}_k^\circ(E'),\ \alpha \mapsto \varphi\alpha\widehat{\varphi}/\deg(\varphi)$$

is an isomorphism of $\mathbb{Q}$-algebras. (The image $\iota_\varphi(\mathrm{End}_k(E))$ is generally not contained in $\mathrm{End}_k(E')$.)

Notice that the choice of isogeny $\varphi\colon E \to E'$ enters the definition of the isomorphism ι_φ between the endomorphism algebras. Indeed, this choice does *not* cancel out: In some cases, the embedding of $\mathrm{End}_k(E)$ into an ambient algebra is highly non-unique, in which case it is common to fix one by choosing a particular curve E_0 as a point of reference, fixing isogenies $\varphi\colon E \to E_0$ from all curves under investigation, and using the corresponding embeddings ι_φ of $\mathrm{End}_k(E)$ into the ambient algebra $\mathrm{End}_k^\circ(E_0)$. See [Wat69] for details. In practice, the embeddings of different endomorphism rings into a common ambient algebra are often omitted, implicitly assuming compatibility with respect to certain (classes of) isogenies.[26] For example, in Chapter 3, we will assume embeddings into an ambient algebra which all map Frobenius to the same element.

Another natural way to acquire the fractions $1/\deg(\varphi)$ required to turn the map $\alpha \mapsto \varphi\alpha\widehat{\varphi}$ into a ring homomorphism is localization at the degree:

Proposition 2.43. *Let $\varphi\colon E \to E'$ be an isogeny defined over k and write $d = \deg(\varphi)$. Then the map*

$$\mathrm{End}_k(E)[1/d] \to \mathrm{End}_k(E')[1/d],\ \alpha \mapsto \varphi\alpha\widehat{\varphi}/d$$

is an isomorphism of $\mathbb{Z}[1/d]$-algebras.

This result has the very useful implication that *isogenies can only change the endomorphism ring locally at the degree*, which Section 2.5.1 relies on crucially.

[25] More formally, this "divisibility" means that $\pi - a$ kills the d-torsion, hence (using Lemma 2.32) there exists a (unique) endomorphism $\tau \in \mathrm{End}_k(E)$ such that $\tau \circ [d] = \pi - a$. This τ is (deservedly) denoted by $\frac{\pi-a}{d}$.

[26] This can be very confusing.

2.4.3 – Kernels and ideals. Before we move on to the classification of endomorphism rings, we introduce an extremely important connection between isogenous curves and ideal classes of the endomorphism ring. Proofs (also covering the case of abelian varieties of higher dimension) can be found in [Wat69].

Throughout this section, let E be an elliptic curve defined over k and write $\mathcal{O} := \mathrm{End}_k(E)$.

Proposition 2.44. *Any non-zero left*[27] *ideal $\mathcal{I} \subseteq \mathcal{O}$ of $\mathcal{O}$ defines a finite subgroup of E by*

$$E[\mathcal{I}] := \bigcap_{\alpha \in \mathcal{I}} \ker(\alpha)\,.$$

Note that iterating over generators α of $\mathcal{I}$ suffices to compute the subgroup $E[\mathcal{I}]$ without enumerating all (infinitely many) elements of $\mathcal{I}$. (When $\mathcal{I} = \mathcal{O} \cdot \vartheta$, then simply $E[\mathcal{I}] = E[\vartheta] := \ker(\vartheta)$.)

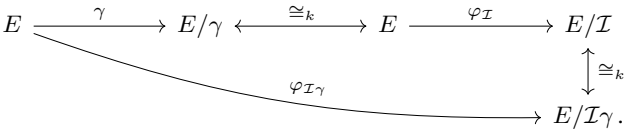
Factor $\mathcal{I}$ as $\mathcal{J} \cdot \pi^r$ with $\mathcal{J} \nsubseteq \mathcal{O}\pi$ and $r \geq 0$. Then the cardinality of $E[\mathcal{I}]$ equals the norm of $\mathcal{J}$, i.e., the greatest common divisor of all norms of elements in $\mathcal{J}$.

In particular, since $E[\mathcal{I}]$ is a finite subgroup, $\mathcal{I}$ defines an isogeny:

Definition 2.45. *For a non-zero left ideal $\mathcal{I} \subseteq \mathcal{O}$, let $\varphi_{\mathcal{I}}$ denote the isogeny with kernel $E[\mathcal{I}]$, thus $\varphi_{\mathcal{I}} = \varphi_{E[\mathcal{I}]}$ in our earlier notation. Write $E/\mathcal{I}$ for the codomain $E/E[\mathcal{I}]$ of $\varphi_{\mathcal{I}}$.*

A word of warning: There may exist subgroups which are not of the form $E[\mathcal{I}]$ for any $\mathcal{I}$.

From the definition of $E[\mathcal{I}]$, it is not very hard to see that multiplying $\mathcal{I}$ from the right by an element $\gamma \in \mathcal{O}\backslash\{0\}$ will not change the codomain (up to k-isomorphism): By construction, $E[\mathcal{I}\gamma] = \gamma^{-1}(E[\mathcal{I}])$ contains $\ker(\gamma)$, hence (using Lemma 2.32) we get a decomposition



Proposition 2.46. *Let $\mathcal{I}$ be a non-zero left ideal of $\mathcal{O}$ and $\gamma \in \mathcal{O}\backslash\{0\}$. Then*

$$E/\mathcal{I}\gamma \;\cong_k\; E/\mathcal{I}\,.$$

In particular, the isogeny codomain defined by a (left) ideal depends only on the class of the ideal.

The significance of this observation can hardly be overstated: It establishes a very strong relationship between the ideal theory of the endomorphism ring and the structure of the isogeny graph, a correspondence which equips us with essential tools to analyze the latter.

2.4.4 – Classification of endomorphism algebras. Let us now delve into the properties of endomorphism rings in various situations. The most important split happens between three major cases, which we distinguish in terms of the endomorphism algebras:

Proposition 2.47. *Let k be a field and E/k an elliptic curve.*

- *If $\mathrm{char}(k) = 0$, then either $\mathrm{End}_k(E) = \mathbb{Z}$ or $\mathrm{End}_k^\circ(E)$ is an imaginary quadratic field.*[28]

[27] Since conjugation $\bar{\ }$ swaps left and right multiplication, we could just as well speak of right ideals; however, left ideals appear to be a bit more popular. When $\mathcal{O}$ is commutative, all sides are of course the same.

[28] If $\mathrm{End}_k(E)$ is larger than $\mathbb{Z}$, then E is said to have *complex multiplication* (or *CM* for short) over k. When the base field is unspecified, the term refers to CM over the closure $\overline{k}$.

- *If $k = \mathbb{F}_q$, then $\mathrm{End}_k^\circ(E)$ is either an imaginary quadratic field or a quaternion algebra.*

To characterize the possible endomorphism rings inside the endomorphism algebra $\mathrm{End}_k^\circ(E)$ in more detail, we require the following definitions:

Definition 2.48. *Let A be a finite-dimensional $\mathbb{Q}$-algebra. A* lattice *in A is a finitely generated subgroup that spans A over $\mathbb{Q}$. An* order *in A is a lattice that is also a subring. The orders in A are partially ordered by inclusion; a* maximal order *of A is one that is not properly contained in any other order in A.*

The next two sections elaborate on specifics of the two "interesting" cases of Proposition 2.47: Quadratic fields and quaternion algebras.

2.4.5 – Quadratic endomorphism rings. For "most" elliptic curves over finite fields, the situation is comparably simple: Scalar multiplications and the Frobenius endomorphism are already "the whole story", at least up to denominators:

Proposition 2.49. *Let $E/\mathbb{F}_q$ with $\#E(\mathbb{F}_q) = q + 1 - t$ points, hence the Frobenius endomorphism π satisfies the equation $\pi^2 - t\pi + q = 0$.*

If $\pi \notin \mathbb{Z}$, then

$$\mathrm{End}_k^\circ(E) = \mathbb{Q}(\pi) \cong \mathbb{Q}(\sqrt{t^2 - 4q}).$$

In particular, $\mathrm{End}_k(E)$ is an order in $\mathbb{Q}(\sqrt{t^2 - 4q})$ containing $\mathbb{Z}[\pi] \cong \mathbb{Z}[\sqrt{t^2 - 4q}]$.

The condition $\pi \notin \mathbb{Z}$ is crucial: If $\pi \in \mathbb{Z}$, the endomorphism ring is non-commutative, which will be discussed in Section 2.4.6. Moreover, note in particular that Proposition 2.49 applies to all ordinary elliptic curves over a finite field, as well as supersingular elliptic curves defined over prime fields $\mathbb{F}_p$ with $p \geq 5$, since the condition $\pi \notin \mathbb{Z}$ is automatically satisfied in these cases.

Recall the following definitions and consequences from algebraic number theory:

Fact 2.50. *Consider the imaginary quadratic field $K = \mathbb{Q}(\sqrt{-d})$ where $d > 0$ is a square-free*[29] *integer. The* discriminant *Δ_K of K is defined to be $-d$ when $-d \equiv 1 \pmod 4$ and $-4d$ otherwise.*

There exists a unique maximal order $\mathcal{O}_K$ in K consisting of all algebraic integers in K; it is given by

$$\mathcal{O}_K = \mathbb{Z}\Big[\frac{\Delta_K + \sqrt{\Delta_K}}{2}\Big].$$

Every (other) order $\mathcal{O}$ in K is of the form

$$\mathcal{O} = \mathbb{Z} + f \cdot \mathcal{O}_K$$

where f is called the conductor *of $\mathcal{O}$ and equals the index $[\mathcal{O}_K : \mathcal{O}]$. The* discriminant *$\Delta_\mathcal{O}$ is $f^2\Delta_K$.*

The conjugation *automorphism $\bar{\ }\colon a + b\sqrt{-d} \mapsto a - b\sqrt{-d}$ of K is an automorphism of every order $\mathcal{O}$, and it extends to ideals of $\mathcal{O}$ via element-wise application. The product $\mathfrak{a}\bar{\mathfrak{a}}$ is a principal ideal generated by a non-negative integer $\mathrm{N}(\mathfrak{a})$ called the* norm *of $\mathfrak{a}$; we have $\mathrm{N}(\mathfrak{a}) = \gcd\{\mathrm{N}(\alpha) \mid \alpha \in \mathfrak{a}\}$.*

A prime number $p \in \mathbb{Z}$ is inert *in $\mathcal{O}$ if $p\mathcal{O}$ is a prime ideal,* split *if $p\mathcal{O} = \mathfrak{p}\bar{\mathfrak{p}}$ with a prime ideal $\mathfrak{p} \subseteq \mathcal{O}$ such that $\mathfrak{p} \neq \bar{\mathfrak{p}}$, and* ramified *if $p\mathcal{O} = \mathfrak{p}^2$. Only finitely many primes in $\mathbb{Z}$ are ramified; the density of split and inert primes in the set of all primes in $\mathbb{Z}$ is $1/2$ each.*

As seen in Proposition 2.46, multiplying left ideals from the right by base ring elements leaves the corresponding isogeny codomain unchanged. To symmetrize the treatment of scaling ideals, it is convenient to allow "ideals with denominators":

[29] This is not a restriction: Clearly $\mathbb{Q}(c^2\sqrt{-d}) = \mathbb{Q}(\sqrt{-d})$ for all $c \in \mathbb{Q}^\times$.

Definition 2.51. *Let A be a $\mathbb{Q}$-algebra and $\mathcal{O} \subseteq A$ an order. A* fractional left ideal *$\mathcal{I}$ of $\mathcal{O}$ is a lattice in A that is closed under left-multiplication by $\mathcal{O}$; similarly on the right.* Integral *(left or right) ideals are ideals in the conventional sense, i.e., fractional ideals of $\mathcal{O}$ contained in $\mathcal{O}$.*

For a non-zero fractional left ideal $\mathcal{I}$ of $\mathrm{End}_k(E)$, we abuse notation and still write $E/\mathcal{I}$ for the codomain $E/d\mathcal{I}$, where d is an appropriate scaling factor in $\mathbb{Z}\backslash\{0\}$ such that $d\mathcal{I}$ is integral; this is well-defined by Proposition 2.46.

In quadratic fields in particular, the notion of fractional ideals lends itself to extending the multiplication of ideals to a group structure:

Fact 2.52. *Let $\mathcal{O}$ be an imaginary quadratic order. A fractional ideal $\mathfrak{a}$ of $\mathcal{O}$ is* invertible *if there exists another fractional ideal $\mathfrak{b}$ of $\mathcal{O}$ such that $\mathfrak{ab} = \mathcal{O}$; if an ideal is invertible, the inverse equals $\bar{\mathfrak{a}}/\mathrm{N}(\mathfrak{a})$. Non-zero ideals of norm coprime to the conductor are always invertible; in particular, this includes all non-zero ideals of the maximal order. The set of invertible fractional ideals forms a group under ideal multiplication. The* ideal-class group $\mathrm{cl}(\mathcal{O})$ *of $\mathcal{O}$ is the quotient of this group of invertible fractional ideals modulo the subgroup of principal fractional ideals: two fractional ideals $\mathfrak{a}, \mathfrak{b}$ of $\mathcal{O}$ are* equivalent *if there exists an element $c \in K^\times$ such that $\mathfrak{a} = \mathfrak{b} \cdot c$. The cardinality of $\mathrm{cl}(\mathcal{O})$ is called the* class number *of $\mathcal{O}$ and denoted by $h(\mathcal{O})$.*

Writing $d = |\Delta_\mathcal{O}|$, it is known that $h(\mathcal{O}) \in O(\sqrt{d} \cdot \ln(d))$, and moreover that $h(\mathcal{O}) \in \Theta(\sqrt{d})$ on average. Assuming GRH, a lower bound is given by $\Omega(\sqrt{d}/\log\log d)$.

For more background on orders in imaginary quadratic fields, we refer to Cox' book [Cox13].

We have seen that whenever $\pi \notin \mathbb{Z}$, the endomorphism ring of an elliptic curve over $\mathbb{F}_q$ is an order in $\mathbb{Q}(\sqrt{t^2 - 4q})$ containing π, where t is the trace of the Frobenius endomorphism π. Conversely, it turns out that most of the quadratic orders that are not "obviously" impossible on the grounds of lacking a Frobenius element occur as endomorphism rings [Sch87, Theorem 4.3]:

Proposition 2.53. *Consider a finite field $\mathbb{F}_q$ where $q = p^r$, and an integer t such that $t^2 - 4q < 0$. Let $\mathcal{O}$ be an order in $K = \mathbb{Q}(\sqrt{t^2 - 4q})$ containing $\mathbb{Z}[\sqrt{t^2 - 4q}]$.*

Suppose that $p \nmid t$, or alternatively that $t = 0$, r is odd, and $p \nmid [\mathcal{O}_K : \mathcal{O}]$. Then $\mathcal{O}$ occurs as an endomorphism ring of an elliptic curve over $\mathbb{F}_q$. (Some more special cases were omitted for simplicity.)

We may arrange the elliptic curves in an isogeny class in a layer structure depending solely on the integer $f = [\mathcal{O}_K : \mathcal{O}]$, which (as we shall see in Section 2.5.1) corresponds to the level in ℓ-isogeny volcanoes for $\ell \mid f$. Also note that the previous result includes some instances of supersingular elliptic curves in particular, which will be used in Chapter 3 in the situation $q = p$ owing to their beneficial properties from a computational perspective.

As explained in Section 2.4.3, ideals define isogenies, and the codomain only depends on the class of an ideal. In addition, one can show that the action of invertible ideals does not change the endomorphism ring, which means we can repeatedly act with more ideals on the codomain — thus defining an extremely important group action on sets of curves with the same (quadratic) endomorphism ring:

Definition 2.54. *For a field k and an quadratic order $\mathcal{O}$, we let*

$$\mathcal{E}\ell\ell_k(\mathcal{O}) = \{E/k \mid \mathrm{End}_k(E) \cong \mathcal{O}\} / \cong_k,$$

where each curve in $\mathcal{E}\ell\ell_k(\mathcal{O})$ implicitly comes equipped with a fixed isomorphism $\iota_E \colon \mathrm{End}_k(E) \xrightarrow{\sim} \mathcal{O}$ that respects k-isogenies.

The condition of "respecting k-isogenies" means: For any k-isogeny $\varphi\colon E \to E'$ between any two curves E, E' in $\mathcal{Ell}_k(\mathcal{O})$, we have

$$\iota_E = \iota_{E'} \circ \left(\iota_\varphi|_{\mathrm{End}_k(E)}\right),$$

with ι_φ being the map from Proposition 2.42.

If k is finite, this simply means that Frobenius is mapped to the same element of $\mathcal{O}$ by all ι_E.

Theorem 2.55 (The CM torsor)**.** *Let k be a field of characteristic $p \geq 0$, and let $\mathcal{O}$ be an order in an imaginary quadratic field K such that $\mathcal{Ell}_k(\mathcal{O})$ is non-empty. The map*

$$\begin{aligned} *\colon \quad \mathrm{cl}(\mathcal{O}) \times \mathcal{Ell}_k(\mathcal{O}) &\longrightarrow \mathcal{Ell}_k(\mathcal{O}) \\ ([\mathfrak{a}], E) &\longmapsto E/\mathfrak{a}, \end{aligned}$$

where $\mathfrak{a} \subseteq \mathcal{O}$ is chosen as an integral representative of its class $[\mathfrak{a}]$, is a well-defined group action.

This group action is free (no two ideal classes act the same on any curve). Whenever $p = 0$ or p is inert in $\mathcal{O}$, it is transitive (every pair of curves is connected); otherwise, there are two orbits. In particular, the cardinality $|\mathcal{Ell}_k(\mathcal{O})|$ equals either $h(\mathcal{O})$ or $2 \cdot h(\mathcal{O})$.

Note that the case of two orbits is extremely rare: At the very least, it requires supersingular elliptic curves defined over $k = \mathbb{F}_q$ with $p^2 \mid t^2 - 4q$; see [Sch87, Theorem 4.5].

Since we will make computational use of the group action from Theorem 2.55 in Chapter 3, let us unfold the definition into something more explicit:

Proposition 2.56. *Let E/k and suppose $\mathcal{O} := \mathrm{End}_k(E)$ equals $\mathbb{Z}[\tau]$ where $\tau = (\pi - m)/f$ with integers m, f. Then any ideal $\mathfrak{a} \subseteq \mathcal{O}$ of norm $N \geq 0$ is either of the form $(\sqrt{N})$ or of the form $(N, \tau - \lambda)$, where $\lambda \in \mathbb{Z}/N$ is an* eigenvalue *of τ on $E[N]$.*

In the latter case, the subgroup $E[\mathfrak{a}]$ is precisely the eigenspace *of τ on $E[N]$ with eigenvalue λ, i.e., the subgroup of N-torsion points on which τ acts as multiplication by λ.*

We refer to [DKS18] for more details, and to Chapter 3 for the specialization $\tau^2 = -p$.

2.4.6 – Quaternionic endomorphism rings. For supersingular elliptic curves, the picture is quite a bit more complicated: They have "extra" endomorphisms which do not all commute with each other:

Example 2.57. *Consider the elliptic curve $E\colon y^2 = x^3 + x$ over $\mathbb{F}_p$ with $p \equiv 3 \pmod 4$. Clearly, we have the $\mathbb{F}_p$-Frobenius endomorphism $\pi\colon E \to E$, $(x, y) \mapsto (x^p, y^p)$. In addition, when viewed as a curve over $\mathbb{F}_{p^2}$, the curve admits the automorphism*

$$\iota\colon E \to E,\ (x, y) \mapsto (-x, \sqrt{-1} \cdot y)$$

of order four, which anti*commutes with π since $\left(\sqrt{-1}\right)^p = -\sqrt{-1}$. Thus, the endomorphism algebra $\mathrm{End}^\circ(E)$ is generated by ι and π, which are subject to the relations $\iota^2 = -1$, $\pi^2 = -p$, and $\pi\iota = -\iota\pi$.*

Similarly, letting $p \equiv 2 \pmod 3$ and fixing a primitive third root of unity $\zeta \in \mathbb{F}_{p^2}$, the curve $E'\colon y^2 = x^3 + 1$ defined over $\mathbb{F}_p$ acquires the automorphism

$$\omega\colon E' \to E',\ (x, y) \mapsto (\zeta \cdot x, y)$$

when the base field is extended to $\mathbb{F}_{p^2}$. Note that again $\pi\omega = -\omega\pi$ where π is the $\mathbb{F}_p$-Frobenius of E'.

These examples may seem somewhat pathological since in both cases the "extra" endomorphisms are actually *auto*morphisms, but the phenomenon extends to many more curves: At the very least, Proposition 2.42 shows that every curve isogenous to these curves will exhibit non-commuting endomorphisms, and with some more work one can show that this happens for all supersingular elliptic curves.

2.4.7 – Quaternion algebras. Example 2.57 shows special cases of *quaternion algebras*:

Definition 2.58. *Let p be a prime number and let a, b denote positive integers specified below. The* quaternion algebra ramified at p and ∞*, denoted by $B_{p,\infty}$, is a four-dimensional $\mathbb{Q}$-algebra spanned by basis elements $1, \mathbf{i}, \mathbf{j}, \mathbf{ij}$ with multiplication law*

$$\mathbf{i}^2 = a, \quad \mathbf{j}^2 = b, \quad \text{and} \quad \mathbf{ij} = -\mathbf{ji}.$$

When $p \equiv 3 \pmod 4$, let $(a, b) = (-1, -p)$. When $p \equiv 1 \pmod 4$, pick a prime $q \equiv 3 \pmod 4$ that is non-square modulo p and let $(a, b) = (-q, -p)$.[30] *When $p = 2$, let $(a, b) = (-1, -1)$. All pairs (a, b) adhering to these conditions yield isomorphic algebras, and $B_{p,\infty}$ is unique up to isomorphism.*

Note that $p = 2$ yields *the* quaternions as discovered by Hamilton in 1843, often written $\mathbb{H}$ in his honour, and all other $B_{p,\infty}$ are in a sense "just" distorted versions of $\mathbb{H}$ with the axes scaled by real quadratic integers. (Algebraically speaking, this change is of course quite significant.)

The most comprehensive reference on quaternion algebras is Voight's book [Voi18]. For now, recall Definitions 2.48 and 2.51.

2.4.8 – Deuring's correspondence. As hinted above, (full) endomorphism rings of supersingular elliptic curves are subrings of a quaternion algebra:

Proposition 2.59. *Let E be a supersingular elliptic curve defined over a field k of characteristic $p > 0$. Then $\mathrm{End}^\circ(E) \cong B_{p,\infty}$, and $\mathrm{End}(E)$ is a maximal order in $B_{p,\infty}$.*

While supersingular endomorphism rings are quaternionic over the closure, generally not all endomorphisms are defined over the base field (see Example 2.57). How large a field extension is needed to acquire all endomorphisms? From Proposition 2.49, we know that $\pi \notin \mathbb{Z}$ implies that the k-endomorphism ring is an imaginary quadratic order. The converse is also true:

Proposition 2.60. *Let $k = \mathbb{F}_q$ be a finite field of characteristic p and E an elliptic curve over k with Frobenius π. Then $\mathrm{End}_k(E)$ is non-commutative, i.e., a maximal order in $B_{p,\infty}$, if and only if $\pi \in \mathbb{Z}$.*[31]

In stark contrast to the commutative case, maximal orders in quaternion algebras are far from unique. Thus, while in the commutative case the endomorphism rings of many curves may be exactly the same (by Theorem 2.55), one may wonder to what extent a supersingular curve can be recovered from its endomorphism ring, a question that was answered by Deuring in an influential 1941 paper [Deu41]:

Theorem 2.61 (The Deuring correspondence)**.** *Let p be a prime and $\sigma\colon \alpha \mapsto \alpha^p$ the nontrivial automorphism of $\mathbb{F}_{p^2}$. Taking endomorphism rings induces a bijection*

$$\{j(E) \mid E/\overline{\mathbb{F}_p} \text{ supersingular}\} / \langle\sigma\rangle \xleftrightarrow{\sim} \{\text{maximal orders } \mathcal{O} \text{ of } B_{p,\infty}\} / \cong.$$

[30] The existence of such a q is guaranteed by Dirichlet's theorem on primes in arithmetic progressions.

[31] Strangely, even though the property $\pi \in \mathbb{Z}$ suggests that the curve "should" end up having fewer endomorphisms than "normal", the exact opposite is the case: These curves are precisely the odd ones out with *more* endomorphisms!

In particular, for every maximal order $\mathcal{O}$ of $B_{p,\infty}$, there exist either one or two isomorphism classes of supersingular elliptic curves over $\overline{\mathbb{F}_p}$ with endomorphism ring (isomorphic to) $\mathcal{O}$. There is only one such curve E if and only if $j(E) \in \mathbb{F}_p$. Otherwise, the two curves have conjugate j-invariants in $\mathbb{F}_{p^2} \setminus \mathbb{F}_p$, so in particular the two curves are connected by the Frobenius isogeny $\pi_p \colon (x, y) \mapsto (x^p, y^p)$.

Even more pleasantly, there is an analogue of the free and transitive group action from Theorem 2.55, except that one-sided ideals do of course not form a group:

Definition 2.62. *The* right order *of a lattice I in $B_{p,\infty}$ is*

$$\mathcal{O}_R(I) = \{\alpha \in B_{p,\infty} \mid I\alpha \subseteq I\};$$

the left order *is defined analogously. In other words, the right (resp. left) order is the largest subring of $B_{p,\infty}$ for which I is a right (resp. left) fractional ideal.*

Two fractional left ideals $\mathcal{I}, \mathcal{J}$ of a maximal order $\mathcal{O} \subseteq B_{p,\infty}$ are (left) equivalent *if there exists $\gamma \in B_{p,\infty}^\times$ such that $\mathcal{J} = \mathcal{I}\gamma$. As usual, we write $[\mathcal{I}]$ for the* class *of $\mathcal{I}$, i.e., the set of all left fractional ideals equivalent to $\mathcal{I}$. The* left (ideal-)class set $\mathrm{Cls}_L(\mathcal{O})$ *is the set of all non-zero ideal classes $[\mathcal{I}]$.*[32]

Proposition 2.63. *Let E_0 be a supersingular elliptic curve over $\overline{\mathbb{F}_p}$ and write $\mathcal{O}_0 := \mathrm{End}(E_0)$. For every supersingular elliptic curve $E/\overline{\mathbb{F}_p}$, there is a unique left ideal class $[\mathcal{I}]$ of $\mathcal{O}_0$ such that $E_0/\mathcal{I} \cong E$.*[33] *The endomorphism ring of the isogeny codomain $E_0/\mathcal{I}$ is isomorphic to the right order*[34] *of $\mathcal{I}$:*

$$\mathrm{End}(E_0/\mathcal{I}) \cong \mathcal{O}_R(\mathcal{I}) \subseteq \mathrm{End}^\circ(E_0) \cong B_{p,\infty}.$$

(The embedding $\mathrm{End}(E_0/\mathcal{I}) \hookrightarrow \mathrm{End}^\circ(E_0)$ is the restriction of the map $\iota_{\widehat{\varphi_\mathcal{I}}}$ from Proposition 2.42.)

Proposition 2.64. *Let E_0, E_1 be supersingular elliptic curves over $\overline{\mathbb{F}_p}$ and write $\mathcal{O}_i := \mathrm{End}(E_i)$ for short. Fix isomorphisms $\mathrm{End}^\circ(E_i) \xrightarrow{\sim} B_{p,\infty}$ and identify $\mathcal{O}_i$ with their images.*

Then, the fractional $\mathcal{O}_0$-left and $\mathcal{O}_1$-right ideal $\mathcal{I} := \mathcal{O}_0 \cdot \mathcal{O}_1$ satisfies $E_0/\mathcal{I} \cong E_1$.

The choice of isomorphisms $\mathrm{End}(E_i) \xrightarrow{\sim} B_{p,\infty}$ in Proposition 2.64 is important: It resolves the apparent contradiction that $\mathcal{I}$ is defined only in terms of endomorphism rings, yet allows to recover the curve exactly, whereas by Theorem 2.61 there are often two curves with isomorphic endomorphism rings.

2.4.9 – Representatives of ideal classes. Recall from Section 2.3.1 that we typically need a separable isogeny to have smooth (or, in general, powersmooth) degree to be able to efficiently compute it from its kernel. This observation carries over to computing an isogeny from a defining ideal, as this (usually) consists of first recovering generators of the kernel subgroup and then applying Vélu-style algorithms. Recalling Proposition 2.44, the complexity of these steps is determined by the prime factorization of the norm of the ideal.

Since the codomain of an isogeny only depends on the ideal class (Proposition 2.46), we may try to find a more suitable representative of an ideal class when the norm is not smooth enough. To this end, it is known abstractly that for all primes $\ell \neq p$, every class of non-zero quaternion left ideals has a representative of ℓ-power norm [Voi18, Main Theorem 28.5.3]:

Proposition 2.65. *Let $\mathcal{O}$ be a maximal order in $B_{p,\infty}$ and $\mathcal{I} \subseteq \mathcal{O}$ a non-zero left ideal. For any prime $\ell \neq p$, there exists a left ideal $\mathcal{J} \in [\mathcal{I}]$ whose norm is a power of ℓ.*

[32] Similarly to the quadratic case, defining this for general orders requires an invertibility condition, but we restrict to maximal orders as the general case will not be needed in the following.

[33] The choice of $\mathcal{I}$ in its class does not matter, though it *does* generally yield different isogenies $\varphi_\mathcal{I} \colon E_0 \to E$.

[34] Mnemonic: "When left with an ideal, the right order is the right order."

Combining this result with Proposition 2.64 yields the important fact that *the supersingular ℓ-isogeny graph is connected* for all primes $\ell \neq p$; see also Section 2.5.3.

It is not known unconditionally how to make Proposition 2.65 effective. However, there is a clever *heuristic* algorithm [KLPT14] to solve this problem (and related problems with other kinds of target norms) in polynomial time for *special* maximal orders, under the assumption that certain integers appearing in the algorithm have "random" factorization properties. Concretely, special orders are those containing $\mathbb{Z} + \mathbb{Z}\mathbf{i} + \mathbb{Z}\mathbf{j} + \mathbb{Z}\mathbf{ij}$, where q in Definition 2.58 is as small as possible.

Theorem 2.66 (The KLPT algorithm)**.** *There is a heuristic algorithm which, given as input a basis of a special maximal order in $B_{p,\infty}$, a basis of a non-zero left ideal $\mathcal{I} \subseteq \mathcal{O}$, and a prime $\ell \neq p$, returns a basis of an ideal $\mathcal{J} \in [\mathcal{I}]$ such that $\mathrm{N}(\mathcal{J}) = \ell^n$ for some integer $n \geq 0$. If the size of the input is polynomial in $\log p$,[35] the runtime is heuristically polynomial in $\log p$.*

Under some more heuristic assumptions on sizes of integers, the norm of $\mathcal{J}$ can be estimated as $p^{7/2}$.

As the KLPT algorithm is quite technical, we refer to [KLPT14; GPS17; DeF+20] for details and only give a short overview: Note that if $\delta \in \mathcal{I}$ has norm $\mathrm{N}(\mathcal{I}) \cdot m$, then $\mathcal{I}\overline{\delta}/\mathrm{N}(\mathcal{I})$ is an equivalent ideal of norm m. (We refer to this fact as *the fact* during this paragraph.) KLPT first reduces to the case of ideals of prime norm: pick random $\delta \in \mathcal{I}$ until $\mathrm{N}(\delta) = \mathrm{N}(\mathcal{I}) \cdot N$ with N prime; hence, by *the fact* we may assume that $\mathcal{I}$ has prime norm N. Then $\mathcal{I}$ can be written as $\mathcal{I} = \mathcal{O}(N, \alpha)$ for some $\alpha \in \mathcal{O}$; such an α is again easily found randomly. The next step (which in turn consists of a few rather technical substeps) is to factor the element α as $\beta\gamma$ modulo N, where $\beta \in \mathcal{O}$ is of norm $N\ell^b$ for some $b \geq 0$ and $\gamma \in \mathbb{Z}\mathbf{j} + \mathbb{Z}\mathbf{ij}$. Finally, the core of the algorithm is an effective version of the *strong approximation theorem*, which allows lifting the element $\gamma \in \mathbb{Z}\mathbf{i} + \mathbb{Z}\mathbf{ij}$ to a pair $(\lambda, \gamma') \in \mathbb{Z} \times \mathcal{O}$ such that $\gamma' \equiv \lambda\gamma \pmod{N}$ and $\mathrm{N}(\gamma') = \ell^c$ for some $c \geq 0$. Finally, note that $\mathcal{O}(N, \beta\gamma') = \mathcal{O}(N, \alpha) = \mathcal{I}$, hence $\beta\gamma' \in \mathcal{I}$, so we can apply *the fact* with $\delta = \beta\gamma'$ to obtain an ideal of norm $N\ell^b\ell^c/N = \ell^{b+c}$ equivalent to $\mathcal{I}$ as desired.

The steps that compute β and γ' are largely agnostic to the kind of target norms they are requested to output; in particular, it is very easy to adapt the algorithm to powersmooth norms.

Notice that the KLPT algorithm can be used to efficiently compute the ring-to-curve direction of the Deuring correspondence (Theorem 2.61): Given any maximal order $\mathcal{O}$ in $B_{p,\infty}$, pick an arbitrary supersingular curve with a known "special" maximal order as endomorphism ring, and apply KLPT to a connecting ideal (which can be computed as shown in Proposition 2.64) to obtain a powersmooth representative. Then, simply compute (a prime-power decomposition of) the subgroup defined by the ideal (see Proposition 2.44), and proceed as explained in Section 2.3.3 to obtain the equation of a curve with endomorphism ring $\mathcal{O}$. The other direction of the Deuring correspondence appears to be much harder: The best known algorithm to compute endomorphism rings of supersingular elliptic curves essentially consists of searching cycles in an isogeny graph (cf. Section 2.5.4) until enough endomorphisms to span the entire ring have been found. See [Koh96, Section 7] and [Eis+20].

2.5 — Isogeny graphs

Armed with the tools from Section 2.4, we can now discuss isogeny graphs in more detail. Recall the notation from Definition 2.36: Let k be a field, $\mathcal{S}$ a set of positive integers, and $G_{k,\mathcal{S}}$ the graph

[35] A priori, it is not obvious that a basis with representation size polynomial in $\log p$ exists for every maximal order in $B_{p,\infty}$; this follows from a geometry-of-numbers argument [Eis+18, Theorem 2]. Similarly, an ideal whose norm is polynomial in p can be represented with size polynomial in $\log p$.

consisting of elliptic curves and isogenies of degree in $\mathcal{S}$, all up to k-isomorphism.

We consider three situations: The ℓ-isogeny graphs and $\mathcal{S}$-isogeny graphs in the "volcanic" case (left side of Figure 2.1 and Figure 2.2), and the ℓ-isogeny graph in the Pizer case (right side of Figure 2.1).

2.5.1 – Volcanoes. The first case are isogeny volcanoes: The kind of connected components one gets by restricting one's attention to a single prime degree in the commutative case. A good overview of the contents of this section is given in Sutherland's *Isogeny Volcanoes* [Sut12b]; see Kohel's PhD thesis [Koh96] for a more in-depth discussion.

Throughout this section, let ℓ denote a prime number not equal to the characteristic of k. Let us first make the informal description of isogeny volcanoes given in Section 2.3 precise:

Definition 2.67. *An* ℓ-volcano *of* depth $d \geq 0$ *is an undirected graph V consisting of a single cycle C (which may be just one node), such that each node of C is the root of a complete tree of depth d, and such that every node has degree* 0, 1, 2, *or* $\ell+1$.

The subgraph C is called surface *or* crater. *The* level *of a node v is the distance from the surface. The* floor *consists of nodes at level d. Isogenies between curves at the same level are called* horizontal, *isogenies from level i to level $i+1$ are* descending, *and isogenies from level i to $i-1$ are* ascending.

Components of isogeny graphs of elliptic curves with imaginary quadratic endomorphism rings (recall from Proposition 2.42 that this property is invariant under k-isogenies) are almost always volcanoes:

Proposition 2.68. *Let k be finite and let V be a connected component of $G_{k,\ell}$ containing a curve E with* $\mathrm{End}_k(E)$ *commutative. Suppose V does not contain a curve with j-invariant* 0 *or* 1728.

Since all curves in V are k-isogenous, their Frobenius orders $\mathbb{Z}[\pi]$ and endomorphism algebras K are the same. Let f denote the conductor of $\mathbb{Z}[\pi]$, i.e., its index in the ring of integers $\mathcal{O}_K$. Define $d = v_\ell(f)$, i.e., the multiplicity of ℓ in the prime factorization of f.

Then V is an ℓ-volcano of depth d. All curves at level $i \in \{0, \ldots, d\}$ have k-rational endomorphism ring $\mathcal{O}_i := \mathbb{Z} + (f/\ell^{d-i})\mathcal{O}_K$. If ℓ splits in $\mathcal{O}_0$, then the size of the crater of V equals the order of $[\mathfrak{l}]$ in the ideal-class group $\mathrm{cl}(\mathcal{O}_0)$, *where $\mathfrak{l}$ is an $\mathcal{O}_0$-ideal of norm ℓ; otherwise, the crater has size* 1.

The proof relies on Proposition 2.43 and Theorem 2.55. The two special cases $j \in \{0, 1728\}$ are not much different, but must be excluded for the technical reason that they may have additional automorphisms which collapse more isogenies into one edge than "usual". In these cases, however, the graph is still very volcano-esque, with a crater of size one, and the only failure is that the crater node has more outgoing than incoming edges; all other structural properties of the graph remain the same. See Figure 2.3 for examples. As the differences are minor, localized at no more than two exceptional nodes, and usually do not impact applications, we will generally largely ignore this issue.

Based on the following theorem of Lenstra, we can moreover see that the structure of the subgroup of k-rational points is the same for all curves on the same level.

Theorem 2.69 [Len96, Theorem 1]**.** *Consider an elliptic curve E over $k = \mathbb{F}_q$ with Frobenius endomorphism π and write $\mathcal{O} = \mathrm{End}_k(E)$. Let $\mathbb{F}_{q^n}$ be a finite extension of $k = \mathbb{F}_q$.*

- *If $\pi \notin \mathbb{Z}$, then $E(\mathbb{F}_{q^n}) \cong \mathcal{O}/(\pi^n - 1)$ as $\mathcal{O}$-modules.*
- *If $\pi \in \mathbb{Z}$, then $E(\mathbb{F}_{q^n}) \cong \mathbb{Z}/(\pi^n - 1) \times \mathbb{Z}/(\pi^n - 1)$ as abelian groups. Furthermore, as left $\mathcal{O}$-modules, we have $E(\mathbb{F}_{q^n}) \oplus E(\mathbb{F}_{q^n}) \cong \mathcal{O}/(\pi^n - 1)$.*

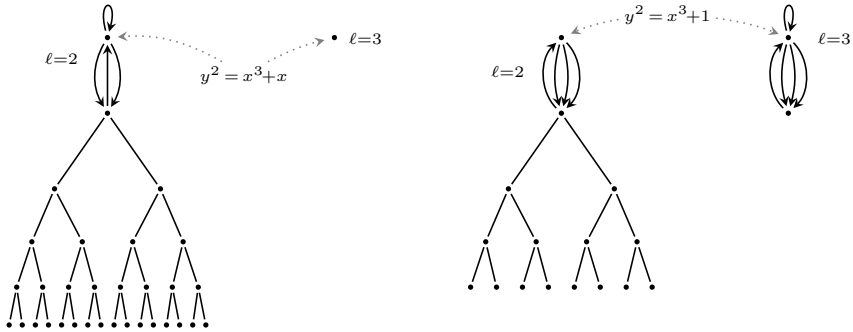


Figure 2.3: Almost-volcanoes with extra automorphisms on the crater: Components of ℓ-isogeny graphs over $\mathbb{F}_{1753}$. Observe that special behaviour occurs only at the crater, whereas all lower levels look like "normal" volcanoes.

Moreover, when the curves have some k-rational ℓ-power torsion, the distance of a curve from the floor often expresses itself in the structure of that subgroup [IJ13, § 2.3]:

Proposition 2.70. *Let $k = \mathbb{F}_q$ and E/k an elliptic curve with Frobenius $\pi \notin \mathbb{Z}$, so that $\mathcal{O} = \mathrm{End}_k(E)$ is commutative. Let f denote the conductor of $\mathbb{Z}[\pi]$ and m the index of $\mathbb{Z}[\pi]$ in $\mathcal{O}$. Furthermore, let $a \in \mathbb{Z}$ with $2a \equiv \mathrm{tr}(\pi) \pmod f$. Then $\mathcal{O} = \mathbb{Z}\left[\frac{\pi - a}{m}\right]$ and*

$$E(k) \;\cong\; \mathbb{Z}/N \times \mathbb{Z}/M\,,$$

where $N = \gcd(a-1, m)$ and $NM = \#E(k)$. Moreover, $N \mid M$ and $N \mid q-1$.

In particular, writing $\nu = v_\ell(N)$, $\mu = v_\ell(M)$, and letting $\delta = v_\ell(m)$ be the distance of E from the floor, we have $\delta \geq \nu$. Moreover, equality $\delta = \nu$ holds if $\nu < \mu$.

We refer to [IJ13] for more details about the structural relationship between volcanoes and rational subgroups. Note that there are examples where $\ell \nmid \#E(k)$ in an ℓ-volcano of depth ≥ 1, hence the rational group structure is strictly weaker information than the conductor of $\mathrm{End}_k(E)$.

Algorithms to compute endomorphism rings of ordinary elliptic curves over finite fields rely on walking in isogeny volcanoes to determine their structure, which (as we have seen) relates the location of a curve in the volcano to its endomorphism ring. See [Koh96] and [Bis12]; the latter also makes use of some facts discussed the following section.

2.5.2 – Schreier graphs. Isogeny volcanoes do not look like very promising candidates for building cryptography: Assuming ℓ-isogenies can be computed efficiently, walking up to the surface is easy (this is in fact done in Kohel's algorithm), and walking on a large cycle does not feature any asymmetry in the complexities of doing the walk vs. recovering a path from start and end nodes: Both take time $O(\#\text{steps})$.

Thus, we consider graphs arising as the union of various ℓ-isogeny volcanoes on the same set of (isogenous) curves, such as the small example depicted in Figure 2.2. The intuition is that combining edges from various isogeny graphs introduces "shortcuts" allowing short walks to any node, in a sense that will be made precise below.

Definition 2.71. *Let $*\colon G \times X \to X$ a free action of an abelian group G on a set X, and consider a subset $T \subseteq G \backslash \{0\}$ closed under inversion. The* Schreier graph *of T on X has vertex set X, and two nodes $x, x' \in X$ are connected by an edge if and only if $x' = g * x$ for some $g \in T$.*

Note that for comparison with traditional group-based Diffie–Hellman, we can reinterpret the well-known square-and-multiply algorithm (see Section 2.1.1) for fast exponentiation in a finite cyclic group $G = \langle g \rangle$ as a particular kind of short walk in the Schreier graph of G acting on itself via multiplication,[36] with the set of generators $T = \{g^{\pm 1}, g^{\pm 2}, g^{\pm 4}, g^{\pm 8}, ..., g^{\pm 2^{\lfloor \log_2 |G| \rfloor}}\}$.

In particular, we can apply the Schreier graph construction to the class-group group action from Theorem 2.55, which yields graphs such as the small example shown in Figure 2.2. However, it is a priori not at all clear that combining isogeny cycles yields better connectivity properties: It might happen that two curves are ℓ'-isogenous if and only if they are ℓ-isogenous, and in this case we would only change the multiplicity of edges. Fortunately, one can prove (assuming the generalized Riemann hypothesis) that combining sufficiently many isogeny volcanoes suffices to guarantee the existence of logarithmically short paths between all nodes, based on the following observation [JMV09, Lemma 2.1]:

Proposition 2.72 (Rapid mixing)**.** *Consider a connected d-regular undirected multigraph Γ with n vertices. Let A denote its adjacency matrix and let $\lambda_1 \geq \cdots \geq \lambda_n$ be the eigenvalues of A. (There are n real eigenvalues as A is symmetric.) Define $\lambda := \max\{|\lambda_2|, ..., |\lambda_n|\}$.*

Suppose $\lambda < d$, and let $K \subseteq \Gamma$ be a subset of size $k \geq 1$. Then, a random walk in Γ of length at least

$$\log_{d/\lambda} \frac{2n}{\sqrt{k}}$$

ends in K with probability between $\frac{1}{2}k/n$ and $\frac{3}{2}k/n$.

Thus, bounding the eigenvalues of the adjacency matrix of an isogeny graph will allow us to determine its mixing properties. We say that a graph is an *expander* if Proposition 2.72 applies with $\lambda/d \leq \kappa$ for some constant $\kappa < 1$. Indeed, we may then conclude that random walks of logarithmic length mix well, and in particular that any two nodes are connected by a logarithmically short path.

As advertised above, the crux is that components with commutative endomorphism rings of $\mathcal{S}$-isogeny graphs for sufficiently large sets of primes $\mathcal{S}$ satisfy the conditions of Proposition 2.72:

Theorem 2.73 [JMV09]**.** *Let q be a prime power and $\mathcal{O}$ an imaginary quadratic order such that $\mathcal{Ell}_q(\mathcal{O})$ is non-empty. Pick a constant $B > 2$ and define*

$$\mathcal{S} = \left\{ \ell \in \mathbb{Z} \text{ prime} \mid \ell \leq (\log 4q)^B \right\}.$$

Then, assuming GRH, the subgraph Γ induced by $G_{\mathbb{F}_q, \mathcal{S}}$ on $\mathcal{Ell}_q(\mathcal{O})$ is an expander as $q \to \infty$.

2.5.3 – Pizer graphs. We shall now discuss supersingular isogeny graphs over $\overline{\mathbb{F}_p}$. Since there are only $p/12 + O(1)$ supersingular j-invariants over $\overline{\mathbb{F}_p}$, the examples for very small p are relatively uneventful and easily analyzed individually, and we will generally assume in this section that p is a "large" prime (certainly $p \geq 5$). First, note that the supersingular component of the isogeny graph over $\overline{\mathbb{F}_p}$ can equivalently be viewed as a supersingular component over $\mathbb{F}_{p^2}$:

Proposition 2.74. *Let $\varphi\colon E_1 \to E_2$ be an isogeny of supersingular elliptic curves defined over $\overline{\mathbb{F}_p}$. Then there exist elliptic curves $E_1', E_2'/\mathbb{F}_{p^2}$ and isomorphisms $\alpha_i\colon E_i \to E_i'$ such that*

$$\varphi' := \alpha_2 \circ \varphi \circ \alpha_1^{-1}$$

is an $\mathbb{F}_{p^2}$-rational isogeny $E_1' \to E_2'$.

[36] The Schreier graph of a group acting on itself by multiplication is much more well-known as its *Cayley graph*.

Hence we may restrict our attention to a supersingular component (say, curves with $(p+1)^2$ points) of isogeny graphs over $\mathbb{F}_{p^2}$, which is computationally much easier to grasp than $\overline{\mathbb{F}_p}$.

The name *Pizer graphs* for supersingular isogeny graphs over $\overline{\mathbb{F}_p}$ stems from the following theorem, which shows that Pizer graphs have mixing properties as good as it gets: They are *Ramanujan graphs*.

Theorem 2.75 [Piz90]**.** *Let p and ℓ be distinct primes with $\ell < p/4$. Then, the supersingular ℓ-isogeny graph over $\overline{\mathbb{F}_p}$ is an expander. In particular, in the notation of Proposition 2.72, we have $\lambda \leq 2\sqrt{\ell}$, which is asymptotically optimal.*

Is there anything else to say about Pizer graphs? Contrary to the volcanic case, there appears to be very little regular structure in Pizer graphs besides the natural subgraph of curves defined over $\mathbb{F}_p$. However, there are actually many more "hidden" volcanic subgraphs in any supersingular ℓ-isogeny graph over $\overline{\mathbb{F}_p}$: it just seems computationally hard to determine if a given curve lies on one of them or not. The underlying raison d'être for the hidden volcanoes is a functorial correspondence between pairs (E, α) where $E/\overline{\mathbb{F}_p}$ and $\alpha \in \mathrm{End}(E)$, and pairs $(\mathcal{E}, A)$ with $\mathcal{E}/\mathbb{C}$ and $A \in \mathrm{End}(\mathcal{E})$, such that reduction modulo a prime ideal of $(\mathcal{E}, A)$ yields (E, α). This connection is given by Deuring's lifting and reduction theorems (Theorem 5.26 and 5.30).

Computationally, the trouble is that it seems difficult to "see" the structure induced by a particular quadratic subring of a maximal order in $B_{p,\infty}$, without first going through the effort of computing endomorphisms — a problem that appears (and is often assumed) to be hard. However, Colò and Kohel [CK19] have made constructive use of quadratic subrings embedded in quaternionic endomorphism rings to construct a key-exchange algorithm named *Oriented Supersingular-Isogeny Diffie–Hellman* (OSIDH); see also [Onu20] for a more detailed explanation of the underlying group action including proofs.

2.5.4 – The isogeny problem. Let us briefly summarize the state of the art of isogeny computation. In cryptographic applications, there are many slightly or substantially different variations of isogeny-finding problems. Probably the most general variant is what we refer to as the "pure" isogeny problem:

Definition 2.76. *The* (pure) isogeny problem *is to compute an isogeny between two given elliptic curves E, E' known to be isogenous over a field k. (Usually, it is implicitly assumed that the inputs are given as a list of Weierstraß coefficients, and that the output is to be represented "efficiently": for instance, as a polynomially-sized description that can be evaluated at points efficiently.)*

Variants of the isogeny problem commonly used in cryptography can be both easier and harder: For instance, it may be required that the isogeny be defined over a specific field, have (power)smooth or small degree, induce a prescribed group homomorphism on some subgroup, or any combination of such constraints. Interestingly, even though prescribed action on a subgroup is an additional demand, it can actually help an attacker: Knowing the restriction of an isogeny of known degree to a sufficiently large torsion subgroup opens the door to *torsion-point attacks*; see Chapter 7 in this thesis. In contemporary isogeny-based cryptography, it is often the case that solving the pure isogeny problem suffices to break the scheme, but it is typically not known whether breaking the cryptosystem always solves the isogeny problem; indeed, known attacks suggest that this may be false in some cases.

To briefly survey the existing algorithms for the isogeny problem over finite fields, let $k = \mathbb{F}_q$ of characteristic p.

Commutative endomorphism rings. For curves with commutative endomorphism rings, the best known algorithms are refinements of an algorithm due to Galbraith [Gal99]. In a nutshell, given two elliptic curves E_1, E_2 with the same number of points over a finite field, the algorithm consists of "walking up" to the crater of every ℓ-volcano, yielding two curves E_1', E_2' with the maximal order as endomorphism rings, hence by Theorem 2.55 they must must be connected by a horizontal isogeny.[37] Such an isogeny is then found either by generic meet-in-the-middle (grow trees from both target nodes using random isogenies of varying degrees until they meet), or quantumly [CJS14] by reducing to a hidden-shift problem as in Example 2.82. The classical algorithm takes exponential time $\widetilde{O}(q^{1/4})$, whereas the quantum algorithm is (under GRH) subexponential with complexity $L_q[1/2, \sqrt{3}/2] \subseteq \exp((\log p)^{\frac{1}{2}+o(1)})$.

Quaternionic endomorphism rings. In the non-commutative case (i.e., supersingular over $\mathbb{F}_{p^2}$), every ℓ-isogeny graph is connected (Theorem 2.75), so running a generic path-finding algorithm on just one ℓ-isogeny graph is sufficient and takes time $\widetilde{O}(p^{1/2})$ with a simple meet-in-the-middle approach.[38] Delfs and Galbraith [DG16] achieve the same time complexity using significantly less memory by splitting the computation into two stages: Approximately a square-root fraction (see Fact 2.50) of all supersingular elliptic curves are in fact already defined over $\mathbb{F}_p$; thus, we first search for a path to that (easily recognizable) subgraph and then solve an easier isogeny problem over $\mathbb{F}_p$ using any of the methods for the commutative case. The asymptotic time complexity is roughly the same as that of the naïve algorithm since the complexity of finding the $\mathbb{F}_p$-subgraph by brute-force random walks remains $\widetilde{O}(p^{1/2})$, but Delfs–Galbraith is essentially memoryless and hence much more efficient than naïve meet-in-the-middle on realistic computer architectures.

Using quantum computers, one can do better: Biasse, Jao, and Sankar [BJS14] noticed that applying Grover's algorithm (Section 2.6.4) to the first stage (finding the $\mathbb{F}_p$-subgraph) reduces the overall complexity of finding an isogeny to $\widetilde{O}(p^{1/4})$. Moreover, a quantum claw-finding algorithm due to Tani [Tan07] has been claimed to recover an isogeny of known smooth degree d using $O(d^{1/3} \cdot \mathrm{polylog}(p))$ operations, but Jaques and Schanck [JS19] argue that the complexity is actually $O(d^{2/3} \cdot \mathrm{polylog}(p))$ when accessing data structures is properly costed.

2.6 — Quantum algorithms

This section gives a brief account of the basics of quantum computing. However, quantum algorithms (and quantum physics even more so) are broad topics and we cannot possibly hope to contain all of it in this short overview. For deeper insights, we refer to Chuang and Nielsen's *Quantum Computation and Quantum Information* [NC11].

2.6.1 – Computational model. We summarize the mathematical formulation of quantum computing. Note that there are multiple proposals for physical realizations of this model, each with different advantages and drawbacks, but the fundamental view of quantum computing does not change — very much like conventional general-purpose computing architectures differ in aspects such as instruction sets and efficiency, but not in their fundamental capabilities.

Qubits. The fundamental unit of data in quantum computing is the *qubit*.[39] Mathematically, it is represented as a two-dimensional complex vector space H. Alluding to conventional digital

[37] Except for the pathological special cases with two orbits, but those can be avoided by twisting the input curves.
[38] Recall that there are $p/12 + O(1)$ supersingular j-invariants.
[39] Not to be confused with the *cubit*, an entirely classical (even ancient) unit of length.

computers, two arbitrary orthogonal vectors are labeled $|0\rangle$ and $|1\rangle$ and called the *computational basis* or *standard basis*.[40] The *states* of a qubit are unit vectors $\alpha|0\rangle + \beta|1\rangle \in H$. When $\alpha, \beta \neq 0$, such a state is referred to as a *superposition* of $|0\rangle$ and $|1\rangle$. The complex coefficients α, β are called *amplitudes* of the corresponding basis states $|0\rangle, |1\rangle$.

Note that generally, multiplying a quantum state by a complex constant of norm 1 yields a physically indistinguishable state: "Global phase does not matter." Hence, the state space of a qubit is actually more correctly viewed as a complex projective line $\mathbb{CP}^1$. In this spirit, we will often abuse notation and omit global scaling factors in $\mathbb{C}$ when writing down quantum states.

Measurements. Qubits can be *measured* with respect to arbitrary orthogonal bases: Mathematically, this simply means sampling a basis state at random with the probability of each basis state given by the square of the norm of its amplitude. Hence, when measuring $\alpha|0\rangle + \beta|1\rangle$ in the computational basis, the outcome is $|0\rangle$ with probability $|\alpha|^2$ and $|1\rangle$ with probability $|\beta|^2$. During the measurement, the state *collapses*, i.e., the qubit remains in the observed basis state.

Entanglement. The crucial physical property that distinguishes quantum computing from classical computing is *entanglement*: Qubits can be combined in such a way that their state spaces become dependent, which most importantly means that the state of one qubit is influenced by measuring the other. Mathematically, this is formalized by viewing the joint state of multiple qubits as a tensor product of the individual state spaces.[41] For example, a two-qubit system is a four-dimensional complex vector space spanned by the elementary tensors $|0\rangle \otimes |0\rangle$, $|0\rangle \otimes |1\rangle$, $|1\rangle \otimes |0\rangle$, $|1\rangle \otimes |1\rangle$. For ease of notation, we will usually write the state $|a\rangle \otimes |b\rangle$ simply as $|ab\rangle$. A state in a joint state space $H_1 \otimes H_2$ is called *entangled* if it is not an elementary tensor, i.e., cannot be written as $\psi_1 \otimes \psi_2$ with $\psi_i \in H_i$. An example: In the two-qubit case above, the state $|00\rangle + |11\rangle$ is entangled, but the state $|00\rangle + |01\rangle$ equals $|0\rangle \otimes \big(|0\rangle + |1\rangle\big)$ and is therefore not entangled.[42] *Separable* states are non-entangled states.

Quantum algorithms are commonly formulated using notation like $|x\rangle$ with x some mathematical object (often an integer): This is shorthand for fixing some encoding $\mathrm{enc}(x) \in \{0,1\}^\ell$ as a bit string and referring to the state $|\mathrm{enc}(x)\rangle$.

Measuring entangled states. When measurements are applied only to a subset of the qubits in an entangled state, the joint state collapses into something compatible with the measurement. For example, measure the first qubit of the state $|00\rangle + |10\rangle + |11\rangle$ in the computational basis: With probability $1/3$, the outcome is $|0\rangle$, leaving the system in the state $|00\rangle$. Otherwise, the outcome is $|1\rangle$, which leaves the system in the (separable) state $|10\rangle + |11\rangle = |1\rangle \otimes \big(|0\rangle + |1\rangle\big)$.

Unitaries. Besides measurements, the only other operations one can apply on a quantum computer are *unitary operators* (or *unitaries* for short): For n-qubit systems, they are described by complex $2^n \times 2^n$ matrices U such that $\overline{U}^T U = 1$. (Equivalently, U is an isometry for $\| \ \|_2$.) Applying a unitary to a n-qubit state then simply consists of a matrix-vector multiplication.[43]

---

[40] The integers 0 and 1 in the *bra-ket* notation $|\ \rangle$ do not have any inherent meaning; the basis states could just as well be called v and w or even $|pineapple\rangle$ and $|pizza\rangle$. However, $|0\rangle$ and $|1\rangle$ are common because the representation of data on qubits resembles classical digital encoding.

[41] When qubit states are formalized as points in $\mathbb{CP}^1$ instead, this tensor product is replaced by the *Segre embedding* $\mathbb{CP}^n \times \mathbb{CP}^m \hookrightarrow \mathbb{CP}^{nm+n+m}$.

[42] These vectors clearly do not have norm 1: Recall that we omit global scaling factors.

[43] In the projectivized view of an n-qubit state as a point in $\mathbb{CP}^{2^n-1}$, we may also quotient unitaries by the subgroup of scalar matrices; hence unitary operators may be viewed as elements of the projective unitary group $\mathrm{PU}(2^n)$.

Real quantum computers are expected to implement only a very small subset of all possible unitaries — these building blocks are called *(quantum) gates* analogous to classical computing — and more complex unitaries[44] have to be constructed by composing potentially very many of these gates. All quantum gates are invertible by definition, which turns out to pose a challenge for algorithms design; see the discussion of "uncomputation" in Section 9.11.3.

Quantum gates. We list some of the most important quantum gates:

- The *Hadamard gate* H maps $|0\rangle$ to $|0\rangle + |1\rangle$ and $|1\rangle$ to $|0\rangle - |1\rangle$. Applied to an array of n qubits simultaneously, the operator $H^{\otimes n}$ is frequently used as a first step in quantum algorithms to set up a uniform superposition of all bit strings of length n.
- The *Pauli X gate* swaps $|0\rangle$ and $|1\rangle$, so it corresponds to the classical `NOT` gate.
- The *phase-shift gate* R_ϕ leaves $|0\rangle$ unchanged and scales $|1\rangle$ by $e^{\mathbf{i}\phi}$. Notably, applying R_ϕ does not immediately influence measurements, but the hidden phase information can subsequently be processed further to be made useful. In fact, *phase estimation* is the core of Shor's and Kuperberg's algorithms (Sections 2.6.2 and 2.6.3).
- For any unitary U operating on n qubits, *controlled-U* is the unitary on $1+n$ qubits leaving all states $|0\rangle \otimes \psi$ unchanged and mapping $|1\rangle \otimes \psi$ to $|1\rangle \otimes U\psi$. Thus, it may be interpreted as applying U in superposition conditioned on the controlling qubit.

A *quantum algorithm* consists of applying a sequence of quantum gates to an array of qubits, possibly intertwined with measurements. (Intermediate measurement outcomes may influence the choice of gates to apply in the future; see [JS19] for an in-depth discussion of the interplay between classical control hardware and quantum experiment.)

Quantum algorithms are often expressed and rendered graphically as circuits, but note that the picture represents a temporal sequence of unitaries applied one after another by control hardware, rather than a spatial arrangement of physical gates (such as when drawing classical electronic circuits).

Quantum $\geq$ classical. It is not immediately clear that quantum computers are no less powerful than classical computers. The first hurdle is gate completeness: Can we express enough Boolean operations in terms of unitaries to build arbitrary circuits? It is not hard to see that classical `XOR` is simply controlled Pauli X, also known as `CNOT`, which maps $(|a\rangle, |b\rangle) \mapsto (|a\rangle, |a \oplus b\rangle)$.

Unfortunately, for nonlinear operations such as `AND`, implementing $(|a\rangle, |b\rangle) \mapsto (|a\rangle, |a \,\&\, b\rangle)$ in place is impossible as the pair (a, b) is not uniquely determined by $(a, a \;\&\; b)$, hence such a transformation cannot be unitary. Instead, we add an auxiliary *ancilla* qubit and compute $(|a\rangle, |b\rangle, |c\rangle) \;\mapsto\; (|a\rangle, |b\rangle, |c \oplus (a \;\&\; b)\rangle)$. This unitary is known as a *Toffoli gate* and consists of simply swapping $|110\rangle$ and $|111\rangle$ while leaving all other basis states unchanged. Applying this operation with $c = 0$ clearly leaves us with $a \,\&\, b$ in the third qubit, and since {`NOT`, `XOR`, `AND`} is a complete set of gates we may conclude that any classical circuit can equivalently be expressed as a quantum computation in principle.

Note there are quite a few more devils in the details of computational models for quantum algorithms, as well as various conversions of classical computation to quantum computation. See Section 9.11 for some extended discussion on this matter.

2.6.2 – Shor's algorithm. After this short introduction to quantum computing, let us have a look at a particular — and perhaps the most famous — quantum algorithm: Shor's algorithm.

[44] No pun intended.

There are several variants of the algorithm: It was originally [Sho94] described for integer factorization and discrete logarithms in $(\mathbb{Z}/n)^\times$, but both of these applications can be recast in the more general framework of recovering the *period lattice* of an efficiently computable periodic function [BL95; Kit96]; see Theorem 2.79 below.

The core technical ingredient in all these results is the *quantum Fourier transform* (QFT):

Definition 2.77. *For a positive integer N, the quantum Fourier transform QFT_N takes $|a\rangle$ to the state*

$$\frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} e^{aj \cdot 2\pi \mathbf{i}/N} \, |j\rangle \, .$$

It was Shor's observation that the QFT can be computed efficiently when N is smooth: Then, one may mimic classical fast Fourier transform algorithms (e.g. Cooley–Tukey) to quickly compute the QFT. For $N = 2^n$ the resulting quantum circuit consists of $O(n^2)$ Hadamard and controlled phase-shift gates. Note that it is not known how to efficiently compute the QFT for arbitrary N; however, in applications it is usually sufficient to approximate QFT_N by another $\mathrm{QFT}_{N'}$ for smooth $N' \geq N$. Bounds on the error introduced by this substitution can be obtained using Fourier analysis.

Given that the QFT is a Fourier transform, it should not come as a huge surprise that we can use it to detect periods in functions:

Example 2.78 [Sho97a]**.** *Let $f : \mathbb{Z} \to S$ be an efficiently computable function to some set S with an unknown period $\lambda \in \mathbb{Z}$ that is no more, but also not much less, than ℓ bits long. Then we may recover λ as follows:*

- *Let $m = 2^{2\ell}$. Perform the quantum operations*

$$\sum_x |0^{2\ell}\rangle \overset{H^{\otimes 2\ell}}{\longmapsto} \sum_x |x\rangle \overset{f}{\longmapsto} \sum_x |x\rangle \, |f(x)\rangle$$
$$\overset{\mathrm{QFT}_m}{\longmapsto} \sum_x \sum_y e^{xy \cdot 2\pi \mathbf{i}/m} \, |y\rangle \, |f(x)\rangle \, ,$$

  where all sums range from 0 to $m-1$ and the QFT is applied to the first register.

- *Measure the first register. The probability of getting a particular outcome $|Y\rangle$ is proportional to*

$$\Big| \sum_{x \equiv Y} e^{xY \cdot 2\pi \mathbf{i}/m} \Big|^2$$

  where the sum ranges over all x between 0 and $m-1$ with $x \equiv Y \pmod{\lambda}$. Write each such x as $k_x \lambda + Y$; substituting this into the expression $xY \cdot 2\pi\mathbf{i}/m$ from above yields

$$xY \cdot 2\pi\mathbf{i}/m = (k_x \lambda Y + Y^2) \cdot 2\pi\mathbf{i}/m =: e_x \, .$$

  When Y is a multiple of λ, then e_x will be close to an integer multiple of $2\pi\mathbf{i}$ regardless of x — hence the amplitudes pile up. If on the other hand Y is not close to a multiple of λ, then the values e_x are well-spread modulo $2\pi\mathbf{i}$ and significant cancellation occurs in the amplitude of $|Y\rangle$.

  Making this argument precise shows that the measurement of $|Y\rangle$ exhibits strong probability peaks centered at multiples of λ. Recovering λ precisely from the observed outcome(s) requires some more work, but is usually feasible; for example, Shor solves this problem using continued fractions.

Famously, we can combine this technique with the well-known fact that factoring reduces to order-finding in multiplicative groups modulo integers to obtain:

Shor's factoring algorithm. Let n be an odd composite positive integer, and suppose we wish to find a proper divisor d of n. Using Example 2.78, we can easily do so:

- Pick a random $a \in \{1, \ldots, n-1\}$. Assume $\gcd(a, n) = 1$; otherwise we are done.
- Define
$$f\colon \mathbb{Z} \to (\mathbb{Z}/n)^\times,\ i \mapsto a^i \bmod n$$
and note this function can be evaluated efficiently using square-and-multiply.
- Apply the period-finding routine from Example 2.78 to f and call the output γ.
- With high probability, γ is close to a multiple of the multiplicative order k of a modulo n; say $\gamma = \mu k$. Since k is likely to be about the size of n, we get $\gamma/m \approx \mu k/k^2 = \mu/k$. Thus, k may be recovered using a continued-fraction expansion, which approximates γ/m as fractions involving smaller integers.
- If k is even, compute $d := \gcd(a^{k/2}-1, n)$ and return d if it is a proper divisor of n. If k is odd or $d \in \{1, n\}$, start over.

One can show using some elementary number theory that the probability of success in the last step is $1 - 2^{-r}$, where r is the number of distinct prime factors of n.

Developing the ideas from Shor's factoring algorithm further leads to the following result:

Theorem 2.79 [BL95]**.** *There is a quantum algorithm which, given an efficiently computable function $f\colon \mathbb{Z}^r \to S$ that factors through a "hidden" lattice $\Lambda \subseteq \mathbb{Z}^r$ as*
$$f\colon\ \mathbb{Z}^r \longrightarrow\!\!\!\!\!\rightarrow\ \mathbb{Z}^r/\Lambda \lhook\joinrel\longrightarrow\ S\,,$$
recovers a basis of Λ in time polynomial in $\log|\det\Lambda|$.

As an immediate application, we can compute discrete logarithms in any group:

Example 2.80. *Let $G = \langle g\rangle$ be a finite group of order q (with efficiently computable operations) and suppose given $h \in G$. Let $a \in \{0, \ldots, q-1\}$ be the (unknown) discrete logarithm of h, hence $h = g^a$.*

To recover a, we define the group homomorphism
$$f\colon\ \mathbb{Z}^2 \to G,\ (x, y) \mapsto g^x h^y\,.$$
Clearly, the kernel of f is contained in $q\mathbb{Z}^2$, and moreover it contains the vector $(a, -1)$. Thus
$$\ker(f)\ =\ \big\langle (q, 0), (a, -1)\big\rangle\,,$$
which we may recover using Theorem 2.79. Simple classical post-processing then suffices to find a vector of the form $(\tau, -1)$ in the lattice $\ker(f)$, which solves the DLP.

2.6.3 – The hidden-shift problem. Evidently, for a general group action $G \times X \to X$, we simply cannot define the period map f from Example 2.80 due to the lack of an efficient meaningful multiplication map $X \times X \to X$. This small observation supports the idea that polynomial-time techniques à la Shor cannot break the vectorization problem for group actions. Instead, the most straightforward formulation as a problem suitable for quantum algorithms seems to be the following:

Definition 2.81. *Let $(G, +)$ be an abelian group with efficiently computable operations. The* (abelian) hidden-shift problem *in G is: Given (efficient descriptions of) two functions $f_0, f_1\colon G \to Y$ such that there exists $\sigma \in G$ with $f_1(x) = f_0(x + \sigma)$ for all $x \in G$, recover such a* hidden shift *σ of (f_0, f_1).*

Clearly, not all pairs (f_0, f_1) determine σ uniquely: For example, if f_0 and f_1 are constant, any $\sigma \in G$ is a correct answer to the problem. Similarly, if f_0 is a group homomorphism, then σ is only defined up to the kernel. When f_0 is injective, the solution to the hidden-shift problem is always unique.

As promised, we can express the vectorization problem as an abelian hidden-shift problem:

Example 2.82. *Let $*\colon G \times X \to X$ be a group action of a finite abelian group G on a set X.*

Suppose given an instance (x_0, x_1) of the vectorization problem; hence, we wish to compute an $\mathfrak{a} \in G$ such that $x_1 = \mathfrak{a} * x_0$. This problem can be reduced to an abelian hidden-shift problem by defining

$$f_i\colon G \to X,\ \mathfrak{g} \mapsto \mathfrak{g} * x_i\,.$$

By construction, $f_1(\mathfrak{g}) = f_0(\mathfrak{g} \cdot \mathfrak{a})$, hence the sought-after secret $\mathfrak{a}$ is a shift of (f_0, f_1).

Note that in this case, the shift is only defined by the pair (f_0, f_1) up to the stabilizer subgroup G_{x_0}; however, any element of the set $\mathfrak{a} + G_{x_0}$ of possible shifts is a valid solution to the vectorization problem and forms an equivalent (correct) key for the group-action Diffie–Hellman scheme.

Remark 2.83. *In a sense, vectorization is really the archetypical example of a hidden-shift problem: The group G acts in a natural way on the set of functions $G \to Y$ by defining*

$$(g * f)\colon\ G \to Y,\ x \mapsto f(g + x)\,.$$

for $g \in G$ and $f\colon G \to Y$. Recovering a hidden shift between f_0 and f_1 simply means finding an element that vectorizes the pair (f_0, f_1) with respect to this action.

Kuperberg's theorem. The best known algorithm to solve the abelian hidden-shift problem is a subexponential quantum algorithm due to Kuperberg [Kup05; Kup13]:

Theorem 2.84 (Kuperberg's algorithm)**.** *On a quantum computer, the abelian hidden-shift problem in a group of size n can be solved in time and space $2^{O(\sqrt{\log n})}$.*

Simplifications. We content ourselves with a rough overview of the algorithm, as the details are a bit intricate and there are numerous configurable parameters to optimize the algorithm for different metrics [BS20; Pei20]. For the sake of simplicity, assume that f_0 is injective (this is true without loss of generality in the group-action application).

First, we restrict our attention to the cyclic group $G = \mathbb{Z}/n$ — the general case works very similarly by decomposing G into an internal direct product $C \times H$ with C cyclic, always clearing the H component in the sieving step below until the C component has been fully determined, and reducing the problem to a hidden shift in H. (The structure of G, as well as isomorphisms to quotients of $\mathbb{Z}^r$, can be computed in quantum polynomial time using Theorem 2.79.)

Moreover, we can replace the group $G = \mathbb{Z}/n$ by $\mathbb{Z}/2^\ell$ for a sufficiently large "bit length" ℓ. (This can be interpreted as representing the group $\mathbb{R}/\mathbb{Z}$, which contains $\frac{1}{n}\mathbb{Z}/\mathbb{Z} \cong \mathbb{Z}/n$, through a finite binary approximation.) Hence, we apply *modulus switching* to the hiding functions (f_0, f_1):

$$F_i\colon\ \mathbb{Z}/2^\ell \mapsto S,\ X \mapsto f_i\big(\lfloor n/2^\ell \cdot X \rfloor\big)\,.$$

Note that unless n is itself a power of two, the functions (F_0, F_1) are not strictly shifts of one another, but for big ℓ this is the case *almost* everywhere: Indeed, defining $\Sigma := \lceil 2^\ell/n \cdot \sigma \rceil$, we have $0 \le n/2^\ell \cdot \Sigma - \sigma < n/2^\ell$, hence

$$F_1(X) = f_1\big(\lfloor n/2^\ell \cdot X \rfloor\big) = f_0\big(\lfloor n/2^\ell \cdot X \rfloor + \sigma\big) = f_0\big(\lfloor n/2^\ell \cdot X + \sigma \rfloor\big)$$

and

$$F_0(X+\Sigma) = f_0\big(\lfloor n/2^\ell \cdot X + \underbrace{n/2^\ell \cdot \Sigma}_{<\sigma+n/2^\ell}\rfloor\big)$$

are guaranteed to be equal whenever

$$\lfloor n/2^\ell \cdot X + n/2^\ell \rfloor = \lfloor n/2^\ell \cdot X \rfloor\,,$$

or equivalently

$$nX \bmod 2^\ell \;<\; 2^\ell - n\,.$$

Over all inputs $X \in \{0,\ldots,2^\ell-1\}$, this property is violated exactly n times, which constitutes an exponentially small fraction as ℓ grows. Thus, by picking a sufficiently large ℓ, we can assume that the input pairs sampled in the algorithm below are indeed shifted by Σ exactly: Counterexamples are so sparse that they should not ever appear in practice.

The algorithm. Finally, the main steps of Kuperberg's algorithm [Kup13] for the important special case $G = \mathbb{Z}/n$, where n is a power of two, are as follows:

- Perform the following quantum computation "many" times:
  - Let $I = \{0,1\} \times G$ and set up the superposition

    $$\sum_{(b,x)\in I} |b\rangle\,|x\rangle\,|f_i(x)\rangle\,.$$

    Measure and discard the third "output" register, resulting in a state

    $$|0\rangle\,|x\rangle + |1\rangle\,|x{+}\sigma\rangle\,.$$

  - Apply QFT_n to the second register; this results in the state

    $$\sum_{j=0}^{n-1} \left(e^{2\pi\mathbf{i}/n\cdot xj}\,|0\rangle + e^{2\pi\mathbf{i}/n\cdot(x+\sigma)j}\,|1\rangle\right)|j\rangle\,.$$

    Measure the second register; this yields a *label* $k \in \{0,\ldots,n-1\}$ and the state

    $$e^{2\pi\mathbf{i}/n\cdot xk}\,|0\rangle + e^{2\pi\mathbf{i}/n\cdot(x+\sigma)k}\,|1\rangle\,.$$

    Writing $\zeta := e^{2\pi\mathbf{i}/n\cdot\sigma}$ and ignoring the global phase yields a so-called *phase vector*

    $$\psi_k \;=\; |0\rangle + \zeta^k|1\rangle\,.$$

    Note that k is known to the algorithm, but cannot a priori be influenced.

  The result is a "large" set V of phase vectors ψ_k.
- The second stage of the algorithm consists of *sieving* the phase vectors in order to obtain a combination suitable for extracting useful information about σ. The goal is to get our hands on $\psi_{n/2}$; we will see below how this helps in finding σ.

  At any given layer of the sieve, we have a set of labeled phase vectors available, and we *combine* them in such a way that new phase vectors emerge with fewer label bits set.

  An easy method, as used in Kuperberg's *first* algorithm [Kup05], is to take two phase vectors ψ_k and ψ_ℓ, apply a controlled NOT operation of ψ_k on ψ_ℓ, and measure the result. This leaves the other qubit in the state $|0\rangle + \zeta^{\ell+k}|1\rangle$ if $|0\rangle$ is measured and $|0\rangle + \zeta^{k-\ell}|1\rangle$ if $|1\rangle$ is

measured. Combining ψ_k and ψ_ℓ thus yields either $\psi_{k+\ell}$ or $\psi_{k-\ell}$, and the measurement tells us which one it is.

Applying this step to states ψ_k and ψ_ℓ whose labels k, ℓ have r trailing bits in common has a $1/2$ probability of yielding a new state with r trailing zero bits. We may thus zero out the labels in our pool of phase vectors from the bottom by arranging suitable combination steps between them in a tree structure, until we reach the desired phase vector $\psi_{n/2}$.

A more efficient way of combining phase vectors is *collimation*, which is the core of Kuperberg's *second* algorithm [Kup13]: The concept of phase vectors is generalized to more than just two basis states, and the combination step involves "filtering" a large product state for phase vectors with fewer label bits set by performing a well-chosen measurement.

- The sieving stage yields the phase vector $\psi_{n/2}$, which equals either $|0\rangle + |1\rangle$ or $|0\rangle - |1\rangle$ depending on the parity of σ. These two states can be distinguished reliably by a single measurement, and thus we learn the least significant bit β of the hidden shift σ.
- Finally, to recover the remaining bits, we replace f_1 by the function

$$f_1' \colon G \to S,\ x \mapsto f_1(x - \beta)\,,$$

such that the pair (f_0, f_1') is shifted by $\sigma - \beta \in 2G$. Hence, replace the group G by the subgroup $2G \cong \mathbb{Z}/(n/2)$ and proceed recursively until all bits of σ have been recovered.

Analyzing the number of bits "cancelled" in each layer of the sieve shows that we should start off with $2^{O(\sqrt{\log n})}$ states in the first layer and cancel $O(\sqrt{\log n})$ bits in each layer to end up with the complexity claimed in Theorem 2.84 and a good chance of finding $\psi_{n/2}$.

Note that the algorithm evaluates the oracle functions a superpolynomial number of times in superposition. The cost of these computations can be very significant, and Chapter 9 in this thesis analyzes the cost of the oracle calls when attacking a particular parameterization of CSIDH (Chapter 3) using Kuperberg's algorithm.

2.6.4 – Grover's algorithm. Another fundamental quantum algorithm that impacts cryptography is due to Grover [Gro96]. It can solve *unstructured search problems* with a square-root speedup; to be precise, given a quantum circuit that computes a function

$$f \colon S \to \{0, 1\}$$

on a set S of size N, with the property that $f(x) = 0$ almost everywhere, the algorithm finds an input $x \in S$ such that $f(x) = 1$ within $O(\sqrt{N})$ evaluations of f and a few additional quantum operations.

"Groverizing" classical search algorithms is a common technique in cryptanalysis: In many cases, it can accelerate attacks or substeps of attacks by an asymptotic square-root speedup, but note that this asymptotic speedup does not always materialize in terms of concrete (in)security due to the potentially high cost of implementing the oracle function f as a quantum circuit. Another practical problem is that the algorithm performs many sequential operations, hence requires the qubits in the quantum computer to remain coherent for a long time.

The internals of Grover's algorithm are rather straightforward, but we will skip the details here since they will not be of much interest in the sequel. See Chapter 7 for examples where Groverization of some steps yields a faster quantum version of an a priori classical attack.

Chapter 3

CSIDH: An efficient post-quantum group action

This chapter is for all practical purposes identical to the paper *CSIDH: an efficient post-quantum commutative group action* [Cas+18] authored jointly with Wouter Castryck, Tanja Lange, Chloe Martindale, and Joost Renes, which was published at Asiacrypt 2018.

3.1 — Introduction

During the past five to ten years, elliptic-curve cryptography (ECC) has taken over public-key cryptography on the internet and in security applications. Many protocols such as Signal or TLS 1.3 rely on the small key sizes and efficient computations to achieve forward secrecy, often meaning that keys are used only once. However, it is also important to notice that security does not break down if keys are reused. Indeed, some implementations of TLS, such as Microsoft's SChannel, reuse keys for some fixed amount of time rather than for one connection [Ber+14]. Google's QUIC protocol relies on servers keeping their keys fixed for a while to achieve quick session resumption. Several more examples are given by Freire, Hofheinz, Kiltz, and Paterson in their paper [FHKP13] formalizing non-interactive key exchange. Some applications require this functionality and for many it provides significant savings in terms of roundtrips or implementation complexity. Finding a post-quantum system that permits non-interactive key exchange while still offering decent performance is considered an open problem. This chapter presents a solution to this problem using isogenies of elliptic curves.

The first proposal of an isogeny-based cryptosystem, made by Couveignes in 1997 [Cou06], described a non-interactive key exchange protocol where the space of public keys equals the set of $\mathbb{F}_q$-isomorphism classes of ordinary elliptic curves over $\mathbb{F}_q$ whose endomorphism ring is a given order $\mathcal{O}$ in an imaginary quadratic field and whose trace of Frobenius has a prescribed value. It is well-known that the ideal-class group $\mathrm{cl}(\mathcal{O})$ acts freely and transitively on this set through the application of isogenies. Couveignes' central observation was that the commutativity of $\mathrm{cl}(\mathcal{O})$ naturally allows for a key-exchange protocol in the style of Diffie and Hellman [DH76]. His work was only circulated privately and thus not picked up by the community; the corresponding paper [Cou06] was never formally published and posted on ePrint only in 2006. The method was eventually independently rediscovered by Rostovtsev and Stolbunov in 2004 (in Stolbunov's master's thesis [Sto04] and published on ePrint as [RS06] in 2006). In 2010, Childs, Jao and Soukharev [CJS14] showed that breaking the Couveignes–Rostovtsev–Stolbunov scheme amounts to solving an instance of the abelian hidden-shift problem, for which quantum algorithms with a time complexity of $L_q[1/2]$ are known to exist; see [Kup05; Reg04]. While this may be tolerable (e.g., classical subexponential factorization methods have not ended the widespread use of RSA), a much bigger concern is that the scheme is unacceptably slow: despite

recent clever speed-ups due to De Feo, Kieffer, and Smith [DKS18; Kie17], several minutes are needed for a single key exchange at a presumed classical security level of 128 bits. Nevertheless, in view of its conceptual simplicity, compactness, and flexibility, it seems a shame to discard the Couveignes–Rostovtsev–Stolbunov scheme.

The attack due to Childs–Jao–Soukharev strongly relies on the fact that $\mathrm{cl}(\mathcal{O})$ is commutative, hence indirectly on the fact that $\mathcal{O}$ is commutative. This led Jao and De Feo [JD11] to consider the use of supersingular elliptic curves, whose full ring of endomorphisms is an order in a quaternion algebra; in particular it is non-commutative. Their resulting (interactive) key-agreement scheme, which nowadays goes under the name "Supersingular Isogeny Diffie–Hellman" (SIDH), has attracted almost the entire focus of isogeny-based cryptography over the past six years. The current state-of-the-art implementation is SIKE [Jao+17], which was recently submitted to the NIST competition on post-quantum cryptography [NIST16].

It should be stressed that SIDH is *not* the Couveignes–Rostovtsev–Stolbunov scheme in which one substitutes supersingular elliptic curves for ordinary elliptic curves; in fact SIDH is much more reminiscent of an isogeny-based cryptographic hash function from 2006 due to Charles, Goren, and Lauter [CLG09]. SIDH's public keys consist of the codomain of a secret isogeny and the image points of certain public points under that isogeny. Galbraith, Petit, Shani, and Ti showed in [GPST16] that SIDH keys succumb to active attacks and thus should not be reused, unless combined with a CCA transform such as the Fujisaki–Okamoto transform [FO99].

In this chapter we show that adapting the Couveignes–Rostovtsev–Stolbunov scheme to supersingular elliptic curves is possible, provided that one restricts to supersingular elliptic curves defined over a prime field $\mathbb{F}_p$. Instead of the full ring of endomorphisms over $\overline{\mathbb{F}_p}$, which is non-commutative, one should consider the subring of $\mathbb{F}_p$-rational endomorphisms, which is again an order $\mathcal{O}$ in an imaginary quadratic field. As before $\mathrm{cl}(\mathcal{O})$ acts via isogenies on the set of $\mathbb{F}_p$-isomorphism classes of elliptic curves whose $\mathbb{F}_p$-rational endomorphism ring is isomorphic to $\mathcal{O}$ and whose trace of Frobenius has a prescribed value; in fact if $p \geq 5$ then there is only one option for this value, namely 0, in contrast with the ordinary case. See e.g. [Wat69, Theorem 4.5], with further details to be found in [Bröo8; DG16] and in Section 3.3. Starting from these observations, the desired adaptation of the Couveignes–Rostovtsev–Stolbunov scheme almost unrolls itself; the details can be found in Section 3.4. We call the resulting scheme CSIDH, where the C stands for "commutative".[1]

While this fails to address Jao and De Feo's initial motivation for using supersingular elliptic curves, which was to avoid the $L_q[1/2]$ quantum attack due to Childs–Jao–Soukharev, we show that CSIDH eliminates the main problem of the Couveignes–Rostovtsev–Stolbunov scheme, namely its inefficiency. Indeed, in Section 3.8 we will report on a proof-of-concept implementation which carries out a non-interactive key exchange at a presumed classical security level of 128 bits and a conjectured post-quantum security level of 64 bits in about 80 milliseconds, while using key sizes of only 64 bytes. This is over 2000 times faster[2] than the current state-of-the-art instantiation of the Couveignes–Rostovtsev–Stolbunov scheme by De Feo, Kieffer and Smith [DKS18; Kie17], which itself presents many new ideas and speedups to even achieve that speed.

For comparison, we remark that SIDH, which is the NIST submission with the smallest combined key and ciphertext length, uses public keys and ciphertexts of over 300 bytes each. More

[1] Since this work was started while being very close to a well-known large body of salt water, we pronounce CSIDH as [ˈsiːˌsaɪd] rather than spelling out all the letters.

[2] This speed-up is explained in part by comparing our own C implementation to the Sage implementation of De Feo, Kieffer, and Smith.

precisely SIKE's version `p503` uses uncompressed keys of 378 bytes long [Jao+17] for achieving CCA security. The optimized SIKE implementation is about ten times faster than our proof-of-concept C implementation, but even at 80 ms, CSIDH is practical.

Another major advantage of CSIDH is that we can efficiently validate public keys, making it possible to reuse a key without the need for transformations to confirm that the other party's key was honestly generated.

Finally we note that just like the original Couveignes–Rostovtsev–Stolbunov scheme, CSIDH relies purely on the isogeny-finding problem; no extra points are sent that could potentially harm security, as argued in [Pet17]; see also Chapter 7.

To summarize, CSIDH is a new cryptographic primitive that can serve as a drop-in replacement for the (EC)DH key-exchange protocol while maintaining security against quantum computers. It provides a *non-interactive* (static–static) key exchange with full public-key validation. The speed is practical while the public-key size is the smallest for key exchange or KEM in the portfolio of post-quantum cryptography. This makes CSIDH particularly attractive in the common scenario of prioritizing bandwidth over computational effort. In addition, CSIDH is compatible with 0-RTT protocols such as QUIC.

Why supersingular? To understand where the bulk of the speed-up comes from, it suffices to record that De Feo–Kieffer–Smith had the idea of choosing a field of characteristic p, where p is congruent to -1 modulo all small odd primes ℓ up to a given bound. They then look for an ordinary elliptic curve $E/\mathbb{F}_p$ such that $\#E(\mathbb{F}_p)$ is congruent to 0 modulo as many of these ℓ's as possible, i.e., such that points of order ℓ exist over $\mathbb{F}_p$. These properties ensure that $\ell\mathcal{O}$ decomposes as a product of two prime ideals $\mathfrak{l} = (\ell, \pi - 1)$ and $\bar{\mathfrak{l}} = (\ell, \pi + 1)$, where π denotes the Frobenius endomorphism. For such primes the action of the corresponding ideal classes $[\mathfrak{l}]$ and $[\bar{\mathfrak{l}}] = [\mathfrak{l}]^{-1}$ can be computed efficiently through an application of Vélu-type formulae to E (resp. its quadratic twist E^t), the reason being that only $\mathbb{F}_p$-rational points are involved. If this works for enough primes ℓ, we can expect that a generic element of $\mathrm{cl}(\mathcal{O})$ can be written as a product of small integral powers of such $[\mathfrak{l}]$, so that the class-group action can be computed efficiently. However, finding an ordinary elliptic curve $E/\mathbb{F}_p$ such that $\#E(\mathbb{F}_p)$ is congruent to 0 modulo many small primes ℓ is hard, and the main focus of De Feo–Kieffer–Smith is on speeding up this search. In the end it is only practical to enforce this for 7 primes, thus they cannot take full advantage of the idea.

However, in the supersingular case the property $\#E(\mathbb{F}_p) = p + 1$ implies that $\#E(\mathbb{F}_p)$ is congruent to 0 modulo *all* primes $\ell \mid p + 1$ that we started from in building p! Concretely, our proof-of-concept implementation uses 74 small odd primes, corresponding to prime ideals $\mathfrak{l}_1, \mathfrak{l}_2, \ldots, \mathfrak{l}_{74}$ for which we heuristically expect that almost all elements of our 256-bit size class group can be written as $[\mathfrak{l}_1]^{e_1}[\mathfrak{l}_2]^{e_2} \cdots [\mathfrak{l}_{74}]^{e_{74}}$, where the exponents e_i are taken from the range $\{-5, \ldots, 5\}$; indeed, one verifies that $\log(2 \cdot 5 + 1)^{74} \approx 255.9979$. The action of such an element can be computed as the composition of at most $5 \cdot 74 = 370$ easy isogeny evaluations. This should be compared to using 7 small primes, where the same approach would require exponents in a range of length about $2^{256/7} \approx 2^{36}$, in view of which De Feo–Kieffer–Smith also resort to other primes with less beneficial properties, requiring to work in extensions of $\mathbb{F}_p$.

The use of supersingular elliptic curves over $\mathbb{F}_p$ has various other advantages. For instance, their trace of Frobenius t is 0, so that the absolute value of the discriminant $|t^2 - 4p| = 4p$ is as large as possible. As a consequence, generically the size of the class group $\mathrm{cl}(\mathcal{O})$ is close to its maximal possible value for a fixed choice of p. Conversely, this implies that for a fixed security level we can make a close-to-minimal choice for p, which directly affects the key size. Note

that this contrasts with the CM construction from [BS07], which could in principle be used to construct ordinary elliptic curves having many points of small order, but whose endomorphism rings have very small class groups, ruling them out for the Couveignes–Rostovtsev–Stolbunov key exchange.

To explain why key validation works, note that we work over $\mathbb{F}_p$ with $p \equiv 3 \pmod 8$ and start from the curve $E_0 \colon y^2 = x^3 + x$ with $\mathbb{F}_p$-rational endomorphism ring $\mathcal{O} = \mathbb{Z}[\pi]$. As it turns out, all Montgomery curves $E_A \colon y^2 = x^3 + Ax^2 + x$ over $\mathbb{F}_p$ that are supersingular appear in the $\mathrm{cl}(\mathcal{O})$-orbit of E_0. Moreover their $\mathbb{F}_p$-isomorphism class is uniquely determined by A. So all one needs to do upon receiving a candidate public key $y^2 = x^3 + Ax^2 + x$ is check for supersingularity, which is an easy task; see Section 3.5. The combination of large size of $\mathrm{cl}(\mathcal{O})$ and representation by a single $\mathbb{F}_p$-element A explains the small key size of 64 bytes.

3.1.1 – One-way group actions. Although non-interactive key exchange is the main application of our primitive, it is actually more general: It is (conjecturally) an instance of Couveignes' *hard homogeneous spaces* [Cou06], ultimately nothing but a finite commutative group action for which some operations are easy to compute while others are hard. Such group actions were first formalized and studied by Brassard and Yung [BY90]. We summarize Couveignes' definition:

Definition 3.1. *A* hard homogeneous space *consists of a finite commutative group G acting freely and transitively on some set X.*
The following tasks are required to be easy (e.g., polynomial-time):

- *Compute the group operations in G.*
- *Sample randomly from G with (close to) uniform distribution.*
- *Decide validity and equality of a representation of elements of X.*
- *Compute the action of a group element $g \in G$ on some $x \in X$.*

The following problems are required to be hard (e.g., not polynomial-time):

- *Given $x, x' \in X$, find $g \in G$ such that $g * x = x'$.*
- *Given $x, x', y \in X$ such that $x' = g * x$, find $y' = g * y$.*

Any such primitive immediately yields a natural Diffie–Hellman protocol: Alice and Bob's private keys are random elements a, b of G, their public keys are $a * x_0$ resp. $b * x_0$, where $x_0 \in X$ is a public fixed element, and the shared secret is $b * (a * x_0) = a * (b * x_0)$. The private keys are protected by the difficulty of the first hard problem above, while the shared secret is protected by the second problem. Note that traditional Diffie–Hellman on a cyclic group C is an instance of this, where X is the set of generators of C and G is the multiplicative group $(\mathbb{Z}/\#C)^*$ acting by exponentiation.

3.1.2 – Notation and terminology. We stress that throughout this chapter, we consider two elliptic curves defined over the same field identical whenever they are isomorphic *over that field*. Note that we do *not* identify curves that are only isomorphic over some extension field, as opposed to what is done in SIDH, for instance. In the same vein, for an elliptic curve E defined over a finite field $\mathbb{F}_p$, we let $\mathrm{End}_p(E)$ be the subring of the endomorphism ring $\mathrm{End}(E)$ consisting of endomorphisms defined over $\mathbb{F}_p$.[3] This subring is always isomorphic to an order in an imaginary quadratic number field. Conversely, for a given order $\mathcal{O}$ in an imaginary quadratic

[3] This constraint only makes a difference for supersingular curves: in the ordinary case, all endomorphisms are defined over the base field.

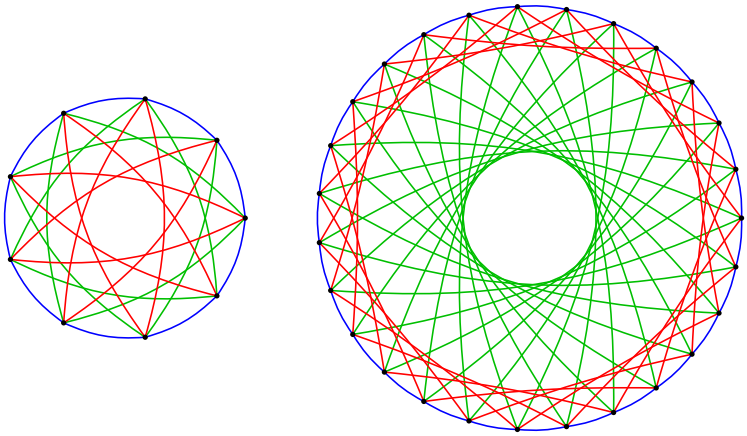


Figure 3.1: Union of the supersingular ℓ-isogeny graphs for $\ell \in \{3, 5, 7\}$ over $\mathbb{F}_{419}$. CSIDH makes use of the larger component, corresponding to curves whose ring of $\mathbb{F}_{419}$-rational endomorphisms is isomorphic to $\mathbb{Z}[\sqrt{-419}]$.

field and an element $\pi \in \mathcal{O}$, we let $\mathcal{E}\ell\ell_p(\mathcal{O}, \pi)$ denote the set of elliptic curves E defined over $\mathbb{F}_p$ with $\mathrm{End}_p(E) \cong \mathcal{O}$ such that π corresponds to the $\mathbb{F}_p$-Frobenius endomorphism of E. In particular, this implies that $\varphi \circ \beta = \beta \circ \varphi$ for all $\mathbb{F}_p$-isogenies φ between two curves in $\mathcal{E}\ell\ell_p(\mathcal{O}, \pi)$ and all $\beta \in \mathcal{O}$ interpreted as endomorphisms.

Ideals are always assumed to be non-zero.

The notation "log" refers to the base-2 logarithm.

Acknowledgements. This project started during a research retreat on post-quantum cryptography, organized by the European PQCRYPTO and ECRYPT-CSA projects in Tenerife from 29 January until 1 February 2018. We would like to thank Jeffrey Burdges, whose quest for a flexible post-quantum key exchange protocol made us look for speed-ups of the Couveignes–Rostovtsev–Stolbunov scheme. We are grateful to Luca De Feo, Jean Kieffer, and Ben Smith for sharing a draft of their paper [DKS18], and to Daniel J. Bernstein, Luca De Feo, Jeroen Demeyer, Léo Ducas, Steven Galbraith, David Jao, and Fré Vercauteren for helpful feedback.

3.2 — Isogeny graphs

Good mixing properties of the underlying isogeny graph are relevant for the security of isogeny-based cryptosystems. Just as in the original Couveignes–Rostovtsev–Stolbunov cryptosystem, in our case this graph is obtained by taking the union of several large subgraphs (each being a union of large isomorphic cycle graphs) on the same vertex set, one for each prime ℓ under consideration; see Figure 3.1 for a (small) example. Such a graph is the *Schreier graph* associated with our class-group action and the chosen generators. We refer to the lecture notes of De Feo [DeF17, §14.1] for more background and to [JMV09] for a discussion of its rapid mixing properties. One point of view on this is that one can quickly move between distant nodes in the subgraph corresponding to one generator by switching to the subgraph corresponding to another generator. This thereby replaces the square-and-multiply algorithm in exponentiation-based cryptosystems (such as classical Diffie–Hellman).

The goal of this section is to analyze the structure of the individual cycles.

Definition 3.2. *For a field k and a prime $\ell \nmid \mathrm{char}(k)$, the k-*rational ℓ*-*isogeny graph $G_{k,\ell}$ *is defined as having all the elliptic curves defined over k as its vertices, and having a directed edge (E_1, E_2) for each k-rational ℓ-isogeny from E_1 to E_2.*[4]

Remark 3.3. *A priori $G_{k,\ell}$ is a directed graph, but given two elliptic curves E_1 and E_2 whose j-invariants are not in $\{0, 1728\}$, there are exactly as many edges (E_2, E_1) as (E_1, E_2), obtained by taking dual isogenies. Annoyingly, the nodes with j-invariants 0 and 1728 are more complicated, since these are exactly the curves with extra automorphisms: an elliptic curve E in $G_{k,\ell}$ has fewer incoming than outgoing edges if and only if either $j(E) = 0$ and $\sqrt{-3} \in k$, or if $j(E) = 1728$ and $\sqrt{-1} \in k$. Throughout this chapter, we will assume for simplicity that $\sqrt{-3}, \sqrt{-1} \notin k$, so that neither of these automorphisms are defined over k and we may view $G_{k,\ell}$ as an undirected graph. In the case of a finite prime field $k = \mathbb{F}_p$, it suffices to restrict to $p \equiv 11 \pmod{12}$, which will be satisfied in the class of instantiations we suggest. See also Section 2.5.1.*

If $k = \mathbb{F}_q$ is a finite field, then $G_{k,\ell}$ is a finite graph that is the disjoint union of ordinary connected components and supersingular connected components. The ordinary components were studied in Kohel's PhD thesis [Koh96]. Due to their regular structure, these components later became known as *isogeny volcanoes*.

In general (e.g. over non-prime fields), the supersingular components may bear no similarity at all to the volcanoes of the ordinary case. Traditionally, following Pizer [Piz90], one instead studies the unique supersingular component of $G_{k,\ell}$ where $k = \overline{\mathbb{F}}_q$, which turns out to be a finite $(\ell+1)$-regular Ramanujan graph and forms the basis for the SIDH protocol.

However, Delfs and Galbraith [DG16] showed that if $k = \mathbb{F}_p$ is a finite prime field, then all connected components are volcanoes, even in the supersingular case (where the depth is at most 1 at $\ell = 2$ and 0 otherwise). We present a special case of a unified statement, restricting our attention to the cases in which $G_{\mathbb{F}_p,\ell}$ is a cycle. Recall that $\mathrm{End}_p(E)$ is an order $\mathcal{O}$ in the imaginary quadratic field

$$\mathrm{End}_p(E) \otimes_{\mathbb{Z}} \mathbb{Q} \cong \mathbb{Q}(\sqrt{t^2 - 4p}) = K,$$

where $|t| \leq 2\sqrt{p}$ denotes the (absolute value of the) trace of the Frobenius endomorphism, and that two curves are isogenous over $\mathbb{F}_p$ if and only if their traces of Frobenius are equal [Tat66, Theorem 1].

Theorem 3.4 (Kohel, Delfs–Galbraith)**.** *Let $p \geq 5$ be a prime number and let V be a connected component of $G_{\mathbb{F}_p,\ell}$. Assume that $p \equiv 11 \pmod{12}$ or that V contains no curve with j-invariant 0 or 1728. Let t be the trace of Frobenius common to all vertices in V, and let K be as above. Assume that $\ell \nmid t^2 - 4p$.*

Then all elliptic curves in V have the same $\mathbb{F}_p$-rational endomorphism ring $\mathcal{O} \subseteq K$, and $\mathcal{O}$ is locally maximal at ℓ. Moreover if $t^2 - 4p$ is a (non-zero) square modulo ℓ, then V is a cycle whose length equals the order of $[\mathfrak{l}]$ in $\mathrm{cl}(\mathcal{O})$, where $\mathfrak{l}$ is a prime ideal dividing $\ell\mathcal{O}$. If not, then V consists of a single vertex and no edges.

Proof. In the case of an ordinary component this is just a special case of [Sut12b, Theorem 7]. In the case of a supersingular component this follows from the proof of [DG16, Theorem 2.7]. (In both cases, we could alternatively (re)prove this theorem by proving that an ℓ-isogeny can only change the conductor of the endomorphism ring of an elliptic curve locally at ℓ and applying Theorem 3.7.) □

[4] Due to our convention of identifying k-isomorphic curves, we also identify isogenies if they are k-isomorphic, i.e., equal up to post-composition with a k-isomorphism.

In the ordinary case a curve and its quadratic twist can never appear in the same component because they have a different trace of Frobenius. This is the main difference with the supersingular case, where this possibility is not excluded. To avoid confusion, we clarify that by the quadratic twist of a given elliptic curve $E\colon y^2 = f(x)$ over $\mathbb{F}_p$ we mean the curve $E^t\colon dy^2 = f(x)$, where $d \in \mathbb{F}_p^*$ is any non-square. If $p \equiv 3 \pmod 4$ and $j(E) = 1728$ then this may deviate from what some readers are used to, because in this case E^t and E are $\mathbb{F}_p$-isomorphic. Note that such a curve is necessarily supersingular.

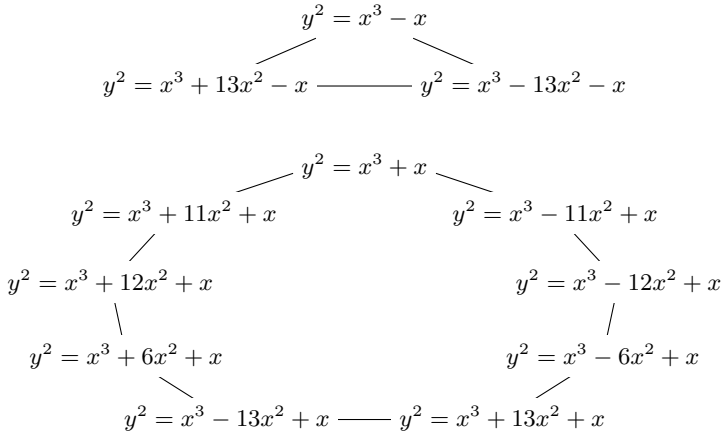


Figure 3.2: The two supersingular components of $G_{\mathbb{F}_{83},3}$. The curves in the top component have $\mathbb{F}_p$-rational endomorphism ring $\mathbb{Z}[(1+\sqrt{-83})/2]$, while those in the lower component correspond to $\mathbb{Z}[\sqrt{-83}]$. Running clockwise through these components corresponds to the repeated action of $[(3, \pi - 1)]$.

Remark 3.5. *In fact, if $p \equiv 3 \bmod 4$ then there are two non-isomorphic curves over $\mathbb{F}_p$ with j-invariant 1728, namely $y^2 = x^3 - x$ and $y^2 = x^3 + x$, whose endomorphism rings are the full ring of integers $\mathbb{Z}[(1+\sqrt{-p})/2]$ and the order $\mathbb{Z}[\sqrt{-p}]$ of conductor 2 respectively. The connected component of each curve is "symmetric": if E is n steps along $G_{\mathbb{F}_p,\ell}$ in one direction from a curve of j-invariant 1728 then the curve that is n steps in the other direction is the quadratic twist of E. In the case of $G_{\mathbb{F}_{83},3}$ we can see this in Figure 3.2, which is taken from [DG16, Figure 8].*

It is also interesting to observe that the symmetry around $j = 1728$ confirms the known fact that the class numbers of $\mathbb{Z}[(1+\sqrt{-p})/2]$ and $\mathbb{Z}[\sqrt{-p}]$ are odd, at least in the case that $p \equiv 3 \pmod 4$; see [Mor61].

3.3 — The class-group action

It is well-known that the ideal-class group of an imaginary quadratic order $\mathcal{O}$ acts freely via isogenies on the set of elliptic curves with $\mathbb{F}_p$-rational endomorphism ring $\mathcal{O}$. Using this group action on a set of ordinary elliptic curves for cryptographic purposes was first put forward by Couveignes [Cou06] and independently rediscovered later by Rostovtsev and Stolbunov [Sto04; RS06]. Our suggestion is to use the equivalent of their construction in the supersingular setting, thus the following discussion covers both cases at once. For concreteness, we focus on prime fields with $p \geq 5$ and point out that the ordinary (but not the supersingular) case generalizes

to all finite fields. We recall the following standard lemma, which is a special case of Proposition 2.30:

Lemma 3.6. *Let $E/\mathbb{F}_p$ be an elliptic curve and G a finite $\mathbb{F}_p$-rational (i.e., stable under the action of the $\mathbb{F}_p$-Frobenius) subgroup of E. Then there exists an elliptic curve $E'/\mathbb{F}_p$ and a separable isogeny $\varphi\colon E \to E'$ defined over $\mathbb{F}_p$ with kernel G. The codomain E' and isogeny φ are unique up to $\mathbb{F}_p$-isomorphism.*[5]

Proof. [Sil09, Proposition III.4.12, Remark III.4.13.2, and Exercise III.3.13e]. □

The ideal-class group. We recall the definitions and basic properties of class groups of quadratic orders that will be needed in the following. This section is based on [Cox13, §7]. Let K be a quadratic number field and $\mathcal{O} \subseteq K$ an order (that is, a subring which is a free $\mathbb{Z}$-module of rank 2). The *norm* of an $\mathcal{O}$-ideal $\mathfrak{a} \subseteq \mathcal{O}$ is defined as $\mathrm{N}(\mathfrak{a}) = |\mathcal{O}/\mathfrak{a}|$; it is equal to $\gcd(\{\mathrm{N}(\alpha) \mid \alpha \in \mathfrak{a}\})$. Norms are multiplicative: $\mathrm{N}(\mathfrak{ab}) = \mathrm{N}(\mathfrak{a})\mathrm{N}(\mathfrak{b})$.

A *fractional ideal* of $\mathcal{O}$ is an $\mathcal{O}$-submodule of K of the form $\alpha\mathfrak{a}$, where $\alpha \in K^*$ and $\mathfrak{a}$ is an $\mathcal{O}$-ideal.[6] Fractional ideals can be multiplied and conjugated in the evident way, and the norm extends multiplicatively to fractional ideals. A fractional $\mathcal{O}$-ideal $\mathfrak{a}$ is *invertible* if there exists a fractional $\mathcal{O}$-ideal $\mathfrak{b}$ such that $\mathfrak{ab} = \mathcal{O}$. If such a $\mathfrak{b}$ exists, we define $\mathfrak{a}^{-1} = \mathfrak{b}$. Clearly all *principal* fractional ideals $\alpha\mathcal{O}$, where $\alpha \in K^*$, are invertible.

By construction, the set of invertible fractional ideals $I(\mathcal{O})$ forms an abelian group under ideal multiplication. This group contains the principal fractional ideals $P(\mathcal{O})$ as a (clearly normal) subgroup, hence we may define the *ideal-class group* of $\mathcal{O}$ as the quotient

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

Every ideal class $[\mathfrak{a}] \in \mathrm{cl}(\mathcal{O})$ has an integral representative, and for any non-zero $M \in \mathbb{Z}$ there even exists an integral representative of norm coprime to M.

There is a unique *maximal order* of K with respect to inclusion called the ring of integers and denoted $\mathcal{O}_K$. The *conductor* of $\mathcal{O}$ (in $\mathcal{O}_K$) is the index $f = [\mathcal{O}_K : \mathcal{O}]$. Away from the conductor, ideals are well-behaved; every $\mathcal{O}$-ideal of norm coprime to the conductor is invertible and factors uniquely into prime ideals.

The class-group action. Fix a prime $p \geq 5$ and an (ordinary or supersingular) elliptic curve E defined over $\mathbb{F}_p$. The Frobenius endomorphism π of E satisfies a characteristic equation

$$\pi^2 - t\pi + p = 0$$

in $\mathrm{End}_p(E)$, where $t \in \mathbb{Z}$ is the trace of Frobenius. The curve E is supersingular if and only if $t = 0$. The $\mathbb{F}_p$-rational endomorphism ring $\mathrm{End}_p(E)$ is an order $\mathcal{O}$ in the imaginary quadratic field $K = \mathcal{O} \otimes_{\mathbb{Z}} \mathbb{Q} \cong \mathbb{Q}(\sqrt{\Delta})$, where $\Delta = t^2 - 4p$. We note that $\mathcal{O}$ always contains the Frobenius endomorphism π, and hence the order $\mathbb{Z}[\pi]$.

Any invertible ideal $\mathfrak{a}$ of $\mathcal{O}$ splits into a product of $\mathcal{O}$-ideals as $(\pi\mathcal{O})^r\mathfrak{a}_s$, where $\mathfrak{a}_s \nsubseteq \pi\mathcal{O}$. This defines an elliptic curve $E/\mathfrak{a}$ and an isogeny

$$\varphi_{\mathfrak{a}}\colon E \to E/\mathfrak{a}$$

[5] This statement remains true in vast generality, but we only need this special case.

[6] Note that the use of the word "ideal" is inconsistent in the literature. We make the convention that "ideal" without qualification refers to an *integral* $\mathcal{O}$-ideal (i.e., an ideal in the sense of ring theory), while fractional ideals are clearly named as such.

of degree $\mathrm{N}(\mathfrak{a})$ as follows [Wat69]: the separable part of $\varphi_{\mathfrak{a}}$ has kernel $\bigcap_{\alpha\in\mathfrak{a}_s}\ker(\alpha)$, and the purely inseparable part consists of r iterations of Frobenius. The isogeny $\varphi_{\mathfrak{a}}$ and codomain $E/\mathfrak{a}$ are both defined over $\mathbb{F}_p$ and are unique up to $\mathbb{F}_p$-isomorphism (by Lemma 3.6), justifying the notation $E/\mathfrak{a}$. Multiplication of ideals corresponds to the composition of isogenies. Since principal ideals correspond to endomorphisms, two ideals lead to the same codomain if and only if they are equal up to multiplication by a principal fractional ideal. Moreover, every $\mathbb{F}_p$-isogeny ψ between curves in $\mathcal{E}\ell\ell_p(\mathcal{O},\pi)$ comes from an invertible $\mathcal{O}$-ideal in this way, and the ideal $\mathfrak{a}_s$ can be recovered from ψ as $\mathfrak{a}_s = \{\alpha\in\mathcal{O} \mid \ker(\alpha)\supseteq\ker(\psi)\}$. In other words:

Theorem 3.7. *Let $\mathcal{O}$ be an order in an imaginary quadratic field and $\pi\in\mathcal{O}$ such that $\mathcal{E}\ell\ell_p(\mathcal{O},\pi)$ is non-empty. Then the ideal-class group $\mathrm{cl}(\mathcal{O})$ acts freely and transitively on the set $\mathcal{E}\ell\ell_p(\mathcal{O},\pi)$ via the map*

$$\begin{array}{ccc}\mathrm{cl}(\mathcal{O})\times\mathcal{E}\ell\ell_p(\mathcal{O},\pi) & \longrightarrow & \mathcal{E}\ell\ell_p(\mathcal{O},\pi)\\ ([\mathfrak{a}],E) & \longmapsto & E/\mathfrak{a},\end{array}$$

in which $\mathfrak{a}$ is chosen as an integral representative.

Proof. See [Wat69, Theorem 4.5]. Erratum: [Sch87, Theorem 4.5]. □

To emphasize the fact that we are dealing with a group action, we will from now on write $[\mathfrak{a}]*E$ or simply $[\mathfrak{a}]E$ for the curve $E/\mathfrak{a}$ defined above.

The structure of the class group. The class group $\mathrm{cl}(\mathcal{O})$ is a finite abelian group whose cardinality is asymptotically [Sie35]

$$\#\mathrm{cl}(\mathcal{O})\approx\sqrt{|\Delta|}.$$

The exact structure of $\mathrm{cl}(\mathcal{O})$ can be computed in subexponential time $L_{|\Delta|}[1/2;\sqrt{2}+o(1)]$ using an algorithm of Hafner and McCurley [HM89]. Unfortunately, this requires too much computation for the sizes of Δ we are working with, but there are convincing heuristics concerning the properties of the class group we need. See Section 3.7.1 for these arguments. If the absolute value $|t|$ of the trace of Frobenius is "not too big", the discriminant Δ is about the size of p, hence by the above approximation we may assume $\#\mathrm{cl}(\mathcal{O})\approx\sqrt{p}$. This holds in particular when E is supersingular, where $t=0$, hence $|\Delta|=4p$.

We are interested in primes ℓ that split in $\mathcal{O}$, i.e., such that there exist (necessarily conjugate) distinct prime ideals $\mathfrak{l},\bar{\mathfrak{l}}$ of $\mathcal{O}$ with $\ell\mathcal{O}=\mathfrak{l}\bar{\mathfrak{l}}$. Such ℓ are known as *Elkies primes* in the point-counting literature. The ideal $\mathfrak{l}$ is generated as $\mathfrak{l}=(\ell,\pi-\lambda)$, where $\lambda\in\mathbb{Z}/\ell$ is an eigenvalue of the Frobenius endomorphism π on the ℓ-torsion, and its conjugate is $\bar{\mathfrak{l}}=(\ell,\pi-p/\lambda)$, where by abuse of notation p/λ denotes any integral representative of that quotient modulo ℓ. Note that ℓ splits in $\mathcal{O}$ if and only if Δ is a non-zero square modulo ℓ.

Computing the group action. Any element of the class group can be represented as a product of small prime ideals [BV07, Propositions 9.5.2 and 9.5.3], hence we describe how to compute $[\mathfrak{l}]E$ for a prime ideal $\mathfrak{l}=(\ell,\pi-\lambda)$. There are (at least) the following ways to proceed, which vary in efficiency depending on the circumstances [DKS18; Kie17]:

- Find $\mathbb{F}_p$-rational roots of the modular polynomial $\Phi_\ell(j(E),Y)$ to determine the two j-invariants of possible codomains (i.e., up to four non-isomorphic curves, though in the ordinary case wrong twists can easily be ruled out); compute the kernel polynomials [Koh96] $\chi\in\mathbb{F}_p[x]$ for the corresponding isogenies (if they exist); if $(x^p,y^p)=[\lambda](x,y)$ modulo χ and the curve equation, then the codomain was correct, else another choice is correct.

- Factor the ℓ^{th} division polynomial $\psi_\ell(E)$ over $\mathbb{F}_p$; collect irreducible factors with the right Frobenius eigenvalues (as above); use Kohel's formulas [Koh96, Section 2.4] to compute the codomain.
- Find a basis of the ℓ-torsion — possibly over an extension field — and compute the eigenspaces of Frobenius; apply Vélu's formulas [Vél71] to a basis point of the correct eigenspace to compute the codomain.

As observed in [DKS18; Kie17], the last method is the fastest if the necessary extension fields are small. The optimal case is $\lambda = 1$; in that case, the curve has a rational point defined over the base field $\mathbb{F}_p$. If in addition $p/\lambda = -1$, the other eigenspace of Frobenius modulo ℓ is defined over $\mathbb{F}_{p^2}$, so both codomains can easily be computed using Vélu's formulas over an at most quadratic extension (but in fact, a good choice of curve model allows for pure prime field computations, see Section 3.8; alternatively one could switch to the quadratic twist). Note that if $p \equiv -1 \pmod{\ell}$, then $\lambda = 1$ automatically implies $p/\lambda = -1$.

Much of De Feo–Kieffer–Smith's work [DKS18; Kie17] is devoted to finding an ordinary elliptic curve E with many small Elkies primes ℓ such that both E and its quadratic twist E^t have an $\mathbb{F}_p$-rational ℓ-torsion point. Despite considerable effort leading to various improvements, the results are discouraging. With the best parameters found within 17 000 hours of CPU time, evaluating one class-group action still requires several minutes of computation to complete. This suggests that without new ideas, the original Couveignes–Rostovtsev–Stolbunov scheme will not become anything close to practical in the foreseeable future.

3.4 — Construction and design choices

In this section, we discuss the construction of our proposed group action and justify our design decisions. For algorithmic details, see Section 3.8. Notice that the main obstacle to performance in the Couveignes–Rostovtsev–Stolbunov scheme — constructing a curve with highly composite order — becomes trivial when using supersingular curves instead of ordinary curves, since for $p \geq 5$ any supersingular elliptic curve over $\mathbb{F}_p$ has exactly $p + 1$ rational points.

The cryptographic group action described below is a straightforward implementation of this construction. Note that we require $p \equiv 3 \pmod{4}$ so that we can easily write down a supersingular elliptic curve over $\mathbb{F}_p$ and so that an implementation may use curves in Montgomery form. It turns out that this choice is also beneficial for other reasons. In principle, this constraint is not necessary for the theory to work, although the structure of the isogeny graph changes slightly (see [DG16] and Remark 3.3 for details).

Parameters. Fix a large prime p of the form $4 \cdot \ell_1 \cdots \ell_n - 1$, where the ℓ_i are small distinct odd primes. Fix the elliptic curve $E_0 \colon y^2 = x^3 + x$ over $\mathbb{F}_p$; it is supersingular since $p \equiv 3 \pmod{4}$. The Frobenius endomorphism π satisfies $\pi^2 = -p$, so its $\mathbb{F}_p$-rational endomorphism ring is an order in the imaginary quadratic field $\mathbb{Q}(\sqrt{-p})$. More precisely, Proposition 3.8 (below) shows $\mathrm{End}_p(E_0) = \mathbb{Z}[\pi]$, which has conductor 2.

Rational Elkies primes. By Theorem 3.4, the choices made above imply that the ℓ_i-isogeny graph is a disjoint union of cycles. Moreover, since $\pi^2 - 1 \equiv 0 \pmod{\ell_i}$ the ideals $\ell_i\mathcal{O}$ split as $\ell_i\mathcal{O} = \mathfrak{l}_i\overline{\mathfrak{l}_i}$, where $\mathfrak{l}_i = (\ell_i, \pi - 1)$ and $\overline{\mathfrak{l}_i} = (\ell_i, \pi + 1)$. In other words, *all* the ℓ_i are Elkies primes. In particular, we can use any one of the three algorithms described at the end of Section 3.3 to walk along the cycles.

Furthermore, the kernel of $\varphi_{\mathfrak{l}_i}$ is the intersection of the kernels of the scalar multiplication $[\ell_i]$ and the endomorphism $\pi - 1$. That is, it is the subgroup generated by a point P of order ℓ_i

which lies in the kernel of $\pi - 1$ or, in other words, is defined over $\mathbb{F}_p$. Similarly, the kernel of $\varphi_{\overline{\mathfrak{l}_i}}$ is generated by a point Q of order ℓ_i that is defined over $\mathbb{F}_{p^2}$ such that $\pi(Q) = -Q$. This greatly simplifies and accelerates the implementation, since it allows performing all computations over the base field (see Section 3.8 for details).

Sampling from the class group. Ideally,[7] we would like to know the exact structure of the ideal-class group $\mathrm{cl}(\mathcal{O})$ to be able to sample elements uniformly at random. However, such a computation is currently not feasible for the size of discriminant we need, hence we resort to heuristic arguments. Assuming that the $\mathfrak{l}_i$ do not have very small order and are "evenly distributed" in the class group, we can expect ideals of the form $\mathfrak{l}_1^{e_1}\mathfrak{l}_2^{e_2}\cdots\mathfrak{l}_n^{e_n}$ for small e_i to lie in the same class only very occasionally. For efficiency reasons, it is desirable to sample the exponents e_i from a short range centered around zero, say $\{-m, \ldots, m\}$ for some integer m. We will argue in Section 3.7.1 that choosing m such that $2m + 1 \geq \sqrt[n]{\#\mathrm{cl}(\mathcal{O})}$ is sufficient. Since the prime ideals $\mathfrak{l}_i$ are fixed global parameters, the ideal $\prod_i \mathfrak{l}_i^{e_i}$ may simply be represented as a vector $(e_1, \ldots, e_n)$.

Evaluating the class-group action. Computing the action of an ideal class represented by $\prod_i \mathfrak{l}_i^{e_i}$ on an elliptic curve E proceeds as outlined in Section 3.3. Since $\pi^2 = -p \equiv 1 \pmod{\ell_i}$, we are now in the favourable situation that the eigenvalues of Frobenius on *all* ℓ_i-torsion subgroups are $+1$ and -1. Hence we can efficiently compute the action of $\mathfrak{l}_i$ (resp. $\overline{\mathfrak{l}_i}$) by finding an $\mathbb{F}_p$-rational point (resp. $\mathbb{F}_{p^2}$-rational with Frobenius eigenvalue -1) of order ℓ_i and applying Vélu-type formulas. This step could simply be repeated for each ideal $\mathfrak{l}_i^{\pm 1}$ whose action is to be evaluated, but see Section 3.8 for a more efficient method.

3.5 — Representing and validating $\mathbb{F}_p$-isomorphism classes

A major unsolved problem of SIDH is its lack of public-key validation, i.e., the inability to verify that a public key was honestly generated. This shortcoming leads to polynomial-time active attacks [GPST16] on static variants for which countermeasures are expensive. For example, the actively secure variant SIKE [Jao+17] applies a transformation proposed by Hofheinz, Hövelmanns, and Kiltz [HHK17] which is similar to the Fujisaki–Okamoto transform [FO99], essentially doubling the running time on the recipient's side compared to an ephemeral key exchange.

The following proposition tackles this problem for the family of CSIDH instantiations we are proposing. Moreover, it shows that the Montgomery coefficient forms a unique representative for the $\mathbb{F}_p$-isomorphism class resulting from the group action, hence may serve as a shared secret without taking j-invariants.

Proposition 3.8. *Let $p \geq 5$ be a prime such that $p \equiv 3 \pmod 8$, and let $E/\mathbb{F}_p$ be a supersingular elliptic curve. Then $\mathrm{End}_p(E) = \mathbb{Z}[\pi]$ if and only if there exists $A \in \mathbb{F}_p$ such that E is $\mathbb{F}_p$-isomorphic to the curve $E_A \colon y^2 = x^3 + Ax^2 + x$. Moreover, if such an A exists then it is unique.*

Proof. First suppose that E is $\mathbb{F}_p$-isomorphic to E_A for some $A \in \mathbb{F}_p$. If E_A has full $\mathbb{F}_p$-rational 2-torsion, then Table 1 of [CS18] shows that either E_A or its quadratic twist must have order divisible by 8. However, both have cardinality $p + 1 \equiv 4 \pmod 8$. Hence E_A can only have one $\mathbb{F}_p$-rational point of order 2. With Theorem 2.7 of [DG16], we can conclude $\mathrm{End}_p(E) = \mathbb{Z}[\pi]$.

Now assume that $\mathrm{End}_p(E) = \mathbb{Z}[\pi]$. By Theorem 3.7, the class group $\mathrm{cl}(\mathbb{Z}[\pi])$ acts transitively on $\mathcal{Ell}_p(\mathbb{Z}[\pi], \pi)$, so in particular there exists $[\mathfrak{a}] \in \mathrm{cl}(\mathbb{Z}[\pi])$ such that $[\mathfrak{a}]E_0 = E$, where

[7] No pun intended.

$E_0 \colon y^2 = x^3 + x$. Choosing a representative $\mathfrak{a}$ that has norm coprime to $2p$ yields a separable $\mathbb{F}_p$-isogeny $\varphi_\mathfrak{a} \colon E_0 \to E$ of odd degree. Thus, by [Ren18, Proposition 1] there exists an $A \in \mathbb{F}_p$ and a separable isogeny $\psi \colon E_0 \to E_A \colon y^2 = x^3 + Ax^2 + x$ defined over $\mathbb{F}_p$ such that $\ker(\psi) = \ker(\varphi_\mathfrak{a})$. As isogenies defined over $\mathbb{F}_p$ with given kernel are unique up to post-composition with $\mathbb{F}_p$-isomorphisms (Lemma 3.6), we conclude that E is $\mathbb{F}_p$-isomorphic to E_A.

Finally, let $B \in \mathbb{F}_p$ such that $E_A \cong E_B \colon Y^2 = X^3 + BX^2 + X$. Then by [Sil09, Proposition III.3.1(b)] there exist $u \in \mathbb{F}_p^*$ and $r, s, t \in \mathbb{F}_p$ such that

$$x = u^2 X + r\,, \quad y = u^3 Y + su^2 X + t\,.$$

Substituting this into the curve equation of E_A and subtracting u^6 times the equation of E_B equals zero in the function field and thus leads to a linear relation over $\mathbb{F}_p$ between the functions 1, X, X^2, Y, and XY. Writing ∞ for the point at infinity of E_B, it follows from Riemann–Roch [Sil09, Thm. 5.4] that $\mathcal{L}(5(\infty))$ is a 5-dimensional $\mathbb{F}_p$-vector space with basis $\{1, X, Y, X^2, XY\}$. Hence the obtained linear relation must be trivial, and a straightforward computation yields

$$s = t = 0\,, \qquad 3r^2 + 2Ar + 1 = u^4\,,$$
$$3r + A = Bu^2\,, \qquad r^3 + Ar^2 + r = 0\,.$$

But since E_A only has a single $\mathbb{F}_p$-rational point of order 2, the only $r \in \mathbb{F}_p$ such that $r^3 + Ar^2 + r = 0$ is simply $r = 0$. In that case $u^4 = 1$, and hence $u = \pm 1$ since $p \equiv 3 \pmod 8$. In particular, $u^2 = 1$ and thus $A = B$. □

Therefore, using a Montgomery coefficient $A \in \mathbb{F}_p$ to represent public keys, Proposition 3.8 guarantees that A represents a curve in the correct isogeny class $\mathcal{E}\ell\ell_p(\mathcal{O}, \pi)$, where $\pi = \sqrt{-p}$ and $\mathcal{O} = \mathbb{Z}[\pi]$, under the assumption that it is smooth (i.e. $A \notin \{\pm 2\}$) and supersingular.

Verifying supersingularity. As $p \geq 5$, an elliptic curve E defined over $\mathbb{F}_p$ is supersingular if and only if $\#E(\mathbb{F}_p) = p + 1$ [Sil09, Exercise 5.10]. In general, proving that an elliptic curve has a given order N is easy if the factorization of N is known; exhibiting a subgroup (or in particular, a single point) whose order d is a divisor of N greater than $4\sqrt{p}$ implies the order must be correct. Indeed, the condition $d > 4\sqrt{p}$ implies that only one multiple of d lies in the Hasse interval $[p + 1 - 2\sqrt{p}; p + 1 + 2\sqrt{p}]$ [Has36]. This multiple is the group order by Lagrange's theorem.

Now note that a random point generally has very large order d. For our curves we have $E(\mathbb{F}_p) \cong \mathbb{Z}/4 \times \prod_{i=1}^n \mathbb{Z}/\ell_i$, so that $\ell_i \mid d$ with probability $(\ell_i - 1)/\ell_i$. Ignoring the even part, this shows that the expected order is lower bounded by

$$\prod_{i=1}^{n} \left(\ell_i - 1 + \frac{1}{\ell_i}\right).$$

This product is about the same size as p, and it is easily seen that a random point will with overwhelming probability have order (much) greater than $4\sqrt{p}$. This observation leads to a straightforward verification method, see Algorithm 3.1.[8]

If the condition $d > 4\sqrt{p}$ does not hold at the end of Algorithm 3.1, the point P had too small order to prove $\#E(\mathbb{F}_p) = p+1$. In this case one may retry with a new random point P (although this outcome has negligible probability and could just be ignored). There is no possibility of wrongly classifying an ordinary curve as supersingular.

[8] The same idea gives rise to a simpler Monte Carlo algorithm which does not require the factorization of $p + 1$ but has a chance of false positives [Sut12a, Section 2.3].

Algorithm 3.1: Verifying supersingularity.

Input: An elliptic curve $E/\mathbb{F}_p$, where $p = 4 \cdot \ell_1 \cdots \ell_n - 1$.

Output: *supersingular* or *ordinary*.

1 Randomly pick a point $P \in E(\mathbb{F}_p)$ and set $d \leftarrow 1$.
2 **for** each ℓ_i **do**
3 Set $Q_i \leftarrow [(p+1)/\ell_i]P$.
4 **If** $[\ell_i]Q_i \neq \infty$ **then return** *ordinary*. // since $\#E(\mathbb{F}_p) \nmid p+1$
5 **If** $Q_i \neq \infty$ **then** set $d \leftarrow \ell_i \cdot d$. // since $\ell_i \mid \operatorname{ord} P$
6 **If** $d > 4\sqrt{p}$ **then return** *supersingular*.

Note moreover that if x-only Montgomery arithmetic is used (as we suggest) and the point P is obtained by choosing a random x-coordinate in $\mathbb{F}_p$, there is no need to differentiate between points defined over $\mathbb{F}_p$ and $\mathbb{F}_{p^2}$; any x-coordinate in $\mathbb{F}_p$ works. Indeed, any point that has an x-coordinate in $\mathbb{F}_p$ but is only defined over $\mathbb{F}_{p^2}$ corresponds to an $\mathbb{F}_p$-rational point on the quadratic twist, which is supersingular if and only if the original curve is supersingular.

There are more optimized variants of this algorithm; the bulk of the work are the scalar multiplications required to compute the points $Q_i = [(p+1)/\ell_i]P$. Since they are all multiples of P with shared factors, one may more efficiently compute all Q_i at the same time using a divide-and-conquer strategy (at the expense of higher memory usage). See Section 10, and in particular Algorithm 3.3, for details.

3.6 — Non-interactive key exchange

Starting from the class-group action on supersingular elliptic curves and the parameter choices outlined in Sections 3.3 and 3.4, one obtains the following non-interactive key-exchange protocol.

Setup. Global parameters of the scheme are a large prime $p = 4 \cdot \ell_1 \cdots \ell_n - 1$, where the ℓ_i are small distinct odd primes, and the supersingular elliptic curve $E_0 \colon y^2 = x^3 + x$ over $\mathbb{F}_p$ with endomorphism ring $\mathcal{O} = \mathbb{Z}[\pi]$.

Key generation. The private key is an n-tuple $(e_1, \ldots, e_n)$ of integers, each sampled randomly from a range $\{-m, \ldots, m\}$. These integers represent the ideal class $[\mathfrak{a}] = [\mathfrak{l}_1^{e_1} \cdots \mathfrak{l}_n^{e_n}] \in \mathrm{cl}(\mathcal{O})$, where $\mathfrak{l}_i = (\ell_i, \pi - 1)$. The public key is the Montgomery coefficient $A \in \mathbb{F}_p$ of the elliptic curve $[\mathfrak{a}]E_0 \colon y^2 = x^3 + Ax^2 + x$ obtained by applying the action of $[\mathfrak{a}]$ to the curve E_0.

Key exchange. Suppose Alice and Bob have key pairs $([\mathfrak{a}], A)$ and $([\mathfrak{b}], B)$. Upon receiving Bob's public key $B \in \mathbb{F}_p \backslash \{\pm 2\}$, Alice verifies that the elliptic curve $E_B \colon y^2 = x^3 + Bx^2 + x$ is indeed in $\mathscr{E}\ell\ell_p(\mathcal{O}, \pi)$ using Algorithm 3.1. She then applies the action of her secret key $[\mathfrak{a}]$ to E_B to compute the curve $[\mathfrak{a}]E_B = [\mathfrak{a}][\mathfrak{b}]E_0$. Bob proceeds analogously with his own secret $[\mathfrak{b}]$ and Alice's public key A to compute the curve $[\mathfrak{b}]E_A = [\mathfrak{b}][\mathfrak{a}]E_0$. The shared secret is the Montgomery coefficient S of the common secret curve $[\mathfrak{a}][\mathfrak{b}]E_0 = [\mathfrak{b}][\mathfrak{a}]E_0$ written in the form $y^2 = x^3 + Sx^2 + x$, which is the same for Alice and Bob due to the commutativity of $\mathrm{cl}(\mathcal{O})$ and Proposition 3.8.

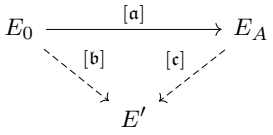


Figure 3.3: A 1-bit identification protocol.

Remark 3.9. *Besides key exchange, we expect that our cryptographic group action will have several other applications, given the resemblance with traditional Diffie–Hellman and the ease of verifying the correctness of public keys. We refer to previous papers on group actions for a number of suggestions in this direction, in particular Brassard–Yung [BY90], Couveignes [Cou06, §4], and Stolbunov [Sto10]. We highlight the following* 1*-bit identification scheme, which in our case uses a key pair* $([\mathfrak{a}], A)$ *as above. One randomly samples an element* $[\mathfrak{b}] \in \mathrm{cl}(\mathcal{O})$ *and commits to a curve* $E' = [\mathfrak{b}]E_0$*. Depending on a challenge bit b, one then releases either* $[\mathfrak{b}]$ *or* $[\mathfrak{c}] := [\mathfrak{b}][\mathfrak{a}]^{-1}$*, as depicted in Figure 3.3. As already pointed out in Stolbunov's PhD thesis [Sto12, §2.B], this can be turned into a signature scheme by repeated application of the* 1*-bit protocol and by applying the Fiat–Shamir [FS86] or Unruh [Unr12] transformation. However, we point out that it is not immediately clear how to represent* $[\mathfrak{c}]$ *in a way that is efficiently computable and leaks no information about the secret key* $[\mathfrak{a}]$*. We leave a resolution of this issue for future research, but mention that a related problem was recently tackled by Galbraith, Petit and Silva [GPS17] who studied a similar triangular identification protocol in the context of SIDH.*[9]

3.7 — Security

The central problem of our new primitive is the following analogue to the classical discrete-logarithm problem.

Problem 3.10 (Key recovery)**.** *Given two supersingular elliptic curves* E, E' *defined over* $\mathbb{F}_p$ *with the same* $\mathbb{F}_p$*-rational endomorphism ring* $\mathcal{O}$*, find an ideal* $\mathfrak{a}$ *of* $\mathcal{O}$ *such that* $[\mathfrak{a}]E = E'$*. This ideal must be represented in such a way that the action of* $[\mathfrak{a}]$ *on a curve can be evaluated efficiently, for instance* $\mathfrak{a}$ *could be given as a product of ideals of small norm.*

Just like in the classical group-based scenario, security notions of Diffie–Hellman schemes built from our primitive rely on slightly different hardness assumptions (cf. Section 3.1.1) that are straightforward translations of the computational and decisional Diffie–Hellman problems; see Section 2.1.3. However, continuing the analogy with the classical case, and since we are not aware of any ideas to attack the key exchange without recovering one of the keys, we will assume in the following analysis that the best approach to breaking the key-exchange protocol is to solve Problem 3.10.

We point out that the "inverse Diffie-Hellman problem" is easy in the context of CSIDH: given $[\mathfrak{a}]E_0$ we can compute $[\mathfrak{a}]^{-1}E_0$ by mere quadratic twisting; see Remark 3.5. This contrasts with the classical group-based setting [Gal12, §21.1]. Note that just like identifying a point (x, y) with its inverse $(x, -y)$ in an ECDLP setting, this may imply a security loss of one bit under some attacks: An attacker may consider the curves $[\mathfrak{a}]E$ and $[\mathfrak{a}]^{-1}E$ identical, which reduces the search space by half.

[9] The "square" SIDH counterparts of this protocol, as considered in [DJP14; GPS17; Yoo+17], are not meaningful in the case of a commutative group action.

No torsion-point images. One of the most worrying properties of SIDH seems to be that Alice and Bob publish the images of known points under their secret isogenies along with the codomain curve, i.e., a public key is of the form $(E', \varphi(P), \varphi(Q))$ where $\varphi\colon E \to E'$ is a secret isogeny and $P, Q \in E$ are publicly known points. Although thus far nobody has succeeded in making use of this extra information to break the original scheme, Petit presented an attack using these points when overstretched, asymmetric parameters are used; see [Pet17] and Chapter 7. The Couveignes–Rostovtsev–Stolbunov scheme, and consequently our new scheme CSIDH, does not transmit such additional points — a public key consists of *only* an elliptic curve. Thus we are confident that a potential future attack against SIDH based on these torsion points would not apply to CSIDH.

Chosen-ciphertext attacks. As explained in Section 3.5, the CSIDH group action features efficient public-key validation. This implies it can be used without applying a CCA transform such as the Fujisaki–Okamoto transform [FO99], thus enabling efficient non-interactive key exchange (see Section 2.1.5) and other applications in a post-quantum world.

3.7.1 – Classical security. We begin by considering classical attacks.

Exhaustive key search. The most obvious approach to attack any cryptosystem is to simply search through all possible keys. In the following, we will argue that our construction provides sufficient protection against key search attacks, including dumb brute force and (less naïvely) a meet-in-the-middle approach.

As explained in Section 3.4, a private key of our scheme is an exponent vector $(e_1, \dots, e_n)$ where each e_i is in the range $\{-m, \dots, m\}$, representing the ideal class $[\mathfrak{l}_1^{e_1}\mathfrak{l}_2^{e_2}\cdots\mathfrak{l}_n^{e_n}] \in \mathrm{cl}(\mathcal{O})$. There may (and typically will) be multiple such vectors that represent the same ideal class and thus form equivalent private keys. However, we argue (heuristically) that the number of *short* representations per ideal class is small. Here and in the following, "short" means that all e_i are in the range $\{-m, \dots, m\}$. The maximum number of such short representations immediately yields the min-entropy[10] of our sampling method, which measures the amount of work a brute-force attacker has to do while conducting an exhaustive search for the key.

We assume in the following discussion that $\mathrm{cl}(\mathcal{O})$ is "almost cyclic" in the sense that it has a very large cyclic component, say of order N not much smaller than $\#\mathrm{cl}(\mathcal{O})$. According to a heuristic of Cohen and Lenstra, this is true with high probability for a "random" imaginary quadratic field [CL84, §9.I], and this conjecture is in line with our own experimental evidence. So suppose

$$\rho\colon \mathrm{cl}(\mathcal{O}) \twoheadrightarrow (\mathbb{Z}/N, +)$$

is a surjective group homomorphism (which may be thought of as a projection to the large cyclic subgroup followed by an isomorphism) and define $\alpha_i = \rho([\mathfrak{l}_i])$. We may assume that $\alpha_1 = 1$; this can be done without loss of generality whenever at least one of the $[\mathfrak{l}_i]$ has order N in the class group. For some fixed $[\mathfrak{a}] \in \mathrm{cl}(\mathcal{O})$, any short representation $[\mathfrak{l}_1^{e_1}\mathfrak{l}_2^{e_2}\cdots\mathfrak{l}_n^{e_n}] = [\mathfrak{a}]$ yields a short solution to the linear congruence

$$e_1 + e_2\alpha_2 + \cdots + e_n\alpha_n \equiv \rho([\mathfrak{a}]) \pmod{N},$$

so counting solutions to this congruence gives an upper bound on the number of short representations of $[\mathfrak{a}]$. These solutions are exactly the points in some shifted version (i.e., a coset) of

[10] The min-entropy of a random variable is the negative logarithm of the probability of the most likely outcome.

the integer lattice spanned by the rows of the matrix

$$L = \begin{pmatrix} N & 0 & 0 & \cdots & 0 \\ -\alpha_2 & 1 & 0 & \cdots & 0 \\ -\alpha_3 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -\alpha_n & 0 & 0 & \cdots & 1 \end{pmatrix},$$

so by applying the Gaussian heuristic [NV10, Chapter 2, Definition 8] one expects

$$\operatorname{vol}[-m;m]^n \,/\, \det(L) = (2m+1)^n/N$$

short solutions. Since we assumed $\mathrm{cl}(\mathcal{O})$ to be almost cyclic, this ratio is not much bigger than $(2m+1)^n/\#\mathrm{cl}(\mathcal{O})$, which is not very large when m is minimal with $(2m+1)^n \geq \#\mathrm{cl}(\mathcal{O})$.

As a result, we expect the complexity of a brute-force search to be around $2^{\log \sqrt{p}-\varepsilon}$ for some positive ε that is small relative to $\log \sqrt{p}$. To verify our claims, we performed computer experiments with many choices of p of up to 40 bits (essentially brute-forcing the number of representations for all elements) and found no counterexamples to the heuristic result that our sampling method loses only a few bits of brute-force security compared to uniform sampling from the class group. For our sizes of p, the min-entropy was no more than 4 bits less than that of a perfectly uniform distribution on the class group (i.e. $\varepsilon \leq 4$). Of course this loss factor may grow in some way with bigger choices of p (a plot of the data points for small sizes suggests an entropy loss proportional to $\log\log p$), but we see no indication for it to explode beyond a few handfuls of bits, as long as we find m and n so that $(2m+1)^n$ is not much larger than $\#\mathrm{cl}(\mathcal{O})$.

Meet-in-the-middle key search. Since a private key trivially decomposes into a product of two smooth ideals drawn from smaller sets (e.g. splitting $[\mathfrak{l}_1^{e_1}\mathfrak{l}_2^{e_2}\cdots\mathfrak{l}_n^{e_n}]$ as $[\mathfrak{l}_1^{e_1}\cdots\mathfrak{l}_\nu^{e_\nu}]\cdot[\mathfrak{l}_{\nu+1}^{e_{\nu+1}}\cdots\mathfrak{l}_n^{e_n}]$ for some $\nu \in \{1,\ldots,n\}$), the usual time-memory trade-offs à la baby-step giant-step [Sha71] with an optimal time complexity of $O\big(\sqrt{\#\mathrm{cl}(\mathcal{O})}\big) \approx O(\sqrt[4]{p})$ apply.[11] Another interpretation of this algorithm is finding a path between two nodes in the underlying isogeny graph by constructing a breadth-first tree starting from each of them, each using a certain subset of the edges, and looking for a collision. Details, including a memoryless variation of this concept, can be found in Delfs and Galbraith's paper [DG16], and for the ordinary case in [Gal99].

Remark 3.11. *The algorithms mentioned thus far scale exponentially in the size of the key space, hence they are asymptotically more expensive than the quantum attacks outlined below which is subexponential in the class-group size. This implies one could possibly balance the costs of the different attacks and use a key space smaller than $\#\mathrm{cl}(\mathcal{O})$ without any loss of security (unless the key space is chosen particularly badly, e.g., as a subgroup), which leads to improved performance. We leave a more thorough analysis of this idea for future work.*

Pohlig–Hellman-style attacks. Notice that the set $\mathscr{E}\ell_p(\mathcal{O},\pi)$ we are acting on does not form a group with efficiently computable operations (that are compatible with the action of $\mathrm{cl}(\mathcal{O})$). Thus there seems to be no way to apply Pohlig–Hellman-style algorithms making use of the decomposition of finite abelian groups. In fact, the Pohlig–Hellman algorithm relies on efficiently computable homomorphisms to proper subgroups, which in the setting at hand would correspond to an efficient algorithm that "projects" a given curve to the orbit of E_0 under a *sub*group

[11] Strictly speaking, the complexity depends on the size of the subset one samples private keys from, rather than the size of the class group, but as was argued before, these are approximately equal for our choice of m and n.

action. Therefore, we believe the structure of the class group to be largely irrelevant (assuming it is big enough); in particular, we do not require it to have a large prime-order subgroup.

3.7.2 – Quantum security. We now discuss the state of quantum algorithms to solve Problem 3.10.

Grover's algorithm and claw finding. Applying Grover search [Gro96] via claw finding as described in [JD11] is fully applicable to CSIDH as well, leading to an attack on Problem 3.10 in $O(\sqrt[6]{p})$ calls to a quantum oracle that computes our group action. The idea is to split the collision search space into a classical $O(\sqrt[6]{p})$ target part and a $O(\sqrt[3]{p})$ search part on which a quantum search is applied. Our choices of p that lead to classical security are also immediately large enough to imply quantum security against this attack (cf. [NIST16, §4.A.5 in Call for Proposals]). That is, the number of queries to our quantum oracle necessary to solve Problem 3.10 is larger than the number of quantum queries to an AES oracle needed to retrieve the key of the corresponding AES instantiation via Grover's algorithm. For example, an AES-128 key can be recovered with approximately 2^{64} (quantum) oracle queries, which requires us to set $p > 2^{384}$. However, p is much larger than that (see Table 3.1) due to the existence of subexponential quantum attacks.

The abelian hidden-shift problem. A crucial result by Kuperberg [Kup05] is an algorithm to solve the hidden-shift problem with time, query and space complexity $2^{O(\sqrt{\log N})}$ in an abelian group H of order N. He also showed that any abelian hidden-shift problem reduces to a dihedral hidden-subgroup problem on a different but closely related oracle. A subsequent alternative algorithm by Regev [Reg04] achieves polynomial quantum space complexity with an asymptotically worse time and query complexity of $2^{O(\sqrt{\log N \log\log N})}$. A follow-up algorithm by Kuperberg [Kup13] uses $2^{O(\sqrt{\log N})}$ time, queries and classical space, but only $O(\log N)$ quantum space. All these algorithms have subexponential time and space complexity.

Attacking the isogeny problem. The relevance of these quantum algorithms to Problem 3.10 has been observed by Childs–Jao–Soukharev [CJS14] in the ordinary case and by Biasse–Jao–Sankar [BJS14] in the supersingular setting. By defining functions $f_0, f_1 \colon \mathrm{cl}(\mathcal{O}) \to \mathcal{E}\ell\ell_p(\mathcal{O}, \pi)$ as $f_0 \colon [\mathfrak{b}] \mapsto [\mathfrak{b}]E$ and $f_1 \colon [\mathfrak{b}] \mapsto [\mathfrak{b}]E' = [\mathfrak{b}][\mathfrak{a}]E$, the problem can be viewed as an abelian hidden-shift problem with respect to f_0 and f_1. We note that each query requires evaluating the functions f_i on arbitrary ideal classes (i.e. without being given a representative that is a product of ideals of small prime norm) which is non-trivial. However, Childs–Jao–Soukharev show this can be done in subexponential time and space [CJS14, §4].

Subexponential vs. practical. An important remark about all these quantum algorithms is that they do not immediately lead to estimates for runtime and memory requirements on concrete instantiations with $H = \mathrm{cl}(\mathcal{O})$. Although the algorithms by Kuperberg and Regev are shown to have subexponential complexity in the limit, this asymptotic behavior is not enough to understand the space and time complexity on actual (small) instances. For example, Kuperberg's first paper [Kup05, Theorem 3.1] mentions $O(2^{3\sqrt{\log N}})$ oracle queries to achieve a non-negligible success probability when N is a power of a small integer. It also presents a second algorithm that runs in $\widetilde{O}(3^{\sqrt{2\log_3 N}}) = O(2^{1.8\sqrt{\log N}})$ [Kup05, Theorem 5.1]. His algorithms handle arbitrary group structures but he does not work out more exact counts for those. Of course, this does not contradict the time complexity of $2^{O(\sqrt{\log N})}$ as stated above, but for a concrete security analysis the hidden constants certainly matter a lot and ignoring the O typically underestimates

the security. Childs–Jao–Soukharev [CJS14, Theorem 5.2] prove a query complexity of

$$L_N\left[1/2, \sqrt{2}\right] = \exp\left[\left(\sqrt{2} + o(1)\right)\sqrt{\ln N \ln\ln N}\right], \tag{3.1}$$

where $N = \#\mathrm{cl}(\mathcal{O})$, for using Regev's algorithm for solving the hidden-shift problem. This estimates only the query complexity, so does not include the cost of queries to the quantum oracle (i.e. the isogeny oracle). Childs–Jao–Soukharev present two algorithms to compute the isogeny oracle, the fastest of which is due to Bisson [Bis12]. In [CJS14, Remark 4.8] Childs–Jao–Soukharev give an upper bound of

$$L_p[1/2, 1/\sqrt{2}] = \exp\left[\left(1/\sqrt{2} + o(1)\right)\sqrt{\ln p \ln\ln p}\right] \tag{3.2}$$

on the running time of Bisson's algorithm.

Remark 3.12. *Childs–Jao–Soukharev compute the total cost for computing the secret isogeny in [CJS14, Remark 5.5] to be $L_p[1/2, 3/\sqrt{2}]$ (using Regev and Bisson's algorithms, requiring only polynomial space). They appear to obtain this by setting $N = p$ when multiplying (3.1) and (3.2), but as $N \sim \sqrt{p}$ this is an overestimation and should be $L_p[1/2, 1 + 1/\sqrt{2}]$. Either way, this is the largest asymptotic complexity of the estimates. Also, [GV18] points out this algorithm actually has superpolynomial space complexity due to the high memory usage of the isogeny oracle in [CJS14], but see [JLLR18].*

Childs–Jao–Soukharev additionally compute the total time $L_p[1/2, 1/\sqrt{2}]$ for computing the secret isogeny combining Kuperberg [Kup05] and Bisson. This requires superpolynomial storage (also before considering the memory usage of the oracle). Note that in this combination the costs of the oracle computation dominate asymptotically.

It is important to mention that asymptotically inferior algorithms may provide practical improvements on our "small" instances over either of the algorithms studied by Childs–Jao–Soukharev: For example, Couveignes [Cou06, §5] provides heuristic arguments that one can find smooth representatives of ideal classes by computing the class-group structure (which can be done in polynomial time on a quantum computer [Hal05]) and applying a lattice-basis-reduction algorithm such as LLL [LLL82] to its lattice of relations. This might be more efficient than using Childs–Jao–Soukharev's subexponential oracle. However, note that this method makes evaluating the oracle several times harder for the attacker than for legitimate users, thus immediately giving a few additional bits of security, since users only evaluate the action of very smooth ideals by construction. We believe further research in this direction is necessary and important, since it will directly impact the cost of an attack, but we consider a detailed analysis of all these algorithms and possible trade-offs to be beyond the scope of this work.[12]

Remark 3.13. *After we posted a first version of the paper this chapter is based on on the Cryptology ePrint Archive, there were several independent attempts at assessing the security of CSIDH.*

Biasse, Iezzi, and Jacobson [BIJ18] work out some more details of the attack ideas mentioned above for Regev's algorithm. They focus on the class-group-computation part of the oracle and they describe how to represent random elements of the class group as a product of small prime ideals. Their analysis is purely asymptotic and an assessment of the actual cost on specific instances is explicitly left for future work.

Bonnetain and Schrottenloher [BS18] determine (quantum) query complexities for breaking CSIDH under the assumption that the quantum memory can be made very large, which implies that Kuperberg's faster algorithms would be applicable. They estimate the number of oracle queries as $(5\pi^2/4)2^{1.8\sqrt{\log N}}$.

[12] The page margins are certainly too narrow to contain such an analysis.

The 1.8 *appears to approximate the* $\sqrt{2\log 3}$ *in Kuperberg [Kup05, Theorem 5.1]. The number of qubits required is stated as* $2^{1.8\sqrt{\log N}+2.3}$.

While we ignored Kuperberg's algorithm due to the large memory costs, they take the stance that "the most time-efficient version is relevant", and so do not ignore this algorithm. For small N *the number of qubits stated in [BS18] might be possible, which would indeed make Kuperberg's algorithm relevant for these sizes. However, in this case the total cost is dominated by the high cost of computing the oracle, which Childs–Jao–Soukharev placed at* $L_p[1/2, 1/\sqrt{2}]$. *Bonnetain and Schrottenloher instead make use of Couveignes' (exponential-time, but perhaps better for small parameters) LLL-based method for the oracle computation, applying BKZ instead for more effective lattice-basis reduction.*

Jao, LeGrow, Leonardi, and Ruiz-Lopez address the issue of superpolynomial space in the oracle computation identified by Galbraith and Vercauteren (stated above) and give a new algorithm for finding short representations of elements. Their paper focuses on the asymptotic analysis of the oracle step so that they achieve overall polynomial quantum space, but does not obtain any concrete cost estimates.

We analyze the cost of quantum evaluation of the CSIDH group action in Chapter 9. Even after introducing several speedups to arithmetic in finite fields and computing isogenies in superposition, for CSIDH-512 it still takes 2^{40} *quantum operations on a quantum computer of* 2^{40} *qubits to compute a single evaluation of the Kuperberg or Regev oracle for success probability* 2^{-32} *and reduced range of exponents.*

See Section 11.4 for an account of more recent developments.

3.7.3 – Instantiations. Finally we present estimates for some sizes of p.

Security estimates. As explained in §3.7.1, the best classical attack has query complexity $O(\sqrt[4]{p})$, and the number of queries has been worked out for different quantum attacks. We consider [CJS14] in combination with Regev and Kuperberg ($L_p\left[1/2, 3/\sqrt{2}\right]$ and $L_p\left[1/2, 1/\sqrt{2}\right]$, respectively) as well as the pure query complexity of Regev's and Kuperberg's algorithms ($L_N\left[1/2, \sqrt{2}\right]$, $O(2^{3\sqrt{\log N}})$, and $O(2^{1.8\sqrt{\log N}})$, respectively). We summarize the resulting attack complexities, ignoring the memory costs and without restricting the maximum depth of quantum circuits, for some sizes of p in Table 3.1. We note again that we expect these complexities to be subject to more careful analysis, taking into account the implicit constants,[13] the (in)feasibility of long sequential quantum operations, and the large memory requirement. We also include the estimates on the query complexity and full attack complexity by [BS18].

We point out a recent analysis [Adj+18] which shows that the classical attack on SIDH (which is the same for CSIDH) is likely slower in practice than current parameter estimates assumed, which is due to the huge memory requirements of the searches. Similarly, the cost of the quantum attacks is significantly higher than just the query complexity multiplied with the cost of the group action, since evaluating the oracle in superposition is significantly more expensive than on a classical machine.

Recall that public keys consist of a single element $A \in \mathbb{F}_p$, which may be represented using $\lceil \log p \rceil$ bits. A private key is represented as a list of n integers in $\{-m, \ldots, m\}$, where m was chosen such that $n \log(2m+1) \approx \log \sqrt{p}$, hence it may be stored using roughly $(\log p)/2$ bits. Therefore the rows of Table 3.1 correspond to public key sizes of 64, 128, and 224 bytes, and private keys are approximately half that size when encoded optimally.

[13] This is illustrated dramatically by the eighth column stating a complexity of $L_p[1/2, 1/\sqrt{2}]$ for [CJS14]-Kuperberg, which we recall arises by multiplying the query complexity of Kuperberg's (first) algorithm and Childs–Jao–Soukharev's estimate $L_p[1/2, 1/\sqrt{2}]$ for the running time of Bisson's algorithm; so here it would make more sense to *add* the corresponding entries of the fourth column, but we decided to leave the numbers as they are in order to be consistent in the way we discard $o(1)$'s.

Table 3.1: Estimated attack complexities ignoring limits on depth. The three rightmost columns state costs for the complete attack; the others state classical and quantum query complexities. All numbers are rounded to whole bits and use $N = \#\mathrm{cl}(\mathcal{O}) = \sqrt{p}$, $o(1) = 0$, and all hidden O-constants 1, except for numbers taken from [BS18].

CSIDH-$(\log p)$	classical $\log \sqrt[4]{p}$	Regev [Reg04] $\log L_N[1/2, \sqrt{2}]$	Kuperberg [Kup05] $3\sqrt{\log N}$	Kuperberg [Kup05] $1.8\sqrt{\log N}$	Table 7 in [BS18]	[CJS14]-Regev $\log L_p[1/2, 3/\sqrt{2}]$	[CJS14]-Kuperberg $\log L_p[1/2, 1/\sqrt{2}]$	Table 8 in [BS18]
CSIDH-512	128	62	48	29	32.5	139	47	71
CSIDH-1024	256	94	68	41	44.5	209	70	88
CSIDH-1792	448	129	90	54	57.5	288	96	104

Security levels. We approximate security levels as proposed by NIST for the post-quantum standardization effort [NIST16, §4.A.5]. That is, the k-bit security level means that the required effort for the best attacks is at least as large as that needed for a key-retrieval attack on a block cipher with a k-bit key (e.g. AES-k for $k \in \{128, 192, 256\}$). In other words, under the assumption that the attacks query an oracle on a circuit at least as costly as AES, we should have a query complexity of at least 2^{k-1} resp. $\sqrt{2^k}$ to a classical resp. quantum oracle. NIST further restricts the power of the quantum computation to circuits of maximum depth 2^{40} up to 2^{96}, meaning that theoretically optimal tradeoffs (such as the formulas in Table 3.1 above) might not be possible for cryptographic sizes.

The parameters for CSIDH-$(\log p)$ were chosen to match the query complexity of Regev's attack on the hidden-shift problem (see the third column in Table 3.1) for roughly $2^{k/2}$, which should match NIST levels 1-3 as the group action computation has depth at least as large as AES.

Some other algorithms give lower estimates which makes it necessary to evaluate the exact cost of the oracle queries or compute the lower-order terms in the complexity. The analysis in [BS18, Table 8] states lower overall costs compared to AES. While this is a significant improvement, it is not clear that this affects our security claim when accounting precisely for the actual cost of oracle queries, as stated above. Our analysis in Chapter 9 shows costs of much more than 2^{40} qubit operations for evaluating the oracle for $\log p \approx 512$, whereas [BS18] assumed only 2^{37}. See also Section 11.4, which discusses CSIDH security claims including more recent developments.

3.8 — Implementation

In this section, we outline our most important tricks to make the system easier to implement or the code faster. As pointed out earlier, the crucial step is to use a field of size $4 \cdot \ell_1 \cdots \ell_n - 1$, where the ℓ_i are small distinct odd primes; this implies that *all* ℓ_i are Elkies primes for a supersingular elliptic curve over $\mathbb{F}_p$ and that the action of ideals $(\ell_i, \pi \pm 1)$ can be computed efficiently using $\mathbb{F}_p$-rational points. See Section 3.4 for these design decisions. The following section focuses on lower-level implementation details.

Montgomery curves. The condition $p + 1 \equiv 4 \pmod 8$ implies (cf. Proposition 3.8) that all curves in $\mathcal{E}\ell\ell_p(\mathbb{Z}[\pi], \pi)$ can be written as $y^2 = x^3 + Ax^2 + x$ with $A \in \mathbb{F}_p$ via an $\mathbb{F}_p$-isomorphism.

This is commonly referred to as the Montgomery form [Mon87] of an elliptic curve and is popular due to the very efficient arithmetic on its x-line. This extends well to computations of isogenies on the x-line, as was first shown by Costello–Longa–Naehrig [CLN16, §3]. Our implementation uses exactly the same formulas for operations on curves. For isogeny computations on Montgomery curves we use a projectivized variant (to avoid almost all inversions) of the formulas from Costello–Hişil [CH17] and Renes [Ren18]. This can be done as follows.

For a fixed prime $\ell \geq 3$, a point P of order ℓ, and an integer $k \in \{1, \dots, \ell - 1\}$, let $(X_k : Z_k)$ be the projectivized x-coordinate of $[k]P$. Then by defining $c_i \in \mathbb{F}_p$ such that

$$\prod_{i=1}^{\ell-1}(Z_i w + X_i) = \sum_{i=0}^{\ell-1} c_i w^i$$

as polynomials in w, we observe that

$$(\tau(A - 3\sigma) : 1) = \big(Ac_0c_{\ell-1} - 3(c_0c_{\ell-2} - c_1c_{\ell-1}) : c_{\ell-1}^2\big),$$

where

$$\tau = \prod_{i=1}^{\ell-1} \frac{X_i}{Z_i}, \quad \sigma = \sum_{i=1}^{\ell-1} \left(\frac{X_i}{Z_i} - \frac{Z_i}{X_i}\right)$$

and A is the Montgomery coefficient of the domain curve. By noticing that $x([k]P) = x([\ell-k]P)$ for all $k \in \{1, \dots, (\ell - 1)/2\}$ we can reduce the computation needed by about half. That is, we can compute $(\tau(A - 3\sigma) : 1)$ iteratively in about $5\ell\mathbf{M} + \ell\mathbf{S}$ operations[14], noting that $\tau(A - 3\sigma)$ is the Montgomery coefficient of the codomain curve of an isogeny with kernel $\langle P \rangle$ [Ren18, Proposition 1]. If necessary, a single division at the end of the computation suffices to obtain an affine curve constant. We refer to the implementation for more details.

Note that for a given prime ℓ, we could reduce the number of field operations by finding an appropriate representative of the isogeny formulas modulo (a factor of) the ℓ-division polynomial ψ_ℓ (as done in [CLN16] for 3- and 4-isogenies). Although this would allow for a more efficient implementation, we do not pursue this now for the sake of simplicity.

Rational points. Recall that the goal is to evaluate the action of (the class of) an ideal $\mathfrak{l}_1^{e_1} \cdots \mathfrak{l}_n^{e_n}$ on a curve $E \in \mathcal{E}\ell_p(\mathbb{Z}[\pi], \pi)$, where each $\mathfrak{l}_i = (\ell_i, \pi - 1)$ is a prime ideal of small odd norm ℓ_i and the e_i are integers in a short range $\{-m, \dots, m\}$. We assume E is given in the form $E_A \colon y^2 = x^3 + Ax^2 + x$.

The obvious way to do this is to consider each factor $\mathfrak{l}_i^{\pm 1}$ in this product and to find the abscissa of a point P of order ℓ_i on E, which (depending on the sign) is defined over $\mathbb{F}_p$ or $\mathbb{F}_{p^2} \backslash \mathbb{F}_p$. This exists by our choice of p and ℓ_i (cf. Section 3.4). Finding such an abscissa amounts to sampling a random $\mathbb{F}_p$-rational x-coordinate, checking whether $x^3 + Ax^2 + x$ is a square or not (for $\mathfrak{l}_i^{+1}$ resp. $\mathfrak{l}_i^{-1}$) in $\mathbb{F}_p$ (and resampling if it was wrong), followed by a multiplication by $(p+1)/\ell_i$ and repeating from the start if the result is ∞. The kernel of the isogeny given by $\mathfrak{l}_i^{\pm 1}$ is then $\langle P \rangle$, so the isogeny may be computed using Vélu-type formulas. Repeating this procedure for all $\mathfrak{l}_i^{\pm 1}$ gives the result.

However, fixing a sign before sampling a random point effectively means wasting about half of all random points, including an ultimately useless square test. Moreover, deciding on a prime ℓ_i before sampling a point and doing the cofactor multiplication wastes another proportion of the points, including both an ultimately useless square test and a scalar multiplication. Both of

[14] Here $\mathbf{M}$ and $\mathbf{S}$ denote a multiplication and squaring in $\mathbb{F}_p$.

these issues can be remedied by not fixing an ℓ_i before sampling a point, but instead taking *any* x-coordinate, determining the smallest field of definition (i.e. $\mathbb{F}_p$ or $\mathbb{F}_{p^2}$) of the corresponding point, and then performing whatever isogeny computations are possible using that point (based on its field of definition and order). The steps are detailed in Algorithm 3.2.

Algorithm 3.2: Evaluating the class-group action.

Input: $A \in \mathbb{F}_p$ and a list of integers $(e_1, \ldots, e_n)$.

Output: B such that $[\mathfrak{l}_1^{e_1} \cdots \mathfrak{l}_n^{e_n}]E_A = E_B$ (where $E_B \colon y^2 = x^3 + Bx^2 + x$).

While some $e_i \neq 0$ **do**
  Sample a random $x \in \mathbb{F}_p$.
  Set $s \leftarrow +1$ if $x^3 + Ax^2 + x$ is a square in $\mathbb{F}_p$, else $s \leftarrow -1$.
  Let $S = \{i \mid e_i \neq 0,\ \operatorname{sign}(e_i) = s\}$. **If** $S = \emptyset$ **then** start over with a new x.
  Let $k \leftarrow \prod_{i \in S} \ell_i$ and compute $Q \leftarrow [(p+1)/k]P$.
  For each $i \in S$ **do**
    Compute $R \leftarrow [k/\ell_i]Q$. **If** $R = \infty$ **then** skip this i.
    Compute an isogeny $\varphi \colon E_A \to E_B \colon y^2 = x^3 + Bx^2 + x$ with $\ker(\varphi) = R$.
    Set $A \leftarrow B$, $Q \leftarrow \varphi(Q)$, $k \leftarrow k/\ell_i$, and finally $e_i \leftarrow e_i - s$.
Return A.

Due to the commutativity of $\mathrm{cl}(\mathcal{O})$, and since we only decrease (the absolute value of) each e_i once we successfully applied the action of $\mathfrak{l}_i^{\pm 1}$ to the current curve, this algorithm indeed computes the action of $[\mathfrak{l}_1^{e_1} \mathfrak{l}_2^{e_2} \cdots \mathfrak{l}_n^{e_n}]$.

Remark 3.14. *Since the probability that a random point has order divisible by ℓ_i (and hence leads to an isogeny step in Algorithm 3.2) grows with ℓ_i, the isogeny steps for big ℓ_i are typically completed before those for small ℓ_i. Hence it may make sense to sample the exponents e_i for ideals $\mathfrak{l}_i$ from different ranges depending on the size of ℓ_i, or to not include any very small ℓ_i in the factorization of $p+1$ at all to reduce the expected number of repetitions of the loop above. Note moreover that doing so may also improve the performance of straightforward constant-time adaptions of our algorithms, since it yields stronger upper bounds on the maximum number of required loop iterations (at the expense of slightly higher cost per isogeny computation). Varying the choice of the ℓ_i can also lead to performance improvements if the resulting prime p has lower Hamming weight. Finding such a p is a significant computational effort but needs to be done only once; all users can use the same finite field.*

Remark 3.15. *Algorithm 3.2 is obviously strongly variable-time when implemented naïvely. Indeed, the number of points computed in the isogeny formulas is linear in the degree, hence the iteration counts of certain loops in our implementation are very directly related to the private key. We note that it would not be very hard to create a constant-time implementation based on this algorithm by always performing the maximal required number of iterations in each loop and only storing the results that were actually needed (using constant-time conditional instructions), although this incurs quite a bit of useless computation, leading to a doubling of the number of curve operations on average. We leave the design of optimized constant-time algorithms for future work.*

Public-key validation. Recall that the public-key validation method outlined in Section 3.5 essentially consists of computing $[(p+1)/\ell_i]P$ for each i, where P is a random point on E. Per-

forming this computation in the straightforward way is simple and effective. On the other hand, a divide-and-conquer approach, such as the following recursive algorithm, yields better speeds at the expense of slightly higher memory usage. Note that Algorithm 3.3 only operates on public data, hence need not be constant-time in a side-channel resistant implementation.

Algorithm 3.3: Batch cofactor multiplication. [Suto7, Algorithm 7.3]

Input: An elliptic-curve point P and positive integers $(k_1, \ldots, k_n)$.

Output: The points $(Q_1, \ldots, Q_n)$, where $Q_i = \left[\prod_{j \neq i} k_j\right] P$.

1 **If** $n = 1$ **then return** (P). // base case

2 Set $m \leftarrow \lceil n/2 \rceil$ and let $u \leftarrow \prod_{i=1}^{m} k_i$, $v \leftarrow \prod_{i=m+1}^{n} k_i$.

3 Compute $L \leftarrow [v]P$ and $R \leftarrow [u]P$.

4 **Recurse** with input $L, (k_1, \ldots, k_m)$ giving $(Q_1, \ldots, Q_m)$. // left half

5 **Recurse** with input $R, (k_{m+1}, \ldots, k_n)$ giving $(Q_{m+1}, \ldots, Q_n)$. // right half

6 **Return** $(Q_1, \ldots, Q_n)$.

This routine can be used for verifying that an elliptic curve $E/\mathbb{F}_p$ is supersingular as follows: Pick a random point $P \in E(\mathbb{F}_p)$ and run Algorithm 3.3 on input $[4]P$ and $(\ell_1, \ldots, \ell_n)$ to obtain the points $Q_i = [(p+1)/\ell_i]P$. Then continue like in Algorithm 3.1 to verify that E is supersingular using these precomputed points.

In practice, it is not necessary to run Algorithm 3.3 as a black-box function until it returns all the points $Q_1, \ldots, Q_n$: The order checking in Algorithm 3.1 can be performed as soon as a new point Q_i becomes available, i.e., in the base case of Algorithm 3.3. This reduces the memory usage (since the points Q_i can be discarded immediately after use) and increases the speed (since the algorithm terminates as soon as enough information was obtained) of public-key validation using Algorithms 3.1 and 3.3. We note that the improved performance of this algorithm compared to Algorithm 3.1 alone essentially comes from a time-space trade-off, hence the memory usage is higher (cf. Section 3.8.1). On severely memory-constrained devices one may instead opt for the naïve algorithm, which requires less space but is slower.

3.8.1 – Performance results. On top of a minimal implementation in the Sage computer algebra system [Sage] for demonstrative purposes, we created a somewhat optimized proof-of-concept implementation of the CSIDH group action for a particular 512-bit prime p. While this implementation features 512-bit field arithmetic written in assembly (for Intel Skylake processors), it also contains generic C code supporting other field sizes and can therefore easily be ported to other computer architectures or parameter sets if desired.[15]

The prime p is chosen as $p = 4 \cdot \ell_1 \cdots \ell_{74} - 1$ where ℓ_1 through ℓ_{73} are the smallest 73 odd primes and $\ell_{74} = 587$ is the smallest prime distinct from the other ℓ_i that renders p prime. This parameter choice implies that public keys have a size of 64 bytes. Private keys are stored in 37 bytes for simplicity, but an optimal encoding would reduce this to only 32 bytes. Table 3.2 summarizes performance numbers for our proof-of-concept implementation. Note that private-key generation is not listed as it only consists of sampling n random integers in a small range $\{-m, \ldots, m\}$, which has negligible cost.

[15] Our code for this chapter is published in the public domain and is available for download at `https://yx7.cc/code/csidh/csidh-latest.tar.xz`.

Table 3.2: Performance numbers for our proof-of-concept implementation (2018-08-26), averaged over 10 000 runs on an Intel Skylake i5 processor clocked at 3.5 GHz.

	Clock cycles	**Wall-clock time**	**Stack memory**
Key validation	$5.5 \cdot 10^6$ cc	2.1 ms	4 368 bytes
Group action	$106 \cdot 10^6$ cc	40.8 ms	2 464 bytes

We emphasize that both our implementations are intended as a proof of concept and unfit for production use; in particular, they are explicitly *not side-channel resistant* and may contain any number of bugs. We leave the design of hardened and more optimized implementations for future work.

Chapter 4

Faster SeaSign signatures through improved rejection sampling

This chapter is for all practical purposes identical to the paper *Faster SeaSign signatures through improved rejection sampling* [DPV19] authored jointly with Thomas Decru and Frederik Vercauteren, which was published at PQCrypto 2019.

4.1 — Introduction

CSIDH's small key sizes prompted De Feo and Galbraith soon afterwards to transform it into a signature scheme called *SeaSign* [DG19]. The construction uses the *Fiat–Shamir with aborts* framework, a technique commonly used in lattice-based cryptography [Lyu09], in combination with an isogeny-based identification scheme going back to Couveignes [Cou06] and independently Stolbunov [Sto12]. Their paper presents three different versions of SeaSign featuring various trade-offs between signature size, public-key size, and secret-key size. One of these versions attains 128 bits of security with signatures of less than one kilobyte. An issue impacting all of these schemes, however, is that the signing and verification times are rather substantial. Indeed, the basic SeaSign scheme takes (on average) almost two days to sign a message on a typical CPU, whereas the variants with smaller signatures or public keys still take almost ten minutes to sign (on average).

In this chapter we tackle this performance issue in the more general setting of using group actions in a "Fiat–Shamir with aborts" scheme. We first discuss two (unfortunately mutually exclusive) adjustments that reduce the likelihood of rejections, which decreases the expected number of failed signing attempts before a success and hence makes signing more efficient. Next, we describe a modification that significantly speeds up the signing process at the cost of a small increase in signature size. The basic idea is to allow the prover to refuse answering a small fixed number of challenges, thereby reducing the overall probability of aborting. To attain a given security level, the total number of challenges — and correspondingly the signature size — will be somewhat larger than for standard Fiat–Shamir with aborts. As an application of these general techniques, we analyze the resulting speed-up for the various versions of the SeaSign signature scheme. The improvement is most noticeable when applied to the basic scheme: the original signing cost goes down from almost two days to just over half an hour. The other two, more advanced variants are still sped up by a factor of four to roughly two minutes per signature. Even though this is still too slow for most (if not all) applications, it is a significant improvement over the state of the art, and the underlying ideas of these speed-ups might be useful for other cryptographic schemes as well.

Acknowledgements. We are thankful to Steven Galbraith for his observation about shorter signatures in Remark 4.3, and to Taechan Kim for pointing out an error in an earlier version of the script in Section 4.4.

4.1.1 – Notation. The notation $[a; b]$ denotes the integer range $\{a, \dots, b\}$.

Fix $n \geq 1$. Throughout, we will consider a transitive action of the abelian group $\mathbb{Z}^n$ on a finite set X, with a fixed element $E_0 \in X$. We will assume that "short" vectors in $\mathbb{Z}^n$ are enough to reach "almost all" elements of X.[1] Moreover, we assume that the cost of computing the action $[\mathbf{v}]E$ of a vector $\mathbf{v} \in \mathbb{Z}^n$ on an element $E \in X$ is linear in the 1-norm $\|\mathbf{v}\|_1 = \sum_{j=1}^n |v_j|$ of $\mathbf{v}$. (We will argue in Section 4.2.1 that these assumptions are satisfied in the CSIDH setting.)

4.2 — Preliminaries

We recall some facts from Chapters 2 and 3.

4.2.1 – CSIDH. Consider a supersingular elliptic curve E defined over $\mathbb{F}_p$, where p is a large prime. While the endomorphism ring $\mathrm{End}(E)$ of E over the algebraic closure of $\mathbb{F}_p$ is noncommutative, the ring $\mathrm{End}_{\mathbb{F}_p}(E)$ of endomorphisms defined over $\mathbb{F}_p$ is an order $\mathcal{O}$ in the imaginary quadratic field $\mathbb{Q}(\sqrt{-p})$.

The ideal class group of $\mathrm{End}_{\mathbb{F}_p}(E) = \mathcal{O}$ is the quotient of the group of fractional invertible ideals in $\mathcal{O}$ by the principal fractional invertible ideals in $\mathcal{O}$, and will be denoted $\mathrm{cl}(\mathcal{O})$. The group $\mathrm{cl}(\mathcal{O})$ acts on the set of $\mathbb{F}_p$-isomorphism classes of elliptic curves with $\mathbb{F}_p$-rational endomorphism ring $\mathcal{O}$ through isogenies. More specifically, when given an $\mathcal{O}$-ideal $\mathfrak{a}$ and an elliptic curve E with $\mathrm{End}_{\mathbb{F}_p}(E) = \mathcal{O}$, we define $[\mathfrak{a}]E$ as the codomain of the isogeny $\varphi_{\mathfrak{a}} \colon E \to E/\mathfrak{a}$ whose kernel is $\bigcap_{\alpha \in \mathfrak{a}} \ker(\alpha)$. This isogeny is well-defined and unique up to $\mathbb{F}_p$-isomorphism.

There are formulas for computing $[\mathfrak{a}]E$. However, for general $\mathfrak{a}$, this computation requires large field extensions and hence has superpolynomial time complexity. To avoid this, CSIDH restricts to ideals of the form $\mathfrak{a} = \prod_{i=1}^n \mathfrak{l}_i^{e_i}$, where all $\mathfrak{l}_i$ are prime ideals of small norm ℓ_i, and such that the action of $\mathfrak{l}_i$ can be computed entirely over the base field $\mathbb{F}_p$. The curve $[\mathfrak{a}]E$ can then be computed by chaining isogenies of degrees ℓ_i. In principle the cost of computing the action of $\mathfrak{l}_i$ is in $\Theta(\ell_i)$, but for small values of ℓ_i it is dominated by a full-size scalar multiplication, which is why assuming cost $|e_1| + \cdots + |e_n|$ for computing the action of $\prod_{i=1}^n \mathfrak{l}_i^{e_i}$, as mentioned in Section 4.1.1, comes close to the truth. (Moreover, in our setting, the $|e_i|$ are all identically distributed, hence the differences in costs between various ℓ_i disappear on average.)

The CSIDH group action is defined as follows.

Parameters. Integers $n \geq 1$, $m \geq 0$. A prime p of the form $4 \cdot \ell_1 \cdots \ell_n - 1$, with ℓ_i small distinct odd primes. The elliptic curve $E_0 \colon y^2 = x^3 + x$ over $\mathbb{F}_p$. Write $\mathcal{El}_p(\mathcal{O})$ for the set of ($\mathbb{F}_p$-isomorphism classes of) elliptic curves over $\mathbb{F}_p$ with $\mathrm{End}_{\mathbb{F}_p}(E) = \mathcal{O} = \mathbb{Z}[\pi]$, where π is the $\mathbb{F}_p$-Frobenius endomorphism.

Group action. A group element is represented[2] by a vector $(e_1, \dots, e_n) \in \mathbb{Z}^n$ sampled uniformly at random from $[-m; m]^n$, which defines the ideal $\mathfrak{a} = \prod_{i=1}^n \mathfrak{l}_i^{e_i}$ with $\mathfrak{l}_i = \langle \ell_i, \pi - 1 \rangle$. A public element is a single coefficient $A \in \mathbb{F}_p$, representing the curve $E_A \colon y^2 = x^3 + Ax^2 + x$. The result of the action of an ideal $\mathfrak{a}$ on a public element $A \in \mathbb{F}_p$, assuming that E_A has the right

[1] In other words: The action of $\mathbb{Z}^n$ on X factors through the quotient $Q = \mathbb{Z}^n/\Lambda$, where $\Lambda \leq \mathbb{Z}^n$ is the stabilizer of any $E \in X$, and we assume that Q is "sufficiently" covered by "short" vectors in $\mathbb{Z}^n$ under the quotient map $\mathbb{Z}^n \twoheadrightarrow Q$.

[2] Note this representation matches the assumptions in Section 4.1.1.

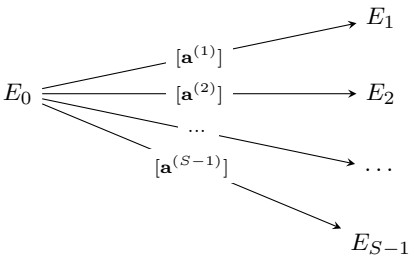


Figure 4.1: Structure of Alice's key pair.

endomorphism ring $\mathcal{O}$, is the coefficient B of the curve $[\mathfrak{a}]E_A \colon y^2 = x^3 + Bx^2 + x$.

The security assumption of the group action is that it is essentially a black-box version of the group $\mathrm{cl}(\mathcal{O})$ on which anyone can efficiently act by translations. In particular, given two elliptic curves $E, E' \in X$, it should be hard to find an ideal $\mathfrak{a}$ of $\mathcal{O}$ such that $E' = [\mathfrak{a}]E$.

Notice that it is not clear in general that the vectors in $[-m; m]^n$ cover the whole group, or even a "large" fraction. Unfortunately, sampling uniformly random from $\mathrm{cl}(\mathcal{O})$ is infeasible for large enough parameters, since there is no known efficient way to compute the structure of $\mathrm{cl}(\mathcal{O})$ in that case. In fact, knowing the exact class group structure would be sufficient to obtain much more efficient signatures, since no rejection sampling would be required [DG19]. Under the right assumptions however, the elements represented by vectors in $[-m; m]^n$ are likely to cover a large fraction of the group as long as $(2m+1)^n \geq \#\mathrm{cl}(\mathcal{O})$. The values suggested for (n, m) in [Cas+18] are $(74, 5)$, which aim to cover a group of size approximately 2^{256}. This results in group elements of 32 bytes, public elements of 64 bytes, and a performance of about 40 ms per group action computation. For more details, see Chapter 3.

As stated in Section 4.1.1, we will from now on abstract away the underlying isogeny-based constructions and work in the setting of the group $(\mathbb{Z}^n, +)$ acting on a finite set X.

4.2.2 – SeaSign. SeaSign [DG19] is a signature scheme based on a sketch of an isogeny-based identification scheme by Couveignes [Cou06] and Stolbunov [Sto10], in combination with the "Fiat–Shamir with aborts" construction [Lyu09] from lattice-based cryptography to avoid leakage. The identification part of SeaSign works as follows. Note that our exposition differs from [DG19] for consistency with the following sections.

Parameters. Like CSIDH, and additionally integers $\delta \geq 1$ and $S \geq 2$. [3]

Keys. Alice's private key is a list $\underline{\mathbf{a}} = (\mathbf{a}^{(1)}, \ldots, \mathbf{a}^{(S-1)})$ of $S-1$ vectors sampled uniformly at random from $[-m; m]^n \subseteq \mathbb{Z}^n$.

For $i \in \{1, \ldots, S-1\}$, write $E_i := [\mathbf{a}^{(i)}]E_0$, that is, the result of applying the group element represented by $\mathbf{a}^{(i)} \in \mathbb{Z}^n$; then Alice's public key is the list $[\underline{\mathbf{a}}]E_0 := (E_1, \ldots, E_{S-1})$ of her secret vectors applied to the starting element E_0.

This situation is summarized in Figure 4.1.

[3] Technically, there is no reason for δ to be an integer: it is sufficient that $\delta \in \frac{1}{m} \cdot \mathbb{Z}$, but we will assume $\delta \in \mathbb{Z}$ throughout for simplicity.

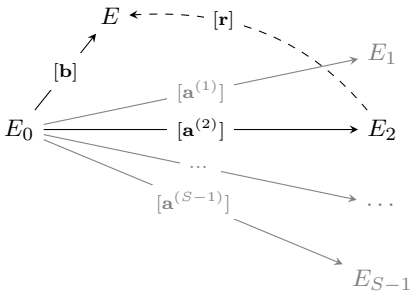


Figure 4.2: The identification scheme in the scenario $c = 2$.

Identification. Alice samples an ephemeral vector $\mathbf{b}$ uniformly random from

$$[-(\delta+1)m;\ (\delta+1)m]^n \subseteq \mathbb{Z}^n\,.$$

She then computes $E = [\mathbf{b}]E_0$ and commits to E. On challenge $c \in \{0,\dots,S-1\}$, she computes $\mathbf{r} = \mathbf{b} - \mathbf{a}^{(c)}$ (where $\mathbf{a}^{(0)}$ is defined as $\mathbf{0}$). If $\mathbf{r} \in [-\delta m; \delta m]^n$, she reveals $\mathbf{r}$; else she rejects the challenge. Bob verifies that $[\mathbf{r}]E_c = E$.

See Figure 4.2 for a visual representation of this protocol.

Since an attacker (who cannot break the underlying isogeny problems) has a $1/S$ chance of winning, this identification scheme provides $\log_2 S$ bits of security. In order to amplify the security level, Alice typically computes $t \geq 1$ independent vectors $\mathbf{b}_1,\dots,\mathbf{b}_t$ instead of just one. The verifier responds with t challenges $c_1,\dots,c_t \in \{0,\dots,S-1\}$. Alice then computes $\mathbf{r}_i = \mathbf{b}_i - \mathbf{a}^{(c_i)}$ for all $1 \leq i \leq t$ and reveals them if all of them are in $[-\delta m; \delta m]^n$; else she rejects the challenge. In order to not have to reject too often, δ must be rather large; more specifically, δ was chosen as nt in [DG19] to achieve a success probability of roughly $1/e$.

As mentioned in the introduction, [DG19] gives three SeaSign constructions. The original idea is the scheme above with $S = 2$, i.e., the public key is a single public element. This results in a large t and therefore a very large signature. The second scheme lets the number of private keys S range from 2 up to 2^{16}, which results in smaller, faster signatures at the expense of larger public-key sizes.[4] The final scheme reduced the size of the public key again by using a Merkle tree, at the cost of increasing the signature size. We will not elaborate on all those variants in detail.

To turn this identification scheme into a non-interactive signature protocol, the standard Fiat–Shamir transformation can be applied [FS86]. In a nutshell, Alice computes the challenges $c_1,\dots,c_t$ herself by hashing the ephemeral public elements $[\mathbf{b}_1]E_0,\dots,[\mathbf{b}_t]E_0$ together with her message. Alice then sends her signature $([\mathbf{b}_1]E_0,\dots,[\mathbf{b}_t]E_0;\ \mathbf{r}_1,\dots,\mathbf{r}_t)$ to Bob, who can recompute the challenges $c_1,\dots,c_t$ to verify that indeed $[\mathbf{r}_i]E_{c_i} = [\mathbf{b}_i]E_0$ for all $i \in \{1,\dots,t\}$.

4.3 — The improved signature scheme

In this section we describe our improvements.

4.3.1 – Core ideas.

[4] In [DG19], S is always a power of 2, but any $S \geq 2$ works.

1. The first improvement is minor (but still has significant implications) and concerns the identification scheme itself: the following observations result in two variants that are more efficient than the basic scheme.[5]

   - Variant $\mathcal{F}$: The ephemeral secret $\mathbf{b}$ is automatically independent of all secrets $\mathbf{a}^{(i)}$, hence can be revealed even if it lies outside of $[-\delta m; \delta m]^n$. We remark that this variant is described in [DG19] already but disregarded as only a single signing attempt is examined. When taking into account the average signing cost, however, it can clearly improve performance, and we will quantify these improvements.
   - Variant $\mathcal{T}$: Depending on the entries of the concrete private keys $\mathbf{a}^{(i)}$, the ephemeral secret $\mathbf{b}$ can be sampled from a smaller set than the worst-case range used in SeaSign to reduce the probability of rejection. Indeed, although the j-th entry in each $\mathbf{a}^{(i)}$ is a priori sampled uniformly in $[-m; m]$, which makes the j-th coefficient of each ephemeral vector $\mathbf{b}$ lie in the interval $[-(\delta+1)m; (\delta+1)m]$, it is useless (since it will *always* lead to a rejection) to sample the j-th coefficient outside the smaller interval $[-\delta m + m_j;\ \delta m + M_j]$ where $m_j = \min\{0, a_j^{(1)}, \ldots, a_j^{(S-1)}\}$ and $M_j = \max\{0, a_j^{(1)}, \ldots, a_j^{(S-1)}\}$.

   It is clear that Variant $\mathcal{F}$ and Variant $\mathcal{T}$ are mutually exclusive: in Variant $\mathcal{T}$ the ephemeral secret $\mathbf{b}$ is sampled from a set that is dependent on the private keys $\mathbf{a}^{(i)}$, whereas for Variant $\mathcal{F}$ to work it is required that this sampling is done completely independently.

2. The second improvement is more significant and modifies the "Fiat–Shamir with aborts" transform as follows: assume the identification scheme uses s-bit challenges (corresponding to a probability of 2^{-s} that an attacker can cheat), and that each execution has probability of rejection ε. The SeaSign approach to attain security level λ is to simultaneously obtain $t = \lceil \lambda/s \rceil$ non-rejected executions of the identification protocol which happens with probability $(1-\varepsilon)^t$. Our approach increases the total number of challenges, but allows the prover to refuse answering a fixed number u of them, since this tolerates much higher rejection probabilities at the cost of a relatively small increase in public-key and signature size.

We now provide more details on each of the above ideas.

4.3.2 – Identification scheme.

Parameters. Integers $S \geq 2$ and $\delta \geq 1$.

Keys. Like in SeaSign (Section 4.2.2).

Identification. Using Alice's key pair $(\underline{\mathbf{a}}, [\underline{\mathbf{a}}]E_0)$, a $(\log_2 S)$-bit identification protocol can be constructed as shown in Figure 4.3.

Lemma 4.1. *The distribution of revealed vectors $\mathbf{r}$ is independent of $\mathbf{a}^{(c)}$.*

Proof. This is trivial in Variant $\mathcal{F}$ in the event $c = 0$. For the other cases, note that I is constructed such that $\mathbf{r} = \mathbf{b} - \mathbf{a}^{(c)}$ is uniformly distributed on a set containing $\Delta := [-\delta m; \delta m]^n$, no matter what $\mathbf{a}^{(c)}$ is. Therefore, the distribution of $\mathbf{r}$ conditioned on the event $\mathbf{r} \in \Delta$ is uniform on Δ independently of $\mathbf{a}^{(c)}$. □

[5] The acronyms $\mathcal{F}$ and $\mathcal{T}$ refer to "**f**ull" and "**t**runcated" ranges, respectively.

Variant $\mathcal{F}$	Variant $\mathcal{T}$
Alice samples a vector $\mathbf{b}$ uniformly random from the set ...	
$I = \big[-(\delta+1)m;\ (\delta+1)m\big]^n \subseteq \mathbb{Z}^n$.	$I = \prod_{j=1}^{n} \big[-\delta m + m_j;\ \delta m + M_j\big] \subseteq \mathbb{Z}^n$, where $m_j = \min\{0, a_j^{(1)}, \ldots, a_j^{(S-1)}\}$; $M_j = \max\{0, a_j^{(1)}, \ldots, a_j^{(S-1)}\}$.
She then computes $E = [\mathbf{b}]E_0$ and commits to E. On challenge $c \in \{0, \ldots, S-1\}$, she computes $\mathbf{r} = \mathbf{b} - \mathbf{a}^{(c)}$ (where $\mathbf{a}^{(0)}$ is defined as $\mathbf{0}$).	
If $c = 0$ or $\mathbf{r} \in [-\delta m; \delta m]^n$, ...	If $\mathbf{r} \in [-\delta m; \delta m]^n$, ...
... then she reveals $\mathbf{r}$; else she rejects the challenge. Bob verifies that $[\mathbf{r}]E_c = E$.	

Figure 4.3: Our $(\log_2 S)$-bit identification scheme

Remark 4.2. *Lemma 4.1 only talks about the conditional distribution of* $\mathbf{r}$ if *it is revealed. Note that in Variant* $\mathcal{T}$*, the probability* that *it can be revealed is still correlated to the entries of* $\mathbf{a}^{(c)}$*, which may have security implications. We show in Section 4.3.3 how to get around this issue in a signature scheme.*

4.3.3 – Signature scheme. Our improved signature scheme is essentially the "Fiat–Shamir with aborts" construction also used in SeaSign (see Section 4.2.2), except that we allow the signer to reject a few challenges in each signature. The resulting scheme is parameterized by two integers $t \geq 0$, denoting the number of challenges the signer must answer correctly, and $u \geq 0$, the number of challenges she may additionally refuse to answer.

Write ID for (one of the variants of) the identification scheme in Section 4.3.2.

Keys. Alice's identity key consists of a key pair $(\underline{\mathbf{a}}, [\underline{\mathbf{a}}]E_0)$ as in ID.

Signing. To sign a message m, Alice first generates a list $\mathbf{b}_1, \ldots, \mathbf{b}_{t+u}$ of random vectors, each sampled like the vector $\mathbf{b}$ in ID. She computes the corresponding list of public elements $[\mathbf{b}_1]E_0, \ldots, [\mathbf{b}_{t+u}]E_0$ and hashes them together with the message m to obtain a list of challenges $c_1, \ldots, c_{t+u} \in \{0, \ldots, S-1\}$. To produce her signature, she then traverses the tuples $(\mathbf{b}_i, c_i)$ in a random order, computing the correct response $\mathbf{r}_i = \mathbf{b}_i - \mathbf{a}^{(c_i)}$ (as in ID) if possible and a rejection ✗ otherwise. Once t successful responses have been generated, the remaining challenges are all rejected in order not to leak any information about the rejection probability; cf. Remark 4.2.[6] Finally, the signature is

$$\big([\mathbf{b}_1]E_0, \ldots, [\mathbf{b}_{t+u}]E_0;\ \mathbf{r}_1, \ldots, \mathbf{r}_{t+u}\big),$$

where exactly u of the $\mathbf{r}_i$ equal ✗. (If less than t challenges could be answered, Alice aborts and retries the whole signing process with new values of $\mathbf{b}_i$.)

[6] This is why the tuples are processed in a random order: Proceeding sequentially and rejecting the remaining tail still leaks, since the number of ✗ at the end would be correlated to the rejection probability.

Verification. This again is standard: Bob first checks that at most u of the $t+u$ values $\mathbf{r}_i$ have been rejected ✗. He then recomputes the challenges $c_1, \ldots, c_{t+u}$ by hashing the message m together with the ephemeral elements $[\mathbf{b}_i]E_0$ and verifies that $[\mathbf{r}_i]E_{c_i} = [\mathbf{b}_i]E_0$ holds for all $i \in \{1, \ldots, t+u\}$ with $\mathbf{r}_i \neq$ ✗.

Remark 4.3. *The signatures can be shortened further: Sending those $[\mathbf{b}_i]E_0$ with $\mathbf{r}_i \neq$ ✗ is wasteful. It is enough to send the hash H of all ephemeral elements $[\mathbf{b}_i]E_0$ instead, since Bob can extract c_i from H, recompute $[\mathbf{b}_i]E_0$ as $[\mathbf{r}_i]E_{c_i}$, and verify in the end that the hash H was indeed correct.*

Remark 4.4. *As mentioned earlier, one can reduce the public-key size by using a Merkle tree, but this does not significantly alter the computation time for any part of the protocol. Given that the main focus of our adjustments to SeaSign is speeding it up, we will therefore not investigate this avenue any further.*

Security. The proof for the security for this scheme is completely analogous to the original SeaSign scheme. This follows from Lemma 4.1 and the fact that there are always a fixed number u of ✗ per signature in random positions. Instead of reproducing the proof here, we refer the reader to [DG19].

4.4 — Analysis and results

In order to quantify our speed-ups compared to the original SeaSign scheme, we analyze our adjustments in the same context as [DG19]. This means $(n, m) = (74, 5)$ and $\log_2 p \approx 512$. Furthermore, we require 128 bits of security and let S range through powers of two between 2 and 2^{16}.

As mentioned before, Variant $\mathcal{F}$ and Variant $\mathcal{T}$ are mutually exclusive. For this reason, we computed the results for both cases to compare which performs better under given conditions. Variant $\mathcal{T}$ clearly converges to the original SeaSign scheme rapidly for growing S, while Variant $\mathcal{F}$ always keeps at least a little bit of advantage. It is clear that from a certain value of S onward, Variant $\mathcal{F}$ will always be better. For small S however, Variant $\mathcal{T}$ will outperform Variant $\mathcal{F}$ rather significantly for average-case key vectors.

We now discuss how to optimize the parameters (t, u, δ) for a given S. The main cost metric is the *expected* signing time[7]

$$\delta \cdot (t+u)/q\,,$$

where q is the probability of a full signing attempt being successful (i.e., at most u rejections ✗). This optimization problem depends on two random variables:

- The number Z of challenges that an *attacker* can successfully answer even though they cannot break the underlying isogeny problems.
- The number Y of challenges that *Alice* can answer without leaking, i.e., the number of non-rejected challenges.

Since the $t+u$ challenges are independent, both Z and Y are binomially distributed with count $t+u$. Let $T_{k,\alpha}$ denote the tail cumulative distribution function of $\mathrm{Bin}_{k,\alpha}$, i.e.,

$$T_{k,\alpha}(x) = \sum_{i=x}^{k} \binom{k}{i} \alpha^i (1-\alpha)^{k-i}\,,$$

[7] Other optimizations could look at the sum of signing and verification time, or even take into account key generation time, but we will not delve into those options.

which is the probability that a $\mathrm{Bin}_{k,\alpha}$-distributed variable attains a value of at least x. The success probability for an attacker is $1/S$, since he knows the correct answer to at most one of S challenges c. In order to achieve 128 bits of security, it is required that

$$\Pr[Z \geq t] = T_{t+u,1/S}(t) \leq 2^{-128}.$$

This condition implies that for fixed S and t, there is a maximal value $u_{max}(t)$ for u, the number of allowed rejections ✗, regardless of δ.

Let $\sigma(\delta)$ denote Alice's probability of being able to answer (i.e., not reject ✗) a single challenge for a given value of δ; hence $Y \sim \mathrm{Bin}_{t+u,\sigma(\delta)}$. In order to find the optimal (u, δ) for a given t, we need to minimize the expression

$$\delta \cdot (t+u)/q(t,u,\delta)\,,$$

where

$$q(t,u,\delta) = \Pr[Y \geq t] = T_{t+u,\sigma(\delta)}(t)$$

is the probability of a full signing attempt being successful. The function σ depends on the variant ($\mathcal{F}$ or $\mathcal{T}$). In case of Variant $\mathcal{F}$ we have

$$\sigma(\delta) = \frac{1}{S} + \frac{S-1}{S}\left(\frac{2\delta m+1}{2(\delta+1)m+1}\right)^n.$$

For Variant $\mathcal{T}$, the function σ even depends on the private keys in use. With fixed private keys $a^{(1)}, ..., a^{(S-1)}$ and writing $m_j = \min\{0, a_j^{(1)}, ..., a_j^{(S-1)}\}$ and $M_j = \max\{0, a_j^{(1)}, ..., a_j^{(S-1)}\}$ as before, the formula becomes

$$\sigma(\delta) = \prod_{j=1}^{n} \frac{2\delta m+1}{2\delta m+1-m_j+M_j}.$$

For our analysis we work with the expected probability over all possible keys.

Our results for the optimization problem can be found in Table 4.1. The [Sage] code that computes these values can be found in Section 4.4; it takes about twelve minutes on a single core. We are quite confident that the values in Table 4.1 are optimal, but cannot strictly claim so since we have not *proven* that the conditions used in the script to terminate the search capture all optimal values, although this seems reasonable to assume.

There are two major differences in the way we present our data compared to [DG19]. First of all, we list the *expected* signing time instead of a single signing *attempt*, which represents the real cost more accurately. Second, we express the time in equivalents of "normal" CSIDH operations instead of in wall-clock time, which makes the results independent of a concrete choice of CSIDH implementation and eases comparison with other work.

Unsurprisingly, the biggest speed-up can be seen for the basic SeaSign scheme (i.e., $S = 2$), since that is where the largest δ could be found. The expected signing time is reduced by a factor of 65, whereas verification is sped up by a factor of roughly 31, at the cost of doubling the signature size. As predicted, Variant $\mathcal{F}$ outperforms Variant $\mathcal{T}$ from a certain point onward, which apparently is for $S \geq 2^4$. The case $S = 2^{16}$ gains a factor of 4.4 in the expected signing time and 6.0 in verification time. Note though that it only has 2.7% faster signing and 21% faster verification than the case $S = 2^{15}$ (which uses public keys half as big), which further emphasizes the importance of choosing the right trade-offs. Perhaps unsurprisingly, taking $u = u_{max}(t)$ often gives the best (expected) signing times, although this is not always the case: for instance, for $S = 2^{16}$ we have $u_{max}(10) = 29$, but $u = 22$ with a bigger δ yields (slightly) better results.

Table 4.1: Parameters for our improved SeaSign variants, optimizing for signing time. All of these choices provide ≥ 128 bits of security (of course assuming that the underlying isogeny problems are hard). Lines with variant "—" refer to the original parameter selection methodology suggested in [DG19]. The signature sizes make use of the observation in Remark 4.3. The "CSIDHs" columns express the computational load in terms of equivalents of a "normal" CSIDH operation, i.e., with exponents in $[-m; m]^n$, making use of the assumption that the cost is linear in the 1-norm of the input vector. Using current implementations [MR18; Cas+18], computing one "CSIDH"-512 takes approximately 40 ms of wall-clock time on a standard processor. Finally, the rightmost column shows the speed-up in signing and verification times compared to the original SeaSign scheme.

S	t	u	δ	Var.	Public-key bytes	Signature bytes	Expected signing attempts	Expected signing CSIDHs	Expected verifying CSIDHs	Speed-up factors
2^1	128	0	9472	—	64 b	19600 b	2.718	3295480	1212416	
2^1	337	79	114	$\mathcal{T}$	64 b	36838 b	1.058	50175	38418	65.7 | 31.6
2^2	64	0	4736	—	192 b	9216 b	2.718	823818	303104	
2^2	144	68	133	$\mathcal{T}$	192 b	18256 b	1.063	29962	19152	27.5 | 15.8
2^3	43	0	3182	—	448 b	5967 b	2.718	371862	136826	
2^3	83	56	141	$\mathcal{T}$	448 b	11695 b	1.078	21119	11703	17.6 | 11.7
2^4	32	0	2368	—	960 b	4320 b	2.718	205928	75776	
2^4	59	58	119	$\mathcal{F}$	960 b	9376 b	1.076	14985	7021	13.7 | 10.8
2^5	26	0	1924	—	1984 b	3442 b	2.717	135937	50024	
2^5	43	50	111	$\mathcal{F}$	1984 b	7301 b	1.085	11198	4773	12.1 | 10.5
2^6	22	0	1628	—	4032 b	2866 b	2.717	97322	35816	
2^6	33	42	108	$\mathcal{F}$	4032 b	5835 b	1.089	8824	3564	11.0 | 10.0
2^7	19	0	1406	—	8128 b	2440 b	2.717	72585	26714	
2^7	26	32	113	$\mathcal{F}$	8128 b	4550 b	1.107	7254	2938	10.0 | 9.1
2^8	16	0	1184	—	16320 b	2020 b	2.717	51469	18944	
2^8	22	30	106	$\mathcal{F}$	16320 b	4028 b	1.114	6139	2332	8.4 | 8.1
2^9	15	0	1110	—	32704 b	1883 b	2.717	45235	16650	
2^9	19	28	101	$\mathcal{F}$	32704 b	3609 b	1.121	5321	1919	8.5 | 8.7
2^{10}	13	0	962	—	65472 b	1609 b	2.717	33974	12506	
2^{10}	17	31	88	$\mathcal{F}$	65472 b	3593 b	1.113	4703	1496	7.2 | 8.4
2^{11}	12	0	888	—	131008 b	1473 b	2.716	28946	10656	
2^{11}	15	27	89	$\mathcal{F}$	131008 b	3155 b	1.126	4208	1335	6.9 | 8.0
2^{12}	11	0	814	—	262080 b	1340 b	2.716	24322	8954	
2^{12}	13	18	106	$\mathcal{F}$	262080 b	2413 b	1.165	3828	1378	6.4 | 6.5
2^{13}	10	0	740	—	524224 b	1207 b	2.716	20099	7400	
2^{13}	12	20	94	$\mathcal{F}$	524224 b	2436 b	1.153	3467	1128	5.8 | 6.6
2^{14}	10	0	740	—	1048512 b	1208 b	2.716	20099	7400	
2^{14}	11	19	92	$\mathcal{F}$	1048512 b	2276 b	1.157	3193	1012	6.3 | 7.3
2^{15}	9	0	666	—	2097088 b	1075 b	2.716	16279	5994	
2^{15}	10	15	100	$\mathcal{F}$	2097088 b	1934 b	1.191	2977	1000	5.5 | 6.0
2^{16}	8	0	592	—	4194240 b	944 b	2.716	12861	4736	
2^{16}	10	22	79	$\mathcal{F}$	4194240 b	2369 b	1.147	2898	790	4.4 | 6.0

Script to produce Table 4.1

```
#!/usr/bin/env sage
RR = RealField(1000)

secbits = 128
pbits = 512
csidhn, csidhm = 74, 5
isz = lambda d: 2*d*csidhm+1    # interval size
sigsize = lambda S, t, u, delta, var = 'O': ceil(1/8 * (0
      + ceil(min(t+u, u*log(t+u,2), t*log(t+u,2)))      # indices of rejections
      + ceil(log(S,2)*(t+u))                            # hash of ephemeral public keys
      + pbits*u                                         # rejected ephemeral public keys
      + t*ceil(log(isz(delta+(var=='F'))**csidhn,2))))  # revealed secret keys
pksize = lambda t, S: ceil(1/8 * (S-1)*pbits)

def Bin(n, p, k):      # Pr[ Bin_n,p >= k ]
    return sum(RR(1) * binomial(n, i) * p**i * (1-p)**(n-i) for i in range(k, n+1))

@cached_function
def joint_minmax_cdf(n, x, y, a, b):
    # Pr that min and max of n independent uniformly random
    # integers in [a;b] satisfy min <= x and max <= y.
    if x < a or y < a: return 0
    if y > b: y = b
    return RR((y-a+1)/(b-a+1))**n - (RR((y-x)/(b-a+1))**n if x < y else 0)

@cached_function
def joint_minmax(n, x, y, a, b):
    # Pr that min and max of n independent uniformly random
    # integers in [a;b] satisfy min = x and max = y.
    F = lambda xx, yy: joint_minmax_cdf(n, xx, yy, a, b)
    return F(x,y) - F(x-1,y) - F(x,y-1) + F(x-1,y-1)

def prob_accept_original(delta, S):
    # sample r from [-(delta+1)*m, (delta+1)*m];
    # reject r and a_c-r outside [-delta*m; +delta*m]
    return (isz(delta) / isz(delta+1)) ** csidhn # entries are independent

def prob_accept_full(delta, S):
    # sample r from [-(delta+1)*m, (delta+1)*m];
    # reject a_c-r outside [-delta*m; +delta*m]
    prob = (isz(delta) / isz(delta+1)) ** csidhn # entries are independent
    prob = 1/S*RR(1) + (S-1)/S*prob # can always reveal r
    return prob

def prob_accept_truncate(delta, S):
    prob = RR(0)
    for x in range(-csidhm, csidhm + 1):
        for y in range(x, csidhm + 1):
            # Pr[min and max coeffs of S-1 secret keys are x and y]
            weight = joint_minmax(S-1, x, y, -csidhm, +csidhm)
            # sample from [min(0,x)-delta*m, max(0,y)+delta*m];
            # reject outside [-delta*m; +delta*m]
            prob += weight * isz(delta) / (isz(delta) + max(0,y) - min(0,x))
    return prob ** csidhn # entries are independent

@cached_function
def max_u(t, S): # largest possible u for given S,t
    u, F = 1, lambda u: Bin(t+u, 1/S, t)
    while F(u) <= 2**-secbits: u *= 2
    lo, hi = u//2, u+1
    while hi - lo > 1:
        m = (lo+hi+1)//2
        if F(m) <= 2**-secbits: lo = m
        else: hi = m
    return lo

def prob_sign(t, u, sigma):
    return Bin(t+u, sigma, t)
```

```
def exp_csidhs_sign(t, u, delta, S, prob):
    pr_single = prob(delta, S)
    pr_all = prob_sign(t, u, pr_single)
    return (t+u) * delta / pr_all

def csidhs_verif(t, delta):
    return t * delta


for s in range(1, 17):
    S = 2**s

    t = ceil(secbits/log(S,2)) - 1
    last_umax = -1

    best_time, no_progress = 1./0, 0
    while True:

        if no_progress >= max(16, t/8): break #XXX hack
        t += 1

        if Bin(t + 4*t, 1/S, t) < 2**-secbits: umax = 4*t #XXX hack
        else: umax = max_u(t,S)

        no_progress_inner = True

        for variant in ('OTF' if t == ceil(secbits/log(S,2)) else 'TF'):

            for u in ([0] if variant == 'O' else reversed(range(last_umax+1, umax+1))):

                print(log(S,2), variant, t, u, no_progress, file=sys.stderr)

                prob = {'O': prob_accept_original,
                        'F': prob_accept_full,
                        'T': prob_accept_truncate}[variant]

                @cached_function
                def f(x): return exp_csidhs_sign(t, u, x, S, prob)

                if variant == 'O':
                    delta = csidhn * t
                else:
                    _, delta = find_local_minimum(f, 1, 2**24, tol=1)
                    delta = min((floor(delta), ceil(delta)), key = f)

                if f(delta) < best_time:
                    print(('logS={:2d} t={:3d} u={:3d} delta={:4d} {} ~> ' \
                           'pksize={:9,d}b sigsize={:7,d}b ' \
                           'tries={:8.6f} signCSIDHs={:9,d} verifCSIDHs={:9,d}') \
                           .format(log(S,2), t, u, delta, variant,
                                   pksize(t,S),
                                   sigsize(S, t, u, delta, variant),
                                   float(1 / prob_sign(t, u, prob(delta, S))),
                                   round(f(delta)),
                                   csidhs_verif(t, delta))
                        )
                    best_time = f(delta)
                    no_progress_inner = False

        no_progress = no_progress + 1 if no_progress_inner else 0

        last_umax = umax
```

Chapter 5

Rational isogenies from irrational endomorphisms

This chapter is for all practical purposes identical to the paper *Rational isogenies from irrational endomorphisms* [CPV20] authored jointly with Wouter Castryck and Frederik Vercauteren, which was published at Eurocrypt 2020.

5.1 — Introduction

In this chapter, we give a polynomial-time algorithm to compute a connecting $\mathcal{O}$-ideal between two supersingular elliptic curves over $\mathbb{F}_p$ with common $\mathbb{F}_p$-endomorphism ring $\mathcal{O}$, from a description of their full endomorphism rings. This algorithm provides a reduction of the security of the CSIDH cryptosystem to the problem of computing endomorphism rings of supersingular elliptic curves. A similar reduction for SIDH appeared at Asiacrypt 2016, but relies on totally different techniques. Furthermore, we also show that any supersingular elliptic curve constructed using the complex-multiplication method can be located precisely in the supersingular isogeny graph by explicitly deriving a path to a known base curve. This result prohibits the use of such curves as a building block for a hash function into the supersingular isogeny graph.

Isogeny-based cryptography is founded on the hardness of computing an isogeny between two given isogenous elliptic curves over a finite field $\mathbb{F}_q$, which appears to remain hard even for quantum computers. The currently most efficient instantiations can be broadly classified into two families, known as SIDH [JD11] and CSIDH [Cas+18], depending on which supersingular elliptic curves and connecting isogenies are being used.

SIDH works in the full supersingular ℓ-isogeny graph, i.e., one considers the graph consisting of all (isomorphism classes of) supersingular elliptic curves defined over $\overline{\mathbb{F}_p}$ for a specifically chosen prime p and connecting isogenies of small prime degree ℓ. The vertices of this graph are the j-invariants of the isomorphism classes and are all contained in $\mathbb{F}_{p^2}$. Finding a path between two given vertices $j(E_1)$ and $j(E_2)$ is equivalent to constructing an isogeny between E_1 and E_2 whose degree is a power of ℓ.

The full endomophism ring of a supersingular elliptic curve is a maximal order in a quaternion algebra. Kohel, Lauter, Petit and Tignol [KLPT14] showed that the above path-finding problem can be solved in (heuristically) expected polynomial time when given the endomorphism rings of E_1 and E_2; we refer to this algorithm as "KLPT" (see Section 2.4.9). Galbraith, Petit, Shani and Ti [GPST16] later extended the KLPT algorithm specifically for the SIDH setting and showed that knowledge of the endomorphism rings of E_1 and E_2 suffices to break SIDH. Results by Eisenträger, Hallgren, Lauter, Morrison and Petit [Eis+18] show that finding a path in the isogeny graph is essentially equivalent to computing endomorphism rings.

CSIDH, on the other hand, restricts the isogeny graph under consideration to supersingular elliptic curves and isogenies *defined over* $\mathbb{F}_p$ and mimics Couveignes' construction of a "hard homogeneous space". In particular, if E is a supersingular elliptic curve over $\mathbb{F}_p$, then its ring of $\mathbb{F}_p$-rational endomorphisms is an imaginary quadratic order $\mathcal{O} \subseteq \mathbb{Q}(\sqrt{-p})$. The letter C in "CSIDH" refers to the commutativity of $\mathcal{O}$, which gives rise to an action of the (commutative) ideal-class group $\mathrm{cl}(\mathcal{O})$ on the set of supersingular elliptic curves over $\mathbb{F}_p$ having $\mathcal{O}$ as their ring of $\mathbb{F}_p$-rational endomorphisms; see Section 2.4.5. This class-group action immediately lends itself to several cryptographic primitives such as identification, non-interactive key agreement, and even signature schemes.

5.1.1 – Contributions. Our first contribution reduces the key recovery problem in CSIDH to computing the full endomorphism ring of the target curve, where in many cases even one non-$\mathbb{F}_p$-rational endomorphism suffices. More precisely, given two supersingular elliptic curves E, E' over $\mathbb{F}_p$ with $\mathbb{F}_p$-rational endomorphism ring $\mathcal{O}$, assuming sufficient knowledge of their full endomorphism rings $\mathrm{End}(E)$ and $\mathrm{End}(E')$, we show how to compute in polynomial time an ideal $\mathfrak{a} \subseteq \mathcal{O}$ such that $E' = [\mathfrak{a}]E$. This result can be seen as an analogon of [GPST16] for SIDH, but uses different techniques, and in particular it does not rely on the KLPT algorithm [KLPT14].

Several remarks on this result are in order:

- In CSIDH all curves have *the same known* $\mathbb{F}_p$-rational endomorphism ring $\mathcal{O}$, which therefore does not contain any information specific to E, nor to $[\mathfrak{a}]$. This explains why we require knowledge of at least one endomorphism of E that is not $\mathbb{F}_p$-rational.
- Since both $\mathrm{End}(E_0)$ and $\mathrm{End}(E)$ are assumed to be known, one can run the KLPT algorithm to obtain an isogeny $\alpha\colon E_0 \to E$. However, this isogeny is most likely not $\mathbb{F}_p$-rational and as such does not correspond to the CSIDH private key. It is easy to verify that the isogeny $\beta = \alpha \circ \pi_{E_0} + \pi_E \circ \alpha$, with π the p-power Frobenius endomorphism on the respective curves, is an $\mathbb{F}_p$-rational isogeny[1] from E_0 to E. Note that β can be evaluated efficiently on points of E_0, but it is unclear how to efficiently derive an invertible ideal $\mathfrak{b} \subseteq \mathcal{O}$ whose action on E_0 corresponds to β. Such an ideal $\mathfrak{b}$ is required to break the CSIDH Diffie–Hellman key agreement and other derived protocols, since it is essentially a curve-independent way of specifying an $\mathbb{F}_p$-rational isogeny.
- Our polynomial-time algorithm returns an ideal $\mathfrak{a}$ whose norm is not necessarily smooth. To efficiently compute the action of $[\mathfrak{a}]$ therefore requires an extra smoothing step, which obtains an ideal of smooth norm in the ideal class $[\mathfrak{a}]$. This smoothing step is standard and consists of a combination of a class-group computation and lattice reduction to solve an instance of the approximate closest-vector problem (CVP). The class-group computation requires subexponential time using classical computers [HM89], but runs in polynomial time on a quantum computer [Kit96]. Using the BKZ algorithm [SE94], one can solve the CVP problem up to a subexponential approximation factor in subexponential time. This last step therefore implies that asymptotically, the smoothing step requires subexponential time. However, we note that for some *practical* instantiations of CSIDH, solving the approximate CVP problem can be done fairly efficiently [BKV19].

Our second contribution is motivated by an important open problem in isogeny-based cryptography, namely how to hash into a supersingular isogeny graph without revealing a path to a known base curve. This problem remains open both in the SIDH (full isogeny graph) and the CSIDH ($\mathbb{F}_p$-rational isogeny graph) setting. The hash function introduced by Charles, Goren and

[1] Unless $\beta = 0$.

Lauter [CLG09] can be used to hash any string into the supersingular isogeny graph, but by construction, the hash function itself leaks an isogeny path from a base curve. To illustrate the issue, we can compare with the standard elliptic-curve discrete-logarithm setting: The equivalent of the CGL construction would start from the public base point $P \in E(\mathbb{F}_q)$ and construct a point Q by multiplying P with a scalar computed deterministically from the message. As such, anyone would know the discrete logarithm of Q with respect to P, which voids cryptographic applications relying on the assumption that the relationship between Q and P cannot be discovered. To remedy this, elliptic-curve cryptosystems instead hash to curve points using maps like Elligator [BHKL13], which computes a point directly without passing through a scalar first, but an equivalent of these constructions in isogeny-based cryptography is not known.

Besides the random-walk approach à la CGL, it is also possible to generate supersingular elliptic curves using the complex-multiplication (CM) method [Brö09]. It is therefore natural to wonder whether CM can be useful to hash into the supersingular isogeny graph, and in particular, whether finding paths between the resulting curves could be computationally hard. Our second result squashes this hope by locating these curves (and therefore also a path to a base curve) in the supersingular isogeny graph, in a surprisingly explicit manner (see Theorem 5.25(iii) for the exact statement).

The remainder of the chapter is organized as follows. In Section 5.2 we recall the necessary mathematical background. In Section 5.3 we introduce the notion of twisting endomorphisms and explain their relation to $\mathbb{F}_p$-rational isogenies. Section 5.4 describes our new algorithm to compute a connecting ideal between two supersingular elliptic curves over $\mathbb{F}_p$ given their endomorphism rings and argues that (at least classically) our approach appears to be optimal. Finally, Section 5.5 shows how to locate supersingular elliptic curves constructed via CM in the isogeny graph, by explicitly deriving a path to a known starting curve.

Acknowledgements. We thank Benjamin Wesolowski, Robert Granger, Christophe Petit, and Ben Smith for interesting discussions regarding this work, and Lixia Luo for pointing out an error in an earlier version of Lemma 5.21, as well as a few smaller mistakes. Thanks to Daniel J. Bernstein for providing key insights regarding the proof of Lemma 5.23.

5.2 — Preliminaries

In this section we recall the required mathematical background and fix notation. Our focus lies on supersingular elliptic curves over finite prime fields $\mathbb{F}_p$, although much of what follows readily generalizes to arbitrary elliptic curves over arbitrary finite fields. Some of the observations below seem new.

For ease of exposition, we shall assume $p \geq 5$ throughout, noting that this is not necessarily a requirement for all of the statements.

5.2.1 – Quadratic twisting. For each odd prime number p we fix a non-square $\xi \in \mathbb{F}_p$ along with a square root $\sqrt{\xi} \in \mathbb{F}_{p^2} \setminus \mathbb{F}_p$; if $p \equiv 3 \pmod 4$ then our default choice is $\xi = -1$ and we write $\mathbf{i} = \sqrt{-1} \in \mathbb{F}_{p^2}$. For an elliptic curve $E\colon y^2 = f(x)$ over $\mathbb{F}_p$ defined by some squarefree cubic polynomial $f(x) \in \mathbb{F}_p[x]$, we call the curve $E^t\colon \xi^{-1}y^2 = f(x)$ the *quadratic twist* of E over $\mathbb{F}_p$. The map $\tau_E\colon E \to E^t$, $(x, y) \mapsto (x, \sqrt{\xi} \cdot y)$ is a non-$\mathbb{F}_p$-rational isomorphism. From $\sqrt{\xi}^p = -\sqrt{\xi}$ one easily sees that

$$\tau_E \circ \pi_E = -\pi_{E^t} \circ \tau_E, \tag{5.1}$$

with π_E and π_{E^t} the respective Frobenius endomorphisms of E and E^t.

It can exceptionally happen that our definition of the quadratic twist is a trivial twist in the sense of [Sil09, § X.2]:

Lemma 5.1. *An elliptic curve $E/\mathbb{F}_p$ is $\mathbb{F}_p$-isomorphic to its quadratic twist E^t if and only if $p \equiv 3 \pmod 4$ and $j(E) = 1728$.*

Proof. After an $\mathbb{F}_p$-isomorphism, we can assume $E\colon y^2 = x^3 + Ax + B$ with $A, B \in \mathbb{F}_p$ satisfying $4A^3 + 27B^2 \neq 0$. Then its quadratic twist is $\mathbb{F}_p$-isomorphic to $y^2 = x^3 + A\xi^2 x + B\xi^3$ for some non-square ξ. According to [Sil09, Prop. III.3.1] this curve is $\mathbb{F}_p$-isomorphic to E if and only if $A\xi^2 = Au^4$ and $B\xi^3 = Bu^6$ for some $u \in \mathbb{F}_p \backslash \{0\}$. This holds if and only if $B = 0$ and ξ^2 is a fourth power, from which the lemma follows. □

5.2.2 – Hard homogeneous spaces on supersingular curves. Fix a prime number $p \geq 5$ and consider the imaginary quadratic number field $K = \mathbb{Q}(\sqrt{-p})$ along with its maximal order $\mathcal{O}_K$. If E is a supersingular elliptic curve defined over $\mathbb{F}_p$, then its ring $\mathrm{End}_p(E)$ of $\mathbb{F}_p$-rational endomorphisms admits an isomorphism to an order $\mathcal{O} \subseteq K$, under which π_E is mapped to $\sqrt{-p}$. In particular, $\mathcal{O}$ always contains the subring $\mathbb{Z}[\sqrt{-p}]$, hence if $p \equiv 1 \pmod 4$ then the only possibility is $\mathcal{O} = \mathcal{O}_K = \mathbb{Z}[\sqrt{-p}]$, while if $p \equiv 3 \pmod 4$ then either $\mathcal{O} = \mathbb{Z}[\sqrt{-p}]$ or $\mathcal{O} = \mathcal{O}_K = \mathbb{Z}[(1+\sqrt{-p})/2]$. Recall from Definition 2.54 that $\mathscr{Ell}_p(\mathcal{O})$ denotes the set of $\mathbb{F}_p$-isomorphism classes of (necessarily supersingular) elliptic curves having endomorphism ring $\mathcal{O}$.

Remark 5.2. *If $p \equiv 3 \pmod 4$, then the $\mathbb{F}_p$-endomorphism ring of a supersingular elliptic curve $E/\mathbb{F}_p$ is determined by its 2-torsion; see [DG16]: either we have $\#E(\mathbb{F}_p)[2] = 2$, in which case $E \in \mathscr{Ell}_p(\mathbb{Z}[\sqrt{-p}])$, or $\#E(\mathbb{F}_p)[2] = 4$, in which case $E \in \mathscr{Ell}_p(\mathbb{Z}[(1+\sqrt{-p})/2])$.*

Every such order $\mathcal{O}$ comes equipped with its (ideal-)class group $\mathrm{cl}(\mathcal{O})$, which consists of invertible ideals modulo non-zero principal ideals; the class of an invertible ideal $\mathfrak{a} \subseteq \mathcal{O}$ is denoted by $[\mathfrak{a}]$. The number of elements of $\mathrm{cl}(\mathcal{O})$ is called the class number and denoted by $h(\mathcal{O})$.

Lemma 5.3. *If $p \equiv 3 \pmod 4$ then $h(\mathcal{O})$ is odd, while if $p \equiv 1 \pmod 4$ then $\mathrm{cl}(\mathcal{O})$ has a unique element of order 2, in particular $h(\mathcal{O})$ is even.*

Proof. This follows from genus theory [Cox13]. □

Through the map

$$\mathrm{cl}(\mathcal{O}) \times \mathscr{Ell}_p(\mathcal{O}) \longrightarrow \mathscr{Ell}_p(\mathcal{O})\colon \quad ([\mathfrak{a}], E) \longmapsto [\mathfrak{a}]E := E/E[\mathfrak{a}]$$

from Theorem 2.55, the class group acts in a free and transitive manner on the set $\mathscr{Ell}_p(\mathcal{O})$. Here $E[\mathfrak{a}]$ denotes the intersection of the kernels of all elements of $\mathfrak{a}$ interpreted as endomorphisms of E; to compute this intersection it suffices to consider a set of generators of $\mathfrak{a}$.

Ignoring constructive issues, this group action (for large enough p) is conjectured to turn $\mathscr{Ell}_p(\mathcal{O})$ into a "hard homogeneous space", in which the following problems are assumed to be computationally infeasible:

Definition 5.4.
(Vectorization problem.) *Given $E, E' \in \mathscr{Ell}_p(\mathcal{O})$, find the ideal class $[\mathfrak{a}] \in \mathrm{cl}(\mathcal{O})$ for which $E' = [\mathfrak{a}]E$.*
(Parallelization problem.) *Given $E, E', E'' \in \mathscr{Ell}_p(\mathcal{O})$, find the curve $[\mathfrak{a}][\mathfrak{b}]E$ where $[\mathfrak{a}], [\mathfrak{b}] \in \mathrm{cl}(\mathcal{O})$ are such that $E' = [\mathfrak{a}]E$ and $E'' = [\mathfrak{b}]E$.*

The hardness of parallelization naturally relates to the security of the Diffie–Hellman-style key exchange protocol built from the above group action: starting from a publicly known base curve $E \in \mathcal{E}\ell\ell_p(\mathcal{O})$, the two parties Alice and Bob secretly sample $[\mathfrak{a}]$ resp. $[\mathfrak{b}]$ from $\mathrm{cl}(\mathcal{O})$, compute $[\mathfrak{a}]E$ resp. $[\mathfrak{b}]E$, and publish the result. The shared secret is then $[\mathfrak{a}][\mathfrak{b}]E$, which Alice computes as $[\mathfrak{a}]([\mathfrak{b}]E)$ and which Bob computes as $[\mathfrak{b}]([\mathfrak{a}]E)$. Clearly, in order to solve the parallelization problem, it suffices to solve the vectorization problem. On a quantum computer, the converse holds as well; see Chapter 6.

For later use we recall the following rule, which was pointed out in Remark 3.5, albeit very briefly and without proof (see also [Arp+19, Prop. 3.31]).

Lemma 5.5. *For all $[\mathfrak{a}] \in \mathrm{cl}(\mathcal{O})$ and all $E \in \mathcal{E}\ell\ell_p(\mathcal{O})$ we have $[\mathfrak{a}]^{-1}E = ([\mathfrak{a}]E^t)^t$.*

Proof. It is convenient to assume that $\mathfrak{a}$ is generated by elements of $\mathbb{Z}[\sqrt{-p}]$, which can be done without loss of generality by scaling with an appropriate principal ideal if needed. We claim that the composition

$$E \xrightarrow{\tau_E} E^t \longrightarrow\!\!\!\!\!\rightarrow E^t/E^t[\mathfrak{a}] = [\mathfrak{a}]E^t \xrightarrow{\tau_{[\mathfrak{a}]E^t}} ([\mathfrak{a}]E^t)^t$$

is an $\mathbb{F}_p$-rational isogeny whose kernel equals the ideal $\bar{\mathfrak{a}}$ obtained from $\mathfrak{a}$ by complex conjugation. This claim implies the lemma because $\mathfrak{a}\bar{\mathfrak{a}}$ is the principal ideal generated by $\mathrm{N}(\mathfrak{a})$.

Let φ be the middle isogeny $E^t \twoheadrightarrow E^t/E^t[\mathfrak{a}]$. Two applications of (5.1) yield

$$\pi_{([\mathfrak{a}]E^t)^t} \circ (\tau_{[\mathfrak{a}]E^t} \circ \varphi \circ \tau_E) = (\tau_{[\mathfrak{a}]E^t} \circ \varphi \circ \tau_E) \circ \pi_E,$$

implying the $\mathbb{F}_p$-rationality. One verifies that $a + b\sqrt{-p} \in \mathfrak{a}$ if and only if $a + b\pi_{E^t}$ vanishes on $\ker(\varphi)$, which holds if and only if $a - b\pi_E$ vanishes on $\ker(\varphi \circ \tau_E)$, from which it follows that $\ker(\tau_{[\mathfrak{a}]E^t} \circ \varphi \circ \tau_E) = \ker(\varphi \circ \tau_E) = E[\bar{\mathfrak{a}}]$. □

5.2.3 – CSIDH. CSIDH (Chapter 3) is an efficient instantiation of the more general supersingular hard-homogeneous-spaces construction described in the previous section. Recall that we let $n \in \mathbb{Z}_{\geq 1}$ and consider a prime p of the form $p = 4\ell_1\ell_2 \cdots \ell_n - 1$, where the ℓ_i's are distinct odd prime numbers. This implies $p \equiv 3 \pmod 8$, so a priori there are two options for $\mathcal{O}$, namely $\mathbb{Z}[\sqrt{-p}]$ and the maximal order $\mathcal{O}_K = \mathbb{Z}[(1+\sqrt{-p})/2]$. CSIDH chooses the former option. Recall from Remark 5.2 that this corresponds to supersingular elliptic curves over $\mathbb{F}_p$ having a unique $\mathbb{F}_p$-rational point of order 2.

Remark 5.6. *In volcano terminology (see Section 2.5.1), the set $\mathcal{E}\ell\ell_p(\mathbb{Z}[\sqrt{-p}])$ is the floor, and the set $\mathcal{E}\ell\ell_p(\mathbb{Z}[(1+\sqrt{-p})/2])$ is the surface of the corresponding 2-volcano. We stress that CSIDH can be set up equally well on the surface, although a convenient feature of the floor is that each $E \in \mathcal{E}\ell\ell_p(\mathbb{Z}[\sqrt{-p}])$ is $\mathbb{F}_p$-isomorphic to a Montgomery curve $E_A\colon y^2 = x^3 + Ax^2 + x$ for a unique coefficient $A \in \mathbb{F}_p$; furthermore, the coefficient defining E^t is then given by $-A$.*

The prime p was chosen such that the primes $\ell_1, \ell_2, \ldots, \ell_n$ exhibit particularly easy splitting behaviour in $\mathbb{Z}[\sqrt{-p}]$, namely

$$(\ell_i) = (\ell_i, \sqrt{-p} - 1)(\ell_i, \sqrt{-p} + 1). \tag{5.2}$$

We refer to the respective factors, which are complex conjugates of each other, by $\mathfrak{l}_i$ and $\overline{\mathfrak{l}_i}$. If we define $\ell_0 := 4$ then (5.2) also applies to $i = 0$, so we can similarly define $\mathfrak{l}_0$ and $\overline{\mathfrak{l}_0}$. All these ideals are clearly invertible, so we can consider their classes $[\mathfrak{l}_i]$ and $[\overline{\mathfrak{l}_i}] = [\mathfrak{l}_i]^{-1}$ inside $\mathrm{cl}(\mathcal{O})$. Although this is not known in general, it seems likely that the $[\mathfrak{l}_i]$'s together generate the entire class group.

Example 5.7. *The concrete instantiation CSIDH-512 from [Cas+18] has $n = 74$, where $\ell_1, \ell_2, \ldots, \ell_{73}$ are the odd primes up to 373 and where $\ell_{74} = 587$. This results in a 511-bit prime p. The structure of $\mathrm{cl}(\mathbb{Z}[\sqrt{-p}])$ was computed by Beullens, Kleinjung and Vercauteren [BKV19], whose results show that $[\mathfrak{l}_1] = [(3, \sqrt{-p} - 1)]$ is in fact a generator.*

The basic idea is then to let Alice and Bob choose their secrets as

$$[\mathfrak{a}] = [\mathfrak{l}_1]^{a_1}[\mathfrak{l}_2]^{a_2} \cdots [\mathfrak{l}_n]^{a_n} \qquad \text{resp.} \qquad [\mathfrak{b}] = [\mathfrak{l}_1]^{b_1}[\mathfrak{l}_2]^{b_2} \cdots [\mathfrak{l}_n]^{b_n},$$

for exponent vectors $(a_1, a_2, \ldots, a_n)$ and $(b_1, b_2, \ldots, b_n)$ sampled randomly from some bounded subset of $\mathbb{Z}^n$, for instance uniformly from a (discrete) hypercube $\{-m, ..., m\}^n$ of cardinality $(2m+1)^n \approx h(\mathbb{Z}[\sqrt{-p}]) \approx \sqrt{p}$. The resulting public keys and shared secret are then computed using $|a_1| + \ldots + |a_n|$ resp. $|b_1| + \ldots + |b_n|$ repeated actions of $[\mathfrak{l}_i]$ or $[\mathfrak{l}_i]^{-1} = [\overline{\mathfrak{l}_i}]$. If $E \in \mathcal{El}_p(\mathbb{Z}[\sqrt{-p}])$ then the subgroups

$$E[\mathfrak{l}_i] = \{\, P \in E[\ell_i] \mid \pi_E(P) = P \,\} = E(\mathbb{F}_p)[\ell_i]$$
$$E[\overline{\mathfrak{l}_i}] = \{\, P \in E[\ell_i] \mid \pi_E(P) = -P \,\}$$

consist of points having $\mathbb{F}_p$-rational x-coordinates; therefore, these actions are easy to evaluate using low-degree Vélu-type formulas and involving only arithmetic in $\mathbb{F}_p$.

We also point out the following class group relations:[2]

Lemma 5.8. *In $\mathrm{cl}(\mathbb{Z}[\sqrt{-p}])$, we have*

$$[\mathfrak{l}_1][\mathfrak{l}_2] \cdots [\mathfrak{l}_n] = [\overline{\mathfrak{l}_0}] \neq [1] \qquad \textit{and} \qquad [\mathfrak{l}_1]^3[\mathfrak{l}_2]^3 \cdots [\mathfrak{l}_n]^3 = [1].$$

Proof. One easily verifies that

$$\mathfrak{l}_1\mathfrak{l}_2 \cdots \mathfrak{l}_n = \Big(\frac{p+1}{4}, \sqrt{-p} - 1\Big) \qquad \text{and} \qquad \mathfrak{l}_0\mathfrak{l}_1\mathfrak{l}_2 \cdots \mathfrak{l}_n = \big(\sqrt{-p} - 1\big).$$

The latter identity implies that $[\mathfrak{l}_1][\mathfrak{l}_2] \cdots [\mathfrak{l}_n] = [\mathfrak{l}_0]^{-1} = [\overline{\mathfrak{l}_0}]$, while the former shows that $[\mathfrak{l}_1][\mathfrak{l}_2] \cdots [\mathfrak{l}_n]$ is an element of order 3. Indeed, it represents a non-trivial ideal class because $\mathbb{Z}[\sqrt{-p}]$ contains no elements of norm $(p+1)/4$, while its order divides 3 since

$$\Big(\frac{p+1}{4}, \sqrt{-p} - 1\Big)\mathcal{O}_K = \frac{1+\sqrt{-p}}{2}\mathcal{O}_K,$$

i.e., it belongs to the kernel of the group homomorphism

$$\mathrm{cl}(\mathcal{O}) \longrightarrow \mathrm{cl}(\mathcal{O}_K), \;\; \mathfrak{a} \longmapsto \mathfrak{a}\mathcal{O}_K$$

which is 3-to-1 by [Con, Thm. 5.2]. □

Note that this allows for reduction of the secret exponent vectors of Alice and Bob modulo $(3, 3, \ldots, 3)$. It also shows that the action of $[\mathfrak{l}_1][\mathfrak{l}_2] \cdots [\mathfrak{l}_n]$ can be evaluated using a single application of $[\overline{\mathfrak{l}_0}] = [(4, \sqrt{-p} + 1)]$. The latter step can be taken using an isogeny of degree 4, or using a composition of two isogenies of degree 2, which necessarily makes us pass through the surface.

[2] After we posted a version of the paper this chapter is based upon online, we learned that this was observed independently and quasi-simultaneously in [OT20], with a more elaborate discussion.

5.2.4 – The full endomorphism ring. The "full" endomorphism ring of a supersingular elliptic curve, as opposed to merely the $\mathbb{F}_p$-rational endomorphisms, plays a fundamental role in the theory of supersingular isogeny graphs.

We recall the following facts from Section 2.4.6: An elliptic curve E is supersingular if and only if $\mathrm{End}(E)$ is non-commutative. In that case, $\mathrm{End}(E)$ embeds as a maximal order into a certain quaternion algebra $B_{p,\infty}$ ramified at p and infinity, which is unique up to isomorphism. Concretely, $B_{p,\infty}$ can be constructed as a four-dimensional $\mathbb{Q}$-algebra of the form $\mathbb{Q} \oplus \mathbb{Q}\mathbf{i} \oplus \mathbb{Q}\mathbf{j} \oplus \mathbb{Q}\mathbf{ij}$, subject to the multiplication rules $\mathbf{i}^2 = -q$, $\mathbf{j}^2 = -p$, and $\mathbf{ji} = -\mathbf{ij}$, for some positive integer q that depends on p. In the common case that $p \equiv 3 \pmod 4$, we can and will use $q = 1$. (Thus $B_{p,\infty}$ may be viewed as two imaginary quadratic fields "glued together" non-commutatively.) We certainly cannot stress enough that the embedding $\mathrm{End}(E) \hookrightarrow B_{p,\infty}$ is *extremely non-unique*; in fact, there are always infinitely many choices, and usually none of them sticks out as being particularly natural.

The notions of dual, degree, and trace of endomorphisms carry over to $B_{p,\infty}$: Taking the dual corresponds to conjugation, which maps $\alpha = a + b\mathbf{i} + c\mathbf{j} + d\mathbf{ij}$ to $\overline{\alpha} = a - b\mathbf{i} - c\mathbf{j} - d\mathbf{ij}$. The degree turns into $\mathrm{N}(\alpha) = \alpha\overline{\alpha} = a^2 + b^2q + c^2p + d^2qp$, and the trace is simply $\mathrm{tr}(\alpha) = \alpha + \overline{\alpha} = 2a$. Moreover, the trace yields a symmetric bilinear map $\langle \alpha, \beta \rangle = \mathrm{tr}(\overline{\alpha}\beta)$ on $B_{p,\infty}$, with respect to which the basis $1, \mathbf{i}, \mathbf{j}, \mathbf{ij}$ is orthogonal. With this, finding *an* embedding $\mathrm{End}(E) \hookrightarrow B_{p,\infty}$ when being given rational maps that span $\mathrm{End}(E)$ in some computationally effective way is easy: A variant of Schoof's point counting algorithm [Sch85] can be used to compute traces of endomorphisms, and thereby the map $\langle \cdot, \cdot \rangle$, which can then be used in the Gram–Schmidt process to compute an orthogonal basis of the given endomorphism ring. Once the basis is orthogonal, some norm computations are necessary to align the given maps with the algebraic properties of the abstract quaternion representation. See [Eis+18, § 5.4] for details. We will commonly use the $\mathbb{Q}$-basis $(1, \mathbf{i}, \mathbf{j}, \mathbf{ij})$ in the forthcoming algorithms to compute with $\mathrm{End}(E)$; the isomorphism to the corresponding rational maps of curves will be made explicit whenever it is realized computationally.

One reason why the endomorphism rings are interesting for cryptographic applications is because they contain all the information necessary to construct an isogeny between two curves: Given $\mathrm{End}(E)$ and $\mathrm{End}(E')$, it is easy to find a *connecting ideal* $\mathcal{I}$ between them; that is, a lattice in $B_{p,\infty}$ that is a left ideal of $\mathrm{End}(E)$ and a right ideal of $\mathrm{End}(E')$. For example, the choice $\mathcal{I} = \mathcal{Q}\mathcal{Q}'$ from Proposition 2.64 works. The intersection of all kernels of endomorphisms contained in (a scaled, integral equivalent of) this ideal is a finite subgroup determining a separable isogeny $E \longrightarrow E'$. Recall (Proposition 2.46) that the codomain curve of the isogeny given by such a left ideal of $\mathrm{End}(E)$ only depends on the left-ideal *class* of $\mathcal{I}$: This is what the Kohel–Lauter–Petit–Tignol algorithm (Section 2.4.9) exploits to find a *smooth-degree*, hence efficiently computable, isogeny between E and E' given their endomorphism rings.

Since we are working with supersingular elliptic curves defined over $\mathbb{F}_p$, our endomorphism rings — maximal orders in $B_{p,\infty}$ — will (by Section 2.4.1) always contain a copy of the Frobenius order $\mathbb{Z}[\sqrt{-p}] \cong \mathbb{Z}[\pi_E] \subseteq \mathrm{End}_p(E)$. It thus makes sense to fix the image of the Frobenius endomorphism π_E when embedding $\mathrm{End}(E)$ into $B_{p,\infty}$ once and for all: We will always assume that π_E is mapped to $\mathbf{j}$.

5.3 — Twisting endomorphisms

As before, we focus on the case of finite fields $\mathbb{F}_p$ with $p \geq 5$ prime.

Definition 5.9. *Let E be an elliptic curve defined over $\mathbb{F}_p$. An endomorphism $\alpha \in \mathrm{End}(E)$ is called a* twisting endomorphism *of E if*

$$\alpha \circ \pi_E = -\pi_E \circ \alpha.$$

(Note that E must necessarily be supersingular for this to be possible.)

Lemma 5.10. *Let E be an elliptic curve defined over $\mathbb{F}_p$. The non-zero twisting endomorphisms of E are precisely the elements of $\mathrm{End}(E)$ that are purely imaginary over $\mathrm{End}_p(E)$.*

Proof. Write $\alpha = a + b\mathbf{i} + c\mathbf{j} + d\mathbf{ij}$ with $a, b, c, d \in \mathbb{Q}$; then using the fact that π_E is mapped to $\mathbf{j}$, the equality $\alpha \circ \pi_E = -\pi_E \circ \alpha$ implies $a = c = 0$. □

Lemma 5.11. *Twisting endomorphisms have kernels defined over $\mathbb{F}_p$. (Thus they always equal either the zero map or an $\mathbb{F}_p$-isogeny followed by an isomorphism.)*

Proof. Since $\pi_E^{-1}(\ker(\alpha)) = \ker(\alpha \circ \pi_E) = \ker(-\pi_E \circ \alpha) = \ker(\alpha)$, the subgroup $\ker(\alpha)$ is stable under the action of $\mathrm{Gal}(\overline{\mathbb{F}_p}/\mathbb{F}_p)$, hence $\mathbb{F}_p$-rational. □

Lemma 5.12. *Let E be an elliptic curve as above and let α be a non-zero twisting endomorphism of E. Then $\tau_E \circ \alpha \colon E \to E^t$ is an $\mathbb{F}_p$-rational isogeny of degree $\mathrm{N}(\alpha)$.*

Proof. Since τ_E is an isomorphism we have $\deg(\tau_E \circ \alpha) = \deg(\alpha) = \mathrm{N}(\alpha)$, so it remains to prove the $\mathbb{F}_p$-rationality, which follows from

$$\tau_E \circ \alpha \circ \pi_E = -\tau_E \circ \pi_E \circ \alpha = \pi_{E^t} \circ \tau_E \circ \alpha$$

where the last equality uses that $\sqrt{\xi} \in \mathbb{F}_{p^2} \setminus \mathbb{F}_p$ and therefore $\sqrt{\xi}^p = -\sqrt{\xi}$. □

5.4 — Isogenies from known endomorphisms

In this section, we describe how to find a connecting ideal between two supersingular elliptic curves over $\mathbb{F}_p$ given their full endomorphism rings.

The basic idea behind our approach is to exploit the symmetry of the isogeny graph over $\mathbb{F}_p$ with respect to quadratic twisting; cf. Lemma 5.5: Intuitively, the distance between a curve and its quadratic twist tells us where in the graph it is located, and combining this information for two curves allows finding the distance between them. See Figure 5.1 below for an illustration.

In more mathematical terms, the "distance" between E and its quadratic twist corresponds to an invertible ideal $\mathfrak{a} \subseteq \mathcal{O}$ that connects E to E^t, i.e., satisfies $[\mathfrak{a}]E = E^t$. We will show in Algorithm 5.1 how to find such an ideal, given the full endomorphism ring of E. Subsequently, given two arbitrary supersingular elliptic curves E, E' with the same $\mathbb{F}_p$-endomorphism ring $\mathcal{O}$ together with such a "twisting ideal" for each of them, Algorithm 5.2 can be used to find a connecting ideal from E to E', i.e., an invertible ideal $\mathfrak{c} \subseteq \mathcal{O}$ such that $[\mathfrak{c}]E = E'$.

The following lemma shows the relationship between ideals in $\mathrm{End}_p(E)$ and $\mathrm{End}(E)$ that determine the same subgroup; it is of crucial significance for the forthcoming algorithms.

Lemma 5.13. *Let E be a supersingular elliptic curve over $\mathbb{F}_p$. Consider a non-zero ideal $\mathfrak{c} \subseteq \mathrm{End}_p(E)$ and a non-zero left ideal $\mathcal{I} \subseteq \mathrm{End}(E)$ such that the corresponding subgroups $E[\mathcal{I}]$ and $E[\mathfrak{c}]$ are equal. Then $\mathcal{I} \cap \mathrm{End}_p(E) = \pi_E^k \mathfrak{c}$ for some $k \in \mathbb{Z}$.*[3]

[3] One could handle the purely inseparable part — powers of π_E — in a unified way by working with scheme-theoretic kernels. Since this issue is only tangential to our work, we will for simplicity avoid this technical complication and deal with π_E explicitly.

Proof. Following [Wat69, Thm. 4.5], we know that for every order $\mathcal{O}$ which can arise as an endomorphism ring, every ideal of $\mathcal{O}$ is a kernel ideal, and thus

$$\mathcal{I} = \{\gamma \in \mathrm{End}(E) \mid \ker(\gamma) \supseteq E[\mathcal{I}]\} \cdot \pi_E^r$$
$$\mathfrak{c} = \{\gamma \in \mathrm{End}_p(E) \mid \ker(\gamma) \supseteq E[\mathfrak{c}]\} \cdot \pi_E^s$$

with non-negative integers $r, s \in \mathbb{Z}$. Now $E[\mathcal{I}] = E[\mathfrak{c}]$ by assumption, hence it follows that $\mathcal{I} \cap \mathrm{End}_p(E) = \pi_E^{r-s}\mathfrak{c}$, which shows the claim. □

5.4.1 – The algorithm. Throughout this section, we write $\mathcal{O}_E := \mathrm{End}_p(E)$ for brevity.

Recall from Section 5.2.4 that we assume $\mathrm{End}(E)$ is represented as a maximal order in $B_{p,\infty}$ with respect to the $1, \mathbf{i}, \mathbf{j}, \mathbf{ij}$ basis, and such that the Frobenius endomorphism π_E is mapped to $\mathbf{j} \in B_{p,\infty}$ under the embedding.

We start off with an algorithm to find an ideal that connects a curve to its quadratic twist, which will be used as a building block for the main algorithm to connect two arbitrary curves with the same $\mathbb{F}_p$-endomorphism ring in the $\mathbb{F}_p$-isogeny graph.

Algorithm 5.1: Connecting ideal of a curve and its twist.

Input: a supersingular $E/\mathbb{F}_p$ and the full endomorphism ring $\mathrm{End}(E)$.

Output: an invertible ideal $\mathfrak{a} \subseteq \mathcal{O}_E$ such that $[\mathfrak{a}]E = E^t$.

1 Find a non-zero element $\alpha \in \mathrm{End}(E)$ of the form $x\mathbf{i} + y\mathbf{ij}$.

2 Compute the ideal $\mathfrak{a} := \big(\mathrm{End}(E) \cdot \alpha\big) \cap \mathcal{O}_E$.

3 **Return** $\mathfrak{a}$.

Lemma 5.14. *Algorithm 5.1 is correct and runs in polynomial time.*

Proof. Note that $\alpha \in \mathbf{i}\mathcal{O}_E$ is a twisting endomorphism of E according to Lemma 5.10. Hence, $E[\mathrm{End}(E) \cdot \alpha] = \ker(\alpha)$ is an $\mathbb{F}_p$-rational subgroup of E giving rise to an $\mathbb{F}_p$-rational isogeny $E \longrightarrow E^t$, which is necessarily horizontal since $\mathcal{O}_E = \mathcal{O}_{E^t}$. Therefore, there exists an invertible ideal $\mathfrak{c}$ of $\mathcal{O}_E$ such that $E[\mathfrak{c}] = \ker \alpha$, and we may apply Lemma 5.13 to conclude that $\mathfrak{a} = \big(\mathrm{End}(E) \cdot \alpha\big) \cap \mathcal{O}_E$ in fact equals the desired ideal $\mathfrak{c}$ — up to powers of π_E, which is an endomorphism.

Regarding the runtime, everything consists of basic arithmetic in $B_{p,\infty}$ and some linear algebra over $\mathbb{Q}$ and $\mathbb{Z}$. □

As mentioned before, the inherent symmetry of the $\mathbb{F}_p$-isogeny graph with respect to quadratic twisting implies that the "location" of a curve E in the graph is somehow related to the properties of ideals that connect E to its quadratic twist E^t. The following lemma makes this intuition precise, in the sense that it determines a connecting ideal between two curves almost uniquely when given a twisting ideal for each of them. This correspondence is then used in an explicit manner to compute a connecting ideal in Algorithm 5.2.

Lemma 5.15. *Let E_0, E_1 be supersingular elliptic curves defined over $\mathbb{F}_p$ with $\mathrm{End}_p(E_0) \cong \mathrm{End}_p(E_1)$, such that we may simply write $\mathcal{O}$ for both. If $\mathfrak{b}, \mathfrak{c} \subseteq \mathcal{O}$ are invertible ideals such that $[\mathfrak{b}]E_0 = E_0^t$ and $[\mathfrak{c}]E_1 = E_1^t$, then the unique ideal class $[\mathfrak{a}]$ such that $[\mathfrak{a}]E_0 = E_1$ satisfies the equation $[\mathfrak{a}]^2 = [\mathfrak{b}][\mathfrak{c}]^{-1}$.*

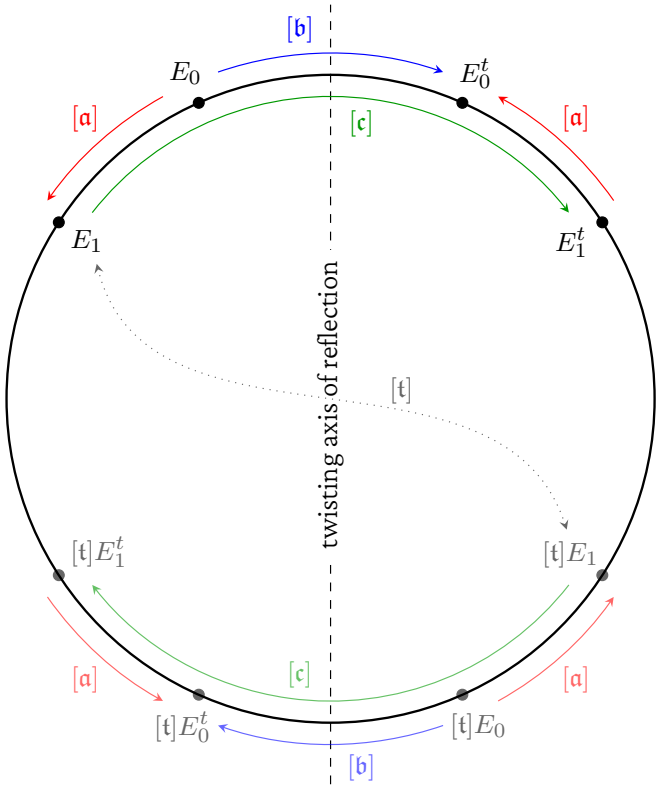


Figure 5.1: Illustration of Lemma 5.15 and the square-root issue in Lemma 5.16. If the ideal $\mathfrak{t} = (2, \sqrt{-p})$ is non-principal and invertible in $\mathcal{O}$, it corresponds to a point symmetry with respect to the "center" of the isogeny cycle, and the entire relationship between $E_{0,1}$ and their twists is replicated on the "opposite" side with the "dual" curves $[\mathfrak{t}]E_{0,1}$ and their twists. This explains why the output of Algorithm 5.2 is a priori only correct up to multiplication by $\mathfrak{t}$; the quadratic equation determining $[\mathfrak{a}]$ simply cannot distinguish whether $[\mathfrak{a}]$ jumps between the two worlds or not.

Proof. By Lemma 5.5, applying the action of an ideal class $[\mathfrak{u}]$ to E^t gives the same result as first applying $[\overline{\mathfrak{u}}] = [\mathfrak{u}]^{-1}$ and then twisting. Hence, if $[\mathfrak{a}]E_0 = E_1$, then $[\mathfrak{a}]^{-1}E_0^t = E_1^t$. However, by the assumptions, we have $[\mathfrak{a}]^{-1}E_0^t = [\mathfrak{a}]^{-1}[\mathfrak{b}]E_0$ on the left-hand side and $E_1^t = [\mathfrak{c}]E_1 = [\mathfrak{c}][\mathfrak{a}]E_0$ on the right-hand side, which implies the claimed equality of ideal classes as the class-group action is free. See Figure 5.1 for a visualization of the situation on an isogeny cycle. □

Lemma 5.16. *Algorithm 5.2 is correct and runs in polynomial time.*

Proof. Most of this follows from Lemmas 5.15 and 5.14. The square root in $\mathrm{cl}(\mathcal{O})$ to determine the ideal $\mathfrak{a}$ can be computed in polynomial time using the algorithm in [BS96, § 6].

Regarding the correctness of the output, recall from Lemma 5.3 that the class number of $\mathcal{O}$ is odd if $p \equiv 3 \pmod 4$, hence the square root $[\mathfrak{a}]$ is unique. On the other hand, if $p \equiv 1 \pmod 4$, then Lemma 5.3 implies that there are exactly two square roots. Now the order $\mathcal{O}$ has discriminant $-4p$, hence $(p) = (\sqrt{-p})^2$ and $(2) = (2, 1+\sqrt{-p})^2$ are the only ramified primes. The principal ideal $(\sqrt{-p})$ becomes trivial in $\mathrm{cl}(\mathcal{O})$. However, $\mathfrak{t} := (2, 1+\sqrt{-p})$ is non-principal as there

Algorithm 5.2: Connecting ideal of two curves.

Input: supersingular elliptic curves $E_0, E_1/\mathbb{F}_p$ with $\mathcal{O}_{E_0} = \mathcal{O}_{E_1} = \mathcal{O}$, together with their full endomorphism rings $\mathrm{End}(E_0)$ and $\mathrm{End}(E_1)$.

Output: an invertible ideal $\mathfrak{a} \subseteq \mathcal{O}$ such that $[\mathfrak{a}]E_0 = E_1$.

1 Using Algorithm 5.1, find an invertible ideal $\mathfrak{b} \subseteq \mathcal{O}$ with $[\mathfrak{b}]E_0 = E_0^t$.

2 Likewise, find an invertible ideal $\mathfrak{c} \subseteq \mathcal{O}$ such that $[\mathfrak{c}]E_1 = E_1^t$.

3 Compute an ideal $\mathfrak{a} \subseteq \mathcal{O}$ such that $[\mathfrak{a}]^2 = [\mathfrak{b}][\mathfrak{c}]^{-1}$ in $\mathrm{cl}(\mathcal{O})$ using [BS96, § 6].

4 If $p \equiv 1 \pmod 4$ and the right order of $\mathrm{End}(E_0) \cdot \mathfrak{a}$ in $B_{p,\infty}$ is not isomorphic to $\mathrm{End}(E_1)$, then replace $\mathfrak{a}$ by $\mathfrak{a} \cdot (2, 1+\sqrt{-p})$.

5 **Return** $\mathfrak{a}$.

is no element of norm 2 in $\mathcal{O}$, hence $[\mathfrak{t}]$ is an element of order 2 in $\mathrm{cl}(\mathcal{O})$. Thus the two square roots of $[\mathfrak{b}][\mathfrak{c}]^{-1}$ are $[\mathfrak{a}]$ and $[\mathfrak{a}\mathfrak{t}]$. The final check in the algorithm identifies the correct choice by lifting $\mathfrak{a}$ to a left $\mathrm{End}(E_0)$-ideal and comparing its right order to the endomorphism ring of E_1; they must be isomorphic if $\mathfrak{a}$ determines an isogeny $E_0 \to E_1$ as intended. □

An example. To illustrate the algorithms in this section, we will show their workings on a concrete, rather special example.

Lemma 5.17. *Assume $p \equiv 3 \pmod 4$ and let E_1 be a supersingular elliptic curve defined over $\mathbb{F}_p$ with $\mathbb{F}_p$-endomorphism ring $\mathcal{O}$. Let E_0 be the elliptic curve in $\mathcal{Ell}_p(\mathcal{O})$ having j-invariant 1728. If $\mathfrak{b} \subseteq \mathcal{O}$ is an invertible ideal such that $[\mathfrak{b}]E_1 = E_1^t$, then the unique ideal class $[\mathfrak{a}]$ such that $[\mathfrak{a}]E_0 = E_1$ is given by $[\mathfrak{b}]^{(h(\mathcal{O})-1)/2}$.*

Proof. This follows from Lemmas 5.1 and 5.15, together with the fact that the class number of $\mathcal{O}$ is odd. □

Example 5.18. *Assume that $p \equiv 11 \pmod{12}$. We illustrate Algorithm 5.2 by computing a connecting ideal $\mathfrak{a}$ between $E_0 \colon y^2 = x^3 + x$ and $E_1 \colon y^2 = x^3 + 1$. Note that both curves are contained in $\mathcal{Ell}_p(\mathbb{Z}[\sqrt{-p}])$, as can be seen by considering $E(\mathbb{F}_p)[2]$. If $\omega \in \mathbb{F}_{p^2} \setminus \mathbb{F}_p$ denotes a primitive 3^{rd} root of unity, then E_1 admits the automorphism $(x, y) \mapsto (\omega x, y)$, which will, by abuse of notation, be denoted by ω as well. According to [McM14, Prop. 3.2],[4] the endomorphism ring of E_1 is isomorphic to the $B_{p,\infty}$-order*

$$\mathcal{Q} = \mathbb{Z} + \mathbb{Z}\frac{-1+\mathbf{i}}{2} + \mathbb{Z}\mathbf{j} + \mathbb{Z}\frac{3+\mathbf{i}+3\mathbf{j}+\mathbf{ij}}{6},$$

where $\mathbf{i}$ corresponds to $2\omega + 1$ and satisfies[5] $\mathbf{i}^2 = -3$, and as usual $\mathbf{j}$ corresponds to the Frobenius endomorphism π_{E_1}. If we choose the twisting endomorphism $\alpha = \mathbf{i}$ in Algorithm 5.1, then we find $\mathcal{Q}\mathbf{i} \cap \mathbb{Z}[\mathbf{j}] = (3, \mathbf{j} - 1)$. (Of course, this also follows from the fact that $2\omega + 1$ is a degree-3 isogeny whose kernel $\{(0, \pm 1), \infty\}$ is $\mathbb{F}_p$-rational.) So $E_1^t = [(3, \sqrt{-p} - 1)]E_1$, and we can take

$$\mathfrak{a} = (3, \sqrt{-p} - 1)^{(h(\mathbb{Z}[\sqrt{-p}])-1)/2} \tag{5.3}$$

[4] Unfortunately, the statement of [McM14, Prop. 3.2] wrongly attributes this description to the quadratic twist of E_1.

[5] Here we deviate from our convention that $\mathbf{i}^2 = -1$ as soon as $p \equiv 3 \pmod 4$.

by Lemma 5.17. Thus, in the 3-isogeny graph associated with $\mathcal{E}\ell\ell_p(\mathbb{Z}[\sqrt{-p}])$, which is a union of cycles, the curve E_1 and its twist $E_1^t\colon y^2 = x^3 - 1$ can be found "opposite" of our starting curve E_0, on the same cycle. We will generalize this example in Section 5.5.

Example 5.19. *In particular, the findings of Example 5.18 hold for a CSIDH prime $p = 4\ell_1\ell_2\cdots\ell_n - 1$ with $\ell_1 = 3$, so that $(3, \sqrt{-p} - 1) = \mathfrak{l}_1$. Note that $E\colon y^2 = x^3 + 1$ is isomorphic to the Montgomery curve $E_{-\sqrt{3}}\colon y^2 = x^3 - \sqrt{3}\cdot x^2 + x$ through*

$$E_{-\sqrt{3}} \longrightarrow E,\ (x, y) \longmapsto (\delta^2 x - 1, \delta^3 y),$$

where $\sqrt{3} \in \mathbb{F}_p$ denotes the square root of 3 which is a square itself, and $\delta^2 = \sqrt{3}$. In view of the class-group computation carried out in [BKV19] for the CSIDH-512 parameter set, the previous example shows that the ideal

$$\mathfrak{l}_1^{12732622111474213758851509300531960108081025715274321179628543048779880586309 5}$$

takes the starting Montgomery coefficient 0 to the coefficient $-\sqrt{3}$, and one further application of $\mathfrak{l}_1$ takes it to $\sqrt{3}$. Smoothing this ideal using the class-group relations of $\mathrm{cl}(\mathbb{Z}[\sqrt{-p}])$ from [BKV19] yields (for instance) the CSIDH-512 exponent vector

$$\begin{array}{cccccccccccccccccc}
(5, & -7, & -1, & 1, & -4, & -5, & -8, & 4, & -1, & 5, & 1, & 0, & -2, & -4, & -2, & 2, & -9, & 4, & 2,\\
5, & 1, & 1, & 1, & 5, & -4, & 2, & 6, & 5, & -1, & 0, & 0, & -4, & -1, & -3, & -1, & -4, & 1, & 7,\\
1, & 4, & 1, & 4, & -7, & 0, & -3, & -1, & 0, & 1, & 2, & 3, & 1, & 2, & -4, & -5, & 9, & -1, & 4,\\
0, & 5, & 1, & 0, & 1, & 1, & 3, & 0, & 2, & 2, & 2, & -1, & 2, & 1, & -1, & 11, & 3), & &
\end{array}$$

which can indeed be readily verified to connect E_0 to $E_{-\sqrt{3}}$ by plugging it into a CSIDH-512 implementation, such as that of [Cas+18], as a private key.

Example 5.20. *If in Example 5.18, we instead choose the twisting endomorphism*

$$\alpha = \frac{\mathbf{i} + \mathbf{ij}}{3} = -1 - \mathbf{j} + 2\frac{3 + \mathbf{i} + 3\mathbf{j} + \mathbf{ij}}{6} \in \mathcal{Q},$$

then we obtain a twisting ideal of norm $(p+1)/3$. In the CSIDH setting of Example 5.19 above, one can deduce that this ideal is nothing but $\bar{\mathfrak{l}}_0\bar{\mathfrak{l}}_2\bar{\mathfrak{l}}_3\cdots\bar{\mathfrak{l}}_n$, so this confirms the first class-group relation stated in Lemma 5.8.

5.4.2 – Incomplete knowledge of endomorphism rings. At first sight, there appears to be no strong reason why one requires the full endomorphism rings to be known exactly in Algorithm 5.1, rather than for instance a full-rank proper subring $\mathcal{Q} \subsetneq \mathrm{End}(E)$ containing $\mathcal{O}$: Twisting endomorphisms α can clearly be found in every full-rank subring, and one can still compute the left ideal $\mathcal{Q}\cdot\alpha$, which can then be intersected with $\mathcal{O}$. The result is indeed an ideal $\mathfrak{a}$ of $\mathcal{O}$, as desired, but unfortunately it turns out that $\mathfrak{a}$ usually falls short of connecting E to its quadratic twist unless in fact $\mathcal{Q} = \mathrm{End}(E)$. This is not surprising: If $\mathcal{Q}$ is contained in multiple non-isomorphic maximal orders, then the algorithm would need to work for all those maximal orders — and therefore elliptic curves — simultaneously, which is absurd. However, luckily, one can prove that $\mathfrak{a}$ is only *locally* "wrong" at the conductor, i.e., the index $f := \big[\mathrm{End}(E) : \mathcal{Q}\big]$.

Lemma 5.21. *Let $\mathcal{Q} \subseteq \mathrm{End}(E)$ a full-rank subring containing $\mathcal{O}$ and $\alpha \in \mathcal{Q}\backslash\{0\}$ a twisting endomorphism. Defining $\mathfrak{a} := (\mathcal{Q}\cdot\alpha) \cap \mathcal{O}$ and $\mathfrak{b}_c := \big(\mathrm{End}(E) \cdot c\alpha\big) \cap \mathcal{O}$ for $c \in \mathbb{Z}$, we have inclusions of $\mathcal{O}$-ideals*

$$\mathfrak{b}_f \subseteq \mathfrak{a} \subseteq \mathfrak{b}_1,$$

where the norm of the quotient $(\mathfrak{b}_1 : \mathfrak{b}_f)$ divides the squared conductor f^2.

Proof. The inclusions are obvious from $\mathrm{End}(E) \cdot f \subseteq \mathcal{Q} \subseteq \mathrm{End}(E)$. Moreover,

$$f\mathfrak{b}_1 = \big(f \cdot \mathrm{End}(E) \cdot \alpha\big) \cap (f \cdot \mathcal{O}) \subseteq \big(\mathrm{End}(E) \cdot f\alpha\big) \cap \mathcal{O} = \mathfrak{b}_f,$$

and the inclusions we have just established imply a chain of surjections

$$\mathfrak{b}_1/f\mathfrak{b}_1 \twoheadrightarrow \mathfrak{b}_1/\mathfrak{b}_f \twoheadrightarrow \mathfrak{b}_1/\mathfrak{a}$$

on the quotients of $\mathfrak{b}_1$. The first module in this sequence is clearly isomorphic to $\mathbb{Z}^2/f\mathbb{Z}^2$, therefore the index $[\mathfrak{b}_1 : \mathfrak{b}_f]$ must be a divisor of $|\mathbb{Z}^2/f\mathbb{Z}^2| = f^2$. □

Note that both ideals $\mathfrak{b}_1$ and $\mathfrak{b}_f$ from Lemma 5.21 would be correct outputs for a generalization of Algorithm 5.1 to proper subrings of $\mathrm{End}(E)$, but $\mathfrak{a}$ typically is not. However, the lemma still suggests an easy strategy for guessing $\mathfrak{b}_1$ after having obtained $\mathfrak{a}$ from the subring variant of Algorithm 5.1, at least when factoring f is feasible and there are not too many prime factors: In that case, one may simply brute-force through all ideals $\mathfrak{c} \subseteq \mathcal{O}$ of norm dividing f^2 and output $\mathfrak{a}\mathfrak{c}$ for each of them. The lemma guarantees that a correct such $\mathfrak{c}$ exists, since the ideal $(\mathfrak{b}_1 : \mathfrak{a})$ is a good choice. This procedure is summarized in Algorithm 5.3.

Algorithm 5.3: Twisting a curve using an endomorphism *sub*ring.

Input: a supersingular $E/\mathbb{F}_p$ and a rank-4 subring $\mathcal{Q} \subseteq \mathrm{End}(E)$ with $\mathcal{Q} \supseteq \mathcal{O}_E$.

Output: a set A of invertible ideals $\mathfrak{a} \subseteq \mathcal{O}_E$ such that $\exists\, \mathfrak{a} \in A$ with $[\mathfrak{a}]E = E^t$.

1 Find a non-zero element $\alpha \in \mathcal{Q}$ of the form $x\mathbf{i} + y\mathbf{ij}$.

2 Compute the ideal $\mathfrak{a} := \big(\mathcal{Q} \cdot \alpha\big) \cap \mathcal{O}_E$.

3 Determine $f = [\mathrm{End}(E) : \mathcal{Q}]$ as the (reduced) discriminant of $\mathcal{Q}$ divided by p.

4 Factor f and iterate through all ideals $\mathfrak{c} \subseteq \mathcal{O}$ of norm dividing f^2 to compute the set $\mathcal{A} := \{\mathfrak{a}\mathfrak{c} \mid \mathfrak{c} \subseteq \mathcal{O} \text{ ideal}, \mathrm{N}(\mathfrak{c}) \mid f^2\}$.

5 **Return** $\mathcal{A}$.

We can bound the size of the set $\mathcal{A}$ returned by the algorithm as follows: If the conductor f factors into primes as $f = \prod_{i=1}^{r} p_i^{e_i}$, then there are at most

$$\prod_{i=1}^{r} \binom{2e_i + 2}{2} \in O\big((\log f)^{2r}\big)$$

distinct $\mathcal{O}$-ideals of norm dividing f^2. Hence, if f is factorable in polynomial time and the number of distinct prime factors r is bounded by a constant, then Algorithm 5.3 takes polynomial time to output polynomially many ideals, and at least one of them is guaranteed to be correct.

5.4.3 – Can we do better? It is a natural question to ask whether one can tweak the KLPT quaternion-ideal algorithm [KLPT14] to simply output an ideal corresponding to an isogeny defined over $\mathbb{F}_p$, while preserving the main characteristics of the algorithm, namely the smoothness of the ideal that is returned and the (heuristic) polynomial runtime.

In this section, we argue that the answer is likely "no", at least for classical algorithms: More concretely, we show that such an algorithm can be used as a black-box oracle to construct, under a few mild assumptions, a polynomial-time algorithm for the discrete-logarithm problem in those imaginary-quadratic class groups where the prospective KLPT variant would apply. In contrast, the currently best known algorithm is only subexponential-time [HM89]. Thus, the

basic conclusion appears to be that either our result is essentially optimal, or there exists an improved classical algorithm to compute class-group discrete logarithms in (at least) some cases.

In a sense, this is not surprising: The requirement that the output be generated by an ideal of the two-dimensional subring $\mathrm{End}_p(E)$ removes about the same amount of freedom as was "adjoined" when moving from $\mathbb{Q}(\sqrt{-p})$ to $B_{p,\infty}$ in the first place. In fact, the KLPT algorithm makes explicit constructive use of a quadratic subring of $B_{p,\infty}$ to achieve its functionality; an advantage that can be expected to cease when imposing strong restrictions on the output.

We formalize the situation as follows. Suppose we are given an algorithm $\mathcal{A}$ with the same interface as Algorithm 5.2, i.e., it takes as input two supersingular elliptic curves $E, E'/\mathbb{F}_p$ with the same $\mathbb{F}_p$-endomorphism ring $\mathcal{O}$, together with their full endomorphism rings, and outputs an ideal $\mathfrak{a} \subseteq \mathcal{O}$ such that $[\mathfrak{a}]E = E'$. In addition, our hypothetical algorithm $\mathcal{A}$ now guarantees that all prime factors of the returned ideal $\mathfrak{a}$ are elements of some polynomially-sized set $S_\mathcal{O}$, which may depend on the prime p or the ring $\mathcal{O}$ but not on the concrete input curves E and E'. For example, $S_\mathcal{O}$ might consist of the prime ideals of $\mathcal{O}$ with norm bounded by a polynomial in $\log p$.[6]

Then, Algorithm 5.5 can use such an oracle $\mathcal{A}$ to compute discrete logarithms in the subgroup of $\mathrm{cl}(\mathcal{O})$ generated by the subset $S_\mathcal{O}$ in expected polynomial time, assuming that querying $\mathcal{A}$ takes polynomial time. Note that the core of the reduction is Algorithm 5.4, which employs $\mathcal{A}$ to decompose class-group elements as a relation over the factor base $S_\mathcal{O}$, and those relations are subsequently used by Algorithm 5.5 in a generic and fairly standard index-calculus procedure.

A remark on notation: we make use of vectors and matrices indexed by finite sets I such as $S_\mathcal{O}$ — in real implementations this would correspond to fixing an ordering of I and simply storing normal vectors or matrices of length $|I|$. We use the notation $|_{I'}$ to restrict a vector or matrix to the columns indexed by a subset $I' \subseteq I$.

Algorithm 5.4: Finding a class-group relation using $\mathcal{A}$.

Input: an oracle $\mathcal{A}$ as above, and an ideal $\mathfrak{a} \subseteq \mathcal{O}$ such that $[\mathfrak{a}] \in \langle [\mathfrak{s}] \mid \mathfrak{s} \in S_\mathcal{O} \rangle$.

Output: a vector $(e_\mathfrak{s} \mid \mathfrak{s} \in S_\mathcal{O}) \in \mathbb{Z}^{S_\mathcal{O}}$ such that $[\mathfrak{a}] = \left[\prod_{\mathfrak{s} \in S_\mathcal{O}} \mathfrak{s}^{e_\mathfrak{s}}\right]$.

1 Find a supersingular $E/\mathbb{F}_p$ with $\mathrm{End}_p(E) = \mathcal{O}$ and known $\mathrm{End}(E)$.
2 Apply KLPT to $\mathrm{End}(E) \cdot \mathfrak{a}$ to get an equivalent powersmooth left ideal $\mathcal{I}$.
3 Find the codomain $E' = [\mathfrak{a}]E$ by computing the isogeny defined by $\mathcal{I}$.
4 Compute $\mathrm{End}(E')$ as the right order of $\mathcal{I}$ in $B_{p,\infty}$.
5 Now query $\mathcal{A}$ to find an ideal $\mathfrak{c} \in \langle S_\mathcal{O} \rangle$ such that $[\mathfrak{c}]E = E' = [\mathfrak{a}]E$.
6 By assumption, $\mathfrak{c}$ is of the form $\prod_{\mathfrak{s} \in S_\mathcal{O}} \mathfrak{s}^{e_\mathfrak{s}}$.
7 **Return** that exponent vector $\underline{e} = (e_\mathfrak{s} \mid \mathfrak{s} \in S_\mathcal{O})$.

Lemma 5.22. *Algorithm 5.4 is correct. It takes polynomial time under the heuristic that the KLPT algorithm runs in polynomial time.*

Proof. Note that finding a curve E as desired is easy: first construct an arbitrary supersingular elliptic curve over $\mathbb{F}_p$ using [Brö09], then potentially walk to the surface or floor of a 2-volcano.

[6] Under GRH, Bach [Bac90] proved that $\mathrm{cl}(\mathcal{O})$ is generated by prime ideals of norm less than $C(\log p)^2$ for an explicitly computable small constant C. It is not known unconditionally whether a polynomial bound on the norms suffices.

Next, note that the curve E' in fact equals $[\mathfrak{a}]E$, since $\mathrm{End}(E) \cdot \mathfrak{a}$ and $\mathfrak{a}$ define the same subgroup of E and $\mathcal{I}$ is equivalent as a left ideal to $\mathrm{End}(E) \cdot \mathfrak{a}$. Computing $\mathrm{End}(E')$ given $\mathcal{I}$ is easy linear algebra. Now, $\mathfrak{c}$ is a product of ideals in $S_{\mathcal{O}}$ by assumption on $\mathcal{A}$, and it must be equivalent to $\mathfrak{a}$ in $\mathrm{cl}(\mathcal{O})$ since the latter acts freely on $\mathscr{El}_p(\mathcal{O})$. In conclusion, Algorithm 5.4 indeed returns a correct relation vector for $\mathfrak{a}$ and takes polynomial time to do so. □

Using Algorithm 5.4, we can then follow the generic index-calculus procedure shown in Algorithm 5.5 to compute discrete logarithms in $\mathrm{cl}(\mathcal{O})$:

Algorithm 5.5: Solving DLP using index calculus (generic).

Input:
- a generating set S of a finite abelian group G.
- an upper bound B on the cardinality $|G|$.
- elements $\mathsf{g}, \mathsf{h} \in G$ such that $\mathsf{h} \in \langle \mathsf{g} \rangle$.
- a probabilistic algorithm $\Delta\colon G \to \mathbb{Z}^S$, such that for all inputs $\mathsf{a} \in G$, we have $\|\Delta(\mathsf{a})\|_\infty < B$ and $\mathsf{a} = \prod_{\mathsf{b} \in S} \mathsf{b}^{\Delta(\mathsf{a})_\mathsf{b}}$.

Output: an integer x such that $\mathsf{g}^x = \mathsf{h}$.

1 Fix a large integer $H \gg B^{2|S|+1}$. (In practice, use much smaller H.)
2 Initialize empty matrices $M \in \mathbb{Z}^{0\times 2}$ and $L \in \mathbb{Z}^{0\times S}$.
3 **for** $n = 1, 2, \ldots$ **do**
4 Pick integers u, v uniformly random in $\{-H, \ldots, H\}$.
5 Invoke Δ to obtain a vector $\underline{e} \in \mathbb{Z}^S$ such that $\mathsf{g}^u\mathsf{h}^v = \prod_{\mathsf{b}\in S} \mathsf{b}^{e_\mathsf{b}}$.
6 Append (u, v) to M as a new row. Append $\underline{e}$ to L as a new row.
7 Compute a basis matrix $K \in \mathbb{Z}^{r\times n}$ of the left kernel of L, which is a lattice in $\mathbb{Z}^n$ of rank r.
8 **If** the row span of $K \cdot M$ contains a vector of the form $(x, -1)$ **then**
9 **Return** x.

Lemma 5.23. *Algorithm 5.5 is correct and runs in expected polynomial time.*[7]

Proof sketch. It is not hard to check that the output of the algorithm is correct if it terminates; we thus only have to bound the expected runtime.

Since the proof is rather technical, we will merely show the overall strategy. Note that it suffices to lower bound the success probability of the algorithm when $r = 2$ by a constant: Since $r \geq n - |S|$ throughout, it is evident that running $|S| + \alpha$ iterations of Algorithm 5.5 has success probability at least as big as $\lfloor \alpha/2 \rfloor$ independent executions of the modified algorithm. We thus want to lower bound the probability that two entries λ_1, λ_2 in the second column of $K \cdot M$ are coprime.

First, since Δ cannot distinguish from which scalars (u, v) the element $\mathsf{g}^u\mathsf{h}^v$ was obtained, the conditional distribution of each coefficient of M after fixing a certain oracle output $\underline{e}$ is close

[7] Note that this does not require *any* assumptions on the output distribution of $\Delta(\mathsf{a})$, other than that the returned vectors are correct. (The algorithm still takes polynomial time if the oracle Δ only succeeds on an inverse polynomial fraction of inputs.)

to uniform on $\{-H, ..., H\}$. As the lattice spanned by the rows of $K \cdot M$ is clearly independent of a basis choice, we may without loss of generality assume that the rows of K form a shortest basis of $\mathbb{Z}^r K$; using lattice techniques, one can then show that the norms of vectors in a shortest basis of $\mathbb{Z}^r K$ are upper bounded by $B^{2|S|}$. (This uses the bound on the size of integers returned by Δ.) Hence λ_i is a "small" coprime linear combination of random variables essentially uniform on $\{-H, ..., H\}$, which in turn implies that λ_i is close to uniform modulo all potential prime divisors. Thus the probability that $\gcd(\lambda_1, \lambda_2) = 1$ is lower bounded by a constant, similar to the well-known fact that the density of coprime pairs in $\mathbb{Z}^2$ is $\zeta(2)^{-1} = 6/\pi^2$. □

For concreteness, we briefly spell out how to instantiate Algorithm 5.5 for our particular application to $\mathrm{cl}(\mathcal{O})$. Clearly, Algorithm 5.4 will serve as the oracle Δ, so the factor base S equals the set $S_{\mathcal{O}}$ from Algorithm 5.4. In order to keep the representation sizes limited and to obtain unique representatives of ideal classes, the required products $\mathsf{g}^u\mathsf{h}^v$ should be computed using the square-and-multiply algorithm combined with reduction of binary quadratic forms; see [Cox13] for more context on the correspondence between quadratic forms and ideals (§ 7.B) and the notion of reduction (§ 2.A). To select the estimate B on the group order, recall the upper bound $h(\mathcal{O}) \in O(\sqrt{p} \log p)$ from the class number formula.

5.5 — Vectorizing CM curves

To the best of our knowledge, there exist two practical methods for constructing supersingular elliptic curves over a large finite field $\mathbb{F}_p$: either one reduces curves having CM by some order $\mathcal{R}$ in an imaginary quadratic field F modulo (appropriately chosen) primes that do not split in F, or one performs isogeny walks starting from known supersingular curves. As pointed out earlier, outside of trusted setup, the latter method is not suitable for most cryptographic applications. In this section we focus on the former method; additional details can be found in Bröker's paper [Brö09] and the references therein. As we will see, from a security point of view, the situation is even more problematic in this case: we show that the vectorization problem associated with a CM-constructed supersingular elliptic curve over $\mathbb{F}_p$ admits a surprisingly easy and explicit solution.

In practice, when constructing supersingular elliptic curves over $\mathbb{F}_p$ one does not explicitly write down CM curves. Rather, one computes the Hilbert class polynomial $H_{\mathcal{R}}(T) \in \mathbb{Z}[T]$ for $\mathcal{R}$, which is a monic irreducible polynomial whose roots are the j-invariants of the curves having CM by $\mathcal{R}$. This polynomial can be computed effectively, although the existing methods are practical for orders having small discriminants only, one reason being that the degree of $H_{\mathcal{R}}(T)$ equals $h(\mathcal{R})$. The roots of $H_{\mathcal{R}}(T) \bmod p \in \mathbb{F}_p[T]$ are precisely those $j \in \overline{\mathbb{F}_p}$ which arise as the j-invariant of a supersingular elliptic curve obtained by reducing an elliptic curve having CM by $\mathcal{R}$. It is well-known that all these j-invariants are in fact elements of $\mathbb{F}_{p^2}$, i.e., the irreducible factors of $H_{\mathcal{R}}(T) \bmod p$ are at most quadratic. The linear factors then correspond to elliptic curves over $\mathbb{F}_p$.

Example 5.24. *The Hilbert class polynomial for $\mathbb{Z}[\sqrt{-17}]$ is given by*

$$H_{\mathbb{Z}[\sqrt{-17}]}(T) = T^4 - 1782110400000T^3 - 758436921600000000T^2 - 31850703872000000000T - 208929750630400000000000000,$$

whose reduction modulo 83 *factors as* $(T-28)(T-50)(T^2+7T+73)$. *This gives rise to two pairs of quadratic twists of elliptic curves over* $\mathbb{F}_{83}$ *that appear as the reduction modulo* 83 *of a curve with CM by* $\mathbb{Z}[\sqrt{-17}]$.

The main result of this section is the following theorem; for conciseness, our focus lies on the setting where $p \equiv 3 \pmod 4$ and where

$$\mathbb{Z}[\sqrt{-\ell}] \subseteq \mathcal{R} \subseteq \mathbb{Q}(\sqrt{-\ell})$$

for some odd prime number ℓ, i.e., we want our CM curves to come equipped with an endomorphism Ψ satisfying $\Psi \circ \Psi = [-\ell]$. This leaves us with two options for $\mathcal{R}$, namely $\mathbb{Z}[\sqrt{-\ell}]$ and $\mathbb{Z}[(1+\sqrt{-\ell})/2]$. In Remark 5.31 we will briefly comment on how to locate curves having CM by more general imaginary quadratic orders.

Theorem 5.25. *Let* $p \equiv 3 \pmod 4$ *and* $\ell < (p+1)/4$ *be primes with* $\left(\frac{-p}{\ell}\right) = 1$.

(i) If $\ell \equiv 1 \pmod 4$ *then*

$$H_{\mathbb{Z}[\sqrt{-\ell}]}(T) \bmod p$$

has precisely two $\mathbb{F}_p$*-rational roots, both corresponding to a pair of quadratic twists of supersingular elliptic curves. One pair is contained in* $\mathcal{E}\ell\ell_p(\mathbb{Z}[\sqrt{-p}])$ *while the other pair is contained in* $\mathcal{E}\ell\ell_p(\mathbb{Z}[(1+\sqrt{-p})/2])$.

(ii) If $\ell \equiv 3 \pmod 4$ *then both*

$$H_{\mathbb{Z}[(1+\sqrt{-\ell})/2]}(T) \bmod p \qquad \text{and} \qquad H_{\mathbb{Z}[\sqrt{-\ell}]}(T) \bmod p$$

have exactly one $\mathbb{F}_p$*-rational root each, in both cases corresponding to a pair of quadratic twists of elliptic curves. The first such pair is contained in* $\mathcal{E}\ell\ell_p(\mathbb{Z}[\sqrt{-p}])$, *while the other pair is contained in* $\mathcal{E}\ell\ell_p(\mathbb{Z}[(1+\sqrt{-p})/2])$.

(iii) Let $\mathcal{O} \in \{\mathbb{Z}[\sqrt{-p}], \mathbb{Z}[(1+\sqrt{-p})/2]\}$ *and let* $E, E^t \in \mathcal{E}\ell\ell_p(\mathcal{O})$ *be a pair of supersingular elliptic curves over* $\mathbb{F}_p$ *arising as above. Up to order, this pair is given by the curves*

$$[\mathfrak{l}]^{(h(\mathcal{O})-1)/2} E_0 \qquad \text{and} \qquad [\mathfrak{l}]^{(h(\mathcal{O})+1)/2} E_0 \tag{5.4}$$

for any prime ideal $\mathfrak{l} \subseteq \mathcal{O}$ *lying above* ℓ. *Here* $E_0 \colon y^2 = x^3 \pm x$ *is the unique curve in* $\mathcal{E}\ell\ell_p(\mathcal{O})$ *with* j*-invariant* 1728.

This theorem can be seen as a vast generalization of (5.3) from Example 5.18, where we dealt with the reduction modulo p of the curve $E \colon y^2 = x^3 + 1$ over $\mathbb{Q}$ having CM by the ring of Eisenstein integers $\mathbb{Z}[e^{2\pi\mathbf{i}/3}] = \mathbb{Z}[(1+\sqrt{-3})/2]$. Up to twisting it is the only such curve: the Hilbert class polynomial for $\mathbb{Z}[(1+\sqrt{-3})/2]$ is just T. An endomorphism Ψ satisfying $\Psi^2 = -3$ can be constructed as $2\omega + 1$, where ω is the automorphism $E \to E,\ (x,y) \mapsto (e^{2\pi\mathbf{i}/3}x, y)$.

One particularly interesting range of parameters satisfying the stated assumptions is where

- $p = 4\ell_1\ell_2\cdots\ell_n - 1$ is a CSIDH prime with $n \geq 2$, and
- $\ell = \ell_i$ for some $i \in \{1, 2, \ldots, n\}$.

If $n = 1$ then $\ell_1 = (p+1)/4$, so Theorem 5.25 can no longer be applied. However, the reasons for excluding the boundary case $\ell = (p+1)/4$ are rather superficial and the statement remains largely valid in this case (the exclusion is related to the possible occurrence of $j = 1728$ as a root of $H_{\mathcal{R}}(T) \bmod p$, which comes with some subtleties in terms of quadratic twisting; see the proof).

5.5.1 – Twisting endomorphisms from Deuring reduction. Before proving Theorem 5.25, we discuss Deuring lifting and reduction, with a focus on how the endomorphism Ψ behaves under reduction.

Theorem 5.26 (Deuring's reduction theorem)**.** *Let p be a prime number and let E be an elliptic curve over a number field K which has CM by some order $\mathcal{R}$ in an imaginary quadratic number field F. Let $\mathfrak{p}$ be a prime of K above p at which E has good reduction. Then $E \bmod \mathfrak{p}$ is supersingular if and only if p ramifies or is inert in F.*

Proof. This is part of [Lan87, Thm. 12 of Ch. 13]. □

When applying this to an elliptic curve E/K having CM by our order $\mathcal{R} \subseteq \mathbb{Q}(\sqrt{-\ell})$ from above, the endomorphism Ψ satisfying $\Psi \circ \Psi = [-\ell]$ reduces modulo $\mathfrak{p}$ to an endomorphism ψ which also satisfies $\psi \circ \psi = [-\ell]$. This is because reduction modulo $\mathfrak{p}$ induces an (injective) homomorphism of endomorphism rings; see for instance [Lan87, § 2 of Ch. 9]. The following proposition gives sufficient conditions for ψ to be a twisting endomorphism.

Proposition 5.27. *Assume $K = \mathbb{Q}(j(E))$, $p > 2$ and $\ell \leq (p+1)/4$. If $E \bmod \mathfrak{p}$ is supersingular and $j(E \bmod \mathfrak{p}) \in \mathbb{F}_p$ then $\deg(\mathfrak{p}) = 1$ and*

$$\pi_{E \bmod \mathfrak{p}} \circ \psi = -\psi \circ \pi_{E \bmod \mathfrak{p}}, \tag{5.5}$$

i.e., ψ anticommutes with the p-power Frobenius endomorphism of $E \bmod \mathfrak{p}$.

The proof of this proposition relies on the following observation:

Lemma 5.28. *Let α be an algebraic integer and $K = \mathbb{Q}(\alpha)$. Consider a prime number p and a prime ideal $\mathfrak{p} \subseteq \mathcal{O}_K$ above p. If $\mathbb{F}_p(\alpha \bmod \mathfrak{p}) \subsetneq \mathcal{O}_K/\mathfrak{p}$, then p divides the discriminant of the minimal polynomial $f(x) \in \mathbb{Z}[x]$ of α over $\mathbb{Q}$.*

Proof. If p does not divide the discriminant of $f(x)$, then

$$\mathfrak{p} = \big(p, g(\alpha)\big),$$

where $g(x) \in \mathbb{Z}[x]$ is a monic polynomial of degree $\deg(\mathfrak{p})$ whose reduction modulo p is an irreducible factor in $\mathbb{F}_p[x]$ of $f(x) \bmod p$; this is a well-known fact, see e.g. [Mar18a, Thm. 27]. But this implies that $\alpha \bmod \mathfrak{p}$ is a generator of $\mathcal{O}_K/\mathfrak{p}$ over $\mathbb{F}_p$, so the lemma follows by contradiction. □

Proof of Proposition 5.27. The minimal polynomial of $j(E)$ over $\mathbb{Q}$ is precisely the Hilbert class polynomial $H_{\mathcal{R}}(T)$ for $\mathcal{R}$. The field $H = \mathbb{Q}(\sqrt{-\ell}, j(E))$ is a quadratic extension of K known as the ring class field for $\mathcal{R}$, see [Cox13, proof of Prop. 1.32]. If $\mathcal{R}$ is a maximal order, then this is better known as the Hilbert class field.

Using that $\ell \leq (p+1)/4$, we see that p does not ramify in $\mathbb{Q}(\sqrt{-\ell})$, hence it must be inert by our assumption that $E \bmod \mathfrak{p}$ is supersingular. This implies that $p\mathcal{O}_H$ splits as a product of prime ideals $\mathfrak{P}$ of degree 2, see [Cox13, Cor. 5.25] for a proof in case $\mathcal{R}$ is a maximal order and [Cox13, proof of Prop. 9.4] for the general case (this is where we use the assumption $p > 2$). Our prime ideal $\mathfrak{p}$ is necessarily dominated by such a $\mathfrak{P}$, so it follows that

- either $\deg(\mathfrak{p}) = 1$, in which case $\mathfrak{p}$ must be inert in H, i.e., $\mathfrak{p}\mathcal{O}_H = \mathfrak{P}$,
- or $\deg(\mathfrak{p}) = 2$, in which case $\mathfrak{p}$ must split in H.

But the latter option would imply that

$$\mathbb{F}_p(j(E) \bmod \mathfrak{p}) = \mathbb{F}_p(j(E \bmod \mathfrak{p})) = \mathbb{F}_p \subsetneq \mathcal{O}_K/\mathfrak{p}$$

and therefore, in view of Lemma 5.28, it would follow that p divides the discriminant of $H_{\mathcal{R}}(T)$. This is impossible: by Gross–Zagier [GZ85, p. 195] the primes p dividing the discriminant of $H_{\mathcal{R}}(T)$ cannot be larger than the absolute value of the discriminant of $\mathcal{R}$, which is at most 4ℓ.

We have thus established that $\deg(\mathfrak{p}) = 1$. Now let Σ be the non-trivial automorphism of H over K. From [Lan87, § 4 of Ch. 10] we see that Ψ is not defined over K and therefore $\Psi^{\Sigma} = -\Psi$. But Σ necessarily descends to the Frobenius automorphism σ of $\mathcal{O}_H/\mathfrak{P} \cong \mathbb{F}_{p^2}$ over $\mathcal{O}_K/\mathfrak{p} \cong \mathbb{F}_p$, from which it follows that $\psi^{\sigma} = -\psi$. This implies (5.5) and thereby concludes the proof. □

We remark that the last part of the preceding proof mimics the proof of [GR04, Prop. 6.1]. However, the statement of [GR04, Prop. 6.1] is lacking an assumption on $\deg(\mathfrak{p})$. For instance, in our case, if $\deg(\mathfrak{p}) = 2$ and therefore $\mathfrak{p}$ splits in H, the reasoning fails because the extension $\mathcal{O}_H/\mathfrak{P}$ over $\mathcal{O}_K/\mathfrak{p}$ becomes trivial. And indeed, in these cases it may happen that the reduction of Ψ mod $\mathfrak{p}$ does *not* anticommute with Frobenius:

Example 5.29. *The discriminant of the Hilbert class polynomial for $\mathbb{Z}[\sqrt{-29}]$ is divisible by 83. More precisely, its reduction modulo 83 factors as $T(T-50)(T-67)^2(T^2+7T+73)$. One can verify that inside $K = \mathbb{Q}[T]/(H_{\mathbb{Z}[\sqrt{-29}]}(T))$, we have*

$$83\mathcal{O}_K = (83, T)(83, T-50)(83, T^2-7)(83, T^2+7T+73),$$

where the third factor is a degree-2 prime ideal $\mathfrak{p}$ modulo which T reduces to 67; note that $67^2 \equiv 7 \pmod{83}$. So in this case we have $\mathbb{F}_p(T \bmod \mathfrak{p}) \subsetneq \mathcal{O}_K/\mathfrak{p}$.

Let E be any of the two elliptic curves over $\mathbb{F}_{83}$ having j-invariant 67. By exhaustive search through the possible kernels of order 29, one can check that E admits four distinct endomorphisms squaring to $[-29]$. These appear in the form $\pm\psi, \pm\psi^{\sigma}$, where as in the proof of Proposition 5.27 we use σ to denote the action of the p-power Frobenius. In particular ψ does not anticommute with π_E. Nevertheless, by Deuring's lifting theorem (recalled below), the pair (E, ψ) must arise as the reduction of some CM curve along with an endomorphism Ψ satisfying $\Psi \circ \Psi = [-29]$. (Note: this also applies to the pair (E, ψ^{σ}), which is reflected in the fact that 67 appears as a double root of $H_{\mathbb{Z}[\sqrt{-\ell}]}(T) \bmod 83$.)

Theorem 5.30 (Deuring's lifting theorem)**.** *Let $E/\overline{\mathbb{F}_p}$ be an elliptic curve and let $\alpha \in \mathrm{End}(E)$. There exists an elliptic curve E' over a number field K along with an endomorphism $\alpha' \in \mathrm{End}(E')$ and a prime $\mathfrak{p}$ of K above p at which E' has good reduction, such that $E' \bmod \mathfrak{p}$ is isomorphic to E and such that α' reduces to α modulo $\mathfrak{p}$.*

Proof. See [Lan87, Thm. 14 of Ch. 13]. □

5.5.2 – Proof of Theorem 5.25.

Proof of Theorem 5.25. Using quadratic reciprocity one checks that

$$\left(\frac{-p}{\ell}\right) = 1 \iff \left(\frac{-\ell}{p}\right) = -1,$$

from which we see that p is inert in $\mathbb{Q}(\sqrt{-\ell})$. Hence a curve with CM by $\mathbb{Z}[\sqrt{-\ell}]$ has supersingular reduction modulo p and therefore the $\mathbb{F}_p$-rational roots of the Hilbert class polynomial

$$H_{\mathbb{Z}[\sqrt{-\ell}]}(T) \bmod p$$

should correspond to pairs of quadratic twists in either the floor $\mathscr{Ell}_p(\mathbb{Z}[\sqrt{-p}])$ or the surface $\mathscr{Ell}_p(\mathbb{Z}[(1+\sqrt{-p})/2])$. If $\ell \equiv 3 \pmod 4$, then the same conclusions apply to $\mathbb{Z}[(1+\sqrt{-\ell})/2]$.

As a side note, we remark that $\ell < (p+1)/4$ implies that $y^2 = x^3 \pm x$ does not admit any twisting endomorphisms of norm ℓ, which is easy to elaborate from [McM14, Prop. 3.1]. In view of Proposition 5.27, we therefore see that the $\mathbb{F}_p$-rational roots of the Hilbert class polynomial never include 1728. Hence by Lemma 5.1 there is no ambiguity in what is meant by "pairs of quadratic twists". (Apart from this ambiguity, the theorem remains true under the weaker assumption $\ell \leq (p+1)/4$.)

We first claim that $\mathscr{Ell}_p(\mathbb{Z}[\sqrt{-p}])$ and $\mathscr{Ell}_p(\mathbb{Z}[(1+\sqrt{-p})/2])$ both contain *at most* one such pair E, E^t. Indeed, using Proposition 5.27 we see that E comes equipped with a twisting endomorphism ψ of degree ℓ, which by Lemma 5.12 corresponds to an $\mathbb{F}_p$-rational degree-ℓ isogeny $E \to E^t$. Its kernel is necessarily of the form $E[\mathfrak{l}]$ for some prime ideal $\mathfrak{l}$ above ℓ, i.e., we must have $E^t = [\mathfrak{l}]E$. But then we can solve the vectorization problem $E = [\mathfrak{a}]E_0$: from Lemma 5.17 we get that $[\mathfrak{a}] = [\mathfrak{l}]^{(h(\mathcal{O})-1)/2}$. Since the pair

$$\left\{[\mathfrak{l}]^{(h(\mathcal{O})-1)/2}, \quad [\mathfrak{l}]^{(h(\mathcal{O})+1)/2} = [\bar{\mathfrak{l}}]^{(h(\mathcal{O})-1)/2}\right\}$$

does not depend on the choice of $\mathfrak{l}$, this shows that the pair $\{E, E^t\}$ is fully characterized by ℓ, implying the claim. At the same time this proves (iii).

Next, let us explain why $\mathscr{Ell}_p(\mathbb{Z}[\sqrt{-p}])$ and $\mathscr{Ell}_p(\mathbb{Z}[(1+\sqrt{-p})/2])$ contain *at least* one such pair E, E^t. We remark that this comes for free if $\ell \equiv 3 \pmod 4$, since in this case the Hilbert class polynomials for $\mathbb{Z}[\sqrt{-\ell}]$ and $\mathbb{Z}[(1+\sqrt{-\ell})/2]$ have odd degree and split over $\mathbb{F}_{p^2}$, their roots being supersingular j-invariants: hence they must admit at least one $\mathbb{F}_p$-rational root. In general, we can reverse the reasoning from the previous paragraph and *define* E, E^t using (5.4), for some choice of prime ideal $\mathfrak{l}$ above ℓ. In particular $E^t = [\mathfrak{l}]E$, which provides us with an $\mathbb{F}_p$-rational degree-ℓ isogeny $\varphi\colon E \to E^t$, which we use to construct an endomorphism $\psi = \tau_{E^t} \circ \varphi$ of E that is not $\mathbb{F}_p$-rational. In contrast, it is easily verified that $\psi \circ \psi$ is $\mathbb{F}_p$-rational. Therefore the minimal polynomial of ψ cannot admit a non-zero linear term, i.e., $\psi \circ \psi$ must be a scalar-multiplication map, necessarily of the form $[\pm\ell]$. By Deuring's lifting theorem E can be lifted to an elliptic curve over a number field carrying an endomorphism Ψ whose reduction modulo a suitable prime above p yields ψ. Since Ψ must belong to an imaginary quadratic ring we see that $\Psi \circ \Psi = [-\ell]$ as wanted.

Altogether this proves (i), while for (ii) it leaves us with the task of showing that if $\ell \equiv 3 \pmod 4$, then the unique $\mathbb{F}_p$-rational root of

$$H_{\mathbb{Z}[(1+\sqrt{-\ell})/2]}(T) \bmod p$$

corresponds to a pair of elliptic curves $\{E, E^t\}$ with endomorphism ring $\mathbb{Z}[\sqrt{-p}]$. Equivalently, we need to show that such curves admit a unique $\mathbb{F}_p$-rational point of order 2, rather than three such points. To this end, let $P \in E$ be an $\mathbb{F}_p$-rational point of order 2 and let φ be the endomorphism of E corresponding to $(1+\sqrt{-\ell})/2$. Proposition 5.27 implies that $\pi_E \circ \varphi = \overline{\varphi} \circ \pi_E$, where $\overline{\varphi}$ corresponds to $(1-\sqrt{-\ell})/2$. But then clearly $(\varphi + \overline{\varphi})(P) = P \neq \infty$, which implies that $\overline{\varphi}(P) \neq \varphi(P)$ and therefore that $\pi_E(\varphi(P)) \neq \varphi(P)$, i.e., $\varphi(P)$ is a non-rational point of order 2. This concludes the proof. □

Remark 5.31. *The above ideas can be generalized to locate reductions mod p of CM curves carrying an endomorphism Ψ such that $\Psi \circ \Psi = [-\ell_1\ell_2\cdots\ell_s]$, where the $\ell_i \leq (p+1)/4$ are distinct odd primes for which*

$$\left(\frac{-\ell_1\ell_2\cdots\ell_s}{p}\right) = -1. \tag{5.6}$$

We did not elaborate this in detail, but assume for instance that each ℓ_i splits in $\mathbb{Q}(\sqrt{-p})$; note that this implies (5.6). Letting $\mathcal{O} \in \{\mathbb{Z}[\sqrt{-p}], \mathbb{Z}[(1+\sqrt{-p})/2]\}$, one expects that 2^{s-1} pairs E, E^t in $\mathcal{E}\ell\ell_p(\mathcal{O})$ can be obtained as the reduction mod p of an elliptic curve carrying such an endomorphism Ψ. Fixing for each $i = 1, 2, \ldots, s$ a prime ideal $\mathfrak{l}_i \subseteq \mathcal{O}$ of norm ℓ_i, these pairs are characterized by

$$E^t = [\mathfrak{l}_1][\mathfrak{l}_2]^{e_2}[\mathfrak{l}_2]^{e_3} \cdots [\mathfrak{l}_s]^{e_s} E$$

with $(e_2, e_3, \ldots, e_s) \in \{\pm 1\}^{s-1}$. As before, an application of Lemma 5.17 then solves the corresponding vectorization problems.

Code. A proof-of-concept [Sage] script demonstrating some of the algorithms in Section 5.4 is available at `https://yx7.cc/files/quat.sage`.

Chapter 6

Quantum equivalence of DLP and CDH for group actions

This chapter is an updated version of the preprint *Quantum equivalence of the DLP and CDHP for group actions* [GPSV18] authored jointly with Steven Galbraith, Benjamin Smith, and Frederik Vercauteren.

6.1 — Introduction

In their seminal 1976 paper [DH76], Diffie and Hellman conjectured that breaking their new key exchange protocol (in the sense of computing the shared secret from the public keys) was as hard as computing discrete logarithms. This polynomial-time equivalence was later proven (assuming knowledge of suitable auxiliary algebraic groups of smooth order) for all groups by Maurer [Mau94], based on earlier results of den Boer [dB88] covering certain special cases.

In this short chapter, we prove an unconditional reduction between the analogous problems for group actions in the quantum setting. This result has important implications for the quantum security of the CSIDH key-exchange scheme (Chapter 3).

Cryptographic group actions. Couveignes in 1997 introduced the notion of a *hard homogeneous space* [Cou06], essentially a free and transitive finite abelian group action $* : G \times X \to X$ which is efficiently computable while other computational problems are hard. In Couveignes' terminology, these problems are *vectorization* and *parallelization*, named by analogy with the archetypical example of a homogeneous space: a vector space acting on affine space by translations (cf. Figure 6.1). The vectorization problem is: given x and $g*x$ in X, compute $g \in G$. The parallelization problem is: given x, $g*x$, and $h*x$ in X, compute $gh*x \in X$. The group-exponentiation analogues of these problems are the *discrete logarithm problem* (DLP) and *computational Diffie–Hellman problem* (CDH); see Section 2.1.2.

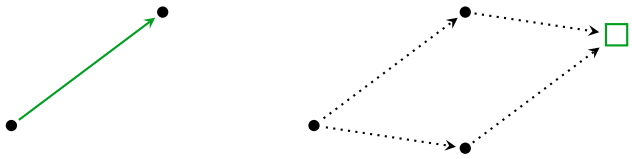


Figure 6.1: The vectorization and parallelization problems.

For twenty years, there was little interest in the hard-homogeneous-spaces framework, since all known (conjectural) instantiations were either painfully slow in practice or already captured by the group-exponentiation point of view. However, interest in these one-way group actions

has reemerged in the more recent past due to the current focus on post-quantum cryptography, where group-exponentiation Diffie–Hellman is broken in polynomial time by Shor's algorithm, but group *actions* are not.

Throughout, let $G \times X \to X$ denote a homogeneous space. In analogy with CSIDH, we write $\mathfrak{a}, \mathfrak{b}, \ldots$ for elements of the group G, and E for elements of the set X.

DLP–CDH reductions. Just like in the classical group-exponentiation setting, it is evident that parallelization reduces to vectorization: recover $\mathfrak{a}$ from $\mathfrak{a} * E$, then apply $\mathfrak{a}$ to $\mathfrak{b} * E$ to obtain $\mathfrak{a}\mathfrak{b} * E$. Traditionally, the other direction is much more subtle. The reduction essentially relies on the existence of auxiliary algebraic groups of smooth group order over $\mathbb{F}_{q_i}$, where the q_i are the prime divisors of the order of the group in which the DLP and CDH are defined. The first result was given by den Boer [dB88] who showed the DLP and CDH to be equivalent in $\mathbb{F}_p^\times$ when p is a prime such that the Euler totient $\varphi(p-1)$ is smooth. The auxiliary groups are simply $\mathbb{F}_{q_i}^\times$ for each prime divisor $q_i \mid p-1$, and the smoothness assumption implies that the DLP in each $\mathbb{F}_{q_i}^\times$ is easy. Maurer [Mau94] generalized this result to arbitrary cyclic groups G, assuming that for each large prime divisor q_i of $|G|$, there exists an efficiently constructible elliptic curve $E/\mathbb{F}_{q_i}$ with smooth group order. On classical computers, these reductions do not apply in the group-action setting [Smi18, §11].

However, we show that there exists a polynomial-time *quantum* reduction from the vectorization to the parallelization problem for group actions without relying on any extra assumptions, thereby proving the polynomial-time equivalence of both problems in the quantum setting.

6.2 — The reduction

Let π be an algorithm that reliably solves the parallelization problem for a homogeneous space $G \times X \to X$. In other words, π takes $\mathfrak{a} * E$ and $\mathfrak{b} * E$ and returns $\mathfrak{a}\mathfrak{b} * E$. We show that quantum access to a quantum circuit that computes π allows one to solve the vectorization problem in $G \times X \to X$ in polynomial time.

Lemma 6.1. *Given an element $\mathfrak{a} * E \in X$ and access to a parallelization oracle π, one can compute $\mathfrak{a}^n * E$ for any integer $n \geq 0$ using $\Theta(\log n)$ queries to π.*

Proof. We perform double-and-add in the "implicit group" [Smi18] of exponents, using the oracle $\pi\colon (\mathfrak{a}^x * E, \mathfrak{a}^y * E) \mapsto \mathfrak{a}^{x+y} * E$ for addition and doubling. □

Theorem 6.2. *Given a perfect (classical or) quantum parallelization algorithm π, one can construct a quantum algorithm that recovers $\mathfrak{a}$ from elements E and $\mathfrak{a} * E$ in X in polynomial time.*

Proof. From the public description of G, one can compute the group structure $\mathbb{Z}/d_1 \times \cdots \times \mathbb{Z}/d_r$ of G together with a basis $\{\mathfrak{g}_1, \ldots, \mathfrak{g}_r\} \subseteq G$ in quantum polynomial time using Boneh–Lipton's [BL95] or Kitaev's [Kit96] generalisation of Shor's algorithm [Sho97a]; see Theorem 2.79.

Now, for $\underline{x} \in \mathbb{Z}^r$, write $\mathfrak{g}^{\underline{x}} = \prod_{i=1}^r \mathfrak{g}_i^{x_i}$ and define

$$\begin{aligned} f\colon\ \mathbb{Z}^r \times \mathbb{Z} &\longrightarrow X \\ (\underline{x}, y) &\longmapsto \mathfrak{g}^{\underline{x}} * (\mathfrak{a}^y * E)\,, \end{aligned}$$

where $\mathfrak{a}^y * E$ is computed using Lemma 6.1.[1] Using the circuit for π one can construct a quantum circuit that computes f. The function f is clearly a group homomorphism (to the implicit group

[1] For negative y, one may generally take a positive representative modulo the exponent $\operatorname{lcm}(d_1, \ldots, d_r)$ of G. This is not needed in the CSIDH setting, since $\mathfrak{a}^{-1} * E$ can be obtained by merely quadratic-twisting $\mathfrak{a} * E$.

on X), hence defines an instance of the hidden-subgroup problem with respect to its kernel, i.e., the lattice

$$L = \{(x, y) \in \mathbb{Z}^r \times \mathbb{Z} : \mathfrak{g}^{\underline{x}+y\underline{v}} = 1 \in G\},$$

where $\underline{v} \in \mathbb{Z}^r$ is any vector such that $\mathfrak{a} = \mathfrak{g}^{\underline{v}}$.[2] This (abelian) hidden-subgroup problem can be solved in polynomial time again using Shor's algorithm. Finally, any vector in L of the form $(\underline{x}, 1)$ satisfies $\mathfrak{g}^{-\underline{x}} = \mathfrak{a}$, hence yields a representation of $\mathfrak{a}$. □

Remark 6.3. *It is unclear how to perform the reduction above when π is only guaranteed to succeed with non-negligible probability α, meaning that the probability over all triples $(E, \mathfrak{a} * E, \mathfrak{b} * E) \in X^3$ that the oracle outputs $\mathfrak{ab} * E$ is at least α.*

In the classical discrete-logarithm setting, it is straightforward to amplify the success probability of CDH oracles using random self-reduction of problem instances [MW96; Sho97b]: one computes lists of possible values of g^{ab} by blinding the inputs and unblinding the outputs, and uses majority vote to determine the correct result. Any exponentially small failure probability can be achieved using polynomially many queries [Sho97b, § 5].

In the group-action setting, however, blinding does not work: The results cited above use a blinding map of the form $g^a \mapsto (g^a)^x g^y = g^{ax+y}$, which relies on the fact that we can multiply two public keys. But the best we can do for a mere group action is to translate the inputs by random elements, i.e., blind as $\mathfrak{a} * E \mapsto \mathfrak{x} * (\mathfrak{a} * E)$ with a random $\mathfrak{x} \in G$, which is insufficient: For example, if $\mathcal{A}$ is a perfect CDH oracle, then the oracle $\mathcal{B}$ that returns the output of $\mathcal{A}$ either unmodified (with some probability ε), or shifted by a fixed element $\mathfrak{z} \in G$, is entirely unaffected by blinding and hence cannot be amplified using this idea. Thus, we must unfortunately leave the case of imperfect oracles as an open problem.

6.3 — Implications for CSIDH

Group actions are a useful, simple abstraction for reasoning about CSIDH and other isogeny-based cryptosystems where the endomorphism rings of the underlying curves are commutative (see Chapters 3, 4). In each of these cryptosystems, the group is $G = \mathrm{cl}(\mathcal{O})$, the ideal-class group of a maximal order $\mathcal{O}$ in some quadratic imaginary field, and the set X is comprised of isomorphism classes of elliptic curves E (over a fixed finite field $\mathbb{F}_q$) such that the endomorphism ring $\mathrm{End}(E)$ is isomorphic to $\mathcal{O}$. The action $G \times X \to X$ is given by $(\mathfrak{a}, E) \mapsto \mathfrak{a} * E := E'$ where E' is the codomain of an isogeny $\phi_{\mathfrak{a}} \colon E \to E'$ with kernel $E[\mathfrak{a}]$, i.e., the finite subgroup of E annihilated by all of the elements of $\mathfrak{a} \subseteq \mathrm{End}(E)$. Public keys are instances $(E, \mathfrak{a} * E)$ of the vectorization problem in this homogeneous space. In CSIDH, the Diffie–Hellman secret shared between Alice and Bob, with public keys $(E, \mathfrak{a} * E)$ and $(E, \mathfrak{b} * E)$, is $\mathfrak{ab} * E$. Recovering the shared secret from the public keys is therefore solving a parallelization problem.

At first glance, then, Theorem 6.2 would appear to imply a polynomial-time equivalence between the Diffie–Hellman problem for CSIDH and recovering CSIDH secrets from public keys. However, this ignores an important subtlety: It is not known how to compute the action of *arbitrary* ideals $\mathfrak{a} \subseteq \mathcal{O}$ in polynomial time. CSIDH gets around this issue by using secret keys of the form $\mathfrak{a} = \prod_i \mathfrak{l}_i^{e_i}$, where $\underline{e} = (e_1, \ldots, e_n) \in \mathbb{Z}^n$ are short exponent vectors and the $\mathfrak{l}_i$ are a fixed set of "small" ideals whose action is efficient.[3] Computationally, this manifests in a linear cost in the 1-norm $\|(e_1, \ldots, e_n)\|_1$ for evaluating the action of $\mathfrak{a}$.

[2] Note that $\underline{v}$ is only defined modulo the relation lattice $R = d_1\mathbb{Z} \oplus \cdots \oplus d_r\mathbb{Z}$ of G with respect to $\mathfrak{g}_1, \ldots, \mathfrak{g}_r$. The choice of $\underline{v}$ does not matter since $L \supseteq R \oplus \{0\}$.

[3] Another way to view this is as an action of the group $(\mathbb{Z}^n, +)$.

When applied to CSIDH, the algorithm in Theorem 6.2 will return *some* presentation of the secret ideal class $[\mathfrak{a}]$ as a product of generators $\mathfrak{g}_i$ (which can be chosen as the ideals $\mathfrak{l}_i$), but in general its action is not known to be computable in polynomial time: the exponent vector $\underline{e}$ may have large norm. We can reduce the norm of $\underline{e} \in \mathbb{Z}^n$ by solving a close(st)-vector problem for the relation lattice $\ker(\mathbb{Z}^n \to \mathrm{cl}(\mathcal{O}))$. But asymptotically, polynomial-time lattice reduction algorithms cannot guarantee that the output will have norm small enough to ensure that the resulting group action is computable in polynomial time.

However, this is not a problem for practical key sizes used in CSIDH. Since the dimensions n used in CSIDH are rather small (e.g. the CSIDH-512 parameter set from [Cas+18] uses 74 prime ideals), an efficient lattice-reduction algorithm such as BKZ [SE94] with moderate block size suffices to obtain highly practical results. For example: reducing a random relation lattice of dimension 74 using BKZ with block size 20 yields exponent vectors only 8 times longer than normal CSIDH-512 private keys. Therefore, our reduction is efficient in the CSIDH context for some practical parameter sizes, despite the aforementioned asymptotical annoyances.

Chapter 7

Weak instances of SIDH variants from improved torsion-point attacks

This chapter is an updated version of the preprint *Weak instances of SIDH variants under improved torsion-point attacks* [Kut+20] authored jointly with Péter Kutas, Chloe Martindale, Christophe Petit, Victoria de Quehen, and Katherine E. Stange.

7.1 — Introduction

In recent years, isogeny-based cryptography has been receiving increased interest, partly due to the fact that isogeny-based key exchange has the smallest key sizes of all current post-quantum candidates while still performing at a reasonable speed. The *Supersingular Isogeny Diffie–Hellman* protocol, or *SIDH*, was the first practical isogeny-based key-exchange protocol, proposed by Jao and De Feo in 2011 [JD11]. The most obvious way to attack SIDH is to solve the following problem:

Problem 7.1. *For a large prime p and smooth coprime integers A and B, given two supersingular elliptic curves $E_0/\mathbb{F}_{p^2}$ and $E/\mathbb{F}_{p^2}$ connected by a degree-A isogeny $\phi\colon E_0 \to E$, and given the action of ϕ on the B-torsion of E_0, recover ϕ.*[1]

Notice that this problem provides the attacker with more information than the 'pure' isogeny problem, where the goal is to find an isogeny between two given curves without any further hints or restrictions. The best known way to break SIDH by treating it as a pure isogeny problem is a claw-finding approach on the isogeny graph having both classical and quantum complexity $O(\sqrt{A} \cdot \mathrm{polylog}(p))$ [JS19].[2] However, Problem 7.1 could be easier than finding isogenies in general, and indeed a line of work started in [Pet17], continued in [Bot+19], and now also with this chapter, suggests that this may hold at least for some instantiations.

The additional torsion-point information clearly does aid *active* attackers: In 2016, Galbraith, Petit, Shani, and Ti [GPST16] presented an active attack against SIDH that sends key-exchange messages with manipulated torsion points and detects whether the key exchange succeeds; this allows recovering the secret within $O(\log A)$ queries. To mitigate this attack, [GPST16] proposes using the Fujisaki–Okamoto transform, which generically renders a CPA-secure public-key encryption scheme CCA-secure, and therefore thwarts those so-called *reaction attacks*. The result of applying (a variant of) the Fujisaki–Okamoto transform to SIDH is called *Supersingular Isogeny Key Encapsulation*, or *SIKE* [Jao+19] for short. It is the only isogeny-based submission to NIST's

[1] These constraints do not necessarily uniquely determine ϕ, but any efficiently computable isogeny from E_0 to E is usually enough to recover the SIDH secret [GPST16]. Moreover, ϕ is unique whenever $B^2 > 4A$. [MP19, § 4]

[2] Note that the naïve meet-in-the-middle approach has prohibitively large memory requirements. Collision finding à la van Oorschot–Wiener thus performs better in practice, although its time complexity is worse in theory [Adj+18].

standardization project for post-quantum cryptography [NIST16] and is currently a Round 3 'Alternate Candidate for Public-key Encryption and Key-establishment Algorithms'.

A particular choice made in SIKE is that one of the two curves, the 'starting curve' E_0, is a special curve: It is defined over $\mathbb{F}_p$ and has small-degree non-scalar endomorphisms, both of which are very rare properties within the set of all supersingular curves defined over $\mathbb{F}_{p^2}$. On its own, this fact does not seem to have any negative security implications for SIKE, but [GPST16] shows that given an explicit description of *both* curves' endomorphism rings, it is (under reasonable heuristic assumptions) possible to recover the secret isogeny; hence, breaking SIKE is no harder than computing endomorphism rings of supersingular elliptic curves in some sufficiently explicit representation.

In 2017, Petit [Pet17] introduced a method to solve some instances of Problem 7.1 based on endomorphisms of the special starting curve. It uses the given action of the secret isogeny on a large torsion subgroup to recover the isogeny itself, giving a *passive* heuristic polynomial-time attack on non-standard variants of SIDH satisfying $B > A^4$ and $A > p$. However, in practice both A and B are taken to be about the size of $\sqrt{p}$ for efficiency reasons; thus this attack does not apply to the SIKE parameters.

7.1.1 – Contributions. We improve upon and extend Petit's 2017 *torsion-point attacks* [Pet17] in several ways. We argue heuristically in Section 7.3 that the imbalance conditions can be relaxed to $[B > A^2$ and $A > p]$ or $[B > A^3$ and $A > p^{1/2}]$, and that furthermore allowing for arbitrarily large B/A gives an attack for $AB \approx p$. We also show that even a mild imbalance of parameters may lead to a heuristic improvement over the generic meet-in-the-middle or claw-finding attack, and we show the relationship between the extremity of the imbalance and the estimated complexity of the torsion-point attack.

Recall also that in SIKE, the starting curve E_0 is taken to be the curve with j-invariant 1728.[3] In Section 7.4 we introduce the notion of a 'trapdoor' curve, which allows breaking or reducing the security of an SIDH key exchange when used as the starting curve: We give a heuristic polynomial-time torsion-point attack on SIDH variants using a trapdoored starting curve when $B > A^2$ (note the absence of a condition on p), and an attack of classical complexity $\tilde{O}(p^{2/5})$ and quantum complexity $\tilde{O}(p^{1/8})$ on SIDH variants using a trapdoored starting curve with $B \approx A \approx p^{1/2}$. Note that this is as in SIKE, but starting from a trapdoored starting curve instead of the curve with j-invariant 1728; thus, such curves could potentially be utilized as backdoors. We also give the relationship between the extremity of the imbalance of the parameters and the complexity of the torsion-point attack applied to this case of trapdoored starting curves, and argue that we expect there to be exponentially many trapdoor curves. Finally, we show that it is possible to construct special primes p, together with an appropriate A and B, for which torsion-point attacks are especially effective, even when using balanced parameters $A \approx B$ and/or using a starting curve with j-invariant 1728.

We emphasize that none of our attacks apply to the NIST candidate SIKE: for each attack described in this chapter, at least one aspect of SIKE needs to be changed (e.g., the balance of the degrees of the secret isogenies, the starting curve, or the base field prime). There are, however, SIDH variants in the literature for which there are realistic parameter sets where our attacks may have practical impact.

Of the existing proposals in the literature, our attacks are the most effective on the recent proposal B-SIDH [Cos20], presented at ASIACRYPT 2020. As discussed in more detail in Section 7.3.3, using parameters $A \approx B \approx p$ as suggested in [Cos20] may affect the security of the

[3] One can also take a neighbour, but this does not affect the security analysis.

scheme due to our torsion-point attacks: Our quantum attack heuristically achieves a complexity of $\tilde{O}(p^{1/3})$, which is asymptotically the best known attack under the assumptions (1) that the cost of breaking B-SIDH even after solving the generic isogeny problem is still greater than $\tilde{O}(p^{1/3})$, and (2) that Tani's quantum claw-finding algorithm [Tan07] has complexity higher than $\tilde{O}(p^{1/3})$. The latter appears to be the case [JS19]; the former is unclear. Note that in any case, none of this violates the rather conservative security claims of [Cos20].

As a second example, we consider the recent n-party group key exchange proposal [AJJS19]; this can be interpreted for cryptanalysis purposes as an unbalanced (two-party) SIDH instance with $B \approx A^{n-1}$ and $AB \approx p$. While we currently cannot break the case $AB \approx p$, the torsion-point attacks we describe in Section 7.3 give rise to, for example, a quantum attack of complexity $O(A^{0.41} \cdot \mathrm{polylog}(p))$ on a 3-party key exchange with $AB \approx p^{1.15}$, a 10-party key exchange with $AB \approx p^{1.04}$, or a 100-party key exchange with $AB \approx p^{1.004}$; these kinds of instantiations are perfectly plausible by combining the group key exchange with ideas from B-SIDH. Furthermore, the attack variant for trapdoored starting curves is heuristically classical polynomial-time for three or more parties.

7.1.2 – Comparison to earlier work. Bottinelli, de Quehen, Leonardi, Mosunov, Pawlega, and Sheth [Bot+19] also gave an improvement on the balance from Petit's 2017 paper [Pet17]. Our work overlaps with theirs (only) in Corollary 7.5, and the only similarity in techniques is in the use of "triangular decomposition" [Bot+19, § 5.1]. Unfortunately, we have not found a way to combine the two methods. Moreover, our results go beyond [Bot+19] in several ways: we consider multiple trade-offs by allowing for superpolynomial attacks, as well as considering other starting curves and base-field primes.

Acknowledgements. Thanks to Daniel J. Bernstein for his help with Section 7.3.5, and to John Voight for answering a question of ours concerning Section 7.4.3. We would also like to thank the anonymous reviewers of an earlier version for their useful feedback.

7.2 — Preliminaries

7.2.1 – Notation. Throughout this chapter, we will neglect factors polynomial in $\log p$. As such, from this point on we will abbreviate $O(g \cdot \mathrm{polylog}(p))$ as $O^*(g)$.[4] Similarly, 'smooth' without further qualification always means $\mathrm{polylog}(p)$-smooth. "Polynomial time" without explicitly mentioning the variables means "polynomial in the representation size of the input" — usually the logarithms of integers. We let $\mathcal{B}_{p,\infty}$ denote the quaternion algebra ramified at p and ∞, for which we use a fixed $\mathbb{Q}$-basis $\langle 1, \mathbf{i}, \mathbf{j}, \mathbf{ij} \rangle$ such that $\mathbf{j}^2 = -p$ and $\mathbf{i}$ is a nonzero endomorphism of minimal norm with $\mathbf{ij} = -\mathbf{ji}$.

7.2.2 – Quantum computation cost assumptions. In the context of NIST's post-quantum cryptography standardization process [NIST16], there is a significant ongoing effort to estimate the quantum cost of fundamental cryptanalysis tasks in practice. In particular, while it seems well-accepted that Grover's algorithm provides a square-root quantum speedup, the complexity of the claimed cube-root claw-finding algorithm of Tani [Tan07] has been disputed by Jaques and Schanck [JS19], and the topic is still subject to ongoing research [JS20].

Several attacks we present in this chapter use claw-finding algorithms as a subroutine, and the state-of-the-art algorithms against which we compare them are also claw-finding algorithms.

[4] Each occurrence of $\mathrm{polylog}(p)$ is shorthand for a concrete, fixed polynomial in $\log p$. (The notation is not meant to imply that all instances of $\mathrm{polylog}(p)$ be the same.)

We stress, however, that the insight provided by our attacks is independent of the choice of the quantum computation model. For concreteness we chose the RAM model studied in detail by Jaques and Schanck in [JS19], in which it is argued that quantum computers do not seem to offer a significant speedup over classical computers for the task of claw-finding. Adapting our various calculations to other existing and future quantum computing cost models, in particular with respect to claw-finding, is certainly possible.

7.2.3 – The Supersingular Isogeny Diffie–Hellman protocol. We give a high-level description of SIDH [JD11]. The public parameters of the system are two smooth coprime numbers A and B, a prime p of the form $p = ABf - 1$, where f is a small cofactor, and a supersingular elliptic curve E_0 defined over $\mathbb{F}_{p^2}$ together with points $P_A, Q_A, P_B, Q_B \in E_0$ such that $E_0[A] = \langle P_A, Q_A\rangle$ and $E_0[B] = \langle P_B, Q_B\rangle$.
The protocol then proceeds as follows:

1. Alice chooses a random cyclic subgroup of $E_0[A]$ as $G_A = \langle P_A + [x_A]Q_A\rangle$ and Bob chooses a random cyclic subgroup of $E_0[B]$ as $G_B = \langle P_B + [x_B]Q_B\rangle$.

2. Alice computes the isogeny $\phi_A : E_0 \to E_0/\langle G_A\rangle =: E_A$ and Bob computes the isogeny $\phi_B : E_0 \to E_0/\langle G_B\rangle =: E_B$.

3. Alice sends the curve E_A and the two points $\phi_A(P_B), \phi_A(Q_B)$ to Bob. Similarly, Bob sends $\big(E_B, \phi_B(P_A), \phi_B(Q_A)\big)$ to Alice.

4. Alice and Bob use the given torsion points to obtain the shared secret curve $E_0/\langle G_A, G_B\rangle$: Alice computes $\phi_B(G_A) = \phi_B(P_A) + [x_A]\phi_B(Q_A)$ and uses the fact that $E_0/\langle G_A, G_B\rangle \cong E_B/\langle\phi_B(G_A)\rangle$. Bob proceeds analogously.

   (Publishing the action of the secret isogeny on public points can be considered *the* core idea behind SIDH: Alice needs $\phi_B(G_A)$ to complete the key exchange, but Bob must keep ϕ_B secret and Alice must keep G_A secret. Handing out the action of ϕ_B on a publicly known group that *contains* the secret G_A is a clever workaround for this problem.)

The SIKE proposal [Jao+19] suggests various choices of (p, A, B) depending on the targeted security level: All parameter sets use powers of two and three for A and B, respectively, with $A \approx B$ and $f = 1$. For example, the smallest parameter set suggested in [Jao+19] uses the prime $p = 2^{216} \cdot 3^{137} - 1$. Other constructions belonging to the SIDH 'family tree' of protocols use different types of parameters [Cos20; AJJS19; SGP19].

We may assume knowledge of $\mathrm{End}(E_0)$: The only known way to construct supersingular elliptic curves is by reduction of elliptic curves with CM by a small discriminant (which implies small-degree endomorphisms: see Chapter 5 or [LB20]), or by isogeny walks starting from such curves (where knowledge of the path reveals the endomorphism ring, thus requiring trusted setup). A common choice when $p \equiv 3 \pmod 4$ is $j(E_0) = 1728$ or a small-degree isogeny neighbour of that curve [Jao+19].

7.2.4 – Petit's torsion-point attacks. "Traditional" attacks on SIDH attempt to solve the general isogeny problem or reduce isogeny finding to computing endomorphism rings. However, SIDH is based on Problem 7.1 introduced above, in which an adversary also gets the action of the secret isogeny on the B-torsion of the starting curve E_0, which is the basis of another attack strategy due to Petit [Pet17].

Remark 7.2. *Problem 7.1 is a slight generalization of the Computational Supersingular Isogeny (CSSI) Problem introduced in [JD11]. Here we do not require A and B to be prime powers (just smooth) and*

we do not require p to have a special form. We remark that some instances of Problem 7.1 require super-polynomial space, as the extension fields required to represent $\ker(\phi)$ and $E_0[B]$ generally have degree at least linear in A and B. Broadly speaking, the interesting cases are the 'efficient' instantiations where computing ϕ and its action on $E_0[B]$ takes time and space polynomial in $\log p$, $\log A$, and $\log B$.

We outline Petit's approach [Pet17] to solve some cases of Problem 7.1. The main steps are:

1. Compute a non-scalar endomorphism $\theta \in \mathrm{End}(E_0)$ and integers $d, e \in \mathbb{Z}$ such that $\deg(\phi \circ \theta \circ \hat{\phi} + [d]) = Be$ with e smooth and relatively small.
2. Recover an efficient representation of $\tau = \phi \circ \theta \circ \hat{\phi} + [d]$ using the fact that the action on the B-torsion of ϕ, hence of τ, is known.
3. Compute $\ker(\tau - [d]) \cap E[A]$ and from that compute ϕ itself.

Notice that step 1 can be done as precomputation as it only depends on E_0, but not on the particular public key under attack. (The degree of τ depends on the degree of ϕ, but not on which particular degree-A isogeny ϕ happens to be.)

First we address steps 2 and 3. In step 2, the endomorphism τ can be decomposed into isogenies $\eta \circ \psi$, where $\deg(\psi) = B$ and $\deg(\eta) = e$. The isogeny ψ can be computed since θ is known and we know the action of ϕ (and thus of $\hat{\phi}$) on $E_0[B]$ (resp. $E[B]$). Then η can be found by meet-in-the-middle using $O^*(\sqrt{e})$ operations. In step 3 we have $\ker(\hat{\phi}) \subseteq \ker(\tau - [d]) \cap E[A]$, and in fact they are usually equal. They are not equal if and only if $\ker(\tau - [d])$ contains $E[M]$ for some divisor M of A; it is shown in [Pet17, Section 4.3] how to resolve this issue.

The complexity of the algorithm clearly depends on the size of e, thus the efficiency of the algorithm is dependent on the effectiveness of step 1. While the endomorphism ring of E_0 is usually known, it is not obvious how to find an element θ as above. For example, in SIKE, the starting curve has j-invariant 1728,[5] whose endomorphism ring is (up to small denominators) generated by Frobenius $\pi : (x, y) \mapsto (x^p, y^p)$ and the automorphism $\iota : (x, y) \mapsto (-x, \sqrt{-1} \cdot y)$, hence step 1 reduces to solving the norm equation

$$A^2(pa^2 + pb^2 + c^2) + d^2 = Be; \tag{7.1}$$

the left-hand-side of this equation is just the degree of $\tau = \phi \circ \theta \circ \hat{\phi} + [d]$ when $\theta = a\iota\pi + b\pi + c\iota$. Petit [Pet17] gives an algorithm to solve Equation (7.1) in the regime $A > p$ and $B > A^4$: The main idea is to choose e such that Be is a square modulo A^2, solve for d modulo A^2, and then solve for c modulo p. What remains is the equation $a^2 + b^2 = \frac{Be - d^2 - c^2A^2}{pA^2}$, which can be solved efficiently by Cornacchia's algorithm if the right-hand side is efficiently factorizable; else the procedure is restarted with a new choice of e. Under the conditions $A > p$ and $B > A^4$, this algorithm can heuristically be expected to find a suitable solution in polynomial time. This already suggests that there exist parameters for which Problem 7.1 is easier than the general supersingular isogeny problem.

7.3 — Improved torsion-point attacks

In this section we generalize and improve the torsion-point attacks from Petit's 2017 paper [Pet17]. Our setup is as in [Pet17]: we study SIDH instances in which Alice and Bob use the starting curve

[5] Note that the newest version of [Jao+19] changed the starting curve to a 2-isogenous neighbour of $j = 1728$, but this does not affect the asymptotic complexity of the (in fact, any) attack and thus we will stick with the original starting curve for simplicity.

$E_0/\mathbb{F}_p : y^2 = x^3 + x$, with p is a prime congruent to $3 \pmod 4$,[6] Alice's secret isogeny has degree $A = p^\alpha$, and Bob's secret isogeny has degree $B = p^\beta$. SIKE consists of such instances with $\alpha \approx \beta \approx 1/2$, but in our analysis we will allow α and β to vary. The case $\alpha+\beta > 1$ may seem artificial to readers mostly familiar with traditional SIDH or SIKE [JD11; Jao+17], but note that [Pet17] and B-SIDH [Cos20] propose cryptographically interesting variants of SIDH with such parameters; furthermore, studying these cases helps our understanding of the case $\alpha + \beta \approx 1$, cf. Figure 7.1. We assume without loss of generality that $A \leq B$ and that we are attacking Alice's key, i.e., the secret isogeny is of degree A and we are given the action of ϕ on the B-torsion of E_0.

Caution: (In)equalities such as $\alpha + \beta > 1$ are not to be interpreted as sharp bounds, but as parameter *regimes*, and the claims may only hold asymptotically.

Petit's 2017 classical, polynomial-time attack [Pet17] requires unbalanced parameters, unlike those in SIKE, namely $\beta > 4\alpha > 4$. In this section, we argue that even a mild imbalance between α and β may result in a better (quantum) attack than the generic claw-finding/meet-in-the-middle algorithm, thus far considered to be the best attack for any parameters not broken by [Pet17] or [Bot+19]. We also reduce the degree of imbalance needed for the polynomial-time algorithm to apply via a different method to [Bot+19]. Once more, we stress that these results are based on heuristic assumptions and ignore factors polynomial in $\log p$. The results of this section are summarized in Figures 7.1 and 7.2, which will be justified by Theorem 7.8 and Heuristic Result 7.10. Note that Figure 7.1 illustrates a trade-off: The closer A and B are to each other, the bigger their product AB must be for the attacks to apply, and conversely reducing AB requires a stronger imbalance. Figure 7.1 shows that allowing for extremely unbalanced parameters $B \gg A$, we approach an attack on $AB \approx p$ as in (for instance) SIKE.

Our results suggest that the choice $\alpha \approx \beta \approx 1/2$ made in SIKE also minimizes the applicability of the torsion-point attack avenue. As we will show in Section 7.4, it is possible to improve on the meet-in-the-middle/claw-finding complexity for balanced parameters with a different starting curve, but with SIKE's starting curve it does not seem possible to get an attack via this method. However, since any imbalance can lead to a lower attack complexity, our results may have an impact on SIDH variants such as B-SIDH [Cos20] and group key exchange [AJJS19]; see Figure 7.2.

Remark 7.3. *A couple of notes on the choices made in Figure 7.1:*

- *Algorithms with complexity polynomial in* $\log p$ *correspond to* $\mathscr{C} = 0$.
- *The complexity of the attack is measured as a power of* A*, the degree of Alice's secret isogeny. Together with our assumption that* $A \leq B$*, this allows for easy comparison with the 'generic attack', i.e., classical or quantum claw finding.*
- *As discussed in Section 7.2.2, we use the RAM model studied in detail for claw-finding by Jaques and Schanck in [JS19] for our quantum computation model. For the classical attack, we compare against the basic meet-in-the-middle algorithm. Therefore, in this chapter, we take the complexity of both classical* and *quantum claw-finding to be* $O^*(A^{1/2})$.

7.3.1 – Improved balance of the polynomial-time attack. Among many other tradeoffs, our work improves from a balance of $B > A^4 > p^4$ as in [Pet17] to a balance of $B > A^3 > p^{3/2}$ or $B > A^2 > p^2$. This improvement comes from one simple trick, explained below.

[6] More generally, these attacks apply for any 'special' starting curve in the sense of [KLPT14].

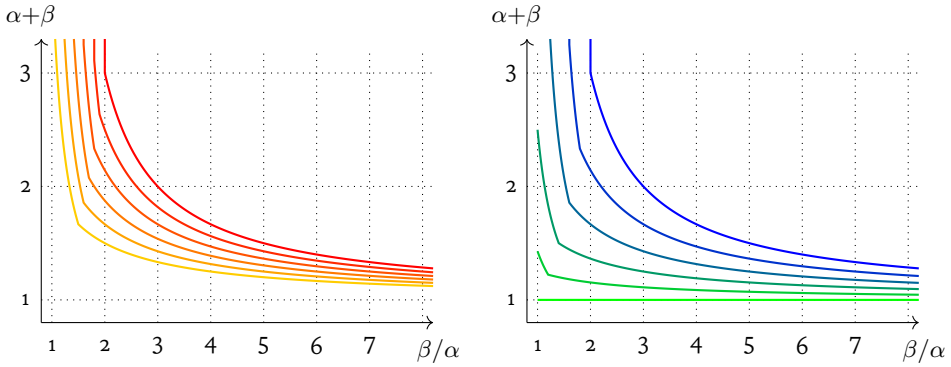


Figure 7.1: Possible choices of (α, β) allowing for a classical attack (left) of complexity $O^*(A^{\mathscr{C}})$ for $\mathscr{C} = 0.5, 0.4, 0.3, 0.2, 0.1, 0.0$, and a quantum attack (right) of complexity $O^*(A^{\mathscr{C}})$ for $\mathscr{C} = 0.5, 0.4, 0.3, 0.2, 0.1, 0.0$. The attack uses Algorithm 7.3 and the complexities are justified in Theorem 7.8.

Petit's attack solves Problem 7.1, in which we want to compute the isogeny ϕ, by computing $\theta \in \mathrm{End}(E_0)$ and $a, b, c, d \in \mathbb{Z}$ such that there exists a small smooth integer e for which

$$A^2(pa^2 + pb^2 + c^2) + d^2 = \deg(\phi \circ \theta \circ \hat{\phi} + [d]) = Be. \tag{7.2}$$

The restrictions $B > A^4$ and $A > p$ are necessary for Petit's algorithm to find a solution (a, b, c, d, e) in polynomial time and output a sufficiently small, smooth e.

We show in the following theorem that (7.2) can be relaxed to

$$A^2(pa^2 + pb^2 + c^2) + d^2 = \deg(\phi \circ \theta \circ \hat{\phi} + [d]) = B^2e.$$

This in turn allows us to relax the balance of A and B to $B > A^3 > p^{3/2}$ or $B > A^2 > p^2$ to find a solution (a, b, c, d, e) with sufficiently small e, just by applying the same algorithm as Petit [Pet17]. We do not repeat the algorithm here for conciseness, as we give a more general algorithm in the next section that also encompasses our non-polynomial time attacks; the balance of A and B is also addressed in the analysis in the following section.

Theorem 7.4. *Let A, B be coprime smooth integers. Let E_0 be a supersingular elliptic curve defined over $\mathbb{F}_{p^2}$. Let ϕ be a secret isogeny of degree A from E_0 to some curve E, and suppose that we are given the action of ϕ on $E_0[B]$. Furthermore, assume we are given a trace-zero endomorphism $\theta \in \mathrm{End}(E_0)$ in a representation that can be efficiently evaluated on $E_0[B]$, an integer d coprime to B, and a smooth integer e such that*

$$\deg(\phi \circ \theta \circ \hat{\phi} + [d]) = B^2e.$$

Then we can compute ϕ in time $O^*(\sqrt{e}) = O(\sqrt{e} \cdot \mathrm{polylog}(p))$.

Proof. Let $\tau = \phi \circ \theta \circ \hat{\phi} + [d]$. Since the degree of τ is B^2e, it can be decomposed as $\tau = \psi' \circ \eta \circ \psi$ where ψ and ψ' are isogenies of degree B and η is an isogeny of degree e. The isogeny ψ can be computed from the given action on the B-torsion as in Section 7.2.4.

To compute the isogeny ψ', we claim that $\ker(\hat{\psi}')$ contains $\tau(E[B])$ with index at most two. Thus, we can first evaluate τ on the B-torsion using the given action of θ, then find $\ker(\hat{\psi}')$ by potentially brute-forcing a 2-isogeny, and finally compute ψ' from $\hat{\psi}'$ and run the rest of the algorithm for each choice of ψ' the brute-force-of-η step yields.

We now prove the claim: First, $\hat{\psi}' \circ \tau = [B] \circ \eta \circ \psi$ establishes that $\ker(\hat{\psi}') \supseteq \tau(E[B])$. We show that $\ker(\tau) \not\supseteq E[m]$ for any $m > 2$ dividing B. Suppose that τ decomposes as $\tau' \circ [m]$ for $\tau' \in \mathrm{End}(E)$, $m \in \mathbb{Z}$. Then m divides $\mathrm{tr}(\tau) = 2d$, but note that $\gcd(m, 2d) \in \{1, 2\}$ since d was assumed coprime to B. Thus, the subgroup of $E[B]$ killed by τ is isomorphic to either $\mathbb{Z}/B$ or $\mathbb{Z}/B \times \mathbb{Z}/2$, which shows that $|\tau(E[B])| \in \{B, B/2\}$ and therefore $[\ker(\hat{\psi}') : \tau(E[B])] \in \{1, 2\}$.

Finally, for each choice of ψ', we attempt to recover the isogeny η by a generic meet-in-the-middle algorithm, which runs in time $O^*(\sqrt{e})$ since e is smooth. Note that if $e \in O^*(1)$, then the entire algorithm runs in time $\mathrm{polylog}(p)$. □

For (a neighbour of) the initial curve used in SIKE [Jao+17] we deduce the following:

Corollary 7.5. *Let $p \equiv 3 \pmod 4$ and $j(E_0) = 1728$. Consider coprime smooth integers A, B and suppose that we are given an integer solution (a, b, c, d, e), with e smooth, to the equation*

$$A^2(pa^2 + pb^2 + c^2) + d^2 = B^2 e. \tag{7.3}$$

Then we can solve Problem 7.1 with the above parameters in time $O^*(\sqrt{e})$.

Proof. For a degree-A isogeny $\varphi\colon E_0 \to E$, the left side of (7.3) is the norm form of

$$\mathbb{Z} + \varphi\mathrm{End}(E_0)\widehat{\varphi} \cong \mathbb{Z} + A\mathrm{End}(E_0).$$

Choosing $\theta = a\iota\pi + b\pi + c\iota \in \mathrm{End}(E_0)$ yields the desired result. □

7.3.2 – Non-polynomial time torsion-point attacks. In this section we generalize Petit's polynomial-time attack to allow for attacks with any complexity better than $O^*(A^{1/2})$, that is, attacks that improve upon the best known generic attack (cf. Section 7.2.2). Recall that $A = p^\alpha$ and $B = p^\beta$ are the degrees of Alice and Bob's secret isogenies respectively, and that we measure the complexity of the overall attack relative to A by writing it as $O^*(A^{\mathscr{C}})$. The attack, following the approach of Petit [Pet17] together with the improvements described above, naturally splits into two stages: First, the 'precomputation' phase (Algorithm 7.1) in which a solution to (7.3) is computed — notably, this depends only on the parameters (p, A, B) and not on the concrete public key under attack. Second, the 'online' phase (Algorithm 7.2) in which we utilize said solution to recover the secret isogeny as in Theorem 7.4 for a specific public key. Our modifications to Petit's method come in three independent guises, and the resulting algorithm is shown in Algorithm 7.3:

- Precomputation phase:
  - **Larger d:** When computing a solution to Equation (7.3), we fix e and then try up to A^δ values for d until the equation has solutions. This allows us to further relax the constraints between A, B, and p, at the price of an exhaustive search of cost $O^*(A^\delta) = O^*(p^{\alpha\delta})$.
- Online phase:
  - **Larger e:** We search for a solution to Equation (7.3) where e is any smooth number $\leq A^\epsilon$ with $\epsilon \in [0, 1]$, whereas in [Pet17] the integer e was required to be polynomial in $\log p$. This relaxes the constraints on A and B, at a price of a $O^*(e^{1/2}) = O^*(p^{\alpha\epsilon/2})$ computation (to retrieve the endomorphism η defined in the proof of Theorem 7.4).

– **Smaller A:** We first naïvely guess part of the secret isogeny and then apply Petit's techniques only on the remaining part for each guess. More precisely, we iterate through isogenies of degree $A^\gamma \mid A$, with $\gamma \in [0,1]$, and for each possible guess we apply Petit's techniques on Problem 7.1 with $A' := A^{1-\gamma} = p^{\alpha(1-\gamma)}$ in place of A. The Diophantine equation to solve thus turns into

$$A'^2(pa^2 + pb^2 + c^2) + d^2 = B^2 e\,. \tag{7.4}$$

Algorithm 7.1: Solving the norm equation; precomputation.

Input: • SIDH parameters $p, A = p^\alpha, B = p^\beta$.
• Attack parameters $\delta, \gamma, \epsilon \in [0,1]$, with $A^\gamma \mid A$.
Output: A solution (a,b,c,d,e) to (7.4) with $A' = A^{1-\gamma}$ and $e \le A^\epsilon$ smooth.

Pick a smooth number $e \le A^\epsilon$ which is a square modulo A'^2.
Compute d_0 as the smallest positive integer such that $d_0^2 \equiv eB^2 \pmod{A'^2}$.
for $d' = 1, 2, \ldots, \lfloor A^\delta \rfloor$ *such that* $d_0 + A'^2 d' < \sqrt{e}B$ **do**
    Let $d = d_0 + A'^2 d'$.
    Find the smallest positive integer c such that $c^2 A'^2 = eB^2 - d^2 \pmod{p}$, or **continue** if no such c exists.
    If $eB^2 > d^2 + c^2 A'^2$ **then**
        Try finding (a,b) such that $a^2 + b^2 = \frac{eB^2 - d^2 - c^2 A'^2}{A'^2 p}$. If a solution is found, **return** (a,b,c,d,e).

Algorithm 7.2: Recovering the secret isogeny; online phase.

Input: • All the inputs of Algorithm 7.1.
• An instance of Problem 7.1 with those parameters, namely a curve E and points $P, Q \in E[B]$ where there exists a degree-A isogeny $\varphi : E_0 \to E$ such that P, Q are the images by φ of a canonical basis of $E_0[B]$.
• $\theta \in \mathrm{End}(E_0)$ and $d, e \in \mathbb{Z}$ such that $\deg(A'\theta + d) = B^2 e$ with $e \le A^\epsilon$ smooth.
Output: An isogeny φ matching the constraints given by the input.

for $\varphi_g : E \to E'$ *an* A^γ*-isogeny* **do**
    Compute $P' = [A^{-\gamma} \bmod B]\,\varphi_g(P)$ and $Q' = [A^{-\gamma} \bmod B]\,\varphi_g(Q)$.
    Use Theorem 7.4 to compute $\varphi' : E_0 \to E'$ of degree $A' = A^{1-\gamma}$, assuming that P' and Q' are the images by φ' of the canonical basis of $E_0[B]$, or conclude that no such isogeny exists.
    If φ' *is found* **then**
        Return $\varphi = \widehat{\varphi_g} \circ \varphi'$.

Algorithm 7.3: Solving Problem 7.1.

Invoke Algorithm 7.1, yielding $a, b, c, d, e \in \mathbb{Z}$, and then Algorithm 7.2 with $\theta = a\iota\pi + b\pi + c\iota$.

Let us analyze the conditions under which Algorithm 7.1 can be expected to succeed:

Heuristic Result 7.6. *We expect Algorithm 7.1 to produce a solution to Equation (7.4) in the regime*

$$2\beta + \alpha\epsilon \;\geq\; \max\{4\alpha + 2\alpha\delta - 4\alpha\gamma,\; 2 + 2\alpha - 2\alpha\delta - 2\alpha\gamma\}.$$

Justification. By construction we expect $d_0 \approx A'^2$, $d \approx A'^2 A^\delta \approx A^{2(1-\gamma)+\delta}$ and $eB^2 \approx A^\epsilon B^2$, so the 'for' loop in Algorithm 7.1 will run for A^δ iterations if

$$2\alpha(2(1-\gamma)+\delta) \leq \alpha\epsilon + 2\beta.$$

The value c is then computed as a square root modulo p. We therefore expect $c \approx p$ most of the time, and $c \approx pA^{-\delta}$ with probability $A^{-\delta}$, thus a constant number of times over all possible choices for d. For this particular c, we have $c^2 A'^2 \approx p^2 A^{-2\delta} A'^2 \approx p^{2-2\alpha\delta+2\alpha(1-\gamma)}$ and we expect to satisfy the second 'if' condition in step 5 when

$$2 - 2\alpha\delta + 2\alpha(1-\gamma) \leq \alpha\epsilon + 2\beta.$$

The two inequalities together give Heuristic Result 7.6. □

Lemma 7.7. *Assume Heuristic Result 7.6 is satisfied.*

1. *The complexity of Algorithm 7.1 is $O^*(A^\delta)$ classically and $O^*(A^{\delta/2})$ quantumly.*
2. *The complexity of Algorithm 7.2 is $O^*(A^{\gamma+\epsilon/2})$ classically and $O^*(A^{(\gamma+\epsilon)/2})$ quantumly.*

Proof. The loop in Algorithm 7.1 has A^δ steps, each with polynomial complexity (use Cornacchia for step 6). Quantumly, this search takes a square root of the classical cost (using Grover).

The loop in Algorithm 7.2 has approximately A^γ steps, and the main cost in each step is an application of Theorem 7.4 with $e \approx A^\epsilon$. On a classical computer the cost is approximately $O^*(A^\gamma e^{1/2}) = O^*(A^{\gamma+\epsilon/2})$. Using quantum search to guess the correct degree-A^γ isogeny φ_g in step 7.2, Algorithm 7.2 has quantum complexity $O^*(A^{\gamma/2} e^{1/2}) = O^*(A^{(\gamma+\epsilon)/2})$. □

Theorem 7.8. *Let $0 < \alpha \leq \beta$ and $0 \leq \mathscr{C} \leq 1/2$, and define*

$$\Gamma \;:=\; \max\left\{\frac{1+3\alpha-2\beta}{3\alpha},\; \frac{2\alpha-\beta}{2\alpha},\; \frac{1+\alpha-\beta}{2\alpha}\right\}.$$

There exists a configuration $(\delta,\gamma,\epsilon) \in [0;1]^3$ of Algorithm 7.3 satisfying the condition given in Heuristic Result 7.6, such that the attack cost according to Lemma 7.7 is at most $O^*(A^{\mathscr{C}})$, if and only if $\mathscr{C} \geq \Gamma$ for classical attacks, or $\mathscr{C} \geq \Gamma/2$ for quantum attacks.

Proof. Write $f = 1$ for classical algorithms and $f = \frac{1}{2}$ for quantum algorithms; hence, the complexity according to Lemma 7.7 equals $\mathscr{C} = \max\{f\delta, f\gamma + \frac{1}{2}\epsilon\}$. Call a tuple $(\delta,\gamma,\epsilon) \in [0;1]^3$ "admissible" if it satisfies the bounds

$$(4 + 2\delta - 4\gamma - \epsilon)\alpha \leq 2\beta \quad \text{and} \quad (2 - 2\delta - 2\gamma - \epsilon)\alpha \leq 2\beta - 2 \tag{$*$}$$

from Heuristic Result 7.6. Suppose given an admissible tuple (δ,γ,ϵ) with cost $\mathscr{C} \leq 1/2$. First, notice that setting $\gamma' := \max\{\delta, \gamma + \frac{1}{2f}\epsilon\}$, the tuple $(\delta,\gamma',0)$ is still admissible with the same $\mathscr{C}$. (Since $\mathscr{C} \leq 1/2$, we have $\gamma' \leq \frac{1}{2f} \leq 1$.) Thus, it suffices to consider admissible tuples $(\delta,\gamma',0)$ with $0 \leq \delta \leq \gamma' \leq 1$ when optimizing. The bounds $(*)$ simplify to

$$1 + \alpha - \beta - \alpha\gamma' \;\leq\; \alpha\delta \;\leq\; \beta - 2\alpha + 2\alpha\gamma', \tag{$*'$}$$

which (leaving out the middle term $\alpha\delta$ and simplifying) implies

$$\gamma' \geq \frac{1+3\alpha-2\beta}{3\alpha}. \tag{$*_1$}$$

This establishes a lower bound on γ', but it is not yet clear which of these values are actually possible: For a given γ', we additionally require a $\delta \in [0;\gamma']$ that satisfies the bounds ($*'$). Hence, the upper bound $\beta - 2\alpha + 2\alpha\gamma'$ on $\alpha\delta$ in ($*'$) must be non-negative, which simplifies to

$$\gamma' \geq \frac{2\alpha-\beta}{2\alpha}. \tag{$*_2$}$$

Similarly, the lower bound $1+\alpha-\beta-\alpha\gamma'$ on $\alpha\delta$ in ($*'$) must not be greater than $\alpha\gamma'$, yielding

$$\gamma' \geq \frac{1+\alpha-\beta}{2\alpha}. \tag{$*_3$}$$

Recalling that $\mathscr{C} = f\gamma'$, this shows the claim. □

7.3.3 – Impact on B-SIDH. A recent proposal called *B-SIDH* [Cos20] consists of instantiating SIDH with parameters where AB is a divisor of p^2-1. Theorem 7.8 suggests that we may expect a quantum attack of complexity $O^*(p^{1/3})$ when $A \approx B \approx p$. This compares to other attack complexities in the literature as follows:

- Tani's quantum claw-finding algorithm [Tan07] was claimed to have complexity $O^*(p^{1/3})$, but [JS19] argues that the complexity is actually no lower than $O^*(p^{2/3})$ when the cost of data-structure operations is properly accounted for.
- A quantum algorithm due to Biasse, Jao, and Sankar [BJS14] finds *some* isogeny between the start and end curve in time $O^*(p^{1/4})$. While there is a heuristic argument for "standard" SIDH/SIKE that any isogeny suffices to find the correct isogeny [GPST16], this argument relies on the fact that the isogeny sought in SIKE has relatively small degree compared to p, which is not true for B-SIDH, so this does currently not yield a complete attack. The B-SIDH paper [Cos20] conservatively views [BJS14] as the best quantum attack.
- The cost of known classical attacks is no lower than $O^*(A^{1/2})$, which is achieved by meet-in-the-middle techniques (using exponential memory) and potentially memoryless by Delfs and Galbraith [DG16] when $A \approx p$ assuming a sufficiently efficient method to produce *the* isogeny from some isogeny.

Thus, assuming our heuristics hold true, Algorithm 7.3 is asymptotically the best known attack against B-SIDH at the moment. Should it turn out in the future that finding *any* isogeny suffices to compute *the right* isogeny in time less than $O^*(p^{1/3})$, then combining that method with Biasse–Jao–Sankar will yield a better quantum attack; at present it is not known how to do this.

Note that for $1/2 < \alpha \approx \beta < 1$, the (quantum) attack cost in terms of p may be lower than $O^*(p^{1/3})$, but it does not get smaller than $O^*(p^{1/4})$ for balanced parameters.

The concrete example parameters in [Cos20] do not allow very strong torsion-attacks since constructing optimal B-SIDH parameters (thus allowing for the most effective attacks) seems difficult. For example, consider the kind of parameters proposed as an alternative for SIKEp610 using the Mersenne prime $p = 2^{521}-1$: In this example, $A = 2^{305}$ and $B \approx 2^{305}$, hence $\alpha \approx \beta \approx 0.58$, and by Theorem 7.8 our methods can be expected to lead to a quantum attack of asymptotic complexity $O^*(A^{0.46}) \subseteq O^*(p^{0.27})$ on parameters with these size ratios. The full range of B-SIDH parameters to which our attacks apply is summarized in Figure 7.1.

7.3.4 – Impact on other variants of SIDH. The group key exchange protocol from [AJJS19] with k parties can be reduced to an instance of Problem 7.1 with $A \approx p^{1/k}$ and $B \approx p^{(k-1)/k}$. Although our attacks only apply for $AB > p$, Figure 7.1 (or equivalently Theorem 7.8) shows that as the imbalance increases, the attack applies for AB approaching p. In particular, for a large number of parties k, the product AB does not have to be much larger than p for an (exponential) torsion-point attack to apply.

Hence, our attacks do not seem to apply to the group key exchange as described in [AJJS19], which (like SIKE) satisfies $AB < p$. However, it is not inconceivable that someone implementing a group key exchange protocol may borrow ideas from B-SIDH in order to improve efficiency, especially given the scarcity of appropriate base field primes for group key exchange following [AJJS19]. Such a combined group key exchange with ideas from B-SIDH could easily yield a torsion-point attack: For example, even for 3 parties, parameters with $AB \approx p^2$ lead to a quantum attack of (heuristic) complexity $O^*(A^{1/8})$, a fourth-root improvement over generic claw finding.

7.3.5 – Improvement prospects. In this section we consider how future improvements on the resolution of Equation (7.3) might impact the hardness of Problem 7.1. We first estimate the minimal size of e for a given set of parameters (p, A, B).

Heuristic Result 7.9. *Solutions (a, b, c, d, e) to Equation (7.3) can be expected to satisfy*

$$e^2 \geq \frac{A^3 p}{B^2}.$$

Justification. We consider solutions with $e \leq M$ for some fixed bound M. Since all summands on the left-hand side are non-negative, they cannot be bigger than the upper bound MB^2 of the right-hand side. This yields the bounds

$$a \leq \frac{\sqrt{M}B}{A\sqrt{p}}; \qquad b \leq \frac{\sqrt{M}B}{A\sqrt{p}}; \qquad c \leq \frac{\sqrt{M}B}{A}; \qquad d \leq \sqrt{M}B.$$

Hence the number of possible assignments of the variables e, a, b, c, d is about

$$M \cdot \frac{\sqrt{M}B}{A\sqrt{p}} \cdot \frac{\sqrt{M}B}{A\sqrt{p}} \cdot \frac{\sqrt{M}B}{A} \cdot \sqrt{M}B = \frac{M^3 B^4}{A^3 p}.$$

Heuristically modelling both left- and right-hand side as uniformly random integers in the range $\{0, \ldots, MB^2\}$, this implies the expected number of solutions is about

$$\frac{M^3 B^4}{A^3 p} / (MB^2) = \frac{M^2 B^2}{A^3 p}.$$

Solving this for one expected solution yields the claimed estimate. □

Heuristic Result 7.10. *Assume that we are given a solution to Equation 7.3 for parameters as in Heuristic Result 7.9. Then we expect to solve Problem 7.1:*

1. *with classical complexity $O^*(1)$ when $B \geq p^{1/2} A^{3/2}$,*
2. *with classical complexity $O^*(A^{1/2})$ when $B \geq p^{1/2} A^{1/2}$, and*
3. *with quantum complexity $O^*(A^{1/2})$ when $B \geq p^{1/2}$.*

Justification. As we are given a solution to Equation (7.3), we no longer need Algorithm 7.1 and can apply Algorithm 7.2 right away. Heuristic Result 7.9 gives the constraint

$$2(\beta + \alpha\epsilon) \geq 1 + 3\alpha(1 - \gamma), \tag{7.5}$$

which we now use in place of Heuristic Result 7.6 to optimally balance parameters.

1. For polynomial-time attacks we need $\epsilon = \gamma = 0$ by Lemma 7.7. Plugging this into (7.5) gives $2\beta > 1 + 3\alpha$, hence the result.

2. Increasing either γ or ϵ will contribute to relaxing Inequality (7.5), and by Lemma 7.7 we need $\gamma + \epsilon/2 \leq 1/2$. Substituting γ for $(1-\epsilon)/2$ in (7.5) gives

$$2\beta \geq 1 + \alpha(3-\epsilon)/2.$$

   Setting $\epsilon = 1$ and $\gamma = 0$ simplifies this to $2\beta \geq 1 + \alpha$, hence the result.

3. Increasing either γ or ϵ will contribute to relaxing Inequality (7.5), and by Lemma 7.7 we need $\gamma + \epsilon < 1$. Substituting γ for $1 - \epsilon$ in (7.5), we get

$$2\beta \geq 1 + \alpha\epsilon.$$

   Setting $\epsilon = 0$ and $\gamma = 1/2$ simplifies this to $2\beta \geq 1$, hence the result.

□

Remark 7.11. *In the group key exchange protocol of [AJJS19] with k parties we have $A \approx p^{1/k}$ and $B \approx p^{(k-1)/k}$. A better solver for Equation (7.3) could give an improved quantum attack when $k > 2$, an improved classical attack when $k > 3$, and a (classical) polynomial-time attack when $k > 5$.*

Remark 7.12. *In contexts where several instances of Problem 7.1 need to be solved with the same parameters, Algorithm 7.1 only needs to be executed once. In this case the algorithm's parameters can be tweaked to reduce the average cost per instance.*

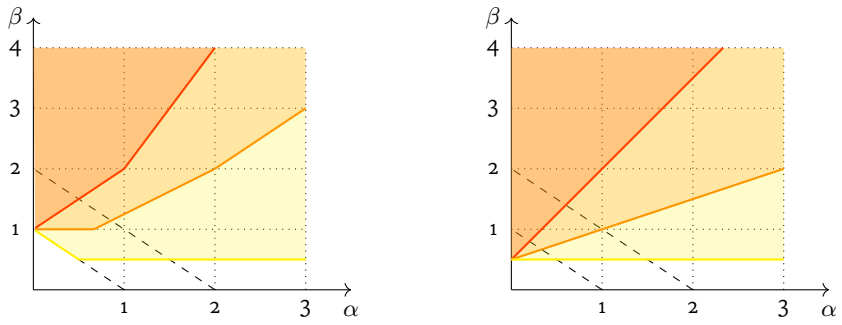


Figure 7.2: Performance of our current attacks. Left: Algorithm 7.3. Right: Hypothetical attack assuming an optimal polynomial-time solver for Equation 7.3 combined with Algorithm 7.2. Here $A = p^\alpha$ and $B = p^\beta$. Parameters (α, β) above the red, orange and yellow lines are parameters admitting a polynomial-time attack, a classical attack in $O^*(A^{1/2})$, and a quantum attack in $O^*(A^{1/2})$, respectively. Parameters below the upper dashed line are those allowing $AB \mid (p^2 - 1)$ as in [Cos20]. Parameters below the lower dashed line are those allowing $AB \mid (p - 1)$ as in [Jao+17; Jao+19].

7.4 — Trapdoor instances

In this section we give a method to specifically create instantiations of the SIDH framework for which we can solve Problem 7.1 more efficiently given some extra information. Recall that we let $A \leq B$ denote the degrees of Alice's and Bob's secret isogenies, respectively, and let $A = p^\alpha$ and $B = p^\beta$. Recall that for all the instances studied in Section 7.3, our attack methods can improve upon the complexity of claw finding only when AB are greater than p (see Figures 7.1 and 7.2), and that we can only expect solutions to Equation 7.3 with a polynomially small[7] value of e when $[B > A^3$ and $A > p^{1/2}]$ or $[B > A^2$ and $A > p]$. However, all of this was only considering cases where the starting curve has j-invariant 1728. In Section 7.4.1 we explore the question: For given A, B can we construct starting curves for which we can solve Problem 7.1 with a better balance? We will call such curves *trapdoor curves* (see Definition 7.13), and quantify the number of trapdoor curves in Section 7.4.3.

In Sections 7.4.4 and 7.4.5, we also consider trapdoored choices of p, A, and B, for which we can solve Problem 7.1 more efficiently even when starting from the curve with j-invariant 1728.

7.4.1 – Trapdoor curves. This section introduces the concept of *trapdoor curves* and how to find such curves. Roughly speaking, these are specially crafted curves which, if used as starting curves for the SIDH protocol, are susceptible to a torsion-point attack by the party who chose the curve, under only moderately imbalanced parameters A, B; in particular, the imbalance is independent of p. In fact, when we allow for non-polynomial time attacks we get an asymptotic improvement on the best general attack for balanced SIDH parameters (but starting from a trapdoor curve). These curves could potentially be utilized as *backdoor* curves, for example by suggesting the use of such a curve as a standardized starting curve. We note that it does not seem obvious how trapdoored curves, such as those generated by Algorithm 7.4, can be detected by other parties: The existence of an endomorphism of large degree which satisfies Equation 7.3 does not seem to be detectible without trying to recover such an endomorphism, which is hard using all currently known algorithms.

The notion of trapdoor curves is dependent on the parameters A, B, which motivates the following definition:

Definition 7.13. *Let A, B be coprime positive integers and $0 \leq \mathscr{C} \leq 1/2$. An $(A, B, \mathscr{C})$-*trapdoor curve *is a tuple (E_0, θ, d, e) of a supersingular elliptic curve E_0 defined over some $\mathbb{F}_{p^2}$, an endomorphism $\theta \in \mathrm{End}(E_0)$ in an efficient representation, and two integers d, e, such that Algorithm 7.2 solves Problem 7.1 for that particular E_0 in time $O^*(A^{\mathscr{C}})$ when given (θ, d, e). An (A, B)-*trapdoor curve *is an $(A, B, 0)$-trapdoor curve, i.e., one for which Algorithm 7.2 takes time polynomial in $\log p$.*

Remark 7.14. *It is important that θ is efficiently represented as it might not have smooth degree.*

We summarize the complexity of our attack on SIDH instances starting at trapdoor curves in Figure 7.3; this figure follows from Theorem 7.21. Note that these attacks do apply to balanced parameters with $AB \approx p$ and give a significant improvement on the meet-in-the-middle claw-finding complexity for these cases. We stress however that this relies on using a special starting curve and hence does not give an attack on SIKE when using the proposed (neighbour of a) starting curve with j-invariant 1728, unless there happens to be short path from this starting curve to a backdoor curve that can be found efficiently.

Algorithm 7.4 computes (A, B)-trapdoor curves in heuristic polynomial time, assuming we have a factoring oracle (see Theorem 7.15).

[7] Recall that this is necessary to obtain a polynomial-time online cost in our attack.

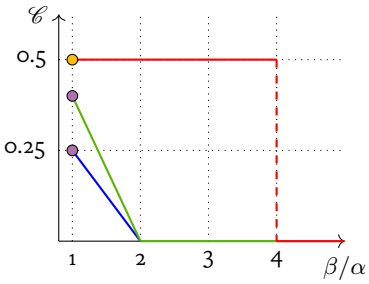


Figure 7.3: Choices of $A = p^\alpha$ and $B = p^\beta$ for which we can (heuristically) find an $(A, B, \mathscr{C})$-trapdoor curve (E_0, θ, d, e) within time $O^*(A^{\mathscr{C}})$. An SIDH variant starting at E_0 can be broken in time $O^*(A^{\mathscr{C}})$ using our attack when (θ, d, e) is given. Red: Complexity of best known attack without having (θ, d, e). Green: Classical complexity of our attack when starting at a $(A, B, \mathscr{C})$-trapdoor curve. Blue: Quantum complexity of our attack when starting at a $(A, B, \mathscr{C})$-trapdoor curve. Yellow: SIKE parameters. Violet: SIKE-like parameters, but starting instead from an $(A, B, \mathscr{C})$-trapdoor curve.

Algorithm 7.4: Generating (A, B)-trapdoor curves.

Input: A prime $p \equiv 3 \pmod 4$ and smooth coprime integers A, B with $B > A^2$.
Output: An (A, B)-trapdoor curve (E_0, θ, d, e) with $E_0/\mathbb{F}_{p^2}$.

Set $e := 1$.
While `true` **do**
  Find an integer d such that $d^2 \equiv B^2 e \pmod{A^2}$.
  If *d is coprime to B* **then**
    If $\frac{B^2e-d^2}{A^2}$ *is square modulo p* **then**
      Find rational a, b, c such that $pa^2 + pb^2 + c^2 = \frac{B^2e-d^2}{A^2}$.
      break
  Set e to the next square.
Set $\vartheta = a\mathbf{ij} + b\mathbf{j} + c\mathbf{i} \in B_{p,\infty}$.
Compute a maximal order $\mathcal{O} \subseteq \mathcal{B}_{p,\infty}$ containing θ.
Compute an elliptic curve E_0 whose endomorphism ring is isomorphic to $\mathcal{O}$.
Construct an efficient representation of the endomorphism θ of E_0 corresponding to ϑ.
Return (E_0, θ, d, e).

Theorem 7.15. *Given an oracle for factoring, Algorithm 7.4 can heuristically be expected to succeed in polynomial time.*

Remark 7.16. *The imbalance $\beta > 2\alpha$ is naturally satisfied for a group key exchange in the style of [AJJS19] with three or more participants; we can break (in polynomial time) such a variant when starting at an (A, B)-trapdoor curve.*

Before proving Theorem 7.15 we need the following easy lemma:

Lemma 7.17. *Let p be a prime congruent to 3 modulo 4. Let D be a positive integer. Then the quadratic form $Q(x_1, x_2, x_3, x_4) = px_1^2 + px_2^2 + x_3^2 - Dx_4^2$ has a nontrivial integer root if and only if D is a quadratic residue modulo p.*

Proof. The proof is essentially a special case of [Sim05, Proposition 10], but we give a brief sketch of the proof here. If D is a quadratic residue modulo p, then $px_1^2 + px_2^2 + x_3^2 - Dx_4^2$ has a solution in $\mathbb{Q}_p$ by setting $x_1 = x_2 = 0$ and $x_4 = 1$ and applying Hensel's lemma to the equation $x_3^2 = D$. The quadratic form Q also has local solutions everywhere else (the 2-adic case involves looking at the equation modulo 8 and applying a 2-adic version of Hensel's lemma). If on the other hand D is not a quadratic residue modulo p, then one has to choose x_3 and x_4 to be divisible by p. Dividing the equation $Q(x_1, x_2, x_3, x_4) = 0$ by p and reducing modulo p yields $x_1^2 + x_2^2 \equiv 0 \pmod{p}$. This does not have a solution as $p \equiv 3 \pmod 4$. Finally, one can show that this implies that Q does not have a root in $\mathbb{Q}_p$. □

Proof of Theorem 7.15. The main idea is to apply Theorem 7.4 in the following way: using Algorithm 7.4, we find integers D, d, and e, with e polynomially small and D a quadratic residue mod p, such that $A^2D + d^2 = B^2e$, and an element $\theta \in \mathcal{B}_{p,\infty}$ of trace zero and such that $\theta^2 = -D$. We then construct a maximal order $\mathcal{O} \subseteq \mathcal{B}_{p,\infty}$ containing θ and an elliptic curve E_0 with $\mathrm{End}(E_0) \cong \mathcal{O}$.

Most steps of Algorithm 7.4 obviously run in polynomial time, although some need further explanation. We expect $d^2 \approx A^4$ since we solved for d modulo B^2, and we expect e to be small since heuristically we find a quadratic residue after a small number of tries. Then the right-hand side in step 6 should be positive since $B > A^2$, so by Lemma 7.17 step 6 returns a solution using Simon's algorithm [Sim05], assuming an oracle for factoring $\frac{B^2e-d^2}{A^2}$. For step 10, we can apply either of the polynomial-time algorithms [IR93; Voi13] for finding maximal orders containing a fixed order in a quaternion algebra, which again assume a factoring oracle. Steps 11 and 12 can be accomplished using the heuristically polynomial-time algorithm from [PL17; Eis+18] which returns both the curve E_0 and (see [Eis+18, § 5.3, Algorithm 5]) an efficient representation of θ. □

Remark 7.18. *The algorithm uses factorization twice. In Section 7.5 we discuss how one can ensure in practice that the numbers to be factored have an easy factorization.*

Remark 7.19. *The main contribution of Simon's paper is a polynomial-time algorithm for finding nontrivial roots of (not necessarily diagonal) quadratic forms which does not rely on an effective version of Dirichlet's theorem. In our case, however, we only need a heuristic polynomial-time algorithm for finding a nontrivial root (x, y, z, u) of a form $px^2 + py^2 + z^2 - Du^2$. We sketch an easy way to do this: Suppose that D is squarefree, and pick a prime $q \equiv 1 \pmod 4$ such that $-pq$ is a quadratic residue modulo every prime divisor of D. It is then easy to see that the quadratic equations $px^2 + py^2 = pq$ and $Du^2 - z^2 = pq$ both admit a nontrivial rational solution which can be found using [CR03].*

Remark 7.20. *Weak curves also have a constructive application: An improvement on the recent paper [SKPS19] using Petit's attack to build a one-way function 'SÉTA'. In this scheme, the secret key is a secret isogeny to a curve E_s that starts from the elliptic curve with j-invariant 1728 and the message is the end point of a secret isogeny from E_s to some curve E_m, together with the image of some torsion points. The reason for using j-invariant 1728 is in order to apply Petit's attack constructively. One could instead use a weak curve; this provides more flexibility to the scheme as one does not need to disclose the starting curve and the corresponding norm equation is easier to solve.*

7.4.2 – Non-polynomial time attacks for trapdoor curves. In this section we give a further generalization of Algorithm 7.3 to utilize some extra techniques available to us when the starting curve E_0 is trapdoored. Recall, as above, that $A \leq B$ are the degrees of Alice's and Bob's

secret isogenies respectively, and $A = p^\alpha$ and $B = p^\beta$. Recall the definition of an $(A, B, \mathscr{C})$-trapdoor curve (E_0, θ, d, e) from Definition 7.13; in particular that such a curve gives rise to a torsion-point attack of complexity $O^*(A^{\mathscr{C}})$.

We show in this section that for $\alpha \approx \beta$, we can modify Algorithm 7.4 to compute a classically $(A, B, 2/5)$-trapdoor curve or a quantumly $(A, B, 1/4)$-trapdoor curve. We also show how the attack on trapdoor curves improves for imbalanced parameters; see Figure 7.3 for a comparison of previous results with Theorem 7.21.

Theorem 7.21. *Heuristically:*

- *Let $\mathscr{C} \in [0, 0.4]$. For A, B such that $B > A^{2-5/2\cdot\mathscr{C}}$, a classical algorithm can construct a $(A, B, \mathscr{C})$-trapdoor curve in time $O^*(A^{\mathscr{C}})$, assuming an oracle for factoring.*
- *Let $\mathscr{C} \in [0, 0.25]$. For every A, B such that $B > A^{2-4\cdot\mathscr{C}}$, a quantum algorithm can construct a $(A, B, \mathscr{C})$-trapdoor curve in polynomial time.*

Proof. Modify Algorithm 7.4 as follows:

- Use $A' = A^{1-\gamma}$ instead of A, namely we will guess part of the isogeny with degree $A^\gamma \mid A$.
- Instead of starting from $e = 1$, start the loop at e such that $B^2 e > A'^4$.
- Choose $A^{\epsilon'}$ random values of $e \leq A^\epsilon$ (note e is not necessarily an integer square) until there exists d such that $d^2 = B^2 e \bmod (A')^2$,

$$B^2 e - d^2 > 0, \tag{7.6}$$

  and $B^2 e - d^2$ is a square modulo p. Once these values of d and e are found, continue like in Algorithm 7.4, step 6.

The attacker can then invoke Algorithm 7.2 to compute the secret isogeny, using the data (θ, d, e) from Algorithm 7.4.

We analyze the complexity of running the modified Algorithm 7.4 followed by Algorithm 7.2. The two quadratic residuosity conditions are heuristically satisfied one in four times, so we ignore them in this analysis. The cost of Algorithm 7.4 modified in this way becomes $O^*(A^{\epsilon'})$ for a classical adversary and $O^*(A^{\epsilon'/2})$ for a quantum adversary.

Note also that by construction we have $e \leq A^\epsilon$, so the cost of running Algorithm 7.2 will be $O^*(A^{\gamma+\epsilon/2})$ for a classical adversary and $O^*(A^{(\gamma+\epsilon)/2})$ for a quantum adversary, following the same reasoning as in the complexity analysis of Algorithm 7.3.

We now look at the conditions for existence of a solution in Algorithm 7.4. Note that d is a priori bounded by $(A')^2 = A^{2(1-\gamma)}$. However, after trying A^ϵ values for e we may hope to find some d bounded by $A^{2(1-\gamma)-\epsilon}$. To satisfy (7.6) we need

$$2\beta > \alpha(4 - 4\gamma - 2\epsilon' - \epsilon),$$

and by construction we also need $\epsilon' \leq \epsilon$.

For a classical adversary, setting $\epsilon = \epsilon' = 2\gamma = \mathscr{C}$ gives the result. For a quantum adversary, setting $\epsilon = \epsilon' = 0$ and $\gamma = 2 \cdot \mathscr{C}$ gives the result. □

Remark 7.22. *We found these choices for $\epsilon, \epsilon', \gamma$ by solving the following optimization problems for $\alpha = \beta = 1/2$, so at least in that case (which corresponds to SIKE) we expect there to be no better choice with respect to overall complexity: For the best classical attack when $\alpha = \beta = 1/2$ we solved the following linear optimization problem:*

$$\min_{\substack{4\gamma+2\epsilon'+\epsilon\geq 2,\\ \epsilon\geq\epsilon'}} \max\left\{\epsilon', \gamma + \epsilon/2\right\}.$$

For the best quantum attack when $\alpha = \beta = 1/2$ we solved the following linear optimization problem:

$$\min_{\substack{4\gamma+2\epsilon'+\epsilon\geq 2\\ \epsilon\geq\epsilon'}} \max\left\{\epsilon'/2, (\gamma+\epsilon)/2\right\}.$$

Remark 7.23. *We have implemented the computation of the maximal orders for the SIKEp434 parameters $A = 2^{216}$ and $B = 3^{137}$.*

7.4.3 – Counting trapdoor curves. Having shown how to construct trapdoor curves and how to exploit them, a natural question to ask is how many of these curves we can find using the methods of the previous section. Recall that the methods above search for an element $\vartheta \in \mathcal{B}_{p,\infty}$ with reduced norm D. Theorem 7.24, due to Onuki [Onu20], suggests they can be expected to produce exponentially (in $\log D$) many different maximal orders, and using Lemma 7.25 we can prove this rigorously for the (indeed interesting) case of (A, B)-trapdoor curves with $AB \approx p$ and $A^2 < B < A^3$ (cf. Theorem 7.15).

We first recall some notation from [Onu20]. The set $\rho(\mathcal{Ell}(\mathcal{O}))$ consists of the reductions modulo p of all elliptic curves over $\overline{\mathbb{Q}}$ with complex multiplication by $\mathcal{O}$. Each curve $E = \mathcal{E} \bmod p$ in this set comes with an optimal embedding $\iota\colon \mathcal{O} \hookrightarrow \mathrm{End}(E)$, referred to as an 'orientation' of E, and conversely, [Onu20, Prop. 3.3] shows that — up to conjugation — each oriented curve (E, ι) defined over $\overline{\mathbb{F}}_p$ is obtained by the reduction modulo p of a characteristic-zero curve; in other words, either (E, ι) or $(E^{(p)}, \iota^{(p)})$ lies in $\rho(\mathcal{Ell}(\mathcal{O}))$. Onuki proves:

Theorem 7.24 [Onu20, Theorem 3.4]**.** *Let K be an imaginary quadratic field such that p does not split in K, and $\mathcal{O}$ an order in K such that p does not divide the conductor of $\mathcal{O}$. Then the ideal class group $\mathrm{cl}(\mathcal{O})$ acts freely and transitively on $\rho(\mathcal{Ell}(\mathcal{O}))$.*

Thus, it follows from well-known results about imaginary quadratic class numbers [Sie35] that asymptotically, there are $h(-D) \in \Omega(D^{1/2-\varepsilon})$ many trapdoor elliptic curves *counted with multiplicities* given by the number of embeddings of $\mathcal{O}$. However, it is not generally clear that this corresponds to many distinct *curves* (or maximal orders). As an (extreme) indication of what could go wrong, consider the following: there seems to be no obvious reason why in some cases the entire orbit of the group action of Theorem 7.24 should *not* consist only of one elliptic curve with lots of independent copies of $\mathcal{O}$ in its endomorphism ring.

We can however at least prove that this does not always happen. In fact, in the case that D is small enough relative to p, one can show that there cannot be more than one embedding of $\mathcal{O}$ into any maximal order in $B_{p,\infty}$, implying that the $h(-D)$ oriented supersingular elliptic curves indeed must constitute $h(-D) \approx \sqrt{D}$ distinct quaternion maximal orders:

Lemma 7.25. *Let $\mathcal{O}$ be a maximal order in $\mathcal{B}_{p,\infty}$. If $D \equiv 3, 0 \pmod 4$ is a positive integer smaller than p, then there exists at most one copy of the imaginary quadratic order of discriminant $-D$ inside $\mathcal{O}$.*

Proof. This follows readily from Theorem 2**'** of [Kan89]. □

This lemma together with Theorem 7.15 shows that there are $\Theta(h(-D))$ many (A, B)-trapdoor maximal orders under the restrictions that $B > A^2$ and $D < p$. Consider the case (of interest) in which $AB \approx p$: Following the same line of reasoning as in the proof of Theorem 7.15 we have that $B^2/A^2 - A^2 \approx D$, which if $D < p \approx AB$ implies that $B \lessapprox A^3$. Hence, as advertised above, Lemma 7.25 suffices to prove that there are $\Theta(h(-D))$ many (A, B)-trapdoor maximal orders under the restriction that $AB \approx p$ and roughly $A^2 < B < A^3$. For larger choices of B, it is no

longer true that there is only one embedding of $\mathcal{O}$ into a quaternion maximal order: indeed, at some point $h(-D)$ will exceed the number $\Theta(p)$ of available maximal orders, hence there must be repetitions. While it seems hard to imagine cases where the orbit of $\mathrm{cl}(\mathbb{Z}[\theta])$ covers only a negligible number of curves (recall that θ was our endomorphism of reduced norm D), we do not currently know how (and under which conditions) to rule out this possibility.

Remark 7.26. *Having obtained any one maximal order $\mathfrak{O}$ that contains θ, it is efficient to compute more such orders (either randomly or exhaustively): For any ideal $\mathfrak{a}$ in $\mathbb{Z}[\theta]$, another maximal order with an optimal embedding of $\mathbb{Z}[\theta]$ is the right order of the left ideal $\mathcal{I} = \mathfrak{O}\mathfrak{a}$. (One way to see this: $\mathfrak{a}$ defines a horizontal isogeny with respect to the subring $\mathcal{O}$; multiplying by the full endomorphism ring does not change the represented kernel subgroup; the codomain of an isogeny described by a quaternion left ideal has endomorphism ring isomorphic to the right order of that ideal. Note that this is similar to a technique used by [CPV20] in the context $\mathcal{O} \subseteq \mathbb{Q}(\pi)$.)*

7.4.4 – Trapdoored p for given A and B with starting vertex $j = 1728$. Another way of constructing trapdoor instances of an SIDH-style key exchange is to keep the starting vertex as $j = 1728$ (or close to it), keep A and B smooth or powersmooth (but not necessarily only powers of 2 and 3 as in SIKE), and construct the base-field prime p to turn $j = 1728$ into an (A, B)-trapdoor curve. In this section, let E_0 denote the curve $E_0 \colon y^2 = x^3 + x$.

An easy way of constructing such a p is to perform steps 1 and 3 of Algorithm 7.4, and then let $D := \frac{B^2e - d^2}{A^2}$. Allowing p to be a variable, we can solve

$$D = p(a^2 + b^2) + c^2$$

in variables $a, b, c, p \in \mathbb{Z}$, p prime, as follows. Factor $D - c^2$ for small c until the result is of the form pm where p is a large prime congruent to 3 modulo 4 and m is a number representable as a sum of squares.[8]

Then, with $\theta = a\iota\pi + b\pi + c\iota$ the tuple (E_0, θ, d, e) is (A, B)-trapdoor. (Note that, in this construction, we cannot expect to satisfy a relationship such as $p = ABf - 1$ with small $f \in \mathbb{Z}$.)

As an (unbalanced) example, let us choose $A = 2^{216}$ and $B = 3^{300}$ and set $e = 1$. Then we can use $d = B \bmod A^2$. Let $D = \frac{B^2 - d^2}{A^2}$, for which we will now produce two primes: First, pick $c = 53$, then $D - c^2$ is a prime number (i.e., $a = 1,\ b = 0$). Second, pick $c = 355$, then $D - c^2$ is 5 times a prime number (i.e., $a = 2, b = 1$). Both of these primes are congruent to 3 modulo 4.

For a powersmooth example, let A be the product of every other prime from 3 up through 317, and let B be the product of all remaining odd primes ≤ 479. With $e = 4$, we can again use $d = B \bmod A^2$ and compute D as above. Then $D - 153^2$ is prime and congruent to 3 modulo 4 (i.e., $a = 1, b = 0$).

7.4.5 – Insecure $A \approx B$ for $j = 1728$. For $A \approx B$, finding (A, B)-trapdoor curves seems difficult. However, in this section we show that certain choices of (power)smooth parameters A and B allow us to find f such that $j = 1728$ can be made insecure over any $\mathbb{F}_{p^2}$ with $p = ABf - 1$.

One approach to this is to find Pythagorean triples $A^2 + d^2 = B^2$ where A and B are coprime and (power)smooth; then $E_0 \colon y^2 = x^3 + x$ is a trapdoor curve with $\theta = \iota$, the d value from the Pythagorean triple, and $e = 1$. With this construction, we can then use *any* $p \equiv 3 \pmod 4$, in particular one of the form $p = ABf - 1$.

[8] Some choices of A and B result in $D \equiv 2 \pmod 4$ which is an obstruction to this method.

Note that given the isogeny degrees A, B, it is easy for anyone to detect if this method has been used by simply checking whether $B^2 - A^2$ is a square; hence, an SIDH key exchange using such degrees is simply *weak* and not just trapdoored.[9]

Problem 7.27. *Find Pythagorean triples $B^2 = A^2 + d^2$ such that A and B are coprime and smooth (or powersmooth).*

Pythagorean triples can be parameterized in terms of Gaussian integers. To be precise, primitive integral Pythagorean triples $a^2 = b^2 + c^2$ are in bijection with Gaussian integers $z = m + ni$ with $\gcd(m, n) = 1$ via the correspondence $(a, b, c) = \big(\mathrm{N}(z), \mathrm{Re}(z^2), \mathrm{Im}(z^2)\big)$. The condition that m and n are coprime is satisfied if we take z to be a product of split Gaussian primes, i.e., $z = \prod_i w_i$ where $\mathrm{N}(w) \equiv 1 \pmod 4$ is prime, taking care to avoid simultaneously including a prime and its conjugate. Thus the following method applies provided that B is taken to be an integer divisible only by primes congruent to 1 modulo 4, and $B > A$.

In order to guarantee that $B = \mathrm{N}(z)$ is powersmooth, one may take many small w_i. In order to guarantee that B is smooth, it is convenient to take $z = w^k$ for a single small Gaussian prime w, and a large composite power k.

It so happens that the sequence of polynomials $\mathrm{Re}(z^k)$ in variables n and m (recall that $z = n + mi$) factors generically into relatively small factors for composite k, so that, when $B^2 = A^2 + d^2$, we can expect that A is frequently smooth or powersmooth. In practice, running a simple search using this method, one very readily obtains example insecure parameters:

$$\begin{aligned} B &= 5^{105} \\ A &= 2^2 \cdot 11 \cdot 19 \cdot 29 \cdot 41 \cdot 59 \cdot 61 \cdot 139 \cdot 241 \cdot 281 \cdot 419 \cdot 421 \cdot 839 \cdot 2381 \cdot 17921 \\ &\quad \cdot 21001 \cdot 39761 \cdot 74761 \cdot 448139 \cdot 526679 \cdot 771961 \cdot 238197121 \\ d &= 3^2 \cdot 13 \cdot 79 \cdot 83 \cdot 239 \cdot 307 \cdot 2801 \cdot 3119 \cdot 3361 \cdot 3529 \cdot 28559 \cdot 36791 \cdot 53759 \\ &\quad \cdot 908321 \cdot 3575762705759 \cdot 23030958433523039 \end{aligned}$$

For this example, if we take $p = 105AB - 1$, we obtain a prime which is 3 modulo 4. Note that here $B \approx 2^{244}$ and $A \approx 2^{238}$. Many other primes can easily be obtained (replacing 105 with 214, 222, etc).

Remark 7.28. *When choosing parameter sets to run B-SIDH [Cos20], if the user is very unlucky, they could hit an instance of such a weak prime. With this in mind, it would be prudent to check that a given combination of A, B, and p does not fall into this category before implementing such a B-SIDH instance.*

7.5 — Implementation

In this section we report on computations regarding Algorithm 7.4 for some concrete parameters. We chose parameters $A = 2^{216}$, $B = 3^{300}$, $p = AB \cdot 277 - 1$. It is easy to see that we can choose $e = 1$ and d equal to B modulo A^2. Now we need to factor $\frac{B^2 - d^2}{A^2}$. The way we chose d makes it easy as $\frac{B^2 - d^2}{A^2} = \frac{B - d}{A^2}(B + d)$. This is something which applies in other cases as well, and to make sure that factorization is easy one can try choices of d until factoring $B + d$ is

[9] We resist the temptation of referring to such instantiations as 'door' instead of 'trapdoor'.

feasible (e.g., $B + d$ is a prime number). For completeness, the factorization of $\frac{B^2-d^2}{A^2}$ is

$$2^2 \cdot 5 \cdot 23 \cdot 359 \cdot 2089 \cdot 39733 \cdot 44059 \cdot 74353 \cdot$$
$$376287243430425811904334555393892643554045789647043 47 \ldots$$
$$\ldots 59039416676945740598806299461624575502089058332472952 \ldots$$
$$\ldots 94279089212441484219144994 63.$$

Once the factorization is known, we apply Simon's algorithm, implemented in Pari/GP [Pari] as `qfsolve()`, to compute a rational solution to the equation $pa^2 + pb^2 + c^2 = \frac{B^2-d^2}{A^2}$. A rational solution is given by

$$a = 32319123496536786843254458765608553095663568521872334 \ldots$$
$$\ldots 297530315749275438736572/z$$
$$b = 37902893736016880777193854875253045553175457573067191 \ldots$$
$$\ldots 240634037840067475117556 0/z$$
$$c = 85437128777417136022423941321585505761757160615798739 \ldots$$
$$\ldots 72406075696054195168847143870020389324092617191284723 \ldots$$
$$\ldots 80905798835064955553407208320599901478282089806543945 \ldots$$
$$\ldots 266931422175906643935346/z,$$

where

$$z = 87978348577011335417453239649099382225650021375809220 \ldots$$
$$\ldots 48203544412114079932641795709491238464691706755851 19.$$

Once θ is computed one has to compute an order $\mathcal{O}_0$ which contains θ. This can be accomplished in various ways. One way is to find a θ' such that $\theta\theta' + \theta'\theta = 0$ and θ'^2 is an integer multiple of the identity. This amounts to finding the kernel of the linear map $\eta \mapsto \theta\eta + \eta\theta$, which is a 2-dimensional vector space over $\mathbb{Q}$ (i.e., one chooses an element in this kernel and then multiplies it with a suitable integer). It is preferable to construct $\mathcal{O}_0$ in this way so that the discriminant of the order is the square of the reduced norm of $\theta\theta'$. In particular, if we choose a θ' whose norm is easy to factor, then the discriminant is also easy to factor. One has a lot of flexibility in choosing θ' and lattice reduction techniques help finding one which is sufficiently small and has an easy factorization. Note that the norm of θ' will always be divisible by p since the discriminant of every order is a multiple of p (and the norm of θ is coprime to p). Finally, one can compute a maximal order containing $\mathcal{O}_0$ using [Magma]'s `MaximalOrder()` function.

7.6 — Additional examples of trapdoored primes

In the examples in Subsection 7.4.4, we let $A = 2^{216}$, $B = 3^{300}$, $e = 1$. We let d equal $B \bmod A^2$, and $D = \frac{B-d^2}{A^2}$, hence

$$D = 16896420333246701930066245846797285820453043046692612 \ldots$$
$$\ldots 34160275705261296847619733634147787139416180071370253 \ldots$$
$$\ldots 15187569458339798745287263097168617279199182380018 0.$$

We first choose $c = 53$, then $D - c^2$ is a prime number (i.e., $a = 1,\ b = 0$),

$$p = 168964203332467019300662458467972858204530430466926 12 \ldots$$
$$\ldots 34160275705261296847619733634147787139416180071370253 \ldots$$
$$\ldots 15187569458339798745287263097168617279199182379737 1.$$

When $c = 355$, then $D - c^2$ is 5 times a prime number, namely,

$$p = 337928406664934038601324916935945716409060860933852 24 \ldots$$
$$\ldots 68320551410522593695239467268295574278832360142740506 \ldots$$
$$\ldots 30375138916679597490574526194337234558398364734831.$$

Both of these primes are congruent to 3 modulo 4.

We also give additional examples of Pythagorean triples as described in Section 7.4.5. In particular, let

$$B = 17^{60},$$
$$A = 2^5 \cdot 3^2 \cdot 5^2 \cdot 7 \cdot 11 \cdot 13 \cdot 19 \cdot 23 \cdot 41 \cdot 47 \cdot 59 \cdot 61 \cdot 101 \cdot 181 \cdot 191 \cdot 199 \cdot 239 \cdot 421$$
$$\cdot 541 \cdot 659 \cdot 769 \cdot 2281 \cdot 16319 \cdot 30119 \cdot 285599 \cdot 391679 \cdot 1039081 \cdot 1109159$$

For this, $177AB - 1 \equiv 3 \pmod 4$ is prime. Finally, a powersmooth example is given by

$$B = 5^8 \cdot 13^4 \cdot 17^4 \cdot 29^4 \cdot 37^4 \cdot 41^4 \cdot 53^4 \cdot 61^4 \cdot 73^4 \cdot 89^4 \cdot 97^4,$$
$$A = 2^4 \cdot 3 \cdot 7 \cdot 11 \cdot 23 \cdot 31 \cdot 127 \cdot 199 \cdot 811 \cdot 2903 \cdot 155383 \cdot 842041 \cdot 933199 \cdot 1900147$$
$$\cdot 8333489 \cdot 21629743 \cdot 30583723 \cdot 69375497$$

For this, $19AB - 1 \equiv 3 \pmod 4$ is prime.

Chapter 8

How to not break SIDH

This chapter is for all practical purposes identical to the paper *How to not break SIDH* [MP19] authored jointly with Chloe Martindale, which was published at CFAIL 2019.

8.1 — Introduction

This chapter's topic of interest is the historically first practical isogeny-based key exchange: *Supersingular Isogeny Diffie–Hellman* (SIDH), conceived by Jao and De Feo in 2011 [JD11], is first and foremost an ephemeral Diffie–Hellman-like key exchange. Unfortunately, it seems impossible to efficiently determine whether a public key was generated honestly; this leads to an active reaction attack which recovers a static private key in a linear (in the key size) number of queries [GPST16]. Based on this observation, SIDH was later transformed into *SIKE* [Jao+17], a key-encapsulation mechanism (KEM) which is currently a second-round contestant in NIST's call for post-quantum cryptographic constructions [NIST16]. In SIKE, one party (the server) can use a static key, while the other party generates a new ephemeral key pair for every connection. The construction is generally the same as SIDH, except that as part of his side of the key exchange, Bob encrypts his private key with the shared secret and sends it to Alice, who can then verify that the public key matches what one would get from Bob's alleged private key when following the protocol honestly. If Alice performs this check before doing anything else with the shared secret, she can be sure not to leak any information to dishonest clients: Bob only learns whether he was honest or not, but he is probably already aware of that.

This chapter summarizes some of our and others' fruitless attempts to cryptanalyze SIDH, including a discussion of the reasons why they failed. We hope that this will be useful to other (in particular, novice) researchers in the field of isogeny-based cryptography: In the past, we have observed a tendency among practitioners to rediscover, and sink time into, some of the ideas outlined in the following. Ideally, this work will provide a shortcut for those poor souls, allowing them to skip past some of the approaches doomed to fail. Finally, we strongly believe that publishing negative results can be valuable: One person's useless observation may be another person's missing link.

Finally, note that we do expect the ideas outlined in the following to strike experienced readers as naïve or foolish. This is by design: Documenting the insight to be gained while debunking — in hindsight — flawed ideas is exactly the point of this work. *"Trivial" is but another word for "we understood it".*

Acknowledgements. The negative results presented here are the result of discussions with many other researchers. We have tried to acknowledge all specific discussions in the relevant subsections. We would like to especially thank Tanja Lange for useful discussions regarding almost every part, if not every part, of this chapter, as well as Dan Bernstein, Dan Boneh, Steven Galbraith, Ben Smith, and Fré Vercauteren for many insightful discussions.

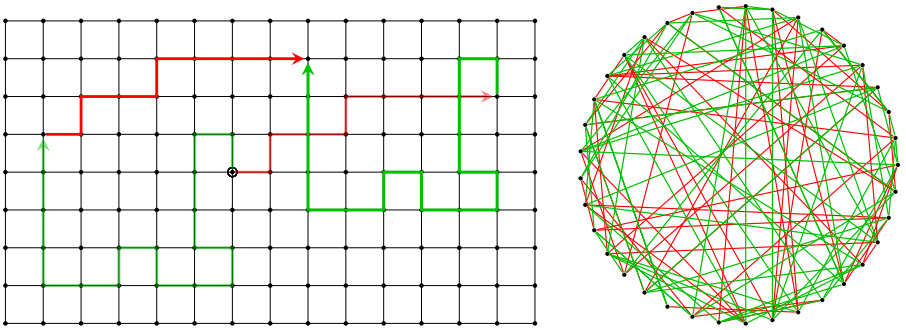


Figure 8.1: Left: Diffie–Hellman on a (too) structured graph. Right: The supersingular $\{2, 3\}$-isogeny graph over $\mathbb{F}_{431^2}$.

8.2 — Preliminaries

In this section, we give an account of the SIDH construction, introduce the problems it poses to cryptanalysts, and finally summarize the most important mathematical properties of the objects of interest.

8.2.1 – The SIDH key-exchange protocol [JD11]. The core idea in isogeny-based key exchange is to compose two random walks on an isogeny graph of elliptic curves in such a way that the end node of both ways of composing is the same. However, the graph used in SIDH is chaotic — it does not carry a computationally useful structure regular enough to support the evident Diffie–Hellman-style key exchange depicted in Figure 8.1.

This creates a serious correctness challenge for key-exchange schemes trying to make use of this graph. The resolution of this problem is the core contribution of SIDH: By sending extra information (so-called "auxiliary points") that helps Alice and Bob orient themselves when walking from the other party's public key node, they are able complete the DH "diamond"  to obtain a shared secret.

Recall from Proposition 2.30 the following fundamental result:

Lemma 8.1. *Let E be an elliptic curve and H a finite subgroup of E. Then there exists an elliptic curve E/H and a separable isogeny $\varphi_H : E \longrightarrow E/H$ whose kernel is H. The codomain E/H and isogeny φ_H are unique up to isomorphism.*

Parameters. The main parameter in SIDH is a large prime p of the form $p = \ell_A^{n_A} \ell_B^{n_B} f - 1$, where ℓ_A, ℓ_B are distinct small primes (typically $2, 3$) and f is a small cofactor (often 1) that is not divisible by ℓ_A or ℓ_B.[1]

Other parameters are: a supersingular elliptic curve $E_0/\mathbb{F}_p$,[2] a basis (P_A, Q_A) of $E_0[\ell_A^{n_A}]$, and a basis (P_B, Q_B) of $E_0[\ell_B^{n_B}]$. Typically, $E_0 : y^2 = x^3 + x$ is used.

Note that the choice of p and E_0 implies that P_A, Q_A, P_B, Q_B are all defined over $\mathbb{F}_{p^2}$, since $E_0(\mathbb{F}_{p^2}) \cong \mathbb{Z}/(p+1) \times \mathbb{Z}/(p+1)$.

We refer to the curves used in SIDH as "SIDH curves". These curves form a complete set of representatives of *all* isomorphism classes of supersingular elliptic curves over $\overline{\mathbb{F}_p}$.

[1] In Chapter 7, the more general notation $A = \ell_A^{n_A}$ and $B = \ell_B^{n_B}$ was used.

[2] In principle, it is not required that E_0 be defined over $\mathbb{F}_p$, but this is beneficial for a variety of reasons. However, there are some reasons to be concerned about special curves like the common choice $j = 1728$; see Section 8.4.3.

Keys. Alice's secret key is an integer $a \in \{0, ..., \ell_A^{n_A} - 1\}$, which defines the cyclic subgroup $A = \langle P_A + [a]Q_A \rangle \leq E_0[\ell_A^{n_A}]$.

Her public key is the curve E_0/A together with the images $\varphi_A(P_B), \varphi_A(Q_B)$ of Bob's public basis under her (secret) isogeny $\varphi_A \colon E_0 \to E_0/A$.

Bob follows the same process: his secret key is an integer $b \in \{0, ..., \ell_B^{n_B} - 1\}$, which defines a cyclic subgroup $B = \langle P_B + [b]Q_B \rangle \leq E_0[\ell_B^{n_B}]$, and his public key is $(E_0/B, \varphi_B(P_A), \varphi_B(Q_A))$.

Key exchange. Bob takes Alice's public key $(E_0/A, \varphi_A(P_B), \varphi_A(Q_B))$ and uses the points contained in it to *shift* his secret $B \leq E_0[\ell_B^{n_B}]$ to E_0/A: He obtains

$$B' := \varphi_A(B) = \langle \varphi_A(P_B) + [b]\varphi_A(Q_B) \rangle \leq (E_0/A)[\ell_B^{n_B}] \,.$$

This allows him to compute the shared secret $(E_0/A)/B' \cong E_0/\langle A, B \rangle$.

Alice proceeds in exactly the same way: she computes $A' := \varphi_B(A)$ to obtain the shared secret $(E_0/B)/A' \cong E_0/\langle A, B \rangle$.

8.2.2 – Basic observations.

Rational points. Tate's Theorem 2.35 implies that E_A and E_B have the same number of points as E_0, that is, $(p+1)^2$. Even stronger, [Waso8, Theorem 4.4] shows that all SIDH curves E have isomorphic groups of $\mathbb{F}_{p^2}$-rational points:

$$E(\mathbb{F}_{p^2}) \;\cong\; \mathbb{Z}/(p+1) \times \mathbb{Z}/(p+1) \,.$$

Among other things, this (together with the smoothness of $p+1$) implies that logarithms in $E(\mathbb{F}_{p^2})$ can be computed in polynomial time, and very efficiently in practice, using the Pohlig–Hellman algorithm. Similarly, the generalization to "two-dimensional discrete logarithms" — or in other words, decomposing a point in $E(\mathbb{F}_{p^2})$ over a basis of the group of rational points — is efficient [Suto7, Algorithm 9.3]. Hence, the information $\varphi_A(P_B), \varphi_A(Q_B)$ and $\varphi_B(P_A), \varphi_B(Q_A)$ that Alice and Bob transmit reveals much more than just the action of the secret on mere two points: it encodes the action of φ_A resp. φ_B on the *entire* $\ell_B^{n_B}$- resp. $\ell_A^{n_A}$-torsion.

The graph structure. As mentioned before, the set of (isomorphism classes of) SIDH curves together with (a subset of) the rational isogenies between them can be viewed as a graph, a very useful viewpoint for understanding and arguing about isogeny-based cryptosystems. For example, for every finite set $S \subseteq \mathbb{Z}_{\geq 2}$, one obtains an S-isogeny graph where the edges are isogenies whose degree is in S; an important special case is $S = \{\ell\}$ where ℓ is a (typically small) prime. One can prove [Eic38] that (up to isomorphism) there are $\lfloor p/12 \rfloor + \varepsilon$ supersingular elliptic curves defined over $\overline{\mathbb{F}_p}$, where $\varepsilon \in \{0, 1, 2\}$.[3] It turns out that all of these isomorphism classes have a representative defined over $\mathbb{F}_{p^2}$, hence the SIDH protocol actually works on the graph of *all* supersingular elliptic curves defined over characteristic-p fields.

Moreover, the ℓ-isogeny graph is always connected (for $p \nmid \ell$), and it has excellent mixing properties [Piz90; JD11]: Any two nodes are expected to be connected via only $O(\log_\ell p)$ steps in the ℓ-isogeny graph, that is, an $\ell^{O(\log_\ell p)}$-isogeny. By counting, it is clear that one cannot hope for faster mixing: Since the ℓ-isogeny graph is $O(\ell)$-regular, there are at most $O(\ell^d)$ nodes at distance $\leq d$ from any given point in the graph. Setting $d \in \Omega(\log_\ell p)$ makes sure one can at least hope to reach all $\Theta(p)$ nodes within d steps, and the theory guarantees that this is indeed true. More careful handling of the constants in the relevant mixing bounds shows that the leading coefficient of the $O(\log_\ell p)$ is in fact a small constant (< 6 for reasonably-sized p), hence the

[3] In the SIDH setting, where $p \equiv 11 \pmod{12}$, we have $\varepsilon = 2$.

SIDH shared secret is close to uniformly random in the supersingular isogeny graph. On the other hand, this is clearly not true for the public keys, which (by counting) lie in a negligibly small subset whose density is only $O(1/\sqrt{p})$.

Endomorphism rings. It is a classical result of Deuring [Deu41] that the (full) endomorphism ring of a supersingular elliptic curve defined over $\overline{\mathbb{F}_p}$ is (isomorphic to) a maximal order in the quaternion algebra $B_{p,\infty}$ ramified at p and ∞. In the SIDH setting,[4] this means there exists a ring isomorphism from the endomorphism algebra $\mathrm{End}^\circ(E) = \mathrm{End}(E) \otimes_{\mathbb{Z}} \mathbb{Q}$ to the $\mathbb{Q}$-algebra $B_{p,\infty} = \mathbb{Q} \oplus \mathbb{Q}\mathbf{i} \oplus \mathbb{Q}\mathbf{j} \oplus \mathbb{Q}\mathbf{ij}$ with multiplication rules $\mathbf{i}^2 = -1$, $\mathbf{j}^2 = -p$, and $\mathbf{ij} = -\mathbf{ji}$. The endomorphism ring $\mathrm{End}(E)$ is thus generated by four linearly independent elements of $B_{p,\infty}$ which span a maximal proper subring with respect to inclusion. The most prominent example is the SIDH starting curve $E_0 \colon y^2 = x^3 + x$, whose endomorphism ring is generated as a ring by the endomorphisms ι and $(\iota + \pi)/2$, where $\iota \colon (x, y) \mapsto (-x, \sqrt{-1} \cdot y)$ is an automorphism of order 4 and $\pi \colon (x, y) \to (x^p, y^p)$ is the p-power Frobenius endomorphism.[5] Hence a $\mathbb{Z}$-basis of $\mathrm{End}(E_0)$ is given by $\langle 1, \iota, \frac{\iota+\pi}{2}, \frac{1+\iota\pi}{2} \rangle$. Note that one can in principle, although there are usually computational hurdles, express the endomorphisms of any other supersingular elliptic curve over $\overline{\mathbb{F}_p}$ with respect to this basis: Fixing an ℓ-isogeny $\psi \colon E_0 \to E$, we get an injective ring homomorphism

$$\mathrm{End}(E) \hookrightarrow \mathrm{End}^\circ(E_0) \cong B_{p,\infty},\ \alpha \mapsto \psi\alpha\widehat{\psi}/\ell. \tag{8.1}$$

Notice that evaluating an endomorphism given in this representation requires first computing an elliptic-curve point division by ℓ, which typically lies in a field extension of degree $\Omega(\ell)$, hence special care needs to be taken to make sure this is feasible: for instance, choose ℓ to be powersmooth [Eis+18, Algorithm 5].

Also note that $\mathrm{End}(E)$ has many commutative subrings, the most important example being $\mathbb{Z}[\pi]$ when E is defined over $\mathbb{F}_p$. In principle, an efficient commutative subring can give rise to a subexponential quantum attack [CJS14], although it seems just as hard to find *an* endomorphism as to break the scheme in the first place. Therefore the only known example of this idea being useful is $\mathbb{Z}[\pi]$. It does mean, however, that finding an isogeny to a curve defined over $\mathbb{F}_p$ can lead to a subexponential quantum attack; cf. Section 8.3.1.

Not only is the endomorphism ring isomorphic to a maximal quaternion order, but the *Deuring correspondence* also works in the other direction: there is a bijection between the set of supersingular elliptic curves over $\overline{\mathbb{F}_p}$, up to isomorphism, and the set of "oriented" maximal orders in $B_{p,\infty}$ [Voi18, Section 42.4]. Simply put, this means for every maximal order $\mathcal{O} \subseteq B_{p,\infty}$ there is a set $\{j, j'\} \subseteq \mathbb{F}_{p^2}$ such that curves with j-invariant j or j' have endomorphism ring $\mathcal{O}$; furthermore, we have $j' = j^p$, hence there is either one such curve, which can be defined over $\mathbb{F}_p$, or the two curves are both defined over $\mathbb{F}_{p^2}$ and Galois conjugates of each other.

Moreover, this correspondence is categorical: Fixing a supersingular elliptic curve E_0 as a base object, every ℓ-isogeny $\alpha \colon E_0 \to E$ corresponds to a left[6] ideal $\mathfrak{a} \subseteq \mathrm{End}(E_0)$ of norm ℓ, and vice-versa (up to post-composition with isomorphisms) [Voi18, Section 42.3]. The codomain E is determined up to isomorphism by the left-ideal *class* of $\mathfrak{a}$, hence finding different representatives of an ideal class corresponds to finding different isogenies between two fixed curves. Notably, given a left ideal $\mathfrak{a} \subseteq E_0$, it is easy to find the endomorphism ring of the image curve of the

[4] The technical condition here is $p \equiv 3 \pmod 4$; the other cases are slightly different but not harder in principle.

[5] To see why $(\iota+\pi)/2$ is an (integral) endomorphism of E_0, note that the affine 2-torsion points of E_0 are all of the form $(\xi, 0)$ where $\xi^3 + \xi = 0$, hence $\xi \in \{0, \pm\sqrt{-1}\}$. Since $\xi^p = -\xi$, we have $(\iota+\pi)(\xi, 0) = (-\xi, 0) + (\xi^p, 0) = [2](-\xi, 0) = \infty$.

[6] Since conjugation swaps left and right multiplication, one could equivalently use right ideals.

corresponding isogeny: Under the embedding $\mathrm{End}(E_0) \hookrightarrow B_{p,\infty}$ given in (8.1), it is isomorphic to another maximal order of $B_{p,\infty}$, and in fact, it turns out that the right order is the adequately named *right order*

$$\mathcal{O}_R(\mathfrak{a}) = \{\, r \in B_{p,\infty} \mid \mathfrak{a}r \subseteq \mathfrak{a} \,\}.$$

It may suggest itself at first that this correspondence will be very useful as an attack tool against SIDH. However, it seems that one simply cannot efficiently transcend into this alternate, equivalent reality: All known approaches to compute the endomorphism ring of a given curve essentially go through first finding an isogeny to either another curve with known endomorphism ring (such that one can compute the right order as above), or to itself [Koh96].

8.2.3 – Attack avenues against SIDH. The obvious way to attack SIDH is to try to recover one of the secret isogenies φ_A, φ_B from the public information. (We will often, without loss of generality, silently assume that we are attacking Alice's key.) A priori, it may seem like one requires one of the actual secret isogenies; however, Galbraith–Petit–Shani–Ti have demonstrated that *any* isogeny ψ between E_0 and one of $\{E_A, E_B\}$ is enough to recover the *right* isogeny and therefore break the system [GPST16]. The reduction makes use of the fact that the secret isogenies in SIDH are relatively "short" compared to a "random" isogeny between two given curves: There are $\Theta(\sqrt{p})$ different secrets, while the graph size is $\Theta(p)$, hence only an exponentially small fraction of SIDH curves can be reached from the starting curve by isogenies shorter than the secret keys. This observation is combined with the fact that isogenies from E_0 correspond to left ideals of $\mathrm{End}(E_0)$, and isogeny *codomains* correspond to left-ideal *classes* (see Section 8.2.2): The reduction first finds the ideal defining the known isogeny $\psi\colon E_0 \to E_A$, then employs lattice-basis reduction to compute an equivalent ideal of small norm. Except for rare cases of bad luck, this small-norm ideal corresponds to the secret isogeny φ_A. The "pure" problem of finding an isogeny between E_0 and a given SIDH curve is discussed in Section 8.3.

The isogeny-finding problem does not capture the full power of an attacker in SIDH. In addition to the target curve, attackers also see the action of the secret isogeny on a coprime torsion subgroup, represented by the action on a few points that span said subgroup. These auxiliary points are the main innovation of SIDH, and the new setting they enable is the reason for SIDH's improved quantum security over other isogeny-based key exchanges [Cou06; RS06; Cas+18], but the additional information that Alice and Bob disclose may also be worrisome: Petit has obtained cryptanalysis results on modified variants of SIDH using these extra points [Pet17]. (Un)fortunately, it seems like there is little hope for his approach to apply to the original, balanced parameters; see Section 8.4.3. Other potential (but fruitless) approaches based on the extra points are outlined in Section 8.4.

Finally, note that analogously to the classical Diffie–Hellman setting, there is of course also the potential for an attack that obtains the shared secret *without* first recovering one party's secret key. Similar to the classical case, we are not aware of any ideas to attack SIDH from this direction.

8.3 — Failed attempts to attack the pure isogeny problem

The *pure isogeny problem* for supersingular elliptic curves is:

> Given supersingular E and $E'/\mathbb{F}_{p^2}$, optionally with the guarantee that E and E' are ℓ^n-isogenous for some ℓ^n, compute an isogeny $\phi\colon E \to E'$.

We refer to this as the "pure" isogeny problem because the hardness assumption on which SIDH is based features a stronger attacker: they also have knowledge of the images of some points

under the isogenies φ_A, φ_B in addition to just the domain and the codomains. Moreover, recall from Section 8.2.3 that it is sufficient to recover *an* isogeny between E_0 and one of E_A, E_B; the correct isogeny can then (usually) be found by employing ideal-based techniques.

The best known classical or quantum attack to find an isogeny $E_0 \to E_A$ in the SIDH setting is essentially a generic approach searching for Alice's secret isogeny φ_A: compute and store random walks of length $n_A/2$ in the ℓ_A-isogeny graph starting from E_0 and E_A until two of them "meet in the middle"; this algorithm takes time $\tilde{O}(p^{1/4})$ as Alice's isogeny from E_0 to E_A has degree approximately $p^{1/2}$. In practice, the memory cost of this algorithm is prohibitively high, so parallel versions of van Oorschot–Wiener's collision search algorithm with almost the same theoretical time complexity but much better time-space tradeoffs and hence superior real-world performance, are considered to be the best known attack against SIDH/SIKE [Adj+18; Cos+20]. Note that Tani's $\tilde{O}(p^{1/6})$ quantum algorithm [Tan07] for the claw-finding problem is deemed unlikely to outperform the classical algorithm of van Oorschot–Wiener:

> Our conclusion is that an adversary with enough quantum memory to run Tani's algorithm with the query-optimal parameters could break SIKE faster by using the classical control hardware to run van Oorschot–Wiener. [JS19]

8.3.1 – Finding the $\mathbb{F}_p$-subgraph. The idea of using the $\mathbb{F}_p$-subgraph to get a better classical attack on the pure isogeny problem was first studied by Delfs and Galbraith [DG16]. Biasse, Jao, and Sankar [BJS14] later[7] applied the same ideas to construct a more efficient quantum algorithm. The (other) attempts at exploiting the $\mathbb{F}_p$-subgraph presented here have certainly been considered by many people, but not written down as it has not (yet?) led to an improved attack on SIDH.

Trying to find a path to a curve in the $\mathbb{F}_p$-subgraph turns out to be common theme in attempts at attacking SIDH, so we now discuss the consequences such an algorithm would have.

Definition 8.2. *Let S be a set of nodes in the SIDH ℓ-isogeny graph G, and let $S' \subseteq S$ be the subset of those nodes that are defined over $\mathbb{F}_p$. We define the* $\mathbb{F}_p$-subgraph *of G to be the full subgraph of G with nodes from S'.*

Fundamentally, the $\mathbb{F}_p$-subgraph forms a distinguished subset of the full isogeny graph that is easily recognizable once we have found it, and it is also easy to identify those edges that go to another node inside this subgraph.

Delfs–Galbraith use this observation to split the problem of finding an ℓ-isogeny between two arbitrary curves E, E' into two smaller subproblems: finding a path from both E and E' to curves defined over $\mathbb{F}_p$, and then connecting these two curves by an isogeny inside the subgraph. The composition of these three isogenies forms an isogeny $E \to E'$.

In total, one can show that there are approximately $\sqrt{p}$ supersingular elliptic curves defined over $\mathbb{F}_p$. The $\mathbb{F}_p$-subgraph G' of the SIDH ℓ-isogeny graph, with ℓ a prime, is either (if ℓ odd) a disjoint union of cycles of the same length, or (if $\ell = 2$) such a union of cycles with one single extra leaf "hanging down" from each node in the cycles. The components of these graphs are known as a *volcanoes*, and we call the set of non-leaf nodes the *surface*.

Note that this implies that the surface subgraph is 2-regular, hence using a single ℓ-isogeny $\mathbb{F}_p$-subgraph leads to a time complexity of $\Theta(\sqrt{p})$ for either finding a path between two given nodes or determining that they do not lie in the same component. Using multiple ℓ yields an

[7] The publication dates suggest the opposite chronology, but a preprint of [DG16] was available online on the arXiv as early as October 2013.

improvement, though: One can show that subexponentially many ℓ are sufficient to connect all nodes, and (under GRH) that random walks on this combined graph mix quickly [JMV09]. Thus the usual meet-in-the-middle techniques apply, reducing the time complexity of connecting two $\mathbb{F}_p$-subgraph curves to $\widetilde{O}(p^{1/4})$. Note how this is not better at attacking SIDH than the easier meet-in-the-middle attack outlined before; this is because the isogenies in SIDH are known to be particularly short, a property which cannot be exploited by the Delfs–Galbraith approach since almost none of the curves on the path are defined over $\mathbb{F}_p$.

Moreover, *finding* the $\mathbb{F}_p$-subgraph in the first place by brute force costs $\widetilde{O}(\sqrt{p})$: The density of that subgraph is roughly $1/\sqrt{p}$, hence random walks can be expected to find a curve defined over $\mathbb{F}_p$ after walking approximately a number of steps that is the reciprocal of this proportion, i.e., $\sqrt{p}$.

With respect to quantum attacks, similar problems apply: Once the $\mathbb{F}_p$-subgraph has been found, isogeny walks can be interpreted as a commutative class-group action of an imaginary quadratic number ring, and therefore two nodes can be connected using a subexponential-time hidden-shift quantum algorithm [Kup05; Kup13]. This was first applied to isogeny graphs of elliptic curves in [CJS14]. However, there is still no known efficient quantum algorithm to find the $\mathbb{F}_p$-subgraph, hence this does not lead to an improved attack.

An $\mathbb{F}_p$-compass? As stated above, the main problem to solve is finding an isogeny to a curve defined over $\mathbb{F}_p$. The evident brute-force approach is not cheaper than breaking SIDH "directly" using meet-in-the-middle or collision finding, and more sophisticated methods seem out of reach. For instance, one observation is that a curve at distance d from the $\mathbb{F}_p$-subgraph in the ℓ-isogeny graph has an endomorphism of degree $\ell^{2d}p$ given by walking to the $\mathbb{F}_p$-subgraph, applying Frobenius, and walking back. Why this may seem a promising approach for detecting the $\mathbb{F}_p$-subgraph, it runs into the same problems as always: Checking whether a curve has an endomorphism of a certain norm seems to boil down to simply trying to *find* that endomorphism, which is infeasible unless (here) the distance d to the $\mathbb{F}_p$-subgraph is already extremely small. We have seen many similar or equivalent, but equally fruitless, attempts in this direction come and go in the past. For example, if a curve E is close to the $\mathbb{F}_p$-subgraph, there is a short isogeny between E and its Galois conjugate $E^{(p)}$, but again there is no known way to detect that isogeny unless we are already close enough to find the $\mathbb{F}_p$-subgraph with a generic approach.

Other subrings? One way to interpret the $\mathbb{F}_p$-subgraph is as the subset of curves with a certain endomorphism of norm p, namely the p-power Frobenius endomorphism. Hence, one is implicitly looking for those supersingular elliptic curves whose endomorphism ring contains the Frobenius order $\mathbb{Z}[\pi]$, and in principle the same sort of subgraph exists for other commutative subrings, like for example $\mathbb{Z}[\iota]$, although in this case it only consists of the single node E_0.

Finding, for instance, a bigger commutative subring than $\mathbb{Z}[\pi]$ that is contained in almost all endomorphism rings in the graph would potentially allow to spend less time on searching for the associated subgraph, but still apply the subexponential quantum attack once it is found.

However, there are a number of problems associated with this approach, one fundamental in nature and the others (as usual) computational: The embedding $\mathrm{End}(E) \hookrightarrow B_{p,\infty}$ is highly *non-canonical*. This means that even if one was able to compute (subrings of) the endomorphism rings of two curves, there is still no way to tell how these rings are related under the embedding from (8.1). The usual strategy to deal with this problem in theory is to make sure the embeddings are always compatible when considering two isogenous curves, but without knowing an isogeny, this of course seems impossible to do in practice. This issue does not apply to $\mathbb{Z}[\pi]$ as,

given a curve $E/\mathbb{F}_p$, the endomorphism π is always trivial to find (it is just $(x, y) \mapsto (x^p, y^p)$), and since (by definition) isogenies defined over $\mathbb{F}_p$ commute with π, we automatically have $\psi\pi\widehat{\psi}/(\deg(\psi)) = \pi\psi\widehat{\psi}/(\deg(\psi)) = \pi$ for all isogenies $\psi\colon E \to E'$ defined over $\mathbb{F}_p$. Therefore it is possible to identify a canonical subring of the endomorphism ring which is automatically compatible between different $\mathbb{F}_p$-isogenous curves.

The computational problems are the usual: It is not clear how to tell whether a given curve E has an endomorphism of a given norm and trace, it seems impossible to make sure these endomorphisms are compatible choices without first finding an isogeny between the two curves in question, and for the quantum part of the attack it must also be efficient to *evaluate* the endomorphisms on points.

8.3.2 – Lifting to characteristic zero. It is relatively well-known that to an ordinary elliptic curve $E/\mathbb{F}_q$ one can canonically associate an elliptic curve $E'/\mathbb{Q}_q$[8] with the same endomorphism ring (viewed as an order in a quadratic number field) — this is normally referred to as the "canonical lift" [LST64] [Mes72, Appendix], and E is the (unique) *reduction* of E'.

It is possible to compute this lift, for example via Satoh's algorithm [Satoo], albeit not efficiently for large characteristic p. Furthermore, it is functorial — we can also lift (and reduce) isogenies. A natural question is:

> Given a supersingular elliptic curve $E/\mathbb{F}_{p^2}$ with endomorphism ring $\mathcal{O}$, is there a way to canonically construct an elliptic curve $E'/\mathbb{C}$ whose endomorphism ring is isomorphic to a (well-chosen) commutative subring of $\mathcal{O}$?

Suppose for the sake of argument that such a construction is efficiently computable and that we can also lift and reduce isogenies. Then to find a path between $E_1/\mathbb{F}_q$ and $E_2/\mathbb{F}_q$ we could first compute their canonical lifts $E_1'/\mathbb{Q}_q$ and $E_2'/\mathbb{Q}_q$ respectively and then compute an isogeny $E_1' \to E_2'$, which one could subsequently hope to reduce back to $\mathbb{F}_q$. As $\mathbb{Q}_q \hookrightarrow \mathbb{C}$, the lifts E_1' and E_2' can be viewed as complex elliptic curves. As a complex elliptic curve is nothing but a torus and an isogeny between two such curves is just a $\mathbb{C}$-linear map, one may hope to be able to easily compute an isogeny over $\mathbb{C}$ using some linear algebra.

Unfortunately, the computational methods for lifting an ordinary elliptic curve $E/\mathbb{F}_q$, such as Satoh's [Satoo], all exploit a known endomorphism τ on E — in their case τ is the Frobenius π — and construct an elliptic curve $E'/\mathbb{C}$ with endomorphism algebra $\mathrm{End}^\circ(E') \cong \mathbb{Q}(\tau)$.

For a generic supersingular elliptic curve $E/\mathbb{F}_{p^2}$, the only endomorphisms we know of are scalar multiplications, i.e., lie in $\mathbb{Z}$. (Recall that in the SIDH case the p^2-power Frobenius is just $[-p]$.) So even if we could lift E in a meaningful and computable way to $E'/\mathbb{C}$ while preserving a known endomorphism, we simply would not know how to find that endomorphism in the first place (as usual).

Computing a path from a generic supersingular elliptic curve $E/\mathbb{F}_{p^2}$ to a curve defined over $\mathbb{F}_p$ would be helpful in this context, but then there would then be easier ways to proceed, see Section 8.3.1.

8.3.3 – Weil restrictions. *Acknowledgements.* Some of the ideas in this section were discussed with participants of the Spontaneous Isogeny Day in Leuven in October 2018. We had particularly enlightening discussions on this topic with Wouter Castryck, Steven Galbraith, Joost Renes, Ben Smith, and Fré Vercauteren (alphabetical order).

[8] The field $\mathbb{Q}_q$, which can be embedded into $\mathbb{C}$, is the fraction field of $\mathbb{Z}_q$, which is a finite extension of the p-adic integers $\mathbb{Z}_p$, which has as elements power series in p.

To any (supersingular) elliptic curve $E/\mathbb{F}_{p^2}$, one can in a natural way associate a (supersingular) principally polarizable abelian surface[9] $W(E)/\mathbb{F}_p$ called the *Weil restriction*.[10] Modulo (many) technical details, the fundamental idea is to interpret the defining equation of E over $\mathbb{F}_{p^2}$ as a set of equations over $\mathbb{F}_p$ instead by plugging in, then splitting over, an $\mathbb{F}_p$-basis of $\mathbb{F}_{p^2}$. The Weil restriction is functorial: isogenies of elliptic curves defined over $\mathbb{F}_{p^2}$ restrict to isogenies of their Weil restrictions over $\mathbb{F}_p$. This means that the isogeny graph of supersingular elliptic curves defined over $\mathbb{F}_{p^2}$ can be viewed as a subgraph of the isogeny graph of supersingular principally polarized abelian surfaces over $\mathbb{F}_p$.

The center of the $\mathbb{F}_p$-rational endomorphism ring $\mathrm{End}_{\mathbb{F}_p}(A)$ of an abelian variety defined over $\mathbb{F}_p$ is is an order in $\mathbb{Q}(\pi)$, where π is the p-power Frobenius of A [Tat66, Theorem 2].

One might hope that in fact $\mathrm{End}_{\mathbb{F}_p}(A)\otimes_{\mathbb{Z}}\mathbb{Q} = \mathbb{Q}(\pi)$, as happens when $\dim(A) = 1$. We considered what the consequences of this might be: Assume that for the Weil restriction $W(E_A)$ of E_A (Alice's public key), the $\mathbb{F}_p$-rational endomorphism ring is commutative. For all but finitely many primes ℓ, we then expect that the (ℓ,ℓ)-isogeny[11] graph of supersingular principally polarized abelian surfaces defined over $\mathbb{F}_p$ is a disjoint union of cycles, as justified at the end of this section. If there is a list $\ell_1,\ldots,\ell_n$ such that the connected component of the union of the $(\ell_1,\ell_1),\ldots,(\ell_n,\ell_n)$-isogeny graphs contains $W(E_0)$ and $W(E_A)$, then the problem of finding a path from $W(E_0)$ to $W(E_A)$ can be viewed as a *hidden shift* problem, for which, if the individual steps in the path, i.e., isogenies, can be efficiently computed, there is a subexponential quantum algorithm due to Kuperberg [Kup05; Kup13].

Any hope? We need the probability of $W(E_0)$ and $W(E_A)$ being in the same connected component C of the union of the $(\ell_1,\ell_1),\ldots,(\ell_n,\ell_n)$-isogeny graphs to be high, which can only happen if C contains the Weil restrictions of almost all the supersingular elliptic curves defined over $\mathbb{F}_{p^2}$.

We expect (as justified at the end of this section) that the (ℓ_i,ℓ_i)-isogeny graphs will be the disjoint union of cycles of length $O(\sqrt{p})$. There exist $\Theta(p)$ (Weil restrictions of) supersingular elliptic curves defined over $\mathbb{F}_{p^2}$, so to have any chance of C covering almost all of these, we would need to take n to be at least $\Omega(\sqrt{p})$.

Currently we cannot compute (ℓ,ℓ)-isogenies efficiently enough unless $\ell = 2$,[12] so we assume for the sake of argument that the complexity of a somewhat optimized algorithm to do this would scale at least as badly as Vélu's formulas for elliptic curves. That is, we assume that the evaluation of an (ℓ,ℓ)-isogeny takes time $\Omega(\ell)$. Since we need to take at least $\Omega(\sqrt{p})$ different primes ℓ, it is then definitely not true that "the individual steps in the path, i.e. isogenies, can be efficiently computed".

More ideas? We considered two variations on this idea:

1. Instead of hoping that $W(E_A)$ is in the same connected component is $W(E_0)$, hope that it is in the same connected component of the Weil restriction of some curve defined over $\mathbb{F}_p$. Approximately one in $\sqrt{p}$ (Weil restrictions of) elliptic curves over $\mathbb{F}_{p^2}$ are (Weil restrictions of) elliptic curves over $\mathbb{F}_p$, so looking at one cycle of length $O(\sqrt{p})$, i.e., just one (ℓ,ℓ)-isogeny graph, might be enough.

[9] To read more about principally polarized abelian varieties, see [EvM07, Chapter 11].
[10] To read more about Weil restrictions, see [DN03].
[11] To read more about (ℓ,ℓ)-isogenies, see [CR15].
[12] To read more about computing (2,2)-isogenies efficiently, see [Cos18].

However, since we do not know which curve over $\mathbb{F}_p$ we are looking for, it seems impossible to phrase this as a hidden shift problem, so Kuperberg's algorithm does not apply.

2. Recall that we assume that $\mathrm{End}_{\mathbb{F}_p}(W(E_A))$ is an order in $\mathbb{Q}(\pi)$, where π is the p-power Frobenius on $W(E_A)$. We explain below that $\pi = \zeta_8\sqrt{p}$, where ζ_8 is an eighth root of unity, and that we then expect that the application of an (ℓ,ℓ)-isogeny to a supersingular principally polarized abelian surface $A/\mathbb{F}_p$ can be viewed as the action of an ideal in the class group $\mathrm{cl}(\mathcal{O}_{\mathbb{Q}(\zeta_8\sqrt{p})})$ (the ℓ are chosen so that $\mathcal{O}_{\mathbb{Q}(\zeta_8\sqrt{p})} = \mathbb{Z}[\zeta_8\sqrt{p}]$ locally at ℓ). The reason that we expect the cycles in the (ℓ,ℓ)-isogeny graph to have length approximately $\sqrt{p}$ comes from this action — this is (heuristically) the size of this class group.

   However $\mathcal{O}_{\mathbb{Q}(\zeta_8\sqrt{p})}$ is *not* the largest commutative subring of $\mathrm{End}_{\overline{\mathbb{F}_p}}(A)$ (locally at ℓ): Since Frobenius commutes with every endomorphism, we could add another endomorphism to get a rank-4 $\mathbb{Z}$-module, the class group of which is likely to have a higher class number. But this is of course equivalent to finding non-obvious endomorphisms, which, if we could do, would lead to a much easier way of attacking SIDH, as explained in Section 8.2.2.

A couple of handwavy mathematical details. As stated above, we expect the following property, under the assumption that $\mathrm{End}_{\mathbb{F}_p}(W(E_A))$ is commutative: For all but finitely many primes ℓ, the (ℓ,ℓ)-isogeny graph of supersingular principally polarized abelian surfaces defined over $\mathbb{F}_p$ is a disjoint union of cycles. We also conjectured that the cycles have length $\Omega(\sqrt{p})$ (subject to some heuristics). We briefly justify our expectations here.

Suppose that ℓ does not divide the index $\left[\mathcal{O}_{\mathbb{Q}(\pi)} : \mathbb{Z}[\pi]\right]$, where π is the p-power Frobenius on $W(E_A)$. Since under our assumptions, for any supersingular abelian surface S over $\mathbb{F}_p$ with commutative $\mathbb{F}_p$-rational endomorphism ring, we have

$$\mathbb{Z}[\pi] \subseteq \mathrm{End}_{\mathbb{F}_p}(S) \subseteq \mathcal{O}_{\mathbb{Q}(\pi)},$$

it follows that every supersingular abelian surface $S/\mathbb{F}_p$ has endomorphism ring $\mathcal{O}_{\mathbb{Q}(\pi)}$ locally at ℓ. An isogeny of abelian surfaces is uniquely determined by its kernel (just like with elliptic curves). In particular, if I is an ideal of $\mathrm{End}_{\mathbb{F}_p}(S)$ then we define f_I to be the isogeny from S with kernel

$$\bigcap_{\alpha\in I} \ker(\alpha).$$

Following exactly the same proof strategy as for elliptic curves, it is believable that the class group of $\mathcal{O}_{\mathbb{Q}(\pi)}$ acts on the set of supersingular abelian surfaces over $\mathbb{F}_p$ with endomorphism ring $\mathcal{O}_{\mathbb{Q}(\pi)}$ via

$$I * E = f_I(E).$$

Going one step further, we suppose for the sake of argument that horizontal (ℓ,ℓ)-isogenies even come from the action of an ideal $\mathfrak{l}$ such that $\ell\mathcal{O}_{\mathbb{Q}(\sqrt{-p})} = \bar{\mathfrak{l}}\mathfrak{l}$. If, as in the elliptic curve case, the results for supersingular abelian surfaces over a prime field turn out to be analogous to results for ordinary abelian surfaces, then such an ideal would send a supersingular abelian surface $S/\mathbb{F}_p$ with endomorphism ring $\mathcal{O}_{\mathbb{Q}(\pi)}$ equipped with a principal polarization $\zeta\colon S \to S^\vee$ to a supersingular abelian surface $f_{\mathfrak{l}}(S)/\mathbb{F}_p$ with endomorphism ring $\mathcal{O}_{\mathbb{Q}(\pi)}$ equipped with a principal polarization $\ell\zeta$. The analogous result for the ordinary case that we refer to here is [Mar18b, Proposition 3.6.1].

If all of this holds, then the (ℓ,ℓ)-isogeny graph of any prime ℓ not dividing $\left[\mathcal{O}_{\mathbb{Q}(\pi)} : \mathbb{Z}[\pi]\right]$ that splits in $\mathcal{O}_{\mathbb{Q}(\pi)}$ is a cycle. Suppose that $\ell\mathcal{O}_{\mathbb{Q}(\pi)} = \mathfrak{l}\bar{\mathfrak{l}}$. Then the length of the cycle is given by the order of $[\mathfrak{l}]$ in $\mathrm{cl}(\mathcal{O}_{\mathbb{Q}(\pi)})$.

Furthermore, by a theorem of Manin and Oort [Oor74, p. 116], the Frobenius π equals $\zeta\sqrt{p}$, where ζ is a root of unity. By a theorem of Tate [Tat66, Theorem 2], our assumption that the endomorphism algebra $B = \mathrm{End}_{\mathbb{F}_p}(W(E_A)) \otimes_{\mathbb{Z}} \mathbb{Q}$ is commutative is equivalent to saying that $[B : \mathbb{Q}] = 4$, so $\zeta = \zeta_8$ is in fact an eighth root of unity, and the characteristic polynomial of Frobenius is $x^4 - p^2$. According to standard class group heuristics [CL84], we expect that $\mathrm{cl}(\mathcal{O}_{\mathbb{Q}(\zeta_8\sqrt{p})})$ is cyclic or almost cyclic, and has order $\Omega(\sqrt{p})$ — hence the (ℓ,ℓ)-isogeny graph, where ℓ satisfies all of the conditions above, is heuristically speaking the disjoint union of cycles of length approximately $\sqrt{p}$.

8.4 — Failed attack attempts that use the auxiliary points

The attacker has more information available than just two isogenous curves: They also get the action of Alice's and Bob's secret isogenies φ_A resp. φ_B on the $\ell_B^{n_B}$- resp. $\ell_A^{n_A}$-torsion. We focus on the problem of recovering the secret from a public key. Without loss of generality, suppose that $\ell_A^{n_A} < \ell_B^{n_B}$ and we are attacking Alice's public key $(E_A, \varphi_A(PB), \varphi_A(QB))$.

First, note that the extra information defines the secret isogeny uniquely: Consider two distinct d-isogenies $\phi,\psi\colon E \to E'$ with the same action on the m-torsion. Then $\ker(\phi-\psi) \supseteq E[m]$, hence $\deg(\phi-\psi) \geq \#\ker(\phi-\psi) \geq m^2$. On the other hand, Lemma V.1.2 of [Sil09] implies $\deg(\phi-\psi) \leq 4d$. Combining these bounds yields $m^2 \leq 4d$. In SIDH, this implies that an $\ell_A^{n_A}$-isogeny is uniquely defined by its action on the $\ell_B^{n_B}$-torsion unless the parameters are highly unbalanced. However, no efficient way to make use of this information is known.

8.4.1 – Interpolation problems. By definition, isogenies are rational maps, hence it is clear that given enough inputs and outputs, one can in principle recover the coefficients of that rational map [GG13, Section 5.8]. One can show [Ren18, Proposition 1] that in the SIDH setting, the isogeny φ_A can be written as

$$\varphi_A\colon (x,y) \longmapsto \big(f(x),\, c_0 y \cdot f'(x)\big)$$

for some rational map $f \in \mathbb{F}_{p^2}(x)$ of degree $\ell_A^{n_A}$ and a constant $c_0 \in \mathbb{F}_q$. Therefore, being given the action of φ_A, and thereby f, on "enough" points, one might hope to recover f and thus Alice's secret isogeny φ_A.

However, this is computationally infeasible: Even printing the *result* of the interpolation takes time linear in the degree, which in SIDH is exponentially large (in the bit length of the involved objects). One might wonder whether it is possible to *evaluate* the function while reconstructing it, thus circumventing the exponentially big output, but all known ways to do (polynomial or rational) interpolation still take time at least linear in the degree. The only conceivable way to succeed with this approach would be to reconstruct the rational map while *at the same time* rewriting it as a *composition* of rational maps, such that each of these maps has a degree polynomially small in ℓ_A. While there are of course methods to decompose polynomials and rational maps into a composition of smaller-degree maps, these algorithms require first storing the input in full.

Generally, the approach of rational-function interpolation seems similar in spirit to the interpolation idea in the next section, except that so far we have not made any use of the group structure underlying the rational maps in question. Since we have been working with less than

all the available structure, it seems reasonable to assume that this approach is fundamentally inferior to the ideas in the next sections.

8.4.2 – Group-theoretic approaches. Perhaps the most obvious idea to make use of the auxiliary points is to try to extrapolate the known action of φ_A on the $\ell_B^{n_B}$-torsion to a bigger torsion subgroup to subsequently recover (part of) the secret.

Unfortunately, it is evident that purely group-theoretic methods are doomed to fail: Let $\gcd(m, \ell_B^{n_B}) = 1$. By the structure theorem of finite abelian groups, the $\ell_B^{n_B}$- and m-torsion subgroups of an elliptic curve are *independent*; i.e., there are simply no nontrivial relations between points of ℓ_B-power order and points of order m in the curve group. (In other words, the $\ell_B^{n_B} m$-torsion subgroup is an internal direct product of the $\ell_B^{n_B}$- and the m-torsion.) Perhaps a reliable extrapolation is too much to ask for, but it seems that even obtaining *any* information about the action on the ℓ_A-torsion with success probability (non-negligibly) better than random guessing seems infeasible. In a sense, this is remarkable, since elliptic curves are also equipped with a *geometric* structure, and many purely group-theoretical morphisms defined on elliptic curve groups do not come from an isogenies, i.e., do not respect the geometric structure. However, nobody has yet discovered an efficient way to exploit this.

An effective Tate's theorem? Rather than extrapolating to a coprime torsion subgroup, one may instead attempt to lift the action of φ_A on the $\ell_B^{n_B}$-torsion to a *higher* ℓ_B-power torsion subgroup. In the limit, this lifting process would yield the action of φ_A on the ℓ_B-adic Tate modules $T_{\ell_B}(E_0)$.[13] Write $\ell = \ell_B$.

If one knew how to do the lifting step, this observation may inspire hope: It is known [Sil09, Theorem 7.7] that the natural map

$$\mathrm{Hom}_{\mathbb{F}_{p^2}}(E_0, E_A) \otimes_{\mathbb{Z}} \mathbb{Z}_\ell \longrightarrow \mathrm{Hom}_{\mathbb{F}_{p^2}}(T_\ell(E_0), T_\ell(E_A))$$

is an isomorphism of $\mathbb{Z}_\ell$-modules, hence the action of an isogeny defined over $\mathbb{F}_{p^2}$ on a sufficiently high ℓ^k-torsion completely determines the map. While this is a priori an abstract result, Petit [Pet17] found a way to turn this into an efficient algorithm assuming k grows big enough; see Section 8.4.3.

However, in any case, it seems that similar obstacles as in the previous section (extrapolating to another torsion subgroup) apply: Group-theoretically, the action on $E[\ell^k]$ can be lifted to an action on $E[\ell^{k+1}]$ in ℓ^4 different ways. Also taking into account the known information about the degree (coprime to ℓ), this expansion factor shrinks slightly,[14] but there still is no hope to learn anything about the action on the ℓ^∞-torsion without making use of the geometry of the underlying elliptic curve.

8.4.3 – Constructing endomorphisms to exploit the auxiliary points. *Acknowledgements.* The ideas in this section are all based on Petit's paper [Pet17], and in particular are the result of discussions with Dan Bernstein (who showed us the technique used below to estimate the expected size of solutions), Tanja Lange, and Christophe Petit (alphabetical order).

[13] The functor T_ℓ is defined as the inverse limit $T_\ell(E) = \varprojlim_n E[\ell^n]$ under the evident restriction maps $[\ell]\colon E[\ell^{n+1}] \twoheadrightarrow E[\ell^n]$; see for instance [Sil09, Section III.7].

Note that if $\ell \neq 0$ in the field of definition of the curve E, then $T_\ell(E) \cong \mathbb{Z}_\ell \times \mathbb{Z}_\ell$.

[14] The expansion factor is smaller, but still significant, for *endo*morphisms with known degree and trace: Forcing the characteristic polynomial limits the amount of choice. Concretely, there are ℓ^2 different ways to lift a known action on the ℓ^n-torsion to the ℓ^{n+1}-torsion while satisfying a given characteristic polynomial $\chi \bmod \ell^{n+1}$.

Recall that in two-party SIDH

$$\ell_A^{n_A} \approx \ell_B^{n_B} \approx \sqrt{p}\,,$$

corresponding to Alice's and Bob's secret isogenies having roughly the same degree. Petit [Pet17] shows how to construct an endomorphism on E_A if instead

$$\ell_B^{n_B} \gg \ell_A^{n_A}\,,$$

such that the capability to evaluate this endomorphism on the $\ell_B^{n_B}$-torsion — which is granted to the attacker in the SIDH setting by means of the auxiliary points (see Section 8.2.2) — allows one to reconstruct Alice's secret isogeny.

Petit's attack. Following the notation of Section 8.2.2, let π be the p-power Frobenius on E_0 and let ι be the order-4 automorphism $(x, y) \mapsto (-x, \sqrt{-1} \cdot y)$ on E_0. Then for any $a, b, c \in \mathbb{Z}$, we have an endomorphism $a\iota\pi + b\pi + c\iota \in \mathrm{End}(E_0)$, and using the (unknown) $\ell_A^{n_A}$-isogeny $\varphi_A \colon E_0 \to E_A$ we can, for every $d \in \mathbb{Z}$, define the endomorphism

$$\alpha = \varphi_A(a\iota\pi + b\pi + c\iota)\widehat{\varphi_A} + d \quad \in \mathrm{End}(E_A)$$

of degree (or equivalently, norm)[15]

$$\deg(\alpha) = \ell_A^{2n_A} p a^2 + \ell_A^{2n_A} p b^2 + \ell_A^{2n_A} c^2 + d^2.$$

Of course, since the attacker does not know φ_A, they cannot compute α directly. However, writing $N_1 = \ell_A^{n_A}$ and $N_2 = \ell_B^{n_B}$, Petit gives conditions under which one can efficiently find $a, b, c, d \in \mathbb{Z}$ such that

$$N_1^2 p a^2 + N_1^2 p b^2 + N_1^2 c^2 + d^2 = e N_2\,, \tag{8.2}$$

where e is a small cofactor controlling the remaining amount of brute-force work the attacker has to do. If $N_2 = \ell_B^{n_B}$ is big enough relative to $N_1 = \ell_A^{n_A}$, then $\ker(\alpha)$, and subsequently the secret $\ker(\varphi_A)$, can be recovered from the action of φ_A on the $\ell_B^{n_B}$-torsion in polynomial time.

Any hope for $\ell_A^{n_A} \approx \ell_B^{n_B} \approx \sqrt{p}$? We can heuristically estimate the expected size of solutions to (8.2) as follows. Suppose we want to count solutions with $e \leq M$ for some fixed bound M. Since all the terms in (8.2) are nonnegative, they cannot be bigger than the right-hand side, which is $\approx M\sqrt{p}$. Hence

$$a, b \lessapprox \sqrt{M} \cdot p^{-3/4}\,; \qquad c \lessapprox \sqrt{M} \cdot p^{-1/4}\,; \qquad d \lessapprox \sqrt{M} \cdot p^{1/4}\,.$$

This means the total number of possible assignments for the variables a, b, c, d, e is approximately

$$M^3 p^{-3/2}\,.$$

Assuming (wrongly, but for the sake of a rough estimate) that for each such assignment, the left- and right-hand side of (8.2) are uniformly random nonnegative integers upper bounded by $\approx M\sqrt{p}$, the expected number of solutions with $e \leq M$ is seen to be about

$$\frac{M^3 p^{-3/2}}{M\sqrt{p}} = M^2 p^{-2}\,,$$

implying that one needs to increase M to approximately p before a solution can be expected. This means that the smallest expected solution to (8.2) features the undesirable property $e \approx p$,

[15] A reader comparing this with the formula given in [Pet17, p. 15] may wonder where q has gone, but the norm of this specific endomorphism ι is $q = 1$.

which means that in this case, Petit's attack performs much worse than simply applying one of the known graph-walking attacks from Section 8.3 directly. We can therefore conclude that at least heuristically, it seems extremely unlikely that Petit's attack can possibly apply to the actual, balanced SIDH parameters.

Chapter 9

Quantum circuits for CSIDH

This chapter is for all practical purposes identical to the paper *Quantum circuits for the CSIDH: optimizing quantum evaluation of isogenies* [BLMP19] authored jointly with Daniel J. Bernstein, Tanja Lange, and Chloe Martindale, which was published at Eurocrypt 2019.

9.1 — Introduction

This chapter is devoted to the study of the cost of breaking CSIDH (Chapter 3) using a quantum computer. When only considering classical attacks, CSIDH (and its predecessor scheme due to Couveignes and Rostovtsev–Stolbunov [Cou06; RS06], which we refer to as *CRS*) has public keys and ciphertexts only about twice as large as traditional elliptic-curve keys and ciphertexts for a similar security level against all known pre-quantum attacks.

For comparison, the SIDH (and SIKE) isogeny-based cryptosystems [JD11; DJP14; Jao+17] are somewhat faster than CSIDH, but they do not support non-interactive key exchange, and their public keys and ciphertexts are 6 times larger[1] than in CSIDH. Furthermore, there are concerns that the extra information in SIDH keys might allow attacks; see [Pet17] and Chapter 7.

These SIDH disadvantages come from avoiding the commutative structure used in CRS and now in CSIDH. SIDH deliberately avoids this structure because the structure allows *quantum* attacks that asymptotically take subexponential time; see below. The CRS/CSIDH key size thus grows superlinearly in the post-quantum security level. For comparison, if the known attacks are optimal, then the SIDH key size grows linearly in the post-quantum security level.

However, even in a post-quantum world, it is not at all clear how much weight to put on these asymptotics. It is not clear, for example, how large the keys will have to be before the subexponential attacks begin to outperform the exponential-time non-quantum attacks or an exponential-time Grover search. It is not clear when the superlinear growth in CSIDH key sizes will outweigh the factor 6 mentioned above. For applications that need non-interactive key exchange in a post-quantum world, the SIDH/SIKE family is not an option, and it is important to understand what influence these attacks have upon CSIDH key sizes. The asymptotic performance of these attacks is stated in Chapter 3, but it is challenging to understand the concrete performance of these attacks for specific CSIDH parameters.

9.1.1 – Contributions. The most important bottleneck in the quantum attacks mentioned above is the cost of evaluating the class-group action, a series of isogenies, in superposition.

[1] When the goal is for pre-quantum attacks to take 2^λ operations (without regard to memory consumption), CRS, CSIDH, SIDH, and SIKE all choose primes $p \approx 2^{4\lambda}$. The CRS and CSIDH keys and ciphertexts use (approximately) $\log_2 p \approx 4\lambda$ bits, whereas the SIDH and SIKE keys and ciphertexts use $6 \log_2 p \approx 24\lambda$ bits for 3 elements of $\mathbb{F}_{p^2}$. There are compressed variants of SIDH that reduce $6 \log_2 p$ to $4 \log_2 p \approx 16\lambda$ (see [Aza+16]) and to $3.5 \log_2 p \approx 14\lambda$ (see [Cos+17] and [Zan+18]), at some cost in runtime.

Each quantum attack incurs this cost many times; see below. The goal of this chapter is to analyze and optimize this cost. We focus on CSIDH because CSIDH is much faster than CRS.

Our main result has the following shape: the CSIDH group action can be carried out in B nonlinear bit operations (counting ANDs and ORs, allowing free XORs and NOTs) with failure probability at most ϵ. (All of our algorithms know when they have failed.) This implies a reversible computation of the CSIDH group action with failure probability at most ϵ using at most $2B$ Toffoli gates (allowing free NOTs and CNOTs). This in turn implies a quantum computation of the CSIDH group action with failure probability at most ϵ using at most $14B$ T-gates (allowing free Clifford gates). Section 9.11 reviews these cost metrics and their relationships.

We explain how to compute pairs (B, ϵ) for any given CSIDH parameters. For example, we show how to compute CSIDH-512 for uniform random exponent vectors in $\{-5, \ldots, 5\}^{74}$ using

- $1118827416420 \approx 2^{40}$ nonlinear bit operations using the algorithm of Section 9.7, or
- $765325228976 \approx 0.7 \cdot 2^{40}$ nonlinear bit operations using the algorithm of Section 9.8,

in both cases with failure probability below 2^{-32}. CSIDH-512 is the smallest parameter set considered in Chapter 3. For comparison, computing the same action with failure probability 2^{-32} using the Jao–LeGrow–Leonardi–Ruiz-Lopez algorithm [JLLR18], with the underlying modular multiplications computed by the same method as in Roetteler–Naehrig–Svore–Lauter [RNSL17], would use approximately 2^{51} nonlinear bit operations.

We exploit a variety of algorithmic ideas, including several new ideas pushing beyond the previous state of the art in isogeny computation, with the goal of obtaining the best pairs (B, ϵ). We introduce a new constant-time variable-degree isogeny algorithm, a new application of the Elligator map, new ways to handle failures in isogeny computations, new combinations of the components of these computations, new speeds for integer multiplication, and more.

9.1.2 – Impact upon quantum attacks. Kuperberg [Kup05] introduced an algorithm using $\exp\big((\log N)^{1/2+o(1)}\big)$ queries to the oracle and $\exp\big((\log N)^{1/2+o(1)}\big)$ quantum operations on $\exp((\log N)^{1/2+o(1)})$ qubits to solve the order-N dihedral hidden-subgroup problem. A variant of the algorithm due to Regev [Reg04] uses only a polynomial number of qubits, although with a worse $o(1)$ for the number of queries and operations. A followup paper by Kuperberg [Kup13] introduced further algorithmic options and tradeoffs.

Childs, Jao, and Soukharev [CJS14] pointed out that these algorithms could be used to attack CRS. They analyzed the asymptotic cost of a variant of Regev's algorithm in this context. This cost is dominated by queries, in part because the number of queries is large but also because the cost of each query is large. Each query evaluates the CRS group action using a superposition of group elements.

We emphasize that computing the exact attack costs for any particular set of CRS or CSIDH parameters is complicated and requires a lot of new work. The main questions are (1) the exact number of queries for various dihedral-hidden-subgroup algorithms, not just asymptotics; and (2) the exact cost of each query, again not just asymptotics.

The first question is outside the scope of this chapter. Some of the simpler algorithms were simulated for small sizes in [Kup05], [BN18], and [BS18], but note that Kuperberg commented in [Kup05, page 5] that his "experiments with this simulator led to a false conjecture for [the] algorithm's precise query complexity".

This chapter addresses the second question for CSIDH: the concrete cost of quantum algorithms for evaluating the action of the class group, which means computing isogenies of elliptic curves in superposition.

9.1.3 – Comparison to previous claims regarding query cost. Bonnetain and Schrottenloher claim in [BS18, online versions 4, 5, and 6] that CSIDH-512 can be broken in "only" 2^{71} quantum gates, where each query uses 2^{37} quantum gates ("Clifford+T" gates; see Section 9.11.4).

We work in the same simplified model of counting operations, allowing any number of qubits to be stored for free. We further simplify by counting only T-gates. We gain considerable performance from optimizations not considered in [BS18]. We take the best possible distribution of input vectors, disregarding the 2^2 overhead estimated in [BS18]. Our final gate counts for each query are nevertheless much higher than the 2^{37} claimed in [BS18]. Even assuming that [BS18] is correct regarding the number of queries, the cost of each query pushes the total attack cost above 2^{80}.

The query-cost calculation in [BS18] is not given in enough detail for full reproducibility. However, some details are provided, and given these details we conclude that costly parts of the computation are overlooked in [BS18] in at least three ways. First, to estimate the number of quantum gates for multiplication in $\mathbb{F}_p$, [BS18] uses a count of nonlinear bit operations for multiplication in $\mathbb{F}_2[x]$, not noticing that all known methods for multiplication in $\mathbb{Z}$ (never mind reduction modulo p) involve many more nonlinear bit operations than multiplication in $\mathbb{F}_2[x]$. Second, at a higher level, the strategy for computing an ℓ-isogeny requires first finding a point of order ℓ, an important cost not noticed in [BS18]. Third, [BS18] counts the number of operations in a *branching* algorithm, not noticing the challenge of building a *non-branching* (constant-time) algorithm for the same task, as required for computations in superposition. Our analysis addresses all of these issues and more.

9.1.4 – Memory consumption. We emphasize that our primary goal is to minimize the number of bit operations. This cost metric pays no attention to the fact that the resulting quantum algorithm for, e.g., CSIDH-512 uses a quantum computer with 2^{40} qubits.

Most of the literature on quantum algorithms pays much more attention to the number of qubits. This is why [CJS14], for example, uses a Regev-type algorithm instead of Kuperberg's algorithm. Similarly, [Cas+18] takes Regev's algorithm "as a baseline" given "the larger memory requirement" for Kuperberg's algorithm.

An obvious reason to keep the number of qubits under control is the difficulty of scaling quantum computers up to a huge number of qubits. Post-quantum cryptography starts from the assumption that there will be enough scalability to build a quantum computer using thousands of logical qubits to run Shor's algorithm, but this does not imply that a quantum computer with millions of logical qubits will be only 1000 times as expensive, given limits on physical chip size and costs of splitting quantum computation across multiple chips.

On the other hand, [BS18] chooses Kuperberg's algorithm, and claims that the number of qubits used in Kuperberg's algorithm is not a problem:

> The algorithm we consider has a subexponential memory cost. More precisely, it needs exactly one qubit per query, plus the fixed overhead of the oracle, which can be neglected.

Concretely, for CSIDH-512, [BS18, online versions 1, 2, 3] claim $2^{29.5}$ qubits, and [BS18, online versions 4, 5, 6] claim 2^{31} qubits. However, no justification is provided for the claim that the number of qubits for the oracle "can be neglected". There is no analysis in [BS18] of the number of qubits used for the oracle.

We are not saying that our techniques *need* 2^{40} qubits. On the contrary: later we mention various ways in which the number of qubits can be reduced with only moderate costs in the

number of operations. However, one cannot trivially extrapolate from the memory consumption of CSIDH software (a few kilobytes) to the number of qubits used in a quantum computation. The requirement of reversibility makes it more challenging and more expensive to reduce space, since intermediate results cannot simply be erased. See Section 9.11.3.

Furthermore, even if enough qubits are available, simply counting qubit operations ignores critical bottlenecks in quantum computation. Fault-tolerant quantum computation corrects errors in every qubit at every time step, even if the qubit is merely being stored; see Section 9.11.5. Communicating across many qubits imposes further costs; see Section 9.11.6. It is thus safe to predict that the actual cost of a quantum CSIDH query will be much larger than indicated by our operation counts. Presumably the gap will be larger than the gap for, e.g., the AES attack in [GLRS16], which has far fewer idle qubits and much less communication overhead.

Acknowledgements. Thanks to Bo-Yin Yang for suggesting factoring the average over vectors of the generating function in Section 9.7.3. Thanks to Joost Renes for his comments.

9.2 — Overview of the computation

We recall the definition of the CSIDH group action, focusing on the computational aspects of the concrete construction rather than discussing the general case of the underlying algebraic theory.

Parameters. The only parameter in CSIDH is a prime number p of the form $p = 4 \cdot \ell_1 \cdots \ell_n - 1$, where $\ell_1 < \cdots < \ell_n$ are (small) odd primes and $n \geq 1$. Note that $p \equiv 3 \pmod 8$ and $p > 3$.

Notation. For each $A \in \mathbb{F}_p$ with $A^2 \neq 4$, define E_A as the Montgomery curve $y^2 = x^3 + Ax^2 + x$ over $\mathbb{F}_p$. This curve E_A is supersingular, meaning that $\#E_A(\mathbb{F}_p) \equiv 1 \pmod p$, if and only if it has trace zero, meaning that $\#E_A(\mathbb{F}_p) = p + 1$. Here $E_A(\mathbb{F}_p)$ means the group of points of E_A with coordinates in $\mathbb{F}_p$, including the neutral element at ∞; and $\#E_A(\mathbb{F}_p)$ means the number of points.

Define S_p as the set of A such that E_A is supersingular. For each $A \in S_p$ and each $i \in \{1, \ldots, n\}$, there is a unique $B \in S_p$ such that there is an ℓ_i-isogeny from E_A to E_B whose kernel is $E_A(\mathbb{F}_p)[\ell_i]$, the set of points $Q \in E_A(\mathbb{F}_p)$ with $\ell_i Q = 0$. Define $\mathcal{L}_i(A) = B$. One can show that $\mathcal{L}_i$ is invertible: specifically, $\mathcal{L}_i^{-1}(A) = -\mathcal{L}_i(-A)$. Hence $\mathcal{L}_i^e$ is defined for each integer e.

Inputs and output. Given an element $A \in S_p$ and a list $(e_1, \ldots, e_n)$ of integers, the CSIDH group action computes $\mathcal{L}_1^{e_1}(\mathcal{L}_2^{e_2}(\cdots(\mathcal{L}_n^{e_n}(A))\cdots)) \in S_p$.

9.2.1 – Distribution of exponents. The performance of our algorithms depends on the distribution of the exponent vectors $(e_1, \ldots, e_n)$, which in turn depends on the context.

Constructively, [Cas+18] proposes to sample each e_i independently and uniformly from a small range $\{-C, \ldots, C\}$. For example, CSIDH-512 in [Cas+18] has $n = 74$ and uses the range $\{-5, \ldots, 5\}$, so there are $11^{74} \approx 2^{256}$ equally likely exponent vectors. We emphasize, however, that all known attacks actually use considerably larger exponent vectors. This means that the distribution of exponents $(e_1, \ldots, e_n)$ our quantum oracle has to process is *not* the same as the distribution used constructively.

The first step in the algorithms of Kuperberg and Regev, applied to a finite abelian group G, is to generate a uniform superposition over all elements of G. The CRS and CSIDH schemes define a map from vectors $(e_1, \ldots, e_n) \in \mathbb{Z}^n$ to elements $\mathfrak{l}_1^{e_1} \cdots \mathfrak{l}_n^{e_n}$ of the ideal-class group G. This map has a high chance of being surjective but it is far from injective: its kernel is a lattice of rank n. Presumably taking, e.g., 17^{74} length-74 vectors with entries in the range $\{-8, \ldots, 8\}$ produces a

close-to-uniform distribution of elements of the CSIDH-512 class group, but the literature does not indicate how Kuperberg's algorithm behaves when each group element is represented as many different strings.

In his original paper on CRS, Couveignes [Cou06] suggested instead generating a unique vector representing each group element as follows. Compute a basis for the lattice mentioned above; on a quantum computer this can be done using Shor's algorithm [Sho97a] which runs in polynomial time, and on a conventional computer this can be done using Hafner and McCurley's algorithm [HM89] which runs in subexponential time. This basis reveals the group size $\#G$ and an easy-to-sample set R of representatives for G, such as $\{(e_1, 0, \ldots, 0) : 0 \leq e_1 < \#G\}$ in the special case that $\mathfrak{l}_1$ generates G; for the general case see, e.g., [Mic01, Section 4.1]. Reduce each representative to a short representative, using an algorithm that finds a close lattice vector. If this algorithm is deterministic (for example, if all randomness used in the algorithm is replaced by pseudorandomness generated from the input) then applying it to a uniform superposition over R produces a uniform superposition over a set of short vectors uniquely representing G.

The same idea was mentioned in the Childs–Jao–Soukharev paper [CJS14] on quantum attacks against CRS, and in the description of quantum attacks in Chapter 3. However, close-vector problems are not easy, even in dimensions as small as 74. Bonnetain and Schrottenloher [BS18] estimate that CSIDH-512 exponent vectors can be found whose 1-norm is 4 times larger than vectors used constructively. They rely on a very large precomputation, and they do not justify their assumption that the 1-norm, rather than the ∞-norm, measures the cost of a class-group action in superposition. Jao, LeGrow, Leonardi, and Ruiz-Lopez [JLLR18] present an algorithm that guarantees $(\log p)^{O(1)}$ bits in each exponent, i.e., in the ∞-norm, but this also requires a subexponential-time precomputation, and the exponents appear to be rather large.

Perhaps future research will improve the picture of how much precomputation time and per-vector computation time is required for algorithms that find vectors of a specified size; or, alternatively, will show that Kuperberg-type algorithms can handle non-unique representatives of group elements. The best conceivable case for the attacker is the distribution used in CSIDH itself, and we choose this distribution as an illustration in analyzing the concrete cost of our algorithms.

9.2.2 – Verification of costs. To ensure that we are correctly computing the number of bit operations in our group-action algorithms, we have built a bit-operation simulator, and implemented our algorithms inside the simulator. The simulator is available from `https://quantum.isogeny.org/software.html`.

The simulator has a very small core that implements — and counts the number of — NOT, XOR, AND, and OR operations. Higher-level algorithms, all the way from basic integer arithmetic up through isogeny computation, are built on top of this core.

The core also encapsulates the values of bits so that higher-level algorithms cannot inspect those values by accident. There is an explicit mechanism to break the encapsulation so that output values can be checked against separate computations in the Sage computer-algebra system.

9.2.3 – Verification of failure probabilities. Internally, each of our algorithms computes the group action by moving the exponent vector $(e_1, \ldots, e_n)$ step by step towards 0. The algorithm fails if the vector does not reach 0 within the specified number of iterations. Analyzing the failure probability requires analyzing how the distribution of exponent vectors interacts with the distribution of curve points produced inside the algorithm; each e_i step relies on finding a point of order ℓ_i.

We mathematically calculate the failure probability in a model where each generated curve point has probability $1-1/\ell_i$ of having order divisible by ℓ_i, and where these probabilities are all independent. The model would be exactly correct if each point were generated independently and uniformly at random. We actually generate points differently, so there is a risk of our failure-probability calculations being corrupted by inaccuracies in the model. To address this risk, we have carried out various point-generation experiments, suggesting that the model is reasonably accurate. Even if the model is inaccurate, one can compensate with a minor increase in costs. See Sections 9.4.3 and 9.5.2.

There is a more serious risk of errors in the failure-probability calculations that we carry out within the model. To reduce this risk, we have carried out 10^7 simple trials of the following type for each algorithm: generate a random exponent vector, move it step by step towards 0 the same way the algorithm does (in the model), and see how many iterations are required. The observed distribution of the number of iterations is consistent with the distribution that we calculate mathematically. Of course, if there is a calculation error that somehow affects only very small probabilities, then this error will not be caught by only 10^7 experiments.

9.2.4 – Structure of the computation. We present our algorithms from bottom up, starting with scalar multiplication in Section 9.3, generation of curve points in Section 9.4, computation of $\mathcal{L}_i$ in Section 9.5, and computation of the entire CSIDH group action in Sections 9.6, 9.7, and 9.8. Lower-level subroutines for basic integer and modular arithmetic appear in Appendices 9.12 and 9.13 respectively.

Various sections and subsections mention ideas for saving time beyond what we have implemented in our bit-operation simulator. These ideas include low-level speedups such as avoiding exponentiations in inversions and Legendre-symbol computations (see Section 9.13.4), and higher-level speedups such as using division polynomials (Section 9.9) and/or modular polynomials (Section 9.10) to eliminate failures in the computation of $\mathcal{L}_i$ for small primes. All of the specific bit-operation counts that we state, such as the $1118827416420 \approx 2^{40}$ nonlinear bit operations mentioned above, have been fully implemented.

9.3 — Scalar multiplication on an elliptic curve

This section analyzes the costs of scalar multiplication on the curves used in CSIDH: supersingular Montgomery curves $E_A : y^2 = x^3 + Ax^2 + x$ over $\mathbb{F}_p$.

For CSIDH-512, our simulator shows (after our detailed optimizations; see Appendices 9.12 and 9.13) that a squaring $\mathbf{S}$ in $\mathbb{F}_p$ can be computed in 349596 nonlinear bit operations, and that a general multiplication $\mathbf{M}$ in $\mathbb{F}_p$ can be computed in 447902 nonlinear bit operations, while addition in $\mathbb{F}_p$ takes only 2044 nonlinear bit operations. We thus emphasize the number of $\mathbf{S}$ and $\mathbf{M}$ in scalar multiplication (and in higher-level operations), although in our simulator we have also taken various opportunities to eliminate unnecessary additions and subtractions.

9.3.1 – How curves are represented. We consider two options for representing E_A. The **affine** option uses $A \in \mathbb{F}_p$ to represent E_A. The **projective** option uses $A_0, A_1 \in \mathbb{F}_p$, with $A_0 \neq 0$, to represent E_A where $A = A_1/A_0$.

The formulas to produce a curve in Section 9.5 naturally produce (A_1, A_0) in projective form. Dividing A_1 by A_0 to produce A in affine form costs an inversion and a multiplication. Staying in projective form is an example of what Section 9.13.5 calls "eliminating inversions", but this requires some extra computation when A is used, as we explain below.

The definition of the class-group action requires producing the output A in affine form at

the end of the computation. It could also be beneficial to convert each intermediate A to affine form, depending on the relative costs of the inversion and the extra computation.

9.3.2 – How points are represented. As in [Mil85, page 425, last paragraph] and [Mon87, page 261], we avoid computing the y-coordinate of a point (x, y) on E_A. This creates some ambiguity, since the points (x, y) and $(x, -y)$ are both represented as $x \in \mathbb{F}_p$, but the ambiguity does not interfere with scalar multiplication.

We again distinguish between affine and projective representations. As in [Ber06], we represent both $(0, 0)$ and the neutral element on E_A as $x = 0$, and (except where otherwise noted) we allow $X/0$, including $0/0$, as a projective representation of $x = 0$. The projective representation thus uses $X, Z \in \mathbb{F}_p$ to represent $x = X/Z$ if $Z \neq 0$, or $x = 0$ if $Z = 0$. These definitions eliminate branches from the scalar-multiplication techniques that we use.

9.3.3 – Computing nP. We use the Montgomery ladder to compute nP, given a b-bit exponent n and a curve point P. The Montgomery ladder consists of b "ladder steps" operating on variables (X_2, Z_2, X_3, Z_3) initialized to $(1, 0, x_1, 1)$, where x_1 is the x-coordinate of P. Each ladder step works as follows:

- Conditionally swap (X_2, Z_2) with (X_3, Z_3), where the condition bit in iteration i is bit n_{b-1-i} of n. This means computing $X_2 \oplus X_3$, ANDing each bit with the condition bit, and XORing the result into both X_2 and X_3; and similarly for Z_2 and Z_3.
- Compute $Y = X_2 - Z_2$, Y^2, $T = X_2 + Z_2$, T^2, $X_4 = T^2Y^2$, $E = T^2 - Y^2$, and $Z_4 = E(Y^2 + ((A + 2)/4)E)$. This is a **point doubling**: it uses $2\mathbf{S} + 3\mathbf{M}$ and a few additions (counting subtractions as additions). We divide $A + 2$ by 4 modulo p before the scalar multiplication, using two conditional additions of p and two shifts.
- Compute $C = X_3 + Z_3$, $D = X_3 - Z_3$, DT, CY, $X_5 = (DT + CY)^2$, and $Z_5 = x_1(DT - CY)^2$. This is a **differential addition**: it also uses $2\mathbf{S} + 3\mathbf{M}$ and a few additions.
- Set $(X_2, Z_2, X_3, Z_3) \leftarrow (X_4, Z_4, X_5, Z_5)$.
- Conditionally swap (X_2, Z_2) with (X_3, Z_3), where the condition bit is again n_{b-1-i}. We merge this conditional swap with the conditional swap at the beginning of the next iteration by using $n_{b-i-i} \oplus n_{b-i-2}$ as condition bit.

Then nP has projective representation (X_2, Z_2) by [BL17, Theorem 4.5]. The overall cost is $4b\mathbf{S} + 6b\mathbf{M}$ plus a small overhead for additions and conditional swaps.

Representing the input point projectively as X_1/Z_1 means computing $X_5 = Z_1(DT+CY)^2$ and $Z_5 = X_1(DT - CY)^2$, and starting from $(1, 0, X_1, Z_1)$. This costs $b\mathbf{M}$ extra. Beware that [BL17, Theorem 4.5] requires $Z_1 \neq 0$.

Similarly, representing A projectively as A_1/A_0 means computing $X_4 = T^2(4A_0Y^2)$ and $Z_4 = E(4A_0Y^2 + (A_1 + 2A_0)E)$, after multiplying Y^2 by $4A_0$. This also costs $b\mathbf{M}$ extra.

Other techniques. The initial $Z_2 = 0$ and $Z_3 = 1$ (for an affine input point) are small, and remain small after the first conditional swap, saving time in the next additions and subtractions. Our framework for tracking sizes of integers recognizes this automatically. The framework does not, however, recognize that half of the output of the last conditional swap is unused. We could use dead-value elimination and other standard peephole optimizations to save bit operations.

Montgomery [Mon87, page 260] considered computing many scalar multiplications at once, using affine coordinates (e.g., $x_2 = X_2/Z_2$), for intermediate points inside each scalar multiplication and batching inversions across the scalar multiplications. This could be slightly less

expensive than the Montgomery ladder for large b, depending on the $\mathbf{S}/\mathbf{M}$ ratio. Our computation of a CSIDH group action involves many scalar multiplications, but not in large enough batches to justify considering affine coordinates for intermediate points. Computing the group action for a batch of inputs might change the picture, but for simplicity we focus on the problem of computing the group action for one input.

A more recent possibility is scalar multiplication on a birationally equivalent Edwards curve. For large b, sliding-window Edwards scalar multiplication is somewhat less expensive than the Montgomery ladder; see generally [BLo8] and [Hiş10]. On the other hand, for constant-time computations it is important to use fixed windows rather than sliding windows. Despite this difficulty, we estimate that small speedups are possible for $b = 512$.

9.3.4 – Computing $P, 2P, 3P, \ldots, kP$. An important subroutine in isogeny computation (see Section 9.5) is to compute the sequence $P, 2P, 3P, \ldots, kP$ for a constant $k \geq 1$.

We compute $2P$ by a doubling, $3P$ by a differential addition, $4P$ by a doubling, $5P$ by a differential addition, $6P$ by a doubling, etc. In other words, each multiple of P is computed by the Montgomery ladder as above, but these computations are merged across the multiples (and conditional swaps are eliminated). This takes $2(k-1)\mathbf{S} + 3(k-1)\mathbf{M}$ for affine P and affine A. Projective P adds $\lfloor (k-1)/2 \rfloor \mathbf{M}$, and projective A adds $\lfloor k/2 \rfloor \mathbf{M}$.

We could instead compute $2P$ by a doubling, $3P$ by a differential addition, $4P$ by a differential addition, $5P$ by a differential addition, $6P$ by a differential addition, etc. This again takes $2(k-1)\mathbf{S} + 3(k-1)\mathbf{M}$ for affine P and affine A, but projective P and projective A now have different effects: projective P adds $(k-2)\mathbf{M}$ if $k \geq 2$, and projective A adds $\mathbf{M}$ if $k \geq 2$. The choice here also has an impact on metrics beyond bit operations: doublings increase space requirements but allow more parallelism.

9.4 — Generating points on an elliptic curve

This section analyzes the cost of several methods to generate a random point on a supersingular Montgomery curve $E_A : y^2 = x^3 + Ax^2 + x$, given $A \in \mathbb{F}_p$. As in Section 9.2, p is a standard prime congruent to 3 modulo 8.

Sometimes one instead wants to generate a point on the twist of the curve. The **twist** is the curve $-y^2 = x^3 + Ax^2 + x$ over $\mathbb{F}_p$; note that -1 is a non-square in $\mathbb{F}_p$. This curve is isomorphic to E_{-A} by the map $(x, y) \mapsto (-x, y)$. Beware that there are several slightly different concepts of "twist" in the literature; the definition here is the most useful definition for CSIDH, as explained in [Cas+18].

9.4.1 – Random point on curve or twist. The conventional approach is as follows: generate a uniform random $x \in \mathbb{F}_p$; compute $x^3 + Ax^2 + x$; compute $y = (x^3 + Ax^2 + x)^{(p+1)/4}$; and check that $y^2 = x^3 + Ax^2 + x$.

One always has $y^4 = (x^3 + Ax^2 + x)^{p+1} = (x^3 + Ax^2 + x)^2$ so $\pm y^2 = x^3 + Ax^2 + x$. About half the time, y^2 will match $x^3 + Ax^2 + x$; i.e., (x, y) will be a point on the curve. Otherwise (x, y) will be a point on the twist.

Since we work purely with x-coordinates (see Section 9.3.2), we skip the computation of y. However, we still need to know whether we have a curve point or a twist point, so we compute the Legendre symbol of $x^3 + Ax^2 + x$ as explained in Section 9.13.4.

The easiest distribution of outputs to mathematically analyze is the uniform distribution over the following $p + 1$ pairs:

- $(x, +1)$ where x represents a curve point;

- $(x, -1)$ where x represents a twist point.

One can sample from this distribution as follows: generate a uniform random $u \in \mathbb{F}_p \cup \{\infty\}$; set x to u if $u \in \mathbb{F}_p$ or to 0 if $u = \infty$; compute the Legendre symbol of $x^3 + Ax^2 + x$; and replace symbol 0 with $+1$ if $u = 0$ or -1 if $u = \infty$.

For computations, it is slightly simpler to drop the two pairs with $x = 0$: generate a uniform random $x \in \mathbb{F}_p^*$ and compute the Legendre symbol of the value $x^3 + Ax^2 + x$. This generates a uniform distribution over the remaining $p - 1$ pairs.

9.4.2 – Random point on curve. What if twist points are useless and the goal is to produce a point specifically on the curve (or vice versa)? One approach is to generate, e.g., 100 random curve-or-twist points as in Section 9.4.1, and select the first point on the curve. This fails with probability $1/2^{100}$. If a computation involves generating 2^{10} points in this way then the overall failure probability is $1 - (1 - 1/2^{100})^{2^{10}} \approx 1/2^{90}$. One can tune the number of generated points according to the required failure probability.

We save time by applying "Elligator" [BHKL13], specifically the Elligator 2 map. Elligator 2 is defined for all the curves E_A that we use, *except* the curve E_0, which we discuss below. For each of these curves E_A, Elligator 2 is a fast injective map from $\{2, 3, \ldots, (p-1)/2\}$ to the set $E_A(\mathbb{F}_p)$ of curve points. This produces only about half of the curve points; see Section 9.5.2 for analysis of the impact of this nonuniformity upon our higher-level algorithms.

Here are the details of Elligator 2, specialized to these curves, further simplified to avoid computing y, and adapted to allow twists as an option:

- Input $A \in \mathbb{F}_p$ with $A^2 \neq 4$ and $A \neq 0$.
- Input $s \in \{1, -1\}$. This procedure generates a point on E_A if $s = 1$, or on the twist of E_A if $s = -1$.
- Input $u \in \{2, 3, \ldots, (p-1)/2\}$.
- Compute $v = A/(u^2 - 1)$.
- Compute e, the Legendre symbol of $v^3 + Av^2 + v$.
- Compute x as v if $e = s$, otherwise $-v - A$.

To see that this works, note first that v is defined since $u^2 \neq 1$, and is nonzero since $A \neq 0$. One can also show that $A^2 - 4$ is nonsquare for all of the CSIDH curves, so $v^3 + Av^2 + v \neq 0$, so e is 1 or -1. If $e = s$ then $x = v$ so $x^3 + Ax^2 + x$ is a square for $s = 1$ and a nonsquare for $s = -1$. Otherwise $e = -s$ and $x = -v - A$ so $x^3 + Ax^2 + x = -u^2(v^3 + Av^2 + v)$, which is a square for $s = 1$ and a nonsquare for $s = -1$. This uses that v and $-v - A$ satisfy $(-v - A)^2 + A(-v - A) + 1 = v^2 + Av + 1$ and $-v - A = -u^2 v$.

The $(p-3)/2$ different choices of u produce $(p-3)/2$ different curve points, but we could produce any particular x output twice since we suppress y.

The case $A = 0$. One way to extend Elligator 2 to the curve E_0 is to set $v = u$ when $A = 0$ instead of $v = A/(u^2 - 1)$. The point of the construction of v is that $x^3 + Ax^2 + x$ for $x = -v - A$ is a non-square times $v^3 + Av^2 + v$, i.e., that $(-v - A)/v$ is a non-square; this is automatic for $A = 0$, since -1 is a non-square.

We actually handle E_0 in a different way: we precompute a particular base point on E_0 whose order is divisible by $(p+1)/4$, and we always return this point if $A = 0$. This makes our higher-level algorithms slightly more effective (but we disregard this improvement in analyzing the success probability of our algorithms), since this point guarantees a successful isogeny com-

putation starting from E_0; see Section 9.5. The same guarantee removes any need to generate other points on E_0, and is also useful to start walks in Section 9.10.

9.4.3 – Derandomization. Rather than generating random points, we generate a deterministic sequence of points by taking $u = 2$ for the first point, $u = 3$ for the next point, etc. We precompute the inverses of $1 - 2^2$, $1 - 3^2$, etc., saving bit operations.

An alternative, saving the same number of bit operations, is to precompute inverses of $1 - u^2$ for various random choices of u, embedding the inverses into the algorithm. This guarantees that the failure probability of the outer algorithm for any particular input A, as the choices of u vary, is the same as the failure probability of an algorithm that randomly chooses u upon demand for each A.

We are heuristically assuming that failures are not noticeably correlated across choices of A. To replace this heuristic with a proof, one can generate the u sequence randomly for each input A. This randomness, in turn, may be replaced by the output of a stream cipher. The stream-cipher inputs are (1) A as a nonce, and (2) a randomly chosen key used for all A. This output is indistinguishable from true randomness if the cipher is secure. In this setting one cannot precompute the reciprocals of $1 - u^2$, but one can still batch the inversions.

9.5 — Computing an ℓ-isogenous curve

This section analyzes the cost of computing a single isogeny in CSIDH. There are two inputs: A, specifying a supersingular Montgomery curve E_A over $\mathbb{F}_p$; and i, specifying one of the odd prime factors ℓ_i of $(p+1)/4 = \ell_1 \cdots \ell_n$. The output is $B = \mathcal{L}_i(A)$. We abbreviate ℓ_i as ℓ and $\mathcal{L}_i$ as $\mathcal{L}$.

Recall that B is characterized by the following property: there is an ℓ-isogeny from E_A to E_B whose kernel is $E_A(\mathbb{F}_p)[\ell]$. Beyond analyzing the costs of computing $B = \mathcal{L}(A)$, we analyze the costs of applying the ℓ-isogeny to a point on E_A, obtaining a point on E_B. See Section 9.5.4.

The basic reason that CSIDH is much faster than CRS is that the CSIDH construction allows (variants of) Vélu's formulas [Vél71; CH17; Ren18] to use points in $E_A(\mathbb{F}_p)$, rather than points defined over larger extension fields. This section focuses on computing B via these formulas. The cost of these formulas is approximately linear in ℓ, assuming that a point of order ℓ is known. There are two important caveats here:

- Finding a point of order ℓ is not easy to do efficiently in constant time; see Section 9.5.1. We follow the obvious approach, namely taking an appropriate multiple of a random point; but this is expensive — recall from Section 9.3 that a 500-bit Montgomery ladder costs $2000\mathbf{S} + 3000\mathbf{M}$ when both A and the input point are affine — and has failure probability approximately $1/\ell$.
- In some of our higher-level algorithms, i is a *variable*. Then $\ell = \ell_i$ is also a variable, and Vélu's formulas are variable-time formulas, while we need constant-time computations. Generic branch elimination produces a constant-time computation taking time approximately linear in $\ell_1 + \ell_2 + \cdots + \ell_n$, which is quite slow. However, we show how to do much better, reducing $\ell_1 + \ell_2 + \cdots + \ell_n$ to $\max\{\ell_1, \ell_2, \ldots, \ell_n\}$, by exploiting the internal structure of Vélu's formulas. See Section 9.5.3.

There are other ways to compute isogenies, as explored in [Kie17; DKS18]:

- The "Kohel" strategy: Compute a univariate polynomial whose roots are the x-coordinates of the points in $E_A(\mathbb{F}_p)[\ell]$. Use Kohel's formulas [Koh96, Section 2.4] to compute an isogeny corresponding to this polynomial. This strategy is (for CSIDH) asymptotically slower

than Vélu's formulas, but could nevertheless be faster when ℓ is very small. Furthermore, this strategy is deterministic and always works.

- The "modular" strategy: Compute the possible j-invariants of E_B by factoring modular polynomials. Determine the correct choice of B by computing the corresponding isogeny kernels or, on subsequent steps, simply by not walking back.

We analyze the Kohel and modular strategies in Sections 9.9 and 9.10.

9.5.1 – Finding a point of order ℓ. We now focus on the problem of finding a point of order ℓ in $E_A(\mathbb{F}_p)$. By assumption $(p+1)/4$ is a product of distinct odd primes $\ell_1, \ldots, \ell_n$; $\ell = \ell_i$ is one of those primes; and $\#E_A(\mathbb{F}_p) = p+1$. One can show that $E_A(\mathbb{F}_p)$ has a point of order 4 and is thus cyclic:

$$E_A(\mathbb{F}_p) \cong \mathbb{Z}/(p+1) \cong \mathbb{Z}/4 \times \mathbb{Z}/\ell_1 \times \cdots \times \mathbb{Z}/\ell_n\,.$$

We can *try* to find a point Q of order ℓ in $E_A(\mathbb{F}_p)$ as follows:

- Pick a random point $P \in E_A(\mathbb{F}_p)$, as explained in Section 9.4.
- Compute a "cofactor" $(p+1)/\ell$. To handle the case $\ell = \ell_i$ for variable i, we first use bit operations to compute the list $\ell'_1, \ldots, \ell'_n$, where $\ell'_j = \ell_j$ for $j \neq i$ and $\ell'_i = 1$; we then use a product tree to compute $\ell'_1 \cdots \ell'_n$. (Computing $(p+1)/\ell$ by a general division algorithm could be faster, but the product tree is simpler and has negligible cost in context.)
- Compute $Q = ((p+1)/\ell)P$ as explained in Section 9.3.

If P is a uniform random element of $E_A(\mathbb{F}_p)$ then Q is a uniform random element of $E_A(\mathbb{F}_p)[\ell] \cong \mathbb{Z}/\ell$. The order of Q is thus the desired ℓ with probability $1 - 1/\ell$. Otherwise Q is ∞, the neutral element on the curve, which is represented by $x = 0$. Checking for $x = 0$ is a reliable way to detect this case: the only other point represented by $x = 0$ is $(0, 0)$, which is outside $E_A(\mathbb{F}_p)[\ell]$ since ℓ is odd.

Different concepts of constant time. Beware that there are two different notions of "constant time" for cryptographic algorithms. One notion is that the time for each operation is independent of *secrets*. This notion allows the CSIDH user to generate a uniform random element of $E_A(\mathbb{F}_p)[\ell]$ and try again if the point is ∞, guaranteeing success with an average of $\ell/(\ell-1)$ tries. The time varies, but the variation is independent of the secret A.

A stricter notion is that the time for each operation is independent of *all* inputs. The time depends on parameters, such as p in CSIDH, but does not depend on random choices. We emphasize that a quantum circuit operating on many inputs in superposition is, by definition, using this stricter notion. We thus choose the sequence of operations carried out by the circuit, and analyze the probability that this sequence fails.

Amplifying the success probability. Having each 3-isogeny fail with probability $1/3$, each 5-isogeny fail with probability $1/5$, etc. creates a correctness challenge for higher-level algorithms that compute many isogenies.

A simple workaround is to generate many points $Q_1, Q_2, \ldots, Q_N$, and use bit operations on the points to select the first point with $x \neq 0$. This fails if all of the points have $x = 0$. Independent uniform random points have overall failure probability $1/\ell^N$. One can make $1/\ell^N$ arbitrarily small by choosing N large enough: for example, $1/3^N$ is below $1/2^{32}$ for $N \geq 21$, and is below $1/2^{256}$ for $N \geq 162$.

We return to the costs of generating so many points, and the costs of more sophisticated alternatives, when we analyze algorithms to compute the CSIDH group action.

9.5.2 – Nonuniform distribution of points. We actually generate random points using Elligator (see Section 9.4.2), which generates only $(p-3)/2$ different curve points P. At most $(p+1)/\ell$ of these points produce $Q = \infty$, so the probability of failure is upper bounded by $(2/\ell)(p+1)/(p-3) \approx 2/\ell$.

This bound cannot be simultaneously tight for $\ell = 3$, $\ell = 5$, and $\ell = 7$ (assuming that $3 \cdot 5 \cdot 7$ divides $p+1$): if it were then the Elligator outputs would include all points having orders dividing $(p+1)/3$ or $(p+1)/5$ or $(p+1)/7$, but this accounts for more than 54% of all curve points; contradiction.

Points generated by Elligator actually appear to be much better distributed modulo each ℓ, with failure chance almost exactly $1/\ell$. Experiments support this conjecture. Readers concerned with the gap between the provable $2/\ell$ and the heuristic $1/\ell$ may prefer to add or subtract a few Elligator 2 outputs, obtaining a distribution provably close to uniform (see [Tib14]) at a moderate cost in performance. A more efficient approach is to accept a doubling of failure probability and use a small number of extra iterations to compensate.

We shall later see other methods of obtaining rational ℓ-torsion points, e.g., by pushing points through ℓ'-isogenies. This does not make a difference in the analysis of failure probabilities.

For comparison, generating a random point on the curve or twist (see Section 9.4.1) has failure probability above $1/2$ at finding a curve point of order ℓ. See Section 9.6.2 for the impact of this difference upon higher-level algorithms.

9.5.3 – Computing an ℓ-isogenous curve from a point of order ℓ. Once we have obtained the x-coordinate of a point Q of order ℓ in $E_A(\mathbb{F}_p)$, we compute the x-coordinates of the points $Q, 2Q, 3Q, \ldots, ((\ell-1)/2)Q$. We use this information to compute $B = \mathcal{L}(A)$, the coefficient determining the ℓ-isogenous curve E_B.

Recall from Section 9.3.4 that computing $Q, 2Q, 3Q, \ldots, ((\ell-1)/2)Q$ has a cost of $(\ell-3)\mathbf{S} + 1.5(\ell-3)\mathbf{M}$ for affine Q and affine A, and just $1\mathbf{M}$ extra for affine Q and projective A. Chapter 3 took more time here, namely $(\ell-3)\mathbf{S} + 2(\ell-3)\mathbf{M}$, to handle projective Q and projective A. We decide, based on comparing ℓ to the cost of an inversion, whether to spend an inversion converting Q to affine coordinates.

Given the x-coordinates of $Q, 2Q, 3Q, \ldots, ((\ell-1)/2)Q$, Chapter 3 took approximately $3\ell\mathbf{M}$ to compute B. Meyer and Reith [MR18] pointed out that CSIDH benefits from using the Edwards-coordinate isogeny formulas from Moody and Shumow [MS16]; we reuse this speedup. These formulas work as follows:

- Compute $a = A + 2$ and $d = A - 2$.
- Compute the Edwards y-coordinates of $Q, 2Q, 3Q, \ldots, ((\ell-1)/2)Q$. The Edwards y-coordinate is related to the Montgomery x-coordinate by $y = (x-1)/(x+1)$. We are given each x projectively as X/Z, and compute y projectively as Y/T where $Y = X - Z$ and $T = X + Z$. Note that Y and T naturally occur as intermediate values in the Montgomery ladder.
- Compute the product of these y-coordinates: i.e., compute $\prod Y$ and $\prod T$. This uses a total of $(\ell-3)\mathbf{M}$.
- Compute $a' = a^\ell(\prod T)^8$ and $d' = d^\ell(\prod Y)^8$. Each ℓth power takes a logarithmic (in ℓ) number of squarings and multiplications; see Section 9.13.4.
- Compute, projectively, $B = 2(a'+d')/(a'-d')$. Subsequent computations decide whether to convert B to affine form.

These formulas are almost three times faster than the formulas used in [Cas+18]. The total cost of computing B from Q is almost two times faster than in [Cas+18].

Handling variable ℓ. We point out that isogeny computations for $\ell = 3$, $\ell = 5$, $\ell = 7$, etc., have a Matryoshka-doll structure, allowing a constant-time computation to handle many different values of ℓ with essentially the same cost as a single computation for the largest value of ℓ.

Concretely, the following procedure takes approximately $\ell_n\mathbf{S} + 2.5\ell_n\mathbf{M}$, and allows any $\ell \leq \ell_n$. If the context places a smaller upper bound upon ℓ then one can replace ℓ_n with that upper bound, saving time; we return to this idea later.

Compute the Montgomery x-coordinates and the Edwards y-coordinates of $Q, 2Q, 3Q, \ldots, ((\ell_n - 1)/2)Q$. Use bit operations to replace each Edwards y-coordinate with 1 after the first $(\ell - 1)/2$ points. Compute the product of these modified y-coordinates; this is the desired product of the Edwards y-coordinates of the first $(\ell - 1)/2$ points. Finish computing B as above. Note that the exponentiation algorithm in Section 9.13.4 allows variable ℓ.

9.5.4 – Applying an ℓ-isogeny to a point. The following formulas define an ℓ-isogeny from E_A to E_B with kernel $E_A(\mathbb{F}_p)[\ell]$. The x-coordinate of the image of a point $P_1 \in E_A(\mathbb{F}_p)$ under this isogeny is

$$x(P_1) \cdot \prod_{j=1}^{(\ell-1)/2} \left(\frac{x(P_1)x(jQ) - 1}{x(P_1) - x(jQ)} \right)^2.$$

Each $x(jQ)$ appearing here was computed above in projective form X/Z, and the expression $(x(P_1)x(jQ) - 1)/(x(P_1) - x(jQ))$ is $(x(P_1)X - Z)/(x(P_1)Z - X)$. This takes $2\mathbf{M}$ to compute projectively if $x(P_1)$ is affine, and thus $(\ell - 1)\mathbf{M}$ across all j. Multiplying the numerators takes $((\ell - 3)/2)\mathbf{M}$, multiplying the denominators takes $((\ell - 3)/2)\mathbf{M}$, squaring both takes $2\mathbf{S}$, and multiplying by $x(P_1)$ takes $1\mathbf{M}$, for a total of $(2\ell - 3)\mathbf{M} + 2\mathbf{S}$.

If $x(P_1)$ is instead given in projective form as X_1/Z_1, computing $X_1X - Z_1Z$ and $X_1Z - Z_1X$ might seem to take $4\mathbf{M}$, but one can instead compute the sum and difference of the products $(X_1 - Z_1)(X + Z)$ and $(X_1 + Z_1)(X - Z)$, using just $2\mathbf{M}$. The only extra cost compared to the affine case is four extra additions. This speedup was pointed out by Montgomery [Mon87] in the context of the Montgomery ladder. The initial CSIDH software accompanying [Cas+18] did not use this speedup but [MR18] mentioned the applicability to CSIDH.

In the opposite direction, if inversion is cheap enough to justify making $x(P_1)$ *and* every $x(jQ)$ affine, then $2\mathbf{M}$ drops to $1\mathbf{M}$, and the total cost drops to approximately $1.5\ell\mathbf{M}$.

As in Section 9.5.3, we allow ℓ to be a variable. The cost of variable ℓ is the cost of a single computation for the maximum allowed ℓ, plus a minor cost for bit operations to select relevant inputs to the product.

9.6 — Computing the action: basic algorithms

Jao, LeGrow, Leonardi, and Ruiz-Lopez [JLLR18] suggested a three-level quantum algorithm to compute $\mathcal{L}_1^{e_1} \cdots \mathcal{L}_n^{e_n}$. This section shows how to make the algorithm an order of magnitude faster for any particular failure probability.

9.6.1 – Baseline: reliably computing each $\mathcal{L}_i$. The lowest level in [JLLR18] *reliably* computes $\mathcal{L}_i$ as follows. Generate r uniform random points on the curve or twist, as in Section 9.4.1. Multiply each point by $(p + 1)/\ell_i$, as in Section 9.5.1, hoping to obtain a point of order ℓ_i on the curve. Use Vélu's formulas to finish the computation, as in Section 9.5.3.

Each point has success probability $(1/2)(1-1/\ell_i)$, where $1/2$ is the probability of obtaining a curve point (rather than a twist point) and $1-1/\ell_i$ is the probability of obtaining a point of order ℓ_i (rather than order 1). The chance of all r points failing is thus $(\ell_i+1)^r/(2\ell_i)^r$, decreasing from $(2/3)^r$ for $\ell_i = 3$ down towards $(1/2)^r$ as ℓ_i grows . One chooses r to obtain a failure probability as small as desired for the isogeny computation, and for the higher levels of the algorithm.

The lowest level optionally computes $\mathcal{L}_i^{-1}$ instead of $\mathcal{L}_i$. The approach in [JLLR18], following [Cas+18], is to use points on the twist instead of points on the curve; an alternative is to compute $\mathcal{L}_i^{-1}(A)$ as $-\mathcal{L}_i(-A)$.

The middle level of the algorithm computes $\mathcal{L}_i^e$, where e is a variable whose absolute value is bounded by a constant C. This level calls the lowest level exactly C times, performing a series of C steps of $\mathcal{L}_i^{\pm 1}$, using bit operations on e to decide whether to retain the results of each step. The ± 1 is chosen as the sign of e, or as an irrelevant 1 if $e = 0$.

The highest level of the algorithm computes $\mathcal{L}_1^{e_1} \cdots \mathcal{L}_n^{e_n}$, where each e_i is between $-C$ and C, by calling the middle level n times, starting with $\mathcal{L}_1^{e_1}$ and ending with $\mathcal{L}_n^{e_n}$. (Our definition of the action applied $\mathcal{L}_n^{e_n}$ first, but the $\mathcal{L}_i$ operators commute with each other, so the order does not matter.)

Importance of bounding each exponent. We emphasize that this algorithm requires each exponent e_i to be between $-C$ and C, i.e., requires the vector $(e_1, \ldots, e_n)$ to have ∞-norm at most C.

We use $C = 5$ for CSIDH-512 as an illustrative example, but all known attacks use larger vectors (see Section 9.2.1). C is chosen in [JLLR18] so that every input, every vector in superposition, has ∞-norm at most C; smaller values of C create a failure probability that needs to be analyzed.

We are not saying that the ∞-norm is the only important feature of the input vectors. On the contrary: our constant-time subroutine to handle variable-ℓ isogenies creates opportunities to share work between separate exponents. See Sections 9.5.3 and 9.7.

Concrete example. For concreteness we suppose that the input vectors are uniformly random in $e \in \{-5, \ldots, 5\}^{74}$. The highest level calls the middle level $n = 74$ times, and the middle level calls the lowest level $C = 5$ times. Taking $r = 70$ guarantees failure probability at most $(2/3)^{70}$ at the lowest level, and thus failure probability at most $1 - (1 - (2/3)^{70})^{74 \cdot 5} \approx 0.750 \cdot 2^{-32}$ for the entire algorithm.

This type of analysis is used in [JLLR18] to select r. We point out that the failure probability of the algorithm is actually lower, and a more accurate analysis allows a smaller value of r. One can, for example, replace $(1 - (2/3)^r)^{74}$ with $\prod_i (1 - (\ell_i + 1)^r/(2\ell_i)^r)$, showing that $r = 59$ suffices for failure probability below 2^{-32}. With more work one can account for the distribution of input vectors e, rather than taking the worst-case e as in [JLLR18]. However, one cannot hope to do better than $r = 55$ here: there is a $10/11$ chance that at least one 3-isogeny is required, and taking $r \leq 54$ means that this 3-isogeny fails with probability at least $(2/3)^{54}$, for an overall failure chance at least $(10/11)(2/3)^{54} > 2^{-32}$.

With the choice $r = 70$ as in [JLLR18], there are $74 \cdot 5 \cdot 70 = 25900$ iterations, in total using more than 100 million multiplications in $\mathbb{F}_p$. In the rest of this section we will reduce the number of iterations by a factor 30, and in Section 9.7 we will reduce the number of iterations by another factor 3, with only moderate increases in the cost of each iteration.

9.6.2 – Fewer failures, and sharing failures. We now introduce Algorithm 9.1, which improves upon the algorithm from [JLLR18] in three important ways. First, we use Elligator to target the curve (or the twist if desired); see Section 9.4.2. This reduces the failure probability of

Algorithm 9.1: Basic class-group action evaluation.

Parameters: Odd primes $\ell_1 < \cdots < \ell_n$ with $n \geq 1$, a prime $p = 4\ell_1 \cdots \ell_n - 1$, and positive integers $(r_1, \ldots, r_n)$.

Input: $A \in S_p$, integers $(e_1, \ldots, e_n)$.

Output: $\mathfrak{l}_1^{e_1} \cdots \mathfrak{l}_n^{e_n}(A)$ or "fail".

for $i \leftarrow 1$ **to** n **do**
  for $j \leftarrow 1$ **to** r_i **do**
    Let $s = \mathrm{sign}(e_i) \in \{-1, 0, +1\}$.
    Find a random point P on E_{sA} using Elligator.
    Compute $Q \leftarrow ((p+1)/\ell_i)P$.
    Compute B with $E_B \cong E_{sA}/\langle Q \rangle$ if $Q \neq \infty$.
    Set $A \leftarrow sB$ if $Q \neq \infty$ and $s \neq 0$.
    Set $e_i \leftarrow e_i - s$ if $Q \neq \infty$.
Set $A \leftarrow$ "fail" if $(e_1, \ldots, e_n) \neq (0, \ldots, 0)$.
Return A.

r points from $(2/3)^r$ to, heuristically, $(1/3)^r$ for $\ell_i = 3$; from $(3/5)^r$ to $(1/5)^r$ for $\ell_i = 5$; from $(4/7)^r$ to $(1/7)^r$ for $\ell_i = 7$; etc.

Second, we allow a separate r_i for each ℓ_i. This lets us exploit the differences in failure probabilities as ℓ_i varies.

Third, we handle failures already at the middle level instead of the lowest level. The strategy in [JLLR18] to compute $\mathfrak{l}_i^e$ with $-C \leq e \leq C$ is to perform C iterations, where each iteration builds up many points on one curve and *reliably* moves to the next curve. We instead perform r_i iterations, where each iteration *tries* to move from one curve to the next by generating just one point. For $C = 1$ this is the same, but for larger C we obtain better tradeoffs between the number of points and the failure probability.

As a concrete example, generating 20 points on one curve with Elligator has failure probability $(1/3)^{20}$ for $\ell_i = 3$. A series of 5 such computations, overall generating 100 points, has failure probability $1 - (1 - (1/3)^{20})^5 \approx 2^{-29.37}$. If we instead perform just 50 iterations, where each iteration generates one point to move 1 step with probability $2/3$, then the probability that we move fewer than 5 steps is just $3846601/3^{50} \approx 2^{-57.37}$; see Section 9.6.3. Our iterations are more expensive than in [JLLR18] — next to each Elligator computation, we always (even when $Q = \infty$) perform the steps for computing an ℓ_i-isogeny — but (for CSIDH-512 etc.) this is not a large effect: the cost of each iteration is dominated by scalar multiplication.

We emphasize that all of our algorithms take constant time. Expressions like "Compute $X \leftarrow Y$ if c" mean that we always compute Y and the bit c, and we then replace the jth bit X_j of X with the jth bit Y_j of Y for each j if c is set, by replacing X_j with $X_j \oplus c(X_j \oplus Y_j)$. This is why Algorithm 9.1 always carries out the bit operations for computing an ℓ_i-isogenous curve, as noted above, even when $Q = \infty$.

9.6.3 – Analysis. We consider the inner loop body of Algorithm 9.1 for a fixed i, hence write $\ell = \ell_i$, $e = e_i$, and $r = r_i$ for brevity.

Heuristically (see Section 9.5.2), we model each point Q as independent and uniform ran-

Table 9.1: Examples of choices of r_i for Algorithm 9.1 for three levels of failure probability for uniform random CSIDH-512 vectors with entries in $\{-5,\ldots,5\}$. Failure probabilities ϵ are rounded to three digits after the decimal point. The "total" column is $\sum r_i$, the total number of iterations. The "[JLLR18]" column is $74 \cdot 5 \cdot r$, the number of iterations in the algorithm of [JLLR18], with r chosen as in [JLLR18] to have $1-(1-(2/3)^r)^{74\cdot 5}$ at most 2^{-1} or 2^{-32} or 2^{-256}. Compare Table 9.2 for $\{-10,\ldots,10\}$.

ϵ \ ℓ_i	3	5	7	11	13	17	…	359	367	373	587	total	[JLLR18]
$0.499 \cdot 2^{-1}$	11	9	8	7	7	6	…	5	5	5	5	406	5920
$0.178 \cdot 2^{-32}$	36	25	21	18	17	16	…	10	10	10	9	869	25900
$0.249 \cdot 2^{-256}$	183	126	105	85	80	73	…	37	37	37	34	3640	167610

Table 9.2: Examples of choices of r_i for Algorithm 9.1 for three levels of failure probability for uniform random CSIDH-512 vectors with entries in $\{-10,\ldots,10\}$. Failure probabilities ϵ are rounded to three digits after the decimal point. The "total" column is $\sum r_i$, the total number of iterations. The "[JLLR18]" column is $74 \cdot 10 \cdot r$, the number of iterations in the algorithm of [JLLR18], with r chosen as in [JLLR18] to have $1-(1-(2/3)^r)^{74\cdot 10}$ at most 2^{-1} or 2^{-32} or 2^{-256}. Compare Table 9.1 for $\{-5,\ldots,5\}$.

ϵ \ ℓ_i	3	5	7	11	13	17	…	359	367	373	587	total	[JLLR18]
$0.521 \cdot 2^{-1}$	20	15	14	13	12	12	…	10	10	10	10	786	13320
$0.257 \cdot 2^{-32}$	48	34	30	25	24	22	…	15	15	15	14	1296	52540
$0.215 \cdot 2^{-256}$	201	139	116	96	90	82	…	43	43	43	41	4185	335960

dom in a cyclic group of order ℓ, so Q has order 1 with probability $1/\ell$ and order ℓ with probability $1-1/\ell$. The number of points of order ℓ through r iterations of the inner loop is binomially distributed with parameters r and $1-1/\ell$. The probability that this number is $|e|$ or larger is $\text{prob}_{\ell,e,r} = \sum_{t=|e|}^{r} \binom{r}{t} (1-1/\ell)^t / \ell^{r-t}$. This is exactly the probability that Algorithm 9.1 successfully performs the $|e|$ desired iterations of $\mathcal{L}^{\text{sign}(e)}$.

Let C be a nonnegative integer. The overall success probability of the algorithm for a particular input vector $(e_1,\ldots,e_n) \in \{-C,\ldots,C\}^n$ is

$$\prod_{i=1}^{n} \text{prob}_{\ell_i,e_i,r_i} \geq \prod_{i=1}^{n} \text{prob}_{\ell_i,C,r_i}.$$

Average over vectors to see that the success probability of the algorithm for a uniform random vector in $\{-C,\ldots,C\}^n$ is $\prod_{i=1}^{n} \left(\sum_{-C\leq e\leq C} \text{prob}_{\ell_i,e,r_i}/(2C+1)\right)$.

9.6.4 – Examples of target failure probabilities. The acceptable level of failure probability for our algorithm depends on the attack using the algorithm. For concreteness we consider three possibilities for CSIDH-512 failure probabilities, namely having the algorithm fail for a uniform random vector with probabilities at most 2^{-1}, 2^{-32}, and 2^{-256}.

Our rationale for considering these probabilities is as follows. Probabilities around 2^{-1} are easy to test, and may be of interest beyond this chapter for constructive scenarios where failing computations can simply be retried. If each computation needs to work correctly, and there are many computations, then failure probabilities need to be much smaller, say 2^{-32}. Asking for every input in superposition to work correctly in one computation (for example, [JLLR18] asks

for this) requires a much smaller failure probability, say 2^{-256}. Performance results for these three cases also provide an adequate basis for estimating performance in other cases.

Table 9.1 presents three reasonable choices of $(r_1, \ldots, r_n)$, one for each of the failure probabilities listed above, for the case of CSIDH-512 with uniform random vectors with entries in $\{-5, \ldots, 5\}$. For each target failure probability δ and each i, the table chooses the minimum r_i such that $\sum_{-C \le e \le C} \text{prob}_{\ell_i,e,r_i}/(2C+1)$ is at least $(1-\delta)^{1/n}$. The overall success probability is then at least $1-\delta$ as desired. The discontinuity of choices of $(r_1, \ldots, r_n)$ means that the actual failure probability ϵ is somewhat below δ, as shown by the coefficients 0.499, 0.178, 0.249 in Table 9.1. We could move closer to the target failure probability by choosing successively $r_n, r_{n-1}, \ldots$, adjusting the probability $(1-\delta)^{1/n}$ at each step in light of the overshoot from previous steps. The values r_i for $\epsilon \approx 0.499 \cdot 2^{-1}$ have been experimentally verified using a modified version of the CSIDH software. To illustrate the impact of larger vector entries, we also present similar data in Table 9.2 for uniform random vectors with entries in $\{-10, \ldots, 10\}$.

The "total" column in Table 9.1 shows that this algorithm uses, e.g., 869 iterations for failure probability $0.178 \cdot 2^{-32}$ with vector entries in $\{-5, \ldots, 5\}$. Each iteration consists mostly of a scalar multiplication, plus some extra cost for Elligator, Vélu's formulas, etc. Overall there are roughly 5 million field multiplications, accounting for roughly 2^{41} nonlinear bit operations, implying a quantum computation using roughly 2^{45} T-gates.

As noted in Section 9.1, using the algorithm of [JLLR18] on top of the modular-multiplication algorithm from [RNSL17] would use approximately 2^{51} nonlinear bit operations for the same distribution of input vectors. We save a factor 30 in the number of iterations compared to [JLLR18], and we save a similar factor in the number of bit operations for each modular multiplication compared to [RNSL17].

We do not analyze this algorithm in more detail: the algorithms we present below are faster.

9.7 — Reducing the top nonzero exponent

Most of the iterations in Algorithm 9.1 are spent on exponents that are already 0. For example, consider the 869 iterations mentioned above for failure probability $0.178 \cdot 2^{-32}$ for uniform random CSIDH-512 vectors with entries in $\{-5, \ldots, 5\}$. Entry e_i has absolute value $30/11$ on average, and needs $(30/11)\ell_i/(\ell_i - 1)$ iterations on average, for a total of $\sum_i (30/11)\ell_i/(\ell_i - 1) \approx 206.79$ useful iterations on average. This means that there are 662.21 useless iterations on average, many more than one would expect to be needed to guarantee this failure probability.

This section introduces a constant-time algorithm that achieves the same failure probability with far fewer iterations. For example, in the above scenario, just 294 iterations suffice to reduce the failure probability below 2^{-32}. Each iteration becomes (for CSIDH-512) about 25% more expensive, but overall the algorithm uses far fewer bit operations.

9.7.1 – Iterations targeting variable ℓ. It is obvious how to avoid useless iterations for variable-time algorithms: when an exponent reaches 0, move on to the next exponent. In other words, always focus on reducing a nonzero exponent, if one exists.

What is new is doing this in constant time. This is where we exploit the Matryoshka-doll structure from Section 9.5.3, computing an isogeny for variable ℓ in constant time. We now pay for an ℓ_n-isogeny in each iteration rather than an ℓ-isogeny, but the iteration cost is still dominated by scalar multiplication. Concretely, for CSIDH-512, an average ℓ-isogeny costs about 600 multiplications, and an ℓ_n-isogeny costs about 2000 multiplications, but a scalar multiplication costs about 5000 multiplications.

Algorithm 9.2: Evaluating CSIDH by reducing the top nonzero exponent.

Parameters: Odd primes $\ell_1 < \cdots < \ell_n$ with $n \geq 1$, a prime $p = 4\ell_1 \cdots \ell_n - 1$, and a positive integer r.

Input: $A \in S_p$, integers $(e_1, \ldots, e_n)$.

Output: $\mathfrak{L}_1^{e_1} \cdots \mathfrak{L}_n^{e_n}(A)$ or "fail".

for $j \leftarrow 1$ **to** r **do**
- Let $i = \max\{k : e_k \neq 0\}$, or $i = 1$ if each $e_k = 0$.
- Let $s = \operatorname{sign}(e_i) \in \{-1, 0, +1\}$.
- Find a random point P on E_{sA} using Elligator.
- Compute $Q \leftarrow ((p+1)/\ell_i)P$.
- Compute B with $E_B \cong E_{sA}/\langle Q \rangle$ if $Q \neq \infty$, using the ℓ_i-isogeny formulas from Section 9.5.3 with maximum degree ℓ_n.
- Set $A \leftarrow sB$ if $Q \neq \infty$ and $s \neq 0$.
- Set $e_i \leftarrow e_i - s$ if $Q \neq \infty$.

Set $A \leftarrow$ "fail" if $(e_1, \ldots, e_n) \neq (0, \ldots, 0)$.

Return A.

We choose to always reduce the top exponent that is not already 0. "Top" here refers to position, not value: we reduce the nonzero e_i where i is maximized. See Algorithm 9.2.

9.7.2 – Upper bounds on the failure probability. One can crudely estimate the failure probability of Algorithm 9.2 in terms of the 1-norm $E = |e_1| + \cdots + |e_n|$ as follows. Model each iteration as having failure probability $1/3$ instead of $1/\ell_i$; this produces a loose upper bound for the overall failure probability of the algorithm.

In this model, the chance of needing exactly r iterations to find a point of order ℓ_i is the coefficient of x^r in the power series

$$(2/3)x + (2/9)x^2 + (2/27)x^3 + \cdots = 2x/(3-x).$$

The chance of needing exactly r iterations to find all E points is the coefficient of x^r in the Eth power of that power series, namely $c_r = \binom{r-1}{E-1}2^E/3^r$ for $r \geq E$. See generally [Wil94] for an introduction to the power-series view of combinatorics; there are many other ways to derive the formula $\binom{r-1}{E-1}2^E/3^r$, but we make critical use of power series for fast computations in Sections 9.7.3 and 9.8.3.

The failure probability of r iterations of Algorithm 9.2 is at most the failure probability of r iterations in this model, namely $f(r, E) = 1 - c_E - c_{E+1} - \cdots - c_r$. The failure probability of r iterations for a uniform random vector with entries in $\{-C, \ldots, C\}$ is at most $\sum_{0 \leq E \leq nC} f(r, E)g[E]$. Here $g[E]$ is the probability that a vector has 1-norm E, which we compute as the coefficient of x^E in the nth power of the polynomial $(1 + 2x + 2x^2 + \cdots + 2x^C)/(2C+1)$. For example, with $n = 74$ and $C = 5$, the failure probability in this model (rounded to 3 digits after the decimal point) is $0.999 \cdot 2^{-1}$ for $r = 302$; $0.965 \cdot 2^{-2}$ for $r = 319$; $0.844 \cdot 2^{-32}$ for $r = 461$; and $0.570 \cdot 2^{-256}$ for $r = 823$. As a double-check, we observe that a simple simulation of the model for $r = 319$ produces 241071 failures in 1000000 experiments, close to the predicted $0.965 \cdot 2^{-2} \cdot 1000000 \approx 241250$.

9.7.3 – Exact values of the failure probability. The upper bounds from the model above are too pessimistic, except for $\ell_i = 3$. We instead compute the exact failure probabilities as follows.

The chance that $\mathcal{L}_1^{e_1} \cdots \mathcal{L}_n^{e_n}$ requires exactly r iterations is the coefficient of x^r in the power series

$$\left(\frac{(\ell_1-1)x}{\ell_1-x}\right)^{|e_1|} \cdots \left(\frac{(\ell_n-1)x}{\ell_n-x}\right)^{|e_n|}.$$

What we want is the average of this coefficient over all vectors $(e_1, \ldots, e_n) \in \{-C, \ldots, C\}^n$. This is the same as the coefficient of the average, and the average factors nicely as

$$\left(\sum_{-C \le e_1 \le C} \frac{1}{2C+1}\left(\frac{(\ell_1-1)x}{\ell_1-x}\right)^{|e_1|}\right) \cdots \left(\sum_{-C \le e_n \le C} \frac{1}{2C+1}\left(\frac{(\ell_n-1)x}{\ell_n-x}\right)^{|e_n|}\right).$$

We compute this product as a power series with rational coefficients: for example, we compute the coefficients of $x^0, \ldots, x^{499}$ if we are not interested in 500 or more iterations. We then add together the coefficients of $x^0, \ldots, x^r$ to find the exact success probability of r iterations of Algorithm 9.2.

As an example we again take CSIDH-512 with $C = 5$. The failure probability (again rounded to 3 digits after the decimal point) is $0.960 \cdot 2^{-1}$ for $r = 207$; $0.998 \cdot 2^{-2}$ for $r = 216$; $0.984 \cdot 2^{-32}$ for $r = 294$; $0.521 \cdot 2^{-51}$ for $r = 319$; and $0.773 \cdot 2^{-256}$ for $r = 468$. We double-checked these averages against the results of Monte Carlo calculations for these values of r. Each Monte Carlo iteration sampled a uniform random 1-norm (weighted appropriately for the initial probability of each 1-norm), sampled a uniform random vector within that 1-norm, and computed the failure probability for that vector using the single-vector generating function.

9.7.4 – Analysis of the cost. We have fully implemented Algorithm 9.2 in our bit-operation simulator. One iteration for CSIDH-512 uses $9208697761 \approx 2^{33}$ bit operations, which includes $3805535430 \approx 2^{32}$ nonlinear bit operations. More than 95% of the cost is explained as follows:

- Each iteration uses a Montgomery ladder with a 511-bit scalar. (We could save a bit here: the largest useful scalar is $(p+1)/3$, which is below 2^{510}.) We use an affine input point and an affine A, so this costs $2044\mathbf{S} + 3066\mathbf{M}$.
- Each iteration uses the formulas from Section 9.5.3 with $\ell = 587$. This takes $602\mathbf{S} + 1472\mathbf{M}$: specifically, $584\mathbf{S}+876\mathbf{M}$ for multiples of the point of order ℓ (again affine); $584\mathbf{M}$ for the product of Edwards y-coordinates; $18\mathbf{S}+10\mathbf{M}$ for two ℓth powers; and $2\mathbf{M}$ to multiply by two 8th powers. (We merge the $6\mathbf{S}$ for the 8th powers into the squarings used for the ℓth powers.)
- Each iteration uses two inversions to obtain affine Q and A, each $507\mathbf{S} + 97\mathbf{M}$, and one Legendre-symbol computation, $506\mathbf{S} + 96\mathbf{M}$.

This accounts for $4166\mathbf{S}+4828\mathbf{M}$ per iteration, i.e., $4166 \cdot 349596 + 4828 \cdot 447902 = 3618887792 \approx 2^{32}$ nonlinear bit operations.

The cost of 294 iterations is simply $294 \cdot 3805535430 = 1118827416420 \approx 2^{40}$ nonlinear bit operations. This justifies the first (B, ϵ) claim in Section 9.1.

9.7.5 – Decreasing the maximum degrees. Always performing isogeny computations capable of handling degrees up to ℓ_n is wasteful: With overwhelming probability, almost all of the 294 iterations required for a failure probability of less than 2^{-32} with the approach discussed so far actually compute isogenies of degree (much) less than ℓ_n. For example, with e uniformly

random in $\{-5, \ldots, 5\}$, the probability that 10 iterations are not sufficient to eliminate all 587-isogenies is approximately 2^{-50}. Therefore, using smaller upper bounds on the isogeny degrees for later iterations of the algorithm will not do much harm to the success probability while significantly improving the performance. We modify Algorithm 9.2 as follows:

- Instead of a single parameter r, we use a list $(r_1, \ldots, r_n)$ of non-negative integers, each r_i denoting the number of times an isogeny computation capable of handling degrees up to ℓ_i is performed.
- The loop iterating from 1 through r is replaced by an outer loop on u from n down to 1, and inside that an inner loop on j from 1 up to r_u. The loop body is unchanged, except that the maximum degree for the isogeny formulas is now ℓ_u instead of ℓ_n.

For a given sequence $(r_1, \ldots, r_n)$, the probability of success can be computed as follows:

- For each $i \in \{1, \ldots, n\}$, compute the generating function

$$\phi_i(x) = \sum_{-C \le e_i \le C} \frac{1}{2C+1} \left(\frac{(\ell_i - 1)x}{\ell_i - x} \right)^{|e_i|}$$

  of the number of ℓ_i-isogeny steps that have to be performed.
- Since we are no longer only interested in the total number of isogeny steps to be computed, but also in their degrees, we cannot simply take the product of all ϕ_i as before. Instead, to account for the fact that failing to compute a ℓ_i-isogeny before the maximal degree drops below ℓ_i implies a total failure, we iteratively compute the product of the ϕ_i from $k = n$ down to 1, but truncate the product after each step. Truncation after some power x^t means eliminating all branches of the probabilistic process in which more than t isogeny steps are needed for the computations so far. In our case we use $t = \sum_{j=i}^{n} r_j$ after multiplying by ϕ_i, which removes all outcomes in which more isogeny steps of degree $\ge \ell_i$ would have needed to be computed.
- After all ϕ_i have been processed (including the final truncation), the probability of success is the sum of all coefficients of the remaining power series.

Note that we have only described a procedure to compute the success probabilty once $r_1, \ldots, r_n$ are known. It is unclear how to find the optimal values r_i which minimize the cost of the resulting algorithm, while at the same time respecting a certain failure probability. We tried various reasonable-looking choices of strategies to choose the r_i according to certain prescribed failure probabilities after each individual step. Experimentally, a good rule seems to be that the failure probability after processing ϕ_i should be bounded by $\epsilon \cdot 2^{2/i-2}$, where ϵ is the overall target failure probability. The results are shown in Table 9.3.

The average degree of the isogenies used constructively in CSIDH-512 is about 174.6, which is not much smaller than the average degree we achieve. Since we still need to control the error probability, it does not appear that one can expect to get much closer to the constructive case.

Also note that the total number of isogeny steps for $\epsilon \approx 2^{-32}$ and $\epsilon \approx 2^{-256}$ is each only one more than the previous number r of isogeny computations, hence one can expect significant savings using this strategy. Assuming that about $1/4$ of the total time is spent on Vélu's formulas (which is close to the real proportion), we get a speedup of about 16% for $\epsilon \approx 2^{-32}$ and about 17% for $\epsilon \approx 2^{-256}$.

Table 9.3: Examples of choices of $r_i, \ldots, r_i$ for Algorithm 9.2 with reducing the maximal degree in Vélu's formulas for uniform random CSIDH-512 vectors with entries in $\{-5, \ldots, 5\}$. Failure probabilities ϵ are rounded to three digits after the decimal point.

ϵ	$r_n \ldots r_1$	$\sum r_i$	avg. ℓ
$0.594 \cdot 2^{-1}$	5 3 4 5 3 5 5 4 3 5 4 3 4 4 3 4 3 4 3 3 3 4 3 3 3 4 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 2 4 2 3 3 3 3 2 3 3 3 2 3 3 2 3 2 3 2 3 2 2 3 2 2 2 1 1 1 0 0	218	205.0
$0.970 \cdot 2^{-32}$	9 5 5 5 5 5 4 5 5 5 4 5 4 5 5 4 5 4 4 5 5 4 4 4 5 4 4 4 4 4 3 5 3 4 4 4 3 4 4 4 3 4 4 3 4 3 4 3 4 4 3 4 3 3 4 4 3 3 4 3 3 4 3 4 3 3 4 3 3 3 4 3 3 4	295	196.0
$0.705 \cdot 2^{-256}$	34 8 6 6 5 6 6 5 5 6 5 6 5 5 5 5 6 5 5 5 5 6 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 4 6 5 5 5 5 5 4 6 5 5 5 5 5 5 5 6 5 5 6 6 6 6 7 7 11 16 38	469	182.7

9.8 — Pushing points through isogenies

Algorithms 9.1 and 9.2 spend most of their time on scalar multiplication. This section pushes points through isogenies to reduce the time spent on scalar multiplication, saving time overall.

The general idea of balancing isogeny computation with scalar multiplication was introduced in [DJP14] in the SIDH context, and was reused in the variable-time CSIDH algorithms in [Cas+18]. This section adapts the idea to the context of constant-time CSIDH computation.

9.8.1 – Why pushing points through isogenies saves time. To illustrate the main idea, we begin by considering a sequence of just two isogenies with the same sign. Specifically, assume that, given distinct ℓ_1 and ℓ_2 dividing $p+1$, we want to compute $\mathcal{L}_1\mathcal{L}_2(A) = B$. Here are two different methods:

- **Method 1.** The method of Algorithm 9.1 uses Elligator to find $P_1 \in E_A(\mathbb{F}_p)$, computes $Q_1 \leftarrow [(p+1)/\ell_1]P_1$, computes $E_{A'} = E_A/\langle Q_1 \rangle$, uses Elligator to find $P_2 \in E_{A'}(\mathbb{F}_p)$, computes $Q_2 \leftarrow [(p+1)/\ell_2]P_2$, and computes $E_B = E_{A'}/\langle Q_2 \rangle$. Failure cases: if $Q_1 = \infty$ then this method computes $A' = A$, failing to compute $\mathcal{L}_1$; similarly, if $Q_2 = \infty$ then this method computes $B = A'$, failing to compute $\mathcal{L}_2$.
- **Method 2.** The method described in this section instead uses Elligator to find $P \in E_A(\mathbb{F}_p)$, computes $R \leftarrow [(p+1)/\ell_1\ell_2]P$, computes $Q \leftarrow [\ell_2]R$, computes $\varphi\colon E_A \to E_{A'} = E_A/\langle Q \rangle$ and $Q' = \varphi(R)$, and computes $E_B = E_{A'}/\langle Q' \rangle$. Failure cases: if $Q = \infty$ then this method computes $Q' = R$ (which has order dividing ℓ_2) and $A' = A$, failing to compute $\mathcal{L}_1$; if $Q' = \infty$ then this method computes $B = A'$, failing to compute $\mathcal{L}_2$.

For concreteness, we compare the costs of these methods for CSIDH-512. The rest of this subsection uses approximations to the costs of lower-level operations to simplify the analysis. The main costs are as follows:

- For p a 512-bit prime, Elligator costs approximately $600\mathbf{M}$.
- Given $P \in E(\mathbb{F}_p)$ and a positive integer k, the computation of $[k]P$ via the Montgomery ladder, as described in Section 9.3.3, costs approximately $10(\log_2 k)\mathbf{M}$, i.e., approximately $(5120 - 10\log_2 \ell)\mathbf{M}$ if $k = (p+1)/\ell$.

- The computation of a degree-ℓ isogeny via the method described in Section 9.5.3 costs approximately $(3.5\ell + 2\log_2 \ell)\mathbf{M}$.
- Given an ℓ-isogeny $\varphi_\ell : E \to E'$ and $P \in E(\mathbb{F}_p)$, the computation of $\varphi_\ell(P)$ via the method described in Section 9.5.4 costs approximately $2\ell\mathbf{M}$.

In conclusion, Method 1 costs approximately

$$(2 \cdot 600 + 2 \cdot 5120 + 3.5\ell_1 + 3.5\ell_2 - 8\log_2 \ell_1 - 8\log_2 \ell_2)\mathbf{M},$$

while Method 2 costs approximately

$$(600 + 5120 + 5.5\ell_1 + 3.5\ell_2 - 8\log_2 \ell_1 + 2\log_2 \ell_2)\mathbf{M}.$$

The savings of $(600+5120)\mathbf{M}$ clearly outweighs the loss of $(2\ell_1+10\log_2 \ell_2)\mathbf{M}$, since the largest value of ℓ_i is 587.

There are limits to the applicability of Method 2: it cannot combine two isogenies of opposite signs, it cannot combine two isogenies using the same prime, and it cannot save time in applying just one isogeny. We will analyze the overall magnitude of these effects in Section 9.8.3.

9.8.2 – Handling the general case, two isogenies at a time. Algorithm 9.3 computes the result of $\mathcal{L}_1^{e_1} \cdots \mathcal{L}_n^{e_n}(A)$ for any exponent vector $(e_1, \ldots, e_n)$. Each iteration of the algorithm tries to perform two isogenies: one for the top nonzero exponent (if the vector is nonzero), and one for the next exponent having the same sign (if the vector has another exponent of this sign). As in Section 9.7, "top" refers to position, not value.

The algorithm pushes the first point through the first isogeny, as in Section 9.8.1, to save the cost of generating a second point. Scalar multiplication, isogeny computation, and isogeny application use the constant-time subroutines described in Sections 9.3.3, 9.5.3, and 9.5.4 respectively. The cost of these algorithms depends on the bound ℓ_n for the prime for the top nonzero exponent and the bound ℓ_{n-1} for the prime for the next exponent. The two prime bounds have asymmetric effects upon costs; we exploit this by applying the isogeny for the top nonzero exponent *after* the isogeny for the next exponent.

Analyzing the correctness of Algorithm 9.3 — assuming that there are enough iterations; see Section 9.8.3 — requires considering three cases. The first case is that the exponent vector is 0. Then i, i', s are initialized to $0, 0, 1$ respectively, and i, i' stay 0 throughout the iteration, so A does not change and the exponent vector does not change.

The second case is that the exponent vector is nonzero and the top nonzero exponent e_i is the only exponent having sign s. Then i' is 0 throughout the iteration, so the "first isogeny" portion of Algorithm 9.3 has no effect. The point $Q = R$ in the "second isogeny" portion is cP where $c = (p+1)/\ell_i$, so $\ell_i Q = \infty$. If $Q = \infty$ then i is set to 0 and the entire iteration has no effect, except for setting A to sA and then back to $s(sA) = A$. If $Q \neq \infty$ then i stays nonzero and A is replaced by $\mathcal{L}_i(A)$, so A at the end of the iteration is $\mathcal{L}_i^s$ applied to A at the beginning of the iteration, while s is subtracted from e_i.

The third case is that the exponent vector is nonzero and that $e_{i'}$ is the next exponent having the same sign s as the top nonzero exponent e_i. By construction $i' < i \leq n$ so $\ell_{i'} \leq \ell_{n-1}$. Now $R = cP$ where $c = (p+1)/(\ell_i \ell_{i'})$. The first isogeny uses the point $Q = \ell_i R$, which is either ∞ or a point of order $\ell_{i'}$. If Q is ∞ then i' is set to 0; both A and the vector are unchanged; the point R must have order dividing ℓ_i; and the second isogeny proceeds as above using this point. If Q has order $\ell_{i'}$ then the first isogeny replaces A with $\mathcal{L}_{i'}(A)$, while subtracting s from $e_{i'}$ and replacing R with a point of order dividing ℓ_i on the new curve (note that the $\ell_{i'}$-isogeny removes

Algorithm 9.3: Evaluating the class-group action by reducing the top nonzero exponent and the next exponent with the same sign.

Parameters: Odd primes $\ell_1 < \cdots < \ell_n$ with $n \geq 1$, a prime $p = 4\ell_1 \cdots \ell_n - 1$, and a positive integer r.

Input: $A \in S_p$, integers $(e_1, \ldots, e_n)$.

Output: $\mathfrak{l}_1^{e_1} \cdots \mathfrak{l}_n^{e_n}(A)$ or "fail".

for $j \leftarrow 1$ **to** r **do**
- Set $I \leftarrow \{k : 1 \leq k \leq n \text{ and } e_k \neq 0\}$.
- Set $i \leftarrow \max I$ and $s \leftarrow \operatorname{sign}(e_i) \in \{-1, 1\}$, or $i \leftarrow 0$ and $s \leftarrow 1$ if $I = \{\}$.
- Set $I' \leftarrow \{k : 1 \leq k < i \text{ and } \operatorname{sign}(e_k) = s\}$.
- Set $i' \leftarrow \max I'$, or $i' \leftarrow 0$ if $I' = \{\}$.
- **Twist.** Set $A \leftarrow sA$.
- **Isogeny preparation.** Find a random point P on E_A using Elligator.
- Compute $R \leftarrow cP$ where $c = 4 \prod_{1 \leq j \leq n, j \neq i, j \neq i'} \ell_j$.
- **First isogeny.** Compute $Q \leftarrow \ell_i R$, where ℓ_0 means 1.
- [Now $\ell_{i'} Q = \infty$ if $i' \neq 0$.] Set $i' \leftarrow 0$ if $Q = \infty$.
- Compute B with $E_B \cong E_A/\langle Q \rangle$ if $i' \neq 0$, using the $\ell_{i'}$-isogeny formulas from Section 9.5.3 with maximum degree ℓ_{n-1}.
- Set R to the image of R in E_B if $i' \neq 0$, using the $\ell_{i'}$-isogeny formulas from Section 9.5.4 with maximum degree ℓ_{n-1}.
- Set $A \leftarrow B$ and $e_{i'} \leftarrow e_{i'} - s$ if $i' \neq 0$.
- **Second isogeny.** Set $Q \leftarrow R$.
- [Now $\ell_i Q = \infty$ if $i \neq 0$.] Set $i \leftarrow 0$ if $Q = \infty$.
- Compute B with $E_B \cong E_A/\langle Q \rangle$ if $i \neq 0$, using the ℓ_i-isogeny formulas from Section 9.5.3 with maximum degree ℓ_n.
- Set $A \leftarrow B$ and $e_i \leftarrow e_i - s$ if $i \neq 0$.
- **Untwist.** Set $A \leftarrow sA$.

Set $A \leftarrow$ "fail" if $(e_1, \ldots, e_n) \neq (0, \ldots, 0)$.

Return A.

any $\ell_{i'}$ from orders of points in the same cyclic subgroup); again the second isogeny proceeds as above.

9.8.3 – Analysis of the failure probability. Consider a modified dual-isogeny algorithm in which the isogeny with a smaller prime is saved to handle later:

- Initialize an iteration counter to 0.
- Initialize an empty bank of positive isogenies.
- Initialize an empty bank of negative isogenies.
- For each ℓ in decreasing order:

– While an ℓ-isogeny needs to be done and the bank has an isogeny of the correct sign: Withdraw an isogeny from the bank, apply the isogeny, and adjust the exponent.
– While an ℓ-isogeny still needs to be done: Apply an isogeny, adjust the exponent, deposit an isogeny with the bank, and increase the iteration counter.

This uses more bit operations than Algorithm 9.3 (since the work here is not shared across two isogenies), but it has the same failure probability for the same number of iterations. We now focus on analyzing the distribution of the number of iterations used by this modified algorithm.

We use three variables to characterize the state of the modified algorithm before each ℓ:

- $i \geq 0$ is the iteration counter;
- $j \geq 0$ is the number of positive isogenies in the bank;
- $k \geq 0$ is the number of negative isogenies in the bank.

The number of isogenies actually applied so far is $2i - (j + k) \geq i$. The distribution of states is captured by the three-variable formal power series $\sum_{i,j,k} s_{i,j,k} x^i y^j z^k$ where $s_{i,j,k}$ is the probability of state (i, j, k). Note that there is no need to track which primes are paired with which; this is what makes the modified algorithm relatively easy to analyze.

If there are exactly h positive ℓ-isogenies to perform then the new state after those isogenies is $(i, j - h, k)$ if $h \leq j$, or $(i + h - j, h - j, k)$ if $h > j$. This can be viewed as a composition of two operations on the power series. First, multiply by y^{-h}. Second, replace any positive power of y^{-1} with the same power of xy; i.e., replace $x^i y^j z^k$ for each $j < 0$ with $x^{i-j} y^{-j} z^k$.

We actually have a distribution of the number of ℓ-isogenies to perform. Say there are h isogenies with probability q_h. We multiply the original series by $\sum_{h \geq 0} q_h y^{-h}$, and then eliminate negative powers of y as above. We similarly handle $h < 0$, exchanging the role of (j, y) with the role of (k, z).

As in the analyses earlier in the chapter, we model each point Q for an ℓ-isogeny as having order 1 with probability $1/\ell$ and order ℓ with probability $1 - 1/\ell$, and we assume that the number of ℓ-isogenies to perform is a uniform random integer $e \in \{-C, \ldots, C\}$. Then q_h for $h \geq 0$ is the coefficient of x^h in $\sum_{0 \leq e \leq C} (((\ell - 1)x)/(\ell - x))^e / (2C + 1)$; also, $q_{-h} = q_h$.

We reduce the time spent on these computations in three ways. First, we discard all states with $i > r$ if we are not interested in more than r iterations. This leaves a cubic number of states for each ℓ: every i between 0 and r inclusive, every j between 0 and i inclusive, and every k between 0 and $i - j$ inclusive.

Second, we use fixed-precision arithmetic, rounding each probability to an integer multiple of (e.g.) 2^{-512}. We round down to obtain lower bounds on success probabilities; we round up to obtain upper bounds on success probabilities; we choose the scale 2^{-512} so that these bounds are as tight as desired. We could save more time by reducing the precision slightly at each step of the computation, and by using standard interval-arithmetic techniques to merge computations of lower and upper bounds.

Third, to multiply the series $\sum_{i,j,k} s_{i,j,k} x^i y^j z^k$ by $\sum_{h \geq 0} q_h y^{-h}$, we instead actually multiply $\sum_j s_{i,j,k} y^j$ by $\sum_{h \geq 0} q_h y^{-h}$ for each (i, k) separately. We use Sage for these multiplications of univariate polynomials with integer coefficients. Sage, in turn, uses fast multiplication algorithms whose cost is essentially bd for d b-bit coefficients, so our total cost for n primes is essentially bnr^3.

Concretely, we used under two hours on one core of a 3.5GHz Intel Xeon E3-1275 v3 to compute lower bounds on all the success probabilities for CSIDH-512 with $b = 512$ and $r = 349$, and

under three hours[2] to compute upper bounds. Our convention of rounding failure probabilities to 3 digits makes the lower bounds and upper bounds identical, so presumably we could have used less precision.

We find, e.g., failure probability $0.943 \cdot 2^{-1}$ after 106 iterations, failure probability $0.855 \cdot 2^{-32}$ after 154 iterations, and failure probability $0.975 \cdot 2^{-257}$ after 307 iterations. Compared to the $207, 294, 468$ single-isogeny iterations required in Section 9.7.3, the number of iterations has decreased to 51.2%, 52.3%, 65.6% respectively.

9.8.4 – Analysis of the cost. We have fully implemented Algorithm 9.3 in our bit-operation simulator. An iteration of Algorithm 9.3 uses $4969644344 \approx 2^{32}$ nonlinear bit operations, about 1.306 times more expensive than an iteration of Algorithm 9.2.

If the number of iterations were multiplied by exactly 0.5 then the total cost would be multiplied by 0.653. Given the actual number of iterations (see Section 9.8.3), the cost is actually multiplied by $0.669, 0.684, 0.857$ respectively. In particular, we reach failure probability $0.855 \cdot 2^{-32}$ with $154 \cdot 4969644344 = 765325228976 \approx 0.7 \cdot 2^{40}$ nonlinear bit operations. This justifies the second (B, ϵ) claim in Section 9.1.

9.8.5 – Variants. The idea of pushing points through isogenies can be combined with the idea of gradually reducing the maximum prime allowed in the Matryoshka-doll isogeny formulas. This is compatible with our techniques for analyzing failure probabilities.

A dual-isogeny iteration very late in the computation is likely to have a useless second isogeny. It should be slightly better to replace some of the last dual-isogeny iterations with single-isogeny iterations. This is also compatible with our techniques for analyzing failure probabilities.

There are many different possible pairings of primes: one can take any two distinct positions where the exponents have the same sign. Possibilities include reducing exponents from the bottom rather than the top; reducing the top nonzero exponent and the bottom exponent with the same sign; always pairing "high" positions with "low" positions; always reducing the largest exponents in absolute value; always reducing e_i where $|e_i|\ell_i/(\ell_i - 1)$ is largest. For some of these ideas it is not clear how to efficiently analyze failure probabilities.

This section has focused on reusing an Elligator computation and large scalar multiplication for (in most cases) two isogeny computations, dividing the scalar-multiplication cost by (nearly) 2, in exchange for some overhead. We could push a point through more isogenies, although each extra isogeny has further overhead with less and less benefit, and computing the failure probability becomes more expensive. For comparison, [Cas+18] reuses one point for every ℓ_i where e_i has the same sign; the number of such ℓ_i is variable, and decreases as the computation continues. For small primes it might also save time to push multiple points through one isogeny, as in [DJP14].

9.9 — Computing ℓ-isogenies using division polynomials

As the target failure probability decreases, the algorithms earlier in this chapter spend more and more iterations handling the possibility of repeated failures for small primes ℓ — especially $\ell = 3$, where each generated point fails with probability $1/3$.

[2] It is unsurprising that lower bounds are faster: many coefficients q_h round down to 0. We could save time in the upper bounds by checking for stretches of coefficients that round up to, e.g., $1/2^{512}$, and using additions to multiply by those stretches.

Algorithm 9.4: ℓ-isogeny using division polynomials.

Parameters: Odd primes $\ell_1 < \cdots < \ell_n$ with $n \geq 1$, a prime $p = 4\ell_1 \cdots \ell_n - 1$, and $\ell \in \{\ell_1, \ldots, \ell_n\}$.

Input: $A \in S_p$.

Output: $\mathcal{L}(A)$.

1 Compute the ℓ-division polynomial $\psi_\ell \in \mathbb{F}_p[X]$ of E_A.

2 Compute $\psi'_\ell = \gcd(X^p - X, \psi_\ell)$.

3 Let $\rho = X^3 + AX^2 + X$ and compute $\chi_\ell = \gcd(\rho^{(p+1)/2} - \rho, \psi'_\ell)$.

4 Use Lemma 9.1 on χ_ℓ to compute B such that $E_B \cong E_A/E_A(\mathbb{F}_p)[\ell]$.

5 **Return** B.

This section presents and analyzes an alternative: a deterministic constant-time subroutine that uses division polynomials to *always* compute ℓ-isogenies. Using division polynomials is more expensive than generating random points, and the cost gap grows rapidly as ℓ increases, but division polynomials have the advantage that each iteration is guaranteed to compute an ℓ-isogeny. See also Section 9.10 for an alternative that uses modular polynomials rather than division polynomials.

Division polynomials can be applied as a first step to any of our class-group evaluation algorithms: compute the group action for some number of powers of $\mathcal{L}_1^{\pm 1}, \ldots, \mathcal{L}_s^{\pm 1}$ (not necessarily C powers of each), and then handle the remaining isogenies as before. Our rough estimates in this section suggest that the optimal choice of s is small: division polynomials are not of interest for large primes ℓ.

9.9.1 – Algorithm. The idea behind the following algorithm is to take the ℓ-division polynomial ψ_ℓ of E_A, whose roots are the x-coordinates of nonzero ℓ-torsion points; identify a divisor χ_ℓ of ψ_ℓ that defines the $\mathbb{F}_p$-rational subgroup of $E_A[\ell]$; and finally use a variant of Kohel's formulas [Koh96, Section 2.4] to compute the codomain of the isogeny defined by χ_ℓ and thus $B = \mathcal{L}(A)$.

Lemma 9.1. *Let $E_A \colon y^2 = x^3 + Ax^2 + x$ be a Montgomery curve defined over a field k with $\operatorname{char}(k) \neq 2$. Consider a finite subgroup $G \leq E$ of odd size $n \geq 3$ and let $\chi \in k[x]$ be a monic squarefree polynomial of degree $d = (n-1)/2$ whose roots are exactly the x-coordinates of all nonzero points in G. Write*

$$\sigma = -\chi[d-1]\,; \quad \tau = (-1)^{d+1} \cdot \chi[1]\,; \quad \pi = (-1)^d \cdot \chi[0]\,,$$

where $\chi[i] \in k$ is the coefficient of x^i in χ. Then there exists an isogeny $E_A \to E_B$ with kernel G, where

$$B = \pi(\pi(A - 6\sigma) + 6\tau)\,.$$

Proof. This is obtained by decomposing the formulas from [Ren18] into elementary symmetric polynomials, which happen to occur as the given coefficients of χ. □

Lemma 9.2. *Algorithm 9.4 is correct.*

Proof. First, $X^p - X = \prod_{a \in \mathbb{F}_p}(X - a)$ implies that ψ'_ℓ is the part of ψ_ℓ that splits into linear factors over $\mathbb{F}_p$. Second, for any $\rho \in \mathbb{F}_p$, choosing $y \in \overline{\mathbb{F}_p}$ such that $y^2 = \rho$ gives

$$\rho^{(p+1)/2} - \rho = y(y^p - y) = y \prod_{\alpha \in \mathbb{F}_p} (y - \alpha).$$

Therefore the roots of χ_ℓ are exactly the x-coordinates of the nonzero $\mathbb{F}_p$-rational ℓ-torsion points on E. Finally, the correctness of the output follows from Lemma 9.1. □

9.9.2 – Cost. To analyze how this approach compares to Vélu's formulas, we focus on rough estimates of how cost scales with ℓ, rather than an exact cost analysis. Finite field squarings $\mathbf{S}$ are counted as $\mathbf{M}$ for simplicity. Let $\mu(d)$ denote the cost of multiplying two d-coefficient polynomials. To establish a rough lower bound, we assume $\mu(d) = (d \log_2 d)\mathbf{M}$, which is a model of the complexity of FFT-based fast multiplication techniques. For a rough upper bound, we use $d^2\mathbf{M}$, which is a model of the cost of schoolbook multiplication.

Computing division polynomials. There are two obvious methods for obtaining division polynomials: Either evaluate the recursive definition directly on a given $A \in \mathbb{F}_p$, or precompute the division polynomials as elements of $\mathbb{F}_p[A, x]$ in advance and evaluate them at a given $A \in \mathbb{F}_p$ at runtime. We estimate the number of operations required for both approaches.

Recursive definition. Ignoring multiplications by small fixed polynomials, the division polynomials satisfy a recursive equation of the form

$$f_\ell = f_a f_b f_c^2 - f_{a'} f_{b'} f_{c'}^2,$$

where the indices a, b, c, a', b', c' are integers within 2 of $\ell/2$ (so there are at most 5 distinct indices). Continuing this recursion involves indices within $2/1 + 2 = 3$ of $\ell/4$ (at most 7 distinct indices), within 3.5 of $\ell/8$ (at most 8), within 3.75 of $\ell/16$ (at most 8), etc.

Each of f_a, f_b, f_c has approximately $\ell^2/8$ coefficients, so computing $f_a f_b f_c^2$ costs $(2\mu(\ell^2/8) + \mu(\ell^2/4))\mathbf{M}$. The rough lower bound is $(\ell^2 \log_2 \ell)\mathbf{M}$, and the rough upper bound is $(3\ell^4/32)\mathbf{M}$.

Computing f_ℓ involves computing both $f_a f_b f_c^2$ and $f_{a'} f_{b'} f_{c'}^2$. The recursion involves at most 5 computations for $\ell/2$, at most 7 computations for $\ell/4$, and at most 8 computations for each subsequent level. The total is

$$(1 + 5/2 + 7/4 + 8/8 + 8/16 + \cdots)(2\ell^2 \log_2 \ell)\mathbf{M} = (29/2)(\ell^2 \log_2 \ell)\mathbf{M}$$

for the rough lower bound, and

$$(1 + 5/4 + 7/16 + 8/64 + 8/256 + \cdots)(3\ell^4/16)\mathbf{M} = (137/256)\ell^4\mathbf{M}$$

for the rough upper bound.

Evaluating precomputed polynomials. The degree of $\Psi_\ell \in \mathbb{F}_p[A, x]$ is $(\ell^2 - 1)/2$ in x and upper bounded by $\ell^2/8 + 1$ in A, so overall Ψ_ℓ has at most about $\ell^4/16$ coefficients. Evaluating a precomputed $\Psi_\ell \in \mathbb{F}_p[x][A]$ at $A \in \mathbb{F}_p$ using Horner's method takes at most about $(\ell^4/16)\mathbf{M}$. This improves the rough upper bound.

Extracting the split part. As stated in Algorithm 9.4, extracting the part ψ'_ℓ of ψ_ℓ that splits over $\mathbb{F}_p$ amounts to computing $\gcd(X^p - X, \psi_\ell)$. The exponentiation $X^p \bmod \psi_\ell$ is computed using square-and-multiply with windows (similar to Section 9.13.5), which uses about $\log_2 p$

Table 9.4: Rough estimates for the number of $\mathbb{F}_p$-multiplications to compute $\mathcal{L}(A)$ using division polynomials (Algorithm 9.4).

ℓ	3	5	7	11	13	17	19	23	29
rough upper bound	$2^{15.1}$	$2^{17.8}$	$2^{19.6}$	$2^{22.1}$	$2^{23.1}$	$2^{24.6}$	$2^{25.3}$	$2^{26.4}$	$2^{27.7}$
rough lower bound	$2^{14.5}$	$2^{16.4}$	$2^{17.6}$	$2^{19.1}$	$2^{19.7}$	$2^{20.6}$	$2^{20.9}$	$2^{21.6}$	$2^{22.3}$

squarings and about $(\log_2 p)/(\log_2 \log_2 p)$ multiplications. For simplicity we count this as a total of $1.2 \log_2 p$ multiplications, which is a reasonable estimate for 512-bit p.

For the number of $\mathbb{F}_p$-multiplications needed to compute $X^p \bmod \psi_\ell$, we obtain approximately $2.4 \log_2 p \cdot \ell^2 \log \ell$ for the lower bound on $\mu(d)$ and $0.6 \log_2 p \cdot \ell^4$ for the upper. Here we assume cost $\mu(d)$ for reducing a degree-$(2d-2)$ polynomial modulo a degree-d polynomial.

The computation of $\gcd\big((X^p \bmod \psi_\ell) - X, \psi_\ell\big)$ can be done using Stevin's algorithm which uses roughly $d^2\mathbf{M}$, where d is the degree of the larger of the input polynomials. In this case, since $\deg(\psi_\ell) \approx \ell^2/2$, this amounts to about $(\ell^4/4)\mathbf{M}$.

Fast arithmetic improves gcd computation to $O(d \cdot (\log_2 d)^2)\mathbf{M}$ asymptotically; see, e.g., [Str83]. We are not aware of literature presenting concrete speeds for fast constant-time gcd computation. For a rough lower bound we assume $2d(\log_2 d)^2\mathbf{M}$, i.e., about $4\ell^2(\log_2 \ell)^2\mathbf{M}$.

Extracting the kernel polynomial. Note that $\deg(\psi'_\ell) = \ell - 1$: Each root in $\mathbb{F}_p$ of ψ_ℓ gives rise to two points of order ℓ in the $+1$ or -1 Frobenius eigenspace, which contain $\ell - 1$ nonzero points each. Hence, as before, the cost of obtaining χ_ℓ from ψ'_ℓ is roughly $(2.4 \log_2 p \cdot \ell \log_2 \ell)\mathbf{M}$ resp. $2.4 \log_2 p \cdot \ell^2\mathbf{M}$ for the exponentiation, plus $2\ell(\log_2 \ell)^2\mathbf{M}$ resp. $\ell^2\mathbf{M}$ for the gcd computation.

Computing the isogeny. Lemma 9.1 is just a simple formula in terms of a few coefficients of χ_ℓ and can be realized using $2\mathbf{M}$ and some additions, hence has negligible cost.

9.9.3 – Total cost. In summary, the cost of Algorithm 9.4 in $\mathbb{F}_p$-multiplications has a rough lower bound of

$$\min\big\{(29/2)\ell^2 \log_2 \ell, \ell^4/16\big\} + 2.4 \log_2 p \cdot \ell^2 \log_2 \ell + 4\ell^2(\log_2 \ell)^2 + 2.4 \log_2 p \cdot \ell \log_2 \ell + 2\ell(\log_2 \ell)^2$$

and a rough upper bound of

$$\ell^4/16 + 0.6 \log_2 p \cdot \ell^4 + \ell^4/4 + 2.4 \log_2 p \cdot \ell^2 + \ell^2.$$

Table 9.4 lists values of these formulas for $\log_2 p \approx 512$ and small ℓ.

The main bottleneck is the computation of $X^p \bmod \psi_\ell$: for each bit of p there is a squaring modulo ψ_ℓ, a polynomial of degree $(\ell^2 - 1)/2$. For comparison, the scalar multiplication in Section 9.5 involves about $10\mathbf{M}$ for each bit of p, no matter how large ℓ is, but is not guaranteed to produce a point of order ℓ.

9.10 — Computing ℓ-isogenies using modular polynomials

One technique suggested by De Feo, Kieffer, and Smith [Kie17; DKS18] to compute the CRS group action is to use the (classical) *modular polynomials* $\Phi_\ell(X, Y)$, which vanish exactly on the pairs of j-invariants that are connected by a cyclic ℓ-isogeny. For prime ℓ, the polynomial $\Phi_\ell(X, Y)$ is

symmetric and has degree $\ell+1$ in the two variables, hence fixing one of the variables to some j-invariant and finding the roots of the resulting univariate polynomial suffices to find neighbours in the ℓ-isogeny graph.

The advantage of modular polynomials over division polynomials is that the degree $\ell+1$ of $\Phi_\ell(j,Y)$ grows more slowly than the degree $(\ell^2-1)/2$ of the ℓ-division polynomial ψ_ℓ used in Section 9.9: modular polynomials are smaller for all $\ell \geq 5$. However, using modular polynomials requires solving two problems: disambiguating twists and disambiguating directions. We address these problems in the rest of this section.

9.10.1 – Disambiguating twists. It may seem that computing ℓ-isogenous curves by finding roots of $\Phi_\ell(X,Y)$ is not applicable to the CSIDH setting, since a single j-invariant almost always defines *two* distinct nodes in the supersingular $\mathbb{F}_p$-rational isogeny graph, namely E_B and E_{-B} for some $B \in S_p$. Knowing $j(E_{\mathcal{L}(A)})$ is not enough information to distinguish $\mathcal{L}(A)$ from $-\mathcal{L}(A)$.

This problem does not arise in CRS: Twists always have the same j-invariant but, in the ordinary case, are not isogenous. A random twist point has negligible chance of being annihilated by the expected group order, so one can reliably recognize the twist at the expense of a scalar multiplication.

For CSIDH, one way to distinguish the cases $\mathcal{L}(A) = B$ and $\mathcal{L}(A) = -B$ is to apply a different isogeny-computation method (from, e.g., Section 9.5 or Section 9.9) to compute $\mathcal{L}(B)$. If $\mathcal{L}(B) = -A$ then $\mathcal{L}(A) = -B$; otherwise $\mathcal{L}(A) = B$.

This might seem to remove any possible advantage of having used modular polynomials to compute $\pm B$ in the first place, since one could simply have used the different method to compute $\mathcal{L}(A)$. However, below we will generalize the same idea to $\mathcal{L}^e(A)$, amortizing the costs of the different method across the costs of e computations using modular polynomials.

An alternative is as follows. The Bostan–Morain–Salvy–Schost algorithm [BMSS08], given a curve C (in short Weierstrass form, but the algorithm is easily adjusted to apply to Montgomery curves) and an ℓ-isogenous curve C', finds a formula for the unique normalized ℓ-isogeny from C to C'. Part of this formula is the kernel polynomial of the isogeny: the monic degree-$(\ell-1)$ polynomial $D \in \mathbb{F}_p[X]$ whose roots are the x-coordinates of the nonzero elements of the kernel of the isogeny. The algorithm uses $\ell^{1+o(1)}$ field operations with fast multiplication techniques. The output of the algorithm can be efficiently verified to be an ℓ-isogeny from C to C', so the algorithm can also be used to test whether two curves are ℓ-isogenous.

Use this algorithm to test whether there is an ℓ-isogeny from E_A to E_B, and, if so, to find the kernel polynomial D. Check whether D divides $X^p - X$, i.e., whether all of the nonzero elements of the kernel have x-coordinates defined over $\mathbb{F}_p$; this takes one exponentiation modulo D. Also check whether D divides $(X^3+AX^2+X)^{(p+1)/2} - (X^3+AX^2+X)$, i.e., whether all of the nonzero elements of the kernel have y-coordinates defined over $\mathbb{F}_p$. These tests are all passed if and only if $B = \mathcal{L}(A)$.

Both of these approaches also incur the cost of computing $B \in \mathbb{F}_p$ given $j(E_B)$, which we handle as follows. First note that there are at most two such B: different Montgomery models of the same curve arise from the choice of point of order 2 which is moved to $(0,0)$; in our setting, there is only one rational order-2 point, hence B is unique up to sign. The j-invariant of E_B is an even rational function of degree 6 in B, hence solving for $B \in \mathbb{F}_p$ given $j(E_B)$ amounts to finding the $\mathbb{F}_p$-roots of a degree-6 polynomial $g \in \mathbb{F}_p[Y^2]$. To do so, we first compute $h = \gcd(Y^p - Y, g)$ to extract the split part; by the above h is a quadratic polynomial. A solution $B \in \mathbb{F}_p$ can then be obtained by computing a square root.

We also mention a further possibility that appears to eliminate all of the costs above: replace the classical modular polynomials for j with modular polynomials for the Montgomery coefficient A. Starting with standard techniques to compute classical modular polynomials, and replacing j with A, appears to produce, at the same speed, polynomials that vanish exactly on the pairs (A, B) where E_A and E_B are connected by a cyclic ℓ-isogeny. The main cost here is in proof complexity: to guarantee that this approach works, one must switch from the well-known theory of classical modular polynomials to a suitable theory of Montgomery (or Edwards) modular polynomials.

9.10.2 – Disambiguating directions. A further problem is that each curve has two neighbors in the ℓ-isogeny graph. The modular polynomial does not contain enough information to distinguish between the two neighbours. Specifically, the roots of $\Phi_\ell(j(E_A), Y)$ in $\mathbb{F}_p$ are $j(E_{\mathcal{L}(A)})$ and $j(E_{\mathcal{L}^{-1}(A)})$, which are almost always different. Switching from j-invariants to other geometric invariants does not solve this problem.

This is already a problem for CRS, and is already solved in [Kie17; DKS18] using the Bostan–Morain–Salvy–Schost algorithm. The application of this algorithm in the CRS context is slightly simpler than the application explained above, since there is no need for isogeny verification: one knows that E_B is ℓ-isogenous to E_A, and the only question is whether the kernel is in the correct Frobenius eigenspace. For CSIDH, the question is whether the y-coordinates in the kernel are defined over $\mathbb{F}_p$.

9.10.3 – Isogeny walks. We now consider the problem of computing $\mathcal{L}^e(A)$. As before, $\mathcal{L}^{-e}(A)$ can be computed as $-\mathcal{L}^e(-A)$, so we focus on the case $e > 0$.

After the first step $\mathcal{L}(A)$ has been computed (see above), identifying the correct direction in each subsequent step is easy, as pointed out in [DKS18, Algorithm ElkiesWalk]. The point is that (except for degenerate cases) another step in the same direction never leads back to the previously visited curve; hence simply avoiding backward steps is enough. The cost of disambiguating directions is thus amortized across all e steps.

We also amortize the cost of disambiguating twists across all e steps as follows. We ascertain the correct direction at the first step. We then compute the sequence of j-invariants for all e steps. At the last step, we compute the corresponding Montgomery coefficient and ascertain the correct twist.

Algorithm 9.5 combines these ideas. For simplicity, it avoids the Bostan–Morain–Salvy–Schost algorithm and uses another isogeny-computation method instead, such as Algorithm 9.4, to disambiguate the direction at the first step and to disambiguate the twist at the last step.

The correctness of Algorithm 9.5 is best explained through the graph picture: Recall that the ℓ-isogeny graph (labelled by A-coefficients) is a disjoint union of cycles which have a natural orientation given by the map $\mathcal{L}$.

Since $-\mathcal{L}(-A) = \mathcal{L}^{-1}(A)$, negating all labels in a cycle $\mathcal{C}$ corresponds to inverting the orientation of the cycle. For a cycle $\mathcal{C}$ as above, let $\mathcal{C}/\pm$ denote the quotient graph of $\mathcal{C}$ by negation. This is the same thing as applying j-invariants to all nodes. If $\mathcal{C}$ contains 0, then $\mathcal{C}/\pm$ is a line with inflection points at the ends; else $\mathcal{C}/\pm$ has the same structure as $\mathcal{C}$. In both cases $\mathcal{C}/\pm$ is unoriented.

For brevity, write $j_i = j(E_{\mathcal{L}^i(A)})$. Algorithm 9.5 starts out on a cycle $\mathcal{C}$ as above by computing one step $\mathcal{L}$ with known-good orientation. It then reduces to $\mathcal{C}/\pm$ and continues walking in the same direction simply by avoiding backwards steps when possible; there are only (up to) two neighbours at all times. Therefore, the property $j_{cur} = j_i$ holds at the end of each iter-

Algorithm 9.5: Isogeny graph walking using modular polynomials.

Parameters: Odd primes $\ell_1 < \cdots < \ell_n$ with $n \geq 1$, a prime $p = 4\ell_1 \cdots \ell_n - 1$, $\ell \in \{\ell_1, \ldots, \ell_n\}$, and an integer $e \geq 1$.

Input: $A \in S_p$.

Output: $\mathcal{L}^e(A)$.

Compute $B = \mathcal{L}(A)$ using another algorithm.
Set $j_{prev} = j(E_A)$ and $j_{cur} = j(E_B)$.
for $i \leftarrow 2$ **to** e **do**
    Compute $f \leftarrow \gcd(Y^p - Y, \Phi_\ell(j_{cur}, Y))$.
    Let $c, d \in \mathbb{F}_p$ be the coefficients of f, such that $f = Y^2 + cY + d$.
    Set $(j_{prev}, j_{cur}) \leftarrow (j_{cur}, c - j_{prev})$.
Find $B \in \mathbb{F}_p$ such that $j(E_B) = j_{cur}$.
Compute $C = \mathcal{L}(B)$ using another algorithm.
Set $B \leftarrow -B$ if $j(E_C) = j_{prev}$.
Return B.

ation of the loop; in particular, arbitrarily lifting j_e to a node with the right j-invariant yields $B \in \{\pm\mathcal{L}^e(A)\}$. Finally, computing and comparing $j(E_{\mathcal{L}(B)}) = j(E_{\mathcal{L}(\pm\mathcal{L}^e(A))}) = j_{e\pm1}$ to the value j_{e-1} known from the previous iteration of the loop reveals the correct sign.

9.10.4 – Cost. Algorithm 9.5 requires two calls to a separate subroutine for $\mathcal{L}$ and some extra work (computing a p^{th} power modulo a degree-6 polynomial), so it is never faster than repeated applications of the separate subroutine when $e \leq 2$. On the other hand, replacing this subroutine with the Bostan–Morain–Salvy–Schost algorithm, and/or replacing classical modular polynomials with modular polynomials for A, might make this approach competitive for $e = 2$ and perhaps even $e = 1$.

No matter how large e is, Algorithm 9.5 requires computing the polynomials $\gcd(Y^p - Y, g)$ and $\gcd(Y^p - Y, \Phi_\ell(j_{cur}, Y))$ for each isogeny. The degree of g is smaller than in Algorithm 9.4 for $\ell \geq 5$, but the gcd cost quickly becomes much more expensive than the "Vélu" method from Section 9.5 as ℓ grows. However, this algorithm may nevertheless be of interest for small values of ℓ. If Algorithm 9.4 (rather than the Vélu method) is used as the separate $\mathcal{L}$ subroutine then Algorithm 9.5 is deterministic and always works.

9.11 — Cost metrics for quantum computation

This section reviews several cost metrics relevant to this chapter.

9.11.1 – Bit operations. Computations on today's non-quantum computers are ultimately nothing more than sequences of bit operations. The hardware carries out a sequence of NOT gates $b \mapsto 1 \oplus b$; AND gates $(a, b) \mapsto ab = \min\{a, b\}$; OR gates $(a, b) \mapsto \max\{a, b\}$; and XOR gates $(a, b) \mapsto a \oplus b$. Some of the results are displayed as outputs.

Formally, a computation is a finite directed acyclic graph where each node has 0, 1, or 2 inputs. Each 0-input node in the graph is labeled as constant 0, constant 1, or a specified input bit. Each 1-input node in the graph is labeled NOT. Each 2-input node in the graph is labeled AND,

OR, or XOR. There is also a labeling of output bits as particular nodes in the graph.

The graph induces a function from sequences of input bits to sequences of output bits. Specifically, given values of the input bits, the graph assigns a value to each node as specified by the label (e.g., the value at an AND node is the minimum of the values of its two input nodes), and in particular computes values of the output bits.

Our primary cost metric in this chapter is the number of **nonlinear bit operations**: i.e., we count the number of ANDs and ORs, disregarding the number of NOTs and XORs (and 0s and 1s). The advantage of choosing this cost metric is comparability to the Toffoli cost metric used in, e.g., [HRS17] and [RNSL17], which in turn is motivated by current estimates of the costs of various quantum operations, as we explain below.

A potential disadvantage of choosing this cost metric is that the cost metric can hide arbitrarily large sequences of linear operations. For example, there are known algorithms to multiply n-coefficient polynomials in $\mathbb{F}_2[x]$ using $\Theta(n)$ nonlinear operations (see, e.g., [PR15]), but this operation count hides $\Theta(n^2)$ linear operations. Other algorithms using $n(\log n)^{1+o(1)}$ total bit operations (see, e.g., [Sch77] and [HHL17]) are much better when n is large, even though they have many more nonlinear bit operations.

This seems to be less of an issue for integer arithmetic than for polynomial arithmetic. Adding nonzero costs for NOT and XOR requires a reevaluation of, e.g., the quantitative cutoff between schoolbook multiplication and Karatsuba multiplication, but does not seem to have broader qualitative impacts on the speedups that we consider in this chapter. Similarly, our techniques can easily be adapted to, e.g., a cost metric that allows NAND gates with lower cost than AND gates, reflecting the reality of computer hardware.

9.11.2 – The importance of constant-time computations. One can object to the simple model of computation explained above as not allowing variable-time computations. The graph uses a constant number of bit operations to produce its outputs, whereas real users often wait input-dependent amounts of time for the results of a computation. If a particular input is processed faster than the worst case, then the time saved can be spent on other useful computations.

However, our primary goal in this chapter is to evaluate the cost of carrying out a CSIDH group action on a huge number of inputs in quantum superposition. Operations are carried out on all of the inputs simultaneously, and then a measurement retroactively selects a particular input. The cost depends on the number of operations carried out on all inputs, not on the number of operations that in retrospect could have been carried out for the selected input.

The same structure has an impact at every level of algorithm design. In conventional algorithm design, if a function calls subroutine X for some inputs and subroutine Y for other inputs, then the cost of the function is the *maximum* of the costs of X and Y. However, a computation graph does not allow this branching. One must instead compute a suitable combination such as

$$bX(\text{inputs}) + (1 \oplus b)Y(\text{inputs}),$$

taking the *total* time for X and Y, or search for ways to overlap portions of the computations of X and Y.

One can provide branches as a higher-level abstraction by building a computation graph that manipulates an input-dependent pointer into an array of instructions, imitating the way that CPU hardware is built. It is important to realize, however, that the number of bit operations required to read an instruction from a variable location in an array grows with the size of the array, so the total number of bit operations in this approach grows much more rapidly than the

number of instructions. A closer look at what actually needs to be computed drastically reduces the number of bit operations.

The speedup techniques considered in this chapter can also be used in constant-time non-quantum software and hardware for CSIDH, reducing the cost of protecting CSIDH users against timing attacks. However, our main focus is the quantum case.

9.11.3 – Reversible bit operations. Bits cannot be erased or copied inside a quantum computation. For example, one cannot simply compute a XOR gate, replacing (a, b) with $a \oplus b$, or an AND gate, replacing (a, b) with ab. However, one can compute a "CNOT" gate, replacing (a, b) with $(a, a \oplus b)$; or a "Toffoli" gate, replacing (a, b, c) with $(a, b, c \oplus ab)$.

In general, an *n-bit reversible computation* begins with a list of n input bits, and then applies a sequence of NOT gates, CNOT gates, and Toffoli gates to specified positions in the list, eventually producing n output bits. Each of these gates is its own inverse, so one can map output back to input by applying the same gates in the reverse order.

Bennett's conversion (see [Ben73], which handles the more complicated case of Turing machines) is a generic transformation from computations, as defined in Section 9.11.1, to reversible computations. Say the original computation maps $x \in \{0,1\}^k$ to $F(x) \in \{0,1\}^\ell$. The reversible computation then maps $(x, y, 0) \in \{0,1\}^{k+\ell+m}$ to $(x, y \oplus F(x), 0) \in \{0,1\}^{k+\ell+m}$, for some choice of m that will be clear in a moment; the m auxiliary zero bits are called *ancillas*. The effect of the reversible computation upon more general inputs $(x, y, z) \in \{0,1\}^{k+\ell+m}$ is more complicated, and usually irrelevant.

For each AND gate $(a, b) \mapsto ab$ in the original computation, the reversible computation allocates an ancilla c and performs $(a, b, c) \mapsto (a, b, c \oplus ab)$ as a Toffoli gate. Note that if the ancilla c begins as 0 then this Toffoli gate produces the desired bit ab. More generally, for each gate in the original computation, the reversible computation allocates an ancilla c and operates reversibly on this ancilla, in such a way that if the ancilla begins with 0 then it ends with the same bit computed by the original gate. For example:

- For each constant-1 gate $() \mapsto 1$, the reversible computation allocates an ancilla c and performs a NOT gate $c \mapsto 1 - c$.
- For each NOT gate $b \mapsto 1 \oplus b$, the reversible computation allocates an ancilla c and performs $(b, c) \mapsto (b, c \oplus 1 \oplus b)$ as a NOT gate and a CNOT gate.
- For each XOR gate $(a, b) \mapsto a \oplus b$, the reversible computation allocates an ancilla c and performs $(a, b, c) \mapsto (a, b, c \oplus a \oplus b)$ as two CNOT gates.

The reversible computation thus maps $(x, y, 0)$ to (x, y, z) where z is the entire sequence of bits in the original computation, including all intermediate results. In particular, z includes the bits of $F(x)$, and ℓ additional CNOT gates produce $(x, y \oplus F(x), z)$. Finally, re-running the computation of z in reverse order has the effect of "uncomputing" z, producing $(x, y \oplus F(x), 0)$ as claimed.

The number of Toffoli gates here is exactly twice the number of nonlinear bit operations in the original computation: once in computing z and once in uncomputing z. There is a larger expansion in the number of NOT and CNOT gates compared to the original number of linear bit operations, but, as mentioned earlier, we focus on nonlinear bit operations.

Sometimes these overheads can be reduced. For example, if the original computation is simply an AND $(a, b) \mapsto ab$, then the reversible computation stated above uses two Toffoli gates and one ancilla —

- $(a, b, y, 0) \mapsto (a, b, y, ab)$ with a Toffoli gate,
- $(a, b, y, ab) \mapsto (a, b, y \oplus ab, ab)$ with a CNOT gate,

- $(a, b, y \oplus ab, ab) \mapsto (a, b, y \oplus ab, 0)$ with another Toffoli gate,

— but it is better to simply compute $(a, b, y) \mapsto (a, b, y \oplus ab)$ with one Toffoli gate and no ancillas. We do not claim that the optimal number of bit operations is a perfect predictor of the optimal number of Toffoli gates; we simply use the fact that the ratio is between 1 and 2.

Note that Bennett's reversible computation operates on an n-bit state where $n = k + \ell + m$ is essentially the number of *bit operations* in the original computation. Perhaps the original computation can fit into a much smaller state (depending on the order of operations, something not expressed by the computation graph), but this often relies on erasing intermediate results, which a reversible computation cannot do. Even in a world where arbitrarily large quantum computers can be built, this number n has an important impact on the cost of the corresponding quantum computation, so it becomes important to consider ways to reduce n, as explained in Section 9.11.5.

9.11.4 – T-gates. The state of n qubits is, by definition, a nonzero element $(v_0, v_1, \ldots)$ of the vector space $\mathbb{C}^{2^n}$. **Measuring** these n qubits produces an n-bit index $i \in \{0, 1, \ldots, 2^n - 1\}$, while modifying the vector to have 1 at position i and 0 elsewhere. The chance of obtaining i is proportional to $|v_i|^2$. One can, if desired, normalize the vectors so that $\sum_i |v_i|^2 = 1$.

An ***n*-qubit quantum computation** applies a sequence of NOT (often written "X"), CNOT, Hadamard ("H"), T, and T^{-1} gates to specified positions within n qubits. There is a standard representation of these gates as the matrices

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \begin{pmatrix} 1 & 0 \\ 0 & \exp(i\pi/4) \end{pmatrix}, \begin{pmatrix} 1 & 0 \\ 0 & \exp(-i\pi/4) \end{pmatrix}$$

respectively; if vectors are normalized then the Hadamard matrix is divided by $\sqrt{2}$. There is also a standard way to interpret these matrices as acting upon vectors in $\mathbb{C}^{2^n}$. For example, applying the NOT gate to qubit 0 of $(v_0, v_1, v_2, v_3, \ldots)$ produces $(v_1, v_0, v_3, v_2, \ldots)$; measuring after the NOT has the same effect as measuring before the NOT and then complementing bit 0 of the result. Applying the NOT gate to qubit 1 of $(v_0, v_1, v_2, v_3, \ldots)$ produces $(v_2, v_3, v_0, v_1, \ldots)$.

The consensus of quantum-computer engineers appears to be that "Clifford operations" such as NOT, CNOT, H, T^2, and T^{-2} are at least two orders of magnitude less expensive than T and T^{-1}. It is thus common practice to allow T^2 ("S" or "P") and T^{-2} as further gates, and to count the number of T and T^{-1}, while disregarding the number of NOT, CNOT, H, T^2, and T^{-2}. The total number of T and T^{-1} is, by definition, the number of *T-gates*.

There is a standard conversion from an n-bit reversible computation to an n-qubit quantum computation. NOT is converted to NOT; CNOT is converted to CNOT; Toffoli is converted to a sequence of 7 T-gates and some Clifford gates. Multiplying 7 by an upper bound on the number of Toffoli gates thus produces an upper bound on the number of T-gates.

As in Section 9.11.3, these overheads can sometimes be reduced. For example:

- All of the quantum gates mentioned here operate on one or two qubits at a time. The intermediate results in a Toffoli computation can often be reused for other computations.
- In a more sophisticated model of quantum computation that allows internal measurements, Jones [Jon12] showed how to implement a Toffoli gate as 4 T-gates, some Clifford gates, and a measurement. We follow [GLRS16] in mentioning but disregarding this alternative.

- In the same model, a recent paper by Gidney [Gid17] showed how to implement n-bit integer addition using about $4n$ T-gates (and a similar number of Clifford gates and measurements). For comparison, a standard "ripple carry" adder uses about $2n$ nonlinear bit operations.

As before, we do not claim that the optimal number of Toffoli gates is a perfect predictor of the number of T-gates; we simply use the fact that the ratio is between 1 and 7.

9.11.5 – Error-correction steps; the importance of parallelism. To recap: Our primary focus is producing an upper bound on the number of nonlinear bit operations. Multiplying by 2 gives an upper bound on the number of Toffoli gates for a reversible computation, and multiplying this second upper bound by 7 gives an upper bound on the number of T-gates for a quantum computation. Linear bit operations (and the corresponding reversible and quantum gates) do not seem to be a bottleneck for the types of computations considered in this chapter.

There is, however, a much more important bottleneck that is ignored in these cost metrics: namely, fault-tolerance seems to require continual error correction of every stored qubit.

Surface codes [FMMC12] are the leading candidates for fault-tolerant quantum computation. A logical qubit is encoded in a particular way as many entangled physical qubits spread over a surface. *Some* of the physical qubits are continually measured, and operations are carried out on the physical qubits to correct any errors revealed by the measurements. The consensus of the literature appears to be that performing a computation on a logical qubit will be only a small constant factor more expensive than storing an idle logical qubit.

One consequence of this structure is that all fault-tolerant quantum computations involve entanglement throughout the entire computation, contrary to the claim in [BS18] that a particular quantum algorithm "does not need to have a highly entangled memory for a long time".

Another consequence of this structure is that the cost of a quantum computation can grow *quadratically* with the number of bit operations. Consider, for example, an n-bit ripple-carry adder, or the adder from [Gid17]. This computation involves $\Theta(n)$ sequential bit operations and finishes in time $\Theta(n)$. Each of the $\Theta(n)$ qubits needs active error correction at each time step, for a total of $\Theta(n^2)$ error-correction steps.

The product of computer size and time is typically called "area-time product" or "AT" in the literature on non-quantum computation; "volume" in the literature on quantum computation; and "price-performance ratio" in the literature on economics. The cost of quantum error correction is not the only argument for viewing this product as the true cost of computation: there is a more fundamental argument stating that the total cost assigned to two separate computations should not depend on whether the computations are carried out in serial (using hardware for twice as much time) or in parallel (using twice as much hardware).

From this perspective, it is much better to use parallel algorithms for integer addition that finish in time $\Theta(\log n)$. This still means $\Theta(n \log n)$ error-correction steps, so the cost is larger by a factor $\Theta(\log n)$ than the number of bit operations.

At a higher level, the CSIDH computation involves various layers for which highly parallel algorithms are not known. For example, modular exponentiation is notoriously difficult to parallelize. A conventional computation of $x \bmod n$, $x^2 \bmod n$, $x^4 \bmod n$, $x^8 \bmod n$, etc. can store each intermediate result on top of the previous result, but Bennett's conversion produces a reversible computation that uses much more storage, and the resulting quantum computation requires continual error correction for all of the stored qubits. Shor's algorithm avoids this issue because it computes a superposition of powers of a *constant* x; this is not helpful in the CSIDH context.

Bennett suggested reducing the number of intermediate results in a reversible computation by checkpointing the computation halfway through:

- Compute the middle as a function of the beginning.
- Uncompute intermediate results, leaving the beginning and the middle.
- Compute the end as a function of the middle.
- Uncompute intermediate results, leaving the beginning, middle, and end.
- Recompute the middle from the beginning.
- Uncompute intermediate results, leaving the beginning and end.

This multiplies the number of qubits by about 0.5 but multiplies the number of gates by about 1.5. See [Ben89] and [Kni95] for analyses of further tradeoffs along these lines.

This chapter focuses on bit operations, as noted above. Beyond this, Section 9.13.6 makes some remarks on the number of qubits required for our computations. We have not attempted to analyze the time required for a parallel computation using a specified number of qubits.

9.11.6 – Error-correction steps on a two-dimensional mesh. There is a further problem with counting bit operations: in many computations, the main bottleneck is communication.

For example, FFT-based techniques multiply n-bit integers using $n^{1+o(1)}$ bit operations, and can be parallelized to use time just $n^{o(1)}$ with area $n^{1+o(1)}$. However, Brent and Kung [BK81] showed that integer multiplication on a two-dimensional mesh of area $n^{1+o(1)}$ requires time $n^{0.5+o(1)}$, even in a model where information travels instantaneously through arbitrarily long wires.

Plausible architectures for fault-tolerant quantum computation, such as [FMMC12], are built from near-neighbor interactions on a two-dimensional mesh. Presumably, as in [BK81], $n^{1+o(1)}$ qubits computing an n-bit product require time $n^{0.5+o(1)}$, and thus $n^{1.5+o(1)}$ error-correction steps. One might hope for quantum teleportation to avoid some of the bottlenecks, but spreading an entangled pair of qubits across distance $n^{0.5+o(1)}$ takes time $n^{0.5+o(1)}$ in the same architectures.

We have not attempted to analyze the impact of these effects for concrete sizes of n. We have also not analyzed communication costs at higher levels of the CSIDH computation. For comparison, attacks against AES [GLRS16] use fewer qubits, and perform much longer stretches of computation on nearby qubits.

9.12 — Basic integer arithmetic

We use b bits $n_0, n_1, n_2, \ldots, n_{b-1}$ to represent the nonnegative integer $n_0 + 2n_1 + 4n_2 + \cdots + 2^{b-1}n_{b-1}$. Each element of $\left\{0, 1, \ldots, 2^b - 1\right\}$ has a unique representation as b bits. This section analyzes the cost of additions, subtractions, multiplications, and squarings in this representation.

9.12.1 – Addition. We use a standard sequential **ripple-carry adder**. If $b \geq 1$ then the sum of the b-bit integers represented by $n_0, n_1, n_2, \ldots, n_{b-1}$ and $m_0, m_1, m_2, \ldots, m_{b-1}$ is the $(b+1)$-

bit integer represented by $s_0, s_1, s_2, \ldots, s_b$ computed as follows:

$$\begin{array}{lll}
x_0 = n_0 \oplus m_0; & s_0 = x_0; & c_0 = n_0 m_0; \\
x_1 = n_1 \oplus m_1; & s_1 = x_1 \oplus c_0; & c_1 = n_1 m_1 \oplus x_1 c_0; \\
x_2 = n_2 \oplus m_2; & s_2 = x_2 \oplus c_1; & c_2 = n_2 m_2 \oplus x_2 c_1; \\
\vdots & & \\
x_{b-1} = n_{b-1} \oplus m_{b-1}; & s_{b-1} = x_{b-1} \oplus c_{b-2}; & c_{b-1} = n_{b-1} m_{b-1} \oplus x_{b-1} c_{b-2}; \\
 & s_b = c_{b-1}. &
\end{array}$$

There are $5b - 3$ bit operations here, including $2b - 1$ nonlinear bit operations. Our primary cost metric is the number of nonlinear bit operations.

More generally, to add a b-bit integer to an a-bit integer with $a \leq b$, we use the formulas above to obtain a $(b + 1)$-bit sum, skipping computations that refer to $m_a, m_{a+1}, \ldots, m_{b-1}$.

Minor speedups: If $a = 0$ then we instead produce a b-bit sum. More generally, we could (but currently do not) track ranges of integers more precisely, and decide based on the output range whether a sum needs b bits or $b + 1$ bits. This is compatible with constant-time computation: the sequence of bit operations being carried out is independent of the values of the bits being processed.

9.12.2 – Subtraction. We use a standard ripple-borrow subtractor to subtract two b-bit integers modulo 2^b, obtaining a b-bit integer. The formulas are similar to the ripple-carry adder. The total number of operations grows from 5 to 7 for each bit but the number of nonlinear operations is still 2 per bit.

9.12.3 – Multiplication. Write $Q(b)$ for the minimum number of nonlinear bit operations for b-bit integer multiplication. We combine Karatsuba multiplication [KO63] and schoolbook multiplication, as explained below, to obtain concrete upper bounds on $Q(b)$ for various values of b. See Table 9.5.

We are not aware of previous analyses of $Q(b)$. It is easy to find literature stating the number of bit operations for schoolbook multiplication, but we do better starting at 14 bits. For $b = 512$ we obtain $Q(512) \leq 241908$ (using an algorithm with a total of 536184 bit operations), while schoolbook multiplication uses 784896 nonlinear bit operations (and a total of 1568768 bit operations).

It is also easy to find literature on the number of bit operations for polynomial multiplication mod 2, but carries make the integer case much more expensive and qualitatively change the analysis. For example, [KS15] uses Karatsuba's method for polynomials all the way down to single-bit multiplication, exploiting the fact that polynomial addition costs 0 nonlinear bit operations; for integer multiplication, Karatsuba's method has much more overhead. Concretely, Karatsuba's method uses just $3^9 = 19683$ nonlinear bit operations to multiply 512-bit polynomials; we use 12 times as many nonlinear bit operations to multiply 512-bit integers.

Schoolbook multiplication. Schoolbook multiplication of two b-bit integers has two stages. The first stage is b^2 parallel multiplications of individual bits. This produces 1 product at position 0; 2 products at position 1; 3 products at position 2; …; b products at position $b-1$; $b-1$ products at position b; …; 1 product at position $2b - 2$. The second stage repeatedly

- adds two bits at position i, obtaining one bit at position i and a carry bit at position $i + 1$, or, more efficiently,

1	1	65	8313	129	25912	193	50221	257	79732	321	114068	385	153686	449	197391
2	6	66	8497	130	26224	194	50631	258	80237	322	114669	386	154350	450	198131
3	18	67	8813	131	26733	195	51310	259	81067	323	115655	387	155448	451	199341
4	36	68	8940	132	26925	196	51563	260	81387	324	116035	388	155866	452	199802
5	60	69	9201	133	27377	197	52161	261	82097	325	116877	389	156807	453	200845
6	90	70	9397	134	27701	198	52594	262	82614	326	117490	390	157494	454	201601
7	126	71	9664	135	28098	199	53124	263	83267	327	118263	391	158354	455	202540
8	168	72	9736	136	28233	200	53296	264	83467	328	118499	392	158615	456	202834
9	216	73	10070	137	28699	201	53930	265	84252	329	119428	393	159655	457	203956
10	270	74	10272	138	28968	202	54295	266	84712	330	119972	394	160261	458	204608
11	330	75	10618	139	29440	203	54924	267	85442	331	120834	395	161233	459	205675
12	396	76	10757	140	29644	204	55200	268	85774	332	121226	396	161674	460	206152
13	468	77	11042	141	29992	205	55657	269	86315	333	121863	397	162385	461	206932
14	535	78	11256	142	30267	206	56017	270	86720	334	122340	398	162923	462	207529
15	630	79	11547	143	30682	207	56565	271	87330	335	123058	399	163738	463	208429
16	684	80	11625	144	30762	208	56669	272	87473	336	123225	400	163918	464	208619
17	795	81	11989	145	31307	209	57384	273	88217	337	124101	401	164926	465	209751
18	851	82	12209	146	31649	210	57835	274	88691	338	124659	402	165568	466	210468
19	974	83	12585	147	32206	211	58537	275	89441	339	125541	403	166571	467	211559
20	1036	84	12736	148	32416	212	58808	276	89718	340	125866	404	166944	468	211981
21	1171	85	13045	149	32910	213	59434	277	90403	341	126671	405	167858	469	212960
22	1239	86	13277	150	33264	214	59872	278	90883	342	127235	406	168495	470	213636
23	1386	87	13592	151	33697	215	60402	279	91444	343	127892	407	169237	471	214440
24	1460	88	13676	152	33844	216	60597	280	91656	344	128140	408	169521	472	214750
25	1608	89	14070	153	34352	217	61190	281	92288	345	128880	409	170347	473	215625
26	1688	90	14308	154	34645	218	61532	282	92644	346	129296	410	170812	474	216126
27	1859	91	14703	155	35159	219	62153	283	93343	347	130115	411	171729	475	217072
28	1934	92	14866	156	35381	220	62411	284	93626	348	130446	412	172097	476	217453
29	2092	93	15188	157	35759	221	62873	285	94130	349	131034	413	172758	477	218153
30	2195	94	15427	158	36058	222	63243	286	94553	350	131529	414	173314	478	218725
31	2369	95	15755	159	36509	223	63788	287	95187	351	132271	415	174142	479	219559
32	2431	96	15845	160	36595	224	63887	288	95275	352	132371	416	174254	480	219694
33	2607	97	16247	161	37188	225	64619	289	96171	353	133423	417	175429	481	220845
34	2726	98	16492	162	37560	226	65072	290	96724	354	134072	418	176152	482	221551
35	2914	99	16917	163	38165	227	65820	291	97632	355	135125	419	177314	483	222704
36	2978	100	17081	164	38393	228	66106	292	97982	356	135535	420	177773	484	223156
37	3172	101	17438	165	38929	229	66750	293	98758	357	136432	421	178755	485	224126
38	3303	102	17706	166	39313	230	67219	294	99323	358	137082	422	179465	486	224834
39	3509	103	18058	167	39782	231	67808	295	100036	359	137904	423	180371	487	225741
40	3579	104	18154	168	39941	232	67990	296	100254	360	138158	424	180650	488	226010
41	3791	105	18597	169	40491	233	68699	297	101111	361	139137	425	181723	489	227102
42	3934	106	18860	170	40808	234	69113	298	101613	362	139712	426	182357	490	227747
43	4158	107	19290	171	41364	235	69781	299	102409	363	140618	427	183334	491	228727
44	4234	108	19477	172	41604	236	70083	300	102771	364	141029	428	183780	492	229181
45	4464	109	19811	173	42012	237	70576	301	103360	365	141703	429	184514	493	229913
46	4619	110	20061	174	42335	238	70949	302	103801	366	142205	430	185052	494	230446
47	4850	111	20423	175	42822	239	71513	303	104465	367	142955	431	185849	495	231243
48	4932	112	20514	176	42914	240	71640	304	104620	368	143134	432	186052	496	231449
49	5169	113	20959	177	43555	241	72338	305	105430	369	144043	433	186996	497	232382
50	5325	114	21237	178	43957	242	72782	306	105946	370	144621	434	187597	498	232980
51	5585	115	21698	179	44599	243	73482	307	106762	371	145536	435	188569	499	233970
52	5673	116	21872	180	44845	244	73743	308	107063	372	145874	436	188919	500	234312
53	5928	117	22278	181	45412	245	74380	309	107808	373	146706	437	189807	501	235231
54	6107	118	22572	182	45815	246	74826	310	108330	374	147290	438	190436	502	235886
55	6349	119	22937	183	46309	247	75351	311	108939	375	147973	439	191165	503	236628
56	6432	120	23056	184	46480	248	75549	312	109169	376	148228	440	191431	504	236899
57	6702	121	23492	185	47050	249	76139	313	109855	377	149000	441	192272	505	237769
58	6868	122	23745	186	47380	250	76473	314	110241	378	149435	442	192742	506	238247
59	7154	123	24183	187	47956	251	77120	315	111000	379	150281	443	193666	507	239231
60	7265	124	24373	188	48203	252	77383	316	111307	380	150625	444	194044	508	239630
61	7510	125	24699	189	48630	253	77853	317	111853	381	151233	445	194697	509	240309
62	7692	126	24954	190	48966	254	78244	318	112312	382	151742	446	195250	510	240901
63	7939	127	25337	191	49467	255	78828	319	113000	383	152505	447	196090	511	241814
64	8009	128	25415	192	49565	256	78914	320	113094	384	152611	448	196197	512	241908

Table 9.5: Upper bounds on $Q(b)$ for $b \leq 512$: e.g., $Q(3) \leq 18$. $Q(b)$ is the minimum number of nonlinear bit operations for b-bit integer multiplication.

- adds three bits at position i, obtaining one bit at position i and a carry bit at position $i+1$,

until there is only one bit at each position.

There are several standard ways to organize the second stage for parallel computation: for example, Wallace trees [Wal64] and Dadda trees [Dad65]. Dadda trees use fewer bit operations since they make sure to add three bits whenever possible rather than two bits. Since parallelism is not visible in our primary cost metric, we simply add sequentially from the bottom bit. Overall we use $6b^2 - 8b$ bit operations for b-bit schoolbook multiplication (if $b \geq 2$), including $3b^2 - 3b$ nonlinear bit operations.

Karatsuba multiplication. When b is not very small, we do better using Karatsuba's method: the product of $X_0 + 2^b X_1$ and $Y_0 + 2^b Y_1$ is $Z_0 + 2^b Z_1 + 2^{2b} Z_2$ where $Z_0 = X_0 Y_0$, $Z_2 = X_1 Y_1$, and $Z_1 = (X_0 + X_1)(Y_0 + Y_1) - (Z_0 + Z_2)$.

Karatsuba's method reduces a $2b$-bit multiplication to two b-bit multiplications for Z_0 and Z_2, two b-bit additions for $X_0 + X_1$ and $Y_0 + Y_1$, one $(b+1)$-bit multiplication, one $2b$-bit addition for $Z_0 + Z_2$, one subtraction modulo 2^{2b+1} for Z_1, and a $4b$-bit addition for $(Z_0 + 2^{2b} Z_2) + 2^b Z_1$. Some operations in the $4b$-bit addition can be eliminated, and counting carefully shows that

$$Q(2b-1) \leq Q(b-1) + Q(b) + Q(b+1) + 17b - 12,$$
$$Q(2b) \leq 2Q(b) + Q(b+1) + 17b - 4.$$

These formulas do better than schoolbook multiplication for $Q(14)$ and for $Q(16), Q(17), \ldots$.

For comparison, similar formulas apply to $M(b)$, the total number of bit operations (linear and nonlinear) for b-bit polynomial multiplication mod 2. The cost of schoolbook multiplication then scales as $2b^2$ rather than $3b^2$. The overhead of "refined Karatsuba" multiplication scales as $7b$ rather than $17b$, already giving improved bounds on $M(6)$, and giving, e.g., $M(512) \leq 109048$.

Other techniques. We have skipped some small speedups. For example, the top bit of $(X_0 + X_1)(Y_0 + Y_1)$ does not need to be computed. As another example, one can use "refined Karatsuba" multiplication for integers; see [HS15] for one way to organize the carry chains. Presumably we have missed some other small speedups.

Toom multiplication [Too63] implies $Q(b) \in b^{1+o(1)}$. FFT-based improvements in the $o(1)$ appear in, e.g., [Pol71], [Nic71, page 532], [SS71], [Für07], [HHL16], and [HH18]. Our Karatsuba-based bounds on $Q(b)$ can thus be improved for sufficiently large values of b, and perhaps for values of b relevant to CSIDH. For comparison, Bernstein [Ber09a] obtained $M(512) \leq 98018$ using Toom multiplication, not a large improvement upon the $M(512) \leq 109048$ bound mentioned above from refined Karatsuba multiplication.

9.12.4 – Squaring. Schoolbook squaring saves about half the work of schoolbook multiplication. Specifically, for each pair (i, j) with $i < j$, schoolbook multiplication adds both $n_i m_j$ and $n_j m_i$ to position $i+j$, while schoolbook squaring adds $n_i n_j$ to position $i+j+1$; also, schoolbook multiplication adds $n_i m_i$ to position $2i$, while schoolbook squaring adds n_i (which is the same as n_i^2) to position $2i$. Overall we use $3b^2 - 6b + 3$ bit operations for b-bit schoolbook squaring, including $1.5b^2 - 2.5b + 1$ nonlinear bit operations.

Karatsuba squaring also has less overhead than Karatsuba multiplication, but the ratio overhead/schoolbook is somewhat larger for squaring than for multiplication, making Karatsuba squaring somewhat less effective. We obtain squaring speedups from Karatsuba squaring — in our primary cost metric, nonlinear bit operations — starting at 22 bits. For 512 bits we use 143587 nonlinear bit operations, about 60% of the nonlinear bit operations that we use for multiplication.

9.12.5 – Multiplication by a constant. We save even more in the multiplications that arise in reduction modulo p (see Section 9.13.1), namely multiplications by large constants. The exact savings depend on the constant; for example, for seven different 512-bit constants, we use

$$107338, 110088, 109574, 111760, 107925, 107711, 108234$$

nonlinear bit operations, about 45% of the multiplication cost. Here the schoolbook method is as follows: if m_j is the constant 1 then add n_i to position $i+j$. We use Karatsuba multiplication starting at 30 bits.

Other techniques. There is some literature studying addition chains (and addition-subtraction chains) with free doublings. For example, [DIZ07] shows that multiplication by a b-bit constant uses $O(b/\log b)$ additions (and [Lef03] shows that most constants require $\Theta(b/\log b)$ additions), for a total of $O(b^2/\log b)$ bit operations. This is asymptotically beaten by Karatsuba multiplication, but could be useful as an intermediate step between schoolbook multiplication and Karatsuba multiplication.

9.13 — Modular arithmetic

CSIDH uses elliptic curves defined over $\mathbb{F}_p$, where p is a standard prime number. For example, in CSIDH-512, p is the prime number $4 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdots 373 \cdot 587 - 1$, between 2^{510} and 2^{511}; all primes between 3 and 373 appear in the product.

Almost all of the bit operations in our computation are consumed by a long series of multiplications modulo p, organized into various higher-level operations such as exponentiation and elliptic-curve scalar multiplication. This section analyzes the performance of modular multiplication, exponentiation, and inversion.

9.13.1 – Reduction. We completely reduce a nonnegative integer z modulo p as follows. Assume that z has c bits (so $0 \le z < 2^c$), and assume $2^{b-1} < p < 2^b$ with $b \ge 2$.

If $c < b$ then there is nothing to do: $0 \le z < 2^{b-1} < p$. Assume from now on that $c \ge b$.

Compute an approximation q to z/p precise enough to guarantee that $0 \le z - qp < 2p$. Here we use the standard idea of multiplying by a precomputed reciprocal:

- Precompute $R = \left\lfloor 2^{c+2}/p \right\rfloor$. Formally, this costs 0 in our primary cost metric, since precomputation is part of *constructing* our algorithm rather than *running* our algorithm. More importantly, our entire algorithm uses only a few small values of c, so this precomputation has negligible cost.
- Compute $q = \left\lfloor \left\lfloor z/2^{b-2} \right\rfloor R/2^{c-b+4} \right\rfloor$. The cost of computing q is the cost of multiplying the $(c-b+2)$-bit integer $\left\lfloor z/2^{b-2} \right\rfloor$ by the constant $(c-b+3)$-bit integer R. Computing $\left\lfloor z/2^{b-2} \right\rfloor$ means simply taking the top $c-b+2$ bits of z.

By construction $R \le 2^{c+2}/p$ and $q \le zR/2^{c+2}$ so $q \le z/p$. Checking that $z/p < q+2$ involves more inequalities:

- $2^{c+2}/p < R+1$ so $z/p < z(R+1)/2^{c+2} < zR/2^{c+2} + 1/4$. This uses the fact that $0 \le z < 2^c$.
- $z/2^{b-2} < \left\lfloor z/2^{b-2} \right\rfloor + 1$, so $(z/2^{b-2})R/2^{c-b+4} < \left\lfloor z/2^{b-2} \right\rfloor R/2^{c-b+4} + 1/2$. This uses the fact that $0 \le R < 2^{c-b+3}$.
- $\left\lfloor z/2^{b-2} \right\rfloor R/2^{c-b+4} < q+1$.

- Hence $z/p < q + 1 + 1/2 + 1/4 = q + 7/4$.

Next replace z with $z - qp$. This involves a multiplication of the $(c - b + 1)$-bit integer q by the constant b-bit integer p, and a subtraction. We save some time here by computing only the bottom $b + 1$ bits of qp and $z - qp$, using the fact that $0 \le z - qp < 2^{b+1}$.

At this point (the new) z is between 0 and $2p - 1$, so all that remains is to subtract p from z if $z \ge p$.

Compute $y = z - p \bmod 2^{b+1}$. Use y_b, the bit at position b of y, to select between the bottom b bits of z and the bottom b bits of y: specifically, compute $y_0 \oplus y_b(y_0 \oplus z_0)$, $y_1 \oplus y_b(y_1 \oplus z_1)$, and so on through $y_{b-1} \oplus y_b(y_{b-1} \oplus z_{b-1})$. If $z \ge p$ then $0 \le z - p < p < 2^b$ so $y = z - p$ and $y_p = 0$, so these output bits are $y_0, y_1, \ldots, y_{b-1}$ as desired; if $z < p$ then $-2^b < -p \le z - p < 0$ so $y = z - p + 2^{b+1}$ and $y_p = 1$, so these output bits are $z_0, z_1, \ldots, z_{b-1}$ as desired.

Other techniques. We could save time in the multiplication by R by skipping most of the computations involved in bottom bits of the product. It is important for the total of the bits thrown away to be at most 2^{c-b+2}, so that q is reduced by at most $1/4$, the gap between $q + 2$ and the $q + 7/4$ mentioned above.

We could vary the number of bits in R, the allowed range of $z - qp$, etc. The literature sometimes recommends repeatedly subtracting p once z is known to be small, but if the range is (e.g.) 0 through $4p - 1$ then it is slightly better to first subtract $2p$ and then subtract p.

Historical notes. Multiplying by a precomputed reciprocal, to compute a quotient and then a remainder, is often called "Barrett reduction", in reference to a 1986 paper [Bar86]. However, Knuth [Knu81, page 264] had already commented in 1981 that Newton's method "for evaluating the reciprocal of a number was extensively used in early computers" and that, for "extremely large numbers", Newton's method and "subsequent multiplication" using fast multiplication techniques can be "considerably faster" than a simple quadratic-time division method.

9.13.2 – Multiplication. To multiply b-bit integers x, y modulo p, we follow the conventional approach of first multiplying x by y, and then reducing the $2b$-bit product xy modulo p as explained in Section 9.13.1.

For example, for CSIDH-512, we use 241814 nonlinear bit operations for 511-bit multiplication, and 206088 nonlinear bit operations for reduction modulo p, for a total of 447902 nonlinear bit operations for multiplication modulo p.

Generic conversion to a quantum algorithm (see Section 9.11.4) produces $14 \cdot 447902 = 6270628$ T-gates. This T-gate count is approximately 48 times larger than the cost "2^{17}" claimed in [BS18, Table 6]. The ratio is actually closer to 100, since [BS18] claims to count "Clifford+T" gates while we count only T-gates. We do not claim that the generic conversion is optimal, but there is no justification for [BS18] using an estimate for the costs of multiplication in a binary field as an estimate for the costs of multiplication in $\mathbb{F}_p$.

Squaring. For CSIDH-512, we use 143508 nonlinear bit operations for 511-bit squaring, and again 206088 nonlinear bit operations for reduction modulo p, for a total of 349596 nonlinear bit operations for squaring modulo p. This is about 78% of the cost of a general multiplication, close to the traditional 80% estimate.

Other techniques. Montgomery multiplication [Mon85] computes $xy/2^b$ modulo p, using a multiple of p to clear the bottom bits of xy. This has the same asymptotic performance as clearing the top bits; it sometimes requires extra multiplications and divisions by 2^b modulo p but might be slightly faster overall.

9.13.3 – Addition. A standard speedup for many software platforms is to avoid reductions after additions. For example, to compute $(x+y)z$ modulo a 511-bit p, one computes the 512-bit sum $x+y$, computes the 1023-bit product $(x+y)z$, and then reduces modulo p.

However, bit operations are not the same as CPU cycles. An intermediate reduction of $x+y$ modulo p (using the last step of the reduction procedure explained in Section 9.13.1, a conditional subtraction of p) involves relatively few bit operations, and saves more bit operations because the multiplication and reduction are working with slightly smaller inputs.

9.13.4 – Exponentiation with small variable exponents. Our isogeny algorithms involve various computations $x^e \bmod p$ where e is a variable having only a few bits, typically under 10 bits.

To compute $x^e \bmod p$ where $e = e_0 + 2e_1 + 4e_2 + \cdots + 2^{b-1}e_{b-1}$, we start with $x^{e_{b-1}}$, square modulo p, multiply by $x^{e_{b-2}}$, square modulo p, and so on through multiplying by x^{e_0}. We compute each x^{e_i} by using the bit e_i to select between 1 and x; this takes a few bit operations per bit of x, as in Section 9.13.1.

Starting at $b = 4$, we instead use "width-2 windows". This means that we perform a sequence of square-square-multiply operations, using two bits of e at a time to select from a precomputed table of $1, x, x^2 \bmod p, x^3 \bmod p$. For example, for 10-bit exponents, we use 9 squarings and 5 general multiplications.

None of the CSIDH parameters that we tested involved variable exponents e large enough to justify window width 3 or larger.

9.13.5 – Inversion. We compute the inverse of x in $\mathbb{F}_p$ as $x^{p-2} \bmod p$. This is different from the situation in Section 9.13.4, in part because the exponent here is a constant and in part because the exponent here has many more bits.

We use fractional sliding windows to compute $x^{p-2} \bmod p$. This means that we begin by computing $x^2, x^3, x^5, x^7, x^9, \ldots, x^W$ modulo p, where W is a parameter; "fractional" means that $W+1$ is not required to be a power of 2. We then recursively compute x^e as $(x^{e/2})^2$ if e is even, and as x^r times x^{e-r} if e is odd, where $r \in \{1, 3, 5, 7, 9, \ldots, W\}$ is chosen to maximize the number of 0 bits at the bottom of $e-r$. For small e we use some minor optimizations listed in [BL08, Section 3]: for example, we compute x^e as $x^{e/2-1}x^{e/2+1}$ if e is a multiple of 4 and $e \le 2W-2$.

We choose W as follows. Given a b-bit target exponent e, we automatically evaluate the cost of the computation described above for each odd $W \le 2b+3$. For this evaluation we model the cost of a squaring as 0.8 times the cost of a general multiplication, without regard to p. We could instead substitute the exact costs for arithmetic modulo p.

For CSIDH-512, we use 537503414 bit operations for inversion, including 220691666 nonlinear bit operations. Here W is chosen as 33. There are 507 squarings, accounting for $507 \cdot 349596 = 177245172$ nonlinear bit operations, and 97 general multiplications, accounting for the remaining $97 \cdot 447902 = 43446494$ nonlinear bit operations.

Batching inversions. We use Montgomery's trick [Mon87] of computing $1/y$ and $1/z$ by first computing $1/yz$ and then multiplying by z and y respectively. This reduces a batch of two inversions to one inversion and three multiplications; a batch of three inversions to one inversion and six multiplications; etc.

Inversion by exponentiation allows input 0 and produces output 0. This extension of the inversion semantics is often convenient for higher-level computations: for example, some of our computations sometimes generate input 0 in settings where the output will later be thrown

away. However, Montgomery's trick does not preserve these semantics: for example, if $y = 0$ and $z \neq 0$ then Montgomery's trick will produce 0 for both outputs.

We therefore tweak Montgomery's trick by replacing each input 0 with input 1 (and replacing the corresponding output with 0; we have not checked whether any of our computations need this). To do this with a constant sequence of bit operations, we compare the input to 0 by ORing all the bits together, and we then XOR the complement of the result into the bottom bit of the input.

Eliminating inversions. Sometimes, instead of dividing x by z, we maintain x/z as a fraction. This skips the inversion of z, but usually costs some extra multiplications. We quantify the effects of this choice in describing various higher-level computations: for example, this is the choice between "affine" and "projective" coordinates for elliptic-curve points in Section 9.3.2.

The Legendre symbol. The Legendre symbol of x modulo p is, by definition, 1 if x is a nonzero square modulo p; -1 if x is a non-square modulo p; and 0 if x is divisible by p. The Legendre symbol is congruent modulo p to $x^{(p-1)/2}$, and we compute it this way.

The cost of the Legendre symbol is marginally smaller than the cost of inversion. For example, for CSIDH-512, there are 506 squarings and 96 general multiplications, in total using 535577602 bit operations, including 218988158 nonlinear bit operations.

Other techniques. It is well known that inversion in $\mathbb{F}_p$ via an extended version of Euclid's algorithm is asymptotically much faster than inversion via exponentiation. Similar comments apply to Legendre-symbol computation.

However, Euclid's algorithm is a variable-time loop, where each iteration contains a variable-time division. This becomes very slow when it is converted in a straightforward way into a constant-time sequence of bit operations. Faster constant-time variants of Euclid's algorithm are relatively complicated and still have considerable overhead; see, e.g., [Bos14] and [RNSL17, Section 3.4].

We encourage further research into these constant-time algorithms. Sufficiently fast inversion and Jacobi-symbol computation could save more than 10% of our overall computation time.

9.13.6 – Fewer qubits. In this subsection we look beyond our primary cost metric and consider some of the other costs incurred by integer arithmetic.

Consider, e.g., the sequence of bit operations described in Section 9.13.5 for inversion in the CSIDH-512 prime field: 537503414 bit operations, including 220691666 nonlinear bit operations. A generic conversion (see Section 9.11.3) produces a reversible computation using $2 \cdot 220691666 = 441383332$ Toffoli gates.

It is important to realize that this reversible computation also uses 537503414 bits of intermediate storage, and the corresponding quantum computation (see Section 9.11.4) requires 537503414 qubits. The factor 2 mentioned in the previous paragraph accounts for the cost of "uncomputation" to recompute these intermediate results in reverse order; all of the results are stored in the meantime. Presumably many of the linear operations can be carried out in place, reducing the intermediate storage, but this improvement is limited: about 40% of the bit operations that we use are nonlinear. The number of qubits is even larger for higher-level computations, such as our algorithms for the CSIDH group action.

In traditional non-reversible computation, the bits used to store intermediate results in one multiplication can be erased and reused to store intermediate results for the next multiplication. Something similar is possible for reversible computation (and quantum computation), but one does not simply erase the intermediate results; instead one immediately uncomputes each

multiplication, doubling the cost of each multiplication. The inversion operation uses many of these double-cost multiplications and accumulates its own sequence of intermediate results, which also need to be uncomputed, again using the double-cost multiplications. To summarize, this reuse of bits doubles the number of Toffoli gates used for inversion from 441383332 to 882766664. Similar comments apply to qubits and T-gates.

The intermediate space used for multiplication outputs in inversion, in scalar multiplication, etc. can similarly be reused, but this produces another doubling of costs. Even after these two doublings, our higher-level computations still require something on the scale of a million qubits.

Quantum algorithms are normally designed to fit into far fewer qubits, even when this means sacrificing many more qubit operations. For example — in the context of applying Shor's attack to an elliptic curve defined over a prime field — Roetteler, Naehrig, Svore, and Lauter [RNSL17, Table 1] squeeze b-bit reversible modular multiplication into

- $5b + 4$ bits using approximately $(16 \log_2 b - 26.3)b^2$ Toffoli gates, or
- $3b + 2$ bits using approximately $(32 \log_2 b - 59.4)b^2$ Toffoli gates.

These are about $2^{24.87}$ or $2^{25.83}$ Toffoli gates for $b = 511$, far more than the number of Toffoli gates we use.

We focus on the challenge of minimizing the number of nonlinear bit operations for the CSIDH class-group action. Understanding the entire tradeoff curve between operations and qubits — never mind more advanced issues such as parallelism (Section 9.11.5) and communication costs (Section 9.11.6) — goes far beyond the scope of this chapter. See [PRM17] for some recent work on improving these tradeoffs for reversible Karatsuba multiplication; see also [Che16], which fits Karatsuba multiplication into fewer bits but does not analyze reversibility.

Chapter 10

CCA security of lattice-based encryption with error correction

This chapter is for all practical purposes identical to the paper *HILA5 Pindakaas: On the CCA security of lattice-based encryption with error correction* [BGLP18] authored jointly with Daniel J. Bernstein, Leon Groot Bruinderink, and Tanja Lange, which was published at Africacrypt 2018.

10.1 — Introduction

HILA5 [Saa17b] is a public-key scheme designed by Saarinen and published at SAC 2017. It was submitted as a "Key Encapsulation Mechanism and Public Key Encryption Algorithm" [Saa17a] to NIST's call [NIST16] for post-quantum proposals. HILA5's design is based on Ring Learning With Errors (RLWE) over NTRU NTT rings. HILA5 takes the same ring parameters as New Hope [ADPS16] and changes the reconciliation method by which Alice and Bob achieve the same key to get a much lower chance of decryption failures.

The HILA5 submission [Saa17a] states

> *This design also provides IND-CCA secure KEM-DEM [CS03] public key encryption if used in conjunction with an appropriate AEAD [Rog02] such as NIST approved AES256-GCM [FIP01, Dwo07].*

In this chapter we show that HILA5 is not CCA secure: We compute Alice's secret key by sending her multiple encapsulation messages and using her answers to determine whether her decapsulated shared secret matches a certain guess or not. Our attack works independently of whether an AEAD is used or not and despite the error correcting code introduced in HILA5.

We have fully implemented our attack and experimentally verified that it works with high probability. We use the HILA5 reference implementation for Alice's part and also to verify that the retrieved secret key works for decryption. We use a slightly modified version of the same software for computations on the attacker's side; of course the attacker need not follow the computations an honest party would.

Acknowledgement. We thank Christine van Vredendaal for helpful discussions.

10.1.1 – Related work. Ajtai–Dwork [AD97] and NTRU [HPS98] are the oldest lattice-based encryption systems. In 1999 Hall, Goldberg, and Schneier [HGS99] developed a reaction attack which recovers the Ajtai–Dwork private key by observing decryption failures for suitably crafted encryptions to the public key. They wrote "We feel that the existance of these attacks effectively limits these ciphers to theoretical considerations only. That is, any implementation of the ciphers will be subject to the attacks we present and hence not safe."

Hoffstein and Silverman [HS00] adapted the attack to NTRU. As a defense, they suggested modifying NTRU to use the Fujisaki–Okamoto transform [FO99]. For a system without decryption failures, this transform turns a CPA-secure system into a CCA-secure one. At the same time this complicates and slows down the cryptosystem. For NTRU, the transform turns out to still allow attacks that exploit occasional decryption failures induced by *valid* ciphertexts; see [How+03].

New Hope [ADPS16] is a key-encapsulation mechanism (KEM), presented as a key-exchange protocol. It allows occasional decryption failures for valid ciphertexts, and explicitly avoids the "changes" that would be required for the Fujisaki–Okamoto transform. To prevent reaction attacks and other chosen-ciphertext attacks by a malicious Bob, New Hope requires using ephemeral keys, meaning keys that change with every execution of the protocol. The New Hope paper warns that reusing a public key in multiple protocol runs ("key caching") would be "disastrous for security", although it does not describe an attack.

Fluhrer [Flu16] showed the details of how to attack key reuse in a similar key-exchange protocol. Followup work [Din+17] extended the attack to more key-exchange protocols.

HILA5 is similar to New Hope, and still does not use the Fujisaki–Okamoto transform. HILA5 includes an error-correction step that practically eliminates decryption failures for valid ciphertexts. HILA5 does not warn against key caching: on the contrary, the most natural interpretation of the HILA5 security claims is that HILA5 is secure against chosen-ciphertext attacks. See Section 10.5. We published our results in December 2017; as of February 2018, the designer of HILA5 has not proposed an alternative interpretation of the security claims.

10.2 — Data flow in the attack

A KEM is defined by three algorithms. Key generation produces a secret key and a public key. Encapsulation produces a ciphertext and a session key, given a public key. Decapsulation produces a session key or failure, given a ciphertext and a secret key. The HILA5 submission document [Saa17a] gives details and reference code for a particular KEM, the "HILA5 KEM".

Our attack is a key-recovery attack against the HILA5 KEM: the attacker, evil Bob, ends up computing the secret key of a target Alice. This secret key gives the attacker the ability to run the decapsulation algorithm using Alice's secret key, and thus the ability to immediately decrypt legitimate ciphertexts sent by other users to Alice.

Our attack is a chosen-ciphertext attack: evil Bob chooses ciphertexts to provide to Alice (different from the legitimate ciphertexts), and learns something from observing the outputs of Alice decapsulating those ciphertexts. Formally, the attack shows that the HILA5 KEM does not provide IND-CCA2 security.

There are two important ways that the attack does not need the full power of a CCA2 decapsulation oracle. First, the attack is what is called a "reaction attack" in [HGS99] or a "sloppy Alice attack" in [VDT02]: evil Bob has a guess for the output of each decapsulation, and learns whether Alice's actual decapsulation output matches this guess. Evil Bob does not need any further information.

Second, evil Bob chooses all of his ciphertexts, and learns the secret key from Alice's reactions, before seeing the legitimate ciphertexts to decrypt. Formally, the attack shows not only that the HILA5 KEM does not provide IND-CCA2 security, but also that it does not provide IND-CCA1 security.

10.2.1 – Hashing the secret key does not stop the attack. One can easily stop key-recovery attacks by defining HILA5Hash as follows. HILA5Hash key generation picks a uniform random

32-byte string s, and then runs HILA5 key generation to obtain a public key, hashing s to generate all randomness used in HILA5 key generation. The HILA5Hash secret key is s. HILA5Hash encapsulation is the same as HILA5 encapsulation. HILA5Hash decapsulation reconstructs the HILA5 secret key from s (again running the HILA5 key-generation algorithm; alternatively, the HILA5 secret key can be cached), and then runs the HILA5 decapsulation algorithm.

Unless the hash function is easy to invert, a key-recovery attack against HILA5 does not produce a key-recovery attack against HILA5Hash. However, this hashing does not prevent the attacker from decrypting legitimate ciphertexts sent by other users to Alice.

10.2.2 – AEAD does not stop the attack. A PKE is defined by three algorithms. Key generation produces a secret key and a public key, as in a KEM. Encryption produces a ciphertext, given a plaintext and a public key. Decryption produces a plaintext or failure, given a ciphertext and a secret key.

The subtitle of the HILA5 submission is "Key Encapsulation Mechanism (KEM) and Public Key Encryption Algorithm". The submission document does not include a definition of a PKE, but NIST had already stated before submission that it would automatically convert each submitted KEM to a PKE using the following "standard conversion technique": "appending to the KEM ciphertext, an AES-GCM ciphertext of the plaintext message" where the AES-GCM key is "the symmetric key output by the encapsulate function". This is the standard Cramer–Shoup "KEM-DEM" construction, using AES-GCM as the DEM. We write "HILA5 PKE" for the PKE that NIST will automatically produce in this way from the HILA5 KEM.[1]

Breaking the IND-CCA2 security of a KEM does not necessarily imply breaking the IND-CCA2 security of a PKE obtained in this way. IND-CCA2 attacks against the KEM can see session keys produced by decapsulation, whereas IND-CCA2 attacks against the PKE are merely able to see the result of AES-GCM decryption using those keys.

However, our attack against the HILA5 KEM is also a key-recovery attack against the HILA5 PKE. It is important here that the attack is a reaction attack: what evil Bob needs to know is merely whether a guessed session key is correct. Starting from this guessed session key, evil Bob produces a valid AES-GCM ciphertext using this guess as an AES key. If decapsulation in fact produces this session key then AES-GCM decryption succeeds and produces the plaintext that evil Bob started with. If decapsulation produces a different session key then AES-GCM decryption is practically guaranteed to fail (anything else would be a surprising security flaw in AES-GCM), so evil Bob sees a decryption failure from the PKE.

To summarize, evil Bob sees decryption failures from the PKE, and learns from this which guesses were correct, which is the same information that evil Bob obtains from the KEM. Evil Bob then computes the secret key from this information. Consequently, the HILA5 PKE does not provide IND-CCA2 security, and does not even provide IND-CCA1 security.

10.2.3 – Black holes would stop the attack. Like other chosen-ciphertext attacks, our attack is inapplicable to scenarios where the results of decapsulation and decryption are hidden from the attacker. For example, if ciphertexts are sent to NSA's public key, and if NSA hides the results of applying its secret key to those ciphertexts, then an attacker outside NSA cannot use our attack to compute NSA's secret key. However, if NSA reacts to those results in a way that

[1] NIST actually deviates slightly from the KEM-DEM construction: it specifies a "randomly generated IV" for AES-GCM, while Cramer and Shoup use a deterministic DEM. For consistency with the ciphertext sizes mentioned in [Saa17a], we actually define "HILA5 PKE" to be the Cramer–Shoup construction using AES-GCM with an all-zero IV. Switching to NIST's construction would expand ciphertext sizes by 12 bytes using the default IV sizes for AES-GCM, and would not affect our attack.

leaks to the attacker which ciphertexts were valid, then the attacker can compute NSA's secret key.

10.2.4 – The Fujisaki–Okamoto transform would stop the attack. We briefly outline a more radical change to HILA5, which we call "HILA5FO". HILA5FO ciphertexts are slightly larger than HILA5 ciphertexts, decapsulation is more complicated, and decapsulation is extrapolated (from reported HILA5 benchmarks) to be several times slower, but HILA5FO would stop our attack.

The idea of the HILA5FO KEM is to reapply the encapsulation algorithm as part of decapsulation, and check whether the resulting ciphertext is identical to the received ciphertext. This is not a new idea: it is used in many other submissions to NIST (with various differences in details), typically with credit to Fujisaki and Okamoto [FO99].

HILA5 does not provide any easy way to reconstruct the randomness used in encapsulation (most importantly Bob's b), so the HILA5FO KEM computes this randomness as a hash of a plaintext recovered as part of decapsulation. The HILA5 KEM does not transmit a plaintext, so the HILA5FO KEM is instead built from the HILA5 PKE.

Encapsulation in the HILA5FO KEM thus chooses a random plaintext, and encrypts this plaintext using the HILA5 PKE (the HILA5 KEM producing a session key for AES-GCM) using a hash of the plaintext to compute all randomness used inside the PKE. Decapsulation applies HILA5 PKE decryption (HILA5 KEM decapsulation producing a session key for AES-GCM decryption), and checks that the resulting plaintext produces the same ciphertext.

Deriving a PKE from the HILA5FO KEM would involve two layers of AES-GCM, which can be compressed to one layer as follows: place 32 bytes of randomness at the beginning of the user-supplied plaintext, and then encrypt this plaintext using the HILA5 PKE, again using a hash of the plaintext to compute all randomness used inside the PKE. The overall ciphertext size is the original plaintext size, plus 32 bytes (the randomness), plus the HILA5 KEM ciphertext size, plus 16 bytes (the AES-GCM authenticator), i.e., 32 bytes more than the HILA5 PKE. The main cost in HILA5FO decryption (for short messages) is reapplying HILA5 KEM encapsulation, which according to [Saa17a, Table 1] is five times slower than HILA5 KEM decapsulation.

10.3 — Preliminaries

This section describes the HILA5 scheme and Fluhrer's attack on RLWE schemes.

10.3.1 – The HILA5 scheme. We describe the scheme as given in [Saa17a, Section 4.9] but leave out formatting and NTT conversions. These are used in the attack implementation to interface with the reference implementation but do not contribute to the security and hamper readability.

The major computations take place in the ring $R = \mathbb{Z}_q[x]/(x^n + 1)$, where $n = 1024$ and $q = 12289$. Alice's secret key is a small, random polynomial $a \in R$, where small (here and in the following) means that the coefficients are chosen from a narrow distribution around zero, more precisely the discrete binomial distribution Ψ_{16} which has integer values in $[-16, 16]$. To compute the public key she picks another small random polynomial $e \in R$ and a random $g \in R$ and computes $A = ga + e$. She publishes (g, A) and keeps a as her secret.

An honest Bob picks two random small polynomials $b, e' \in R$ and computes $B = gb + e'$ and $y = Ab$. Bob sends B to Alice. The second value

$$y = Ab = (ga + e)b = gab + eb \approx gab$$

is very close to what Alice can compute using her secret:

$$x = aB = a(gb + e') = gab + e'a \approx gab,$$

because a, b, e, e' are all small.

A simple rounding operation to achieve a shared secret, such as taking the top bits of each coefficient, will induce differences between Alice's and Bob's version with too high probability. For example, Bob could take $k[i] = \lfloor 2\,y[i]/q \rfloor$ and Alice could take $k'[i] = \lfloor 2\,x[i]/q \rfloor$, where we use $t[i]$ to denote the ith coefficient of polynomial or vector t, but for indices with $(gab)[i] \approx 0$ (or $q/2$) the error-terms can cause the values to flip to a different bit, i.e., $k[i] \neq k'[i]$. For this rounding operation, we call elements of $\{0, q/2\}$ the "edges", as these are the values for which it is probable that errors occur.

This is why Bob sends a second vector, a binary reconciliation vector c, to help Alice recover the same k as Bob. Basically, this means that the scheme uses two pairs of edges. If $y[i]$ was close to one edge of a certain pair, Bob will choose the other pair of edges, so that Alice can still successfully recover the shared secret. In previous work [Pei14], the reconciliation vector achieves a successful shared secret with high probability, as long as $|x[i] - y[i]| < q/8$.

HILA5 differs in how these reconciliation bits are computed. For each coefficient $y[i]$ of y Bob computes $k[i] = \lfloor 2\,y[i]/q \rfloor$, $c[i] \equiv \lfloor 4\,y[i]/q \rfloor \bmod 2$, and

$$d[i] = \begin{cases} 1 & \text{if } |(y[i] \bmod \lfloor q/4 \rceil) - \lfloor q/8 \rfloor| \leq \beta \\ 0 & \text{otherwise,} \end{cases}$$

where $\beta = 799$. He then selects the first 496 positions i for which $d[i] = 1$ and restarts with fresh b and e' if there are fewer. Positions with $d[i] = 1$ are those for which it is likely that Alice and Bob recover the same value. In other words, for these indices the value $(gab)[i]$ is likely to be far away from an edge, thus further reducing the probability of errors in the shared secret. (Note that the description suggests to discard some positions if there are more than 496 such positions while the code deterministically discards the later ones by setting $d[j] = 0$ for them.)

The encapsulation consists of B, d, c, and an extra part r described below; here d covers the full n positions while c can be compressed to those positions i where $d[i] = 1$.

Alice recovers the $k[i]$ at the selected 496 positions by computing

$$k'[i] = \lfloor 2\,(x[i] - c[i] \cdot \lfloor q/4 \rceil + \lfloor q/8 \rceil \bmod q)/q \rfloor.$$

The HILA5 submission shows that $k'[i] = k[i]$ with probability $1 - 2^{-36}$. Let k (resp. k') be the 496-bit string given by the concatenation of the $k[i]$ (resp. $k'[i]$).

The role of r is not well described but the HILA5 design overview says that is an encrypted encoding of a part of k. It is computed by splitting k as $k = m \| z$, where m gets the first 256 bits and z the remaining 240 bits. HILA5 uses a custom-designed error-correcting code XE5 that corrects at least 5 errors to compute a 240-bit checksum s of m and then computes $r = s \oplus z$, where $\oplus$ denotes bitwise addition (XOR).

Alice computes $k' = m' \| z'$, the checksum s' on m', and applies the XE5 error correction to m', s', z' and r to correct m' to m.

10.3.2 – Fluhrer's attack. The chosen-ciphertext attack on HILA5 that we present is a variant of the following attack against key reuse in RLWE-based key exchange protocols presented by Fluhrer in 2016 [Flu16]. This section assumes that Bob computes the $c[i]$ and $k[i]$ in a way similar to the previous section. The $d[i]$ were added in HILA5 and will be considered in the next section.

Recall that Alice's version of the shared secret key is

$$gab + e'a,$$

where g is some large public generator element, a and b are Alice's and Bob's small private keys, and e' is a small noise vector chosen by Bob. This version of the shared secret differs from Bob's by some small error, hence they need to employ a reconciliation mechanism to arrive at the same secret bit string.

The general strategy of an evil Bob is to artificially force one (say, the first) coefficient of gab to be close to the edge M between the intervals that are mapped to bits 0 and 1 during reconciliation. An honest user would set the reconciliation bit $c[0]$ in that case, so Alice would use another mapping that is less likely to produce an error; but evil Bob does not. Since evil Bob proceeds honestly except for the first bit, he knows two possibilities for Alice's key, hence he can query Alice with one of these guesses and distinguish between 0 and 1 based on her reaction. If we assume for the moment that evil Bob can choose, hence knows, $(gab)[0]$, this tells him that $(e'a)[0]$ lies in a certain interval.

After a few queries using binary search with varying values for $(gab)[0]$, evil Bob knows the exact distance of $(e'a)[0]$ from the edge, and if he sets $e' = 1$, this distance is nothing but the first coefficient of Alice's secret key a. Note that in Fluhrer's setting the edge M is at zero and he uses b with $(gab)[0] = 1$, hence evil Bob can just multiply that b by small distances to obtain a prescribed $(gab)[0]$ when searching for $(e'a)[0]$. In our adaptation of the attack to HILA5, this step is more involved; see Section 10.4.2.

One could apply this method individually to each coefficient to extract Alice's full secret key. However, being able to recover the coefficient at one position is enough: due to the structure of the underlying ring, evil Bob can shift the ith coefficient of a into the constant term of $e'a$ by setting e' to $-x^{n-i}$, i.e., a vector with one entry of -1 and 0 elsewhere.

We now come back to the assumption made above. Notice that evil Bob does not a priori know a vector $b \in R$ such that $(gab)[0] = 1$, but he can still reasonably guess one: Alice's public key is $ga + e$ for small vectors a and e, hence if b is a small low-weight vector such that $(b \cdot (ga + e))[0]$ is close to 1, there is a good chance that in fact $(gab)[0] = 1$. Thus, while evil Bob does not have a deterministic method to find an "evil" b, he can still just make educated guesses based on Alice's public key until he finds one that works. Finding $b \in R$ with $(b \cdot (ga + e))[0]$ close to 1 is an offline computation using only Alice's public key; testing for $(gab)[0] = 1$ requires interaction with Alice.

There are several follow-ups to Fluhrer's paper, e. g. the recently posted [Din+17], but a small and new generalization of Fluhrer's attack is sufficient to attack HILA5.

10.4 — Chosen-ciphertext attack on HILA5

In this section, we describe how we circumvent the error-correction code and how to adapt Fluhrer's attack to the HILA5 case.

10.4.1 – Working around error correction. The HILA5 construction includes XE5 as an error-correcting code that is applied to the shared secret after decapsulation. Both Alice and Bob compute their version of a redundancy check, which will help Alice to correct up to 5 errors in the shared secret. The redundancy part r is divided into ten subcodewords $r = r_0, \ldots, r_9$ of variable sizes. For the purpose of the attack, these sizes do not matter, but we use the same notation L_i for the size, as in the HILA5 paper. This means we can index each $r_i = r_{(i,0)} \ldots r_{(i,L_i-1)}$ for $i \in \{0, \ldots, 9\}$.

Bob first computes his part of the HILA5 encapsulation, i.e., he computes his version of the shared secret, selects the indices that are safe to use by Alice and computes the reconciliation vector. The last 240 bits of Bob's shared secret are used in XE5 error-correction. From these bits, Bob constructs his redundancy check r', and sends this as part of the ciphertext.

Upon receiving Bob's ciphertext, Alice first computes her part of the HILA5 decapsulation, i.e., she computes her version of the shared secret. Then she computes her own redundancy check r and computes the distance r^{Δ} with Bob's r' from the ciphertext:

$$r^{\Delta} = r' \oplus r$$

To determine which bits in the shared secret are erroneous, Alice determines a weight value $w_k^{\Delta} \in [0, 10]$ for each of the 256 bits by the following formula:

$$w_k^{\Delta} = r_{0,\lfloor k/16 \rfloor}^{\Delta} + \sum_{j=1}^{9} r_{j,k \bmod L_j}^{\Delta}$$

Now, if a single bit k of Alice's shared secret is flipped, it means $w_k^{\Delta} = 10$ [Saa17a, Lemma 2], and it is therefore detectable and correctable by Alice. Moreover, it is shown that XE5 corrects bit k as long as $w_k^{\Delta} \geq 6$ [Saa17a, Theorem 1], which means XE5 can correct at least 5 bits in the shared secret. This means that applying Fluhrer's original attack directly to HILA5 will not work, as Fluhrer's original attack depends crucially on the attacker's ability to detect single-bit errors in Alice's version of the shared secret. Thus, to apply Fluhrer's attack, we have to work around these error-correction abilities.

In the attack described in the next section, we focus on inducing errors only in the first bit $k = 0$ of the shared secret. This means the attacker evil Bob needs to force w_0^{Δ} to be less than 6, as this means XE5 is no longer capable of correcting the first bit. However, evil Bob needs to leave the remaining error-correction in place, otherwise he still does not know if the first bit was the only flipped bit. In order to do that, evil Bob needs to change his redundancy check r' to do exactly that. As w_0^{Δ} is obtained by summing up the first bits of the subcodeword distances r_i^{Δ}, he can flip any 5 of the bits labeled $r'_{(0,0)}$ through $r'_{(9,0)}$ to force $w_0^{\Delta} < 6$. Our attack flips the first 5 of these bits. This means in the following section we consider the issue of error-correction solved and can directly apply a modification of Fluhrer's attack.

10.4.2 – Details of the attack. This section elaborates evil Bob's approach to recover Alice's secret key. As mentioned before, the general procedure mimics Fluhrer's attack (Section 10.3.2). The major steps are:

1. Guess a small low-weight secret b_0 such that $(gab_0)[0]$ is at the edge M.
2. For each $\delta \in \{-16, \ldots, 16\}$, compute b_δ such that $(gab_\delta)[0] = M + \delta$.
3. For each target coefficient of Alice's secret:
   a) Choose e' such that $(e'a)[0]$ is the target coefficient.
   b) Perform a binary search using the b_δ to recover the target coefficient. (Alice's coefficient $(gab_\delta + e'a)[0]$ maps to a 1 bit iff $(-e'a)[0] > \delta$.)
4. If the results look "bad" after recovering a few coefficients in this way, the guess for b_0 was probably wrong and evil Bob should start over at step 1.

Note that for each oracle query, i.e., for every interaction with Alice, Bob proceeds honestly except for using specially crafted b_δ and e', setting $d_0 = c_0 = 1$, and flipping a few bits in the error correction as described in Section 10.4.1. We now explain and analyze the steps above in more detail.

Forcing coefficients near the edge. In HILA5's reconciliation mechanism, there is no edge at zero for any choice of reconciliation bit, hence Fluhrer's attack does not apply without modifications. We chose to set the reconciliation bit c_0 to 1 and attack the edge at

$$M = \lfloor q/8 \rceil = 1536.$$

To perform the binary search for Alice's secret coefficients in the attack, we need to find small low-weight vectors b_δ such that

$$(gab_\delta)[0] = M + \delta$$

for all δ with $|\delta| \leq 16$. (As mentioned in Section 10.3.2, Fluhrer's evil Bob attacked $M = 0$, thus he could guess b_1 based on Alice's public key and set $b_\delta = \delta \cdot b_1$.) One could of course try to guess each b_δ individually based on Alice's public key, but as we want to get all b_δ right at the same time, this has exponentially low success probability. Instead, we make use of a special property of the M used in HILA5: The inverse

$$M^{-1} \bmod q = -8$$

is small.[2] Hence, as soon as evil Bob successfully guessed b_0, he may simply set

$$b_\delta = (1 + \delta M^{-1} \bmod q) \cdot b_0.$$

In our case, we choose b_0 with only two non-zero coefficients from $\{\pm 1\}$, thus b_δ will have only two non-zero coefficients bounded by $1+8\delta$. This property is necessary to make sure evil Bob can actually know what Alice's version of the shared secret will be (except for the target bit that leaks information): If the coefficients of b_δ are too large, the error $eb - e'a$ between Alice's and Bob's shared secrets becomes too large to recover from and their secrets will mismatch no matter what the value of the attacked bit is. In theory, with these parameters we still expect a tiny possibility of unintended errors, but this happens so rarely that it is not an issue in practice. If it ever does occur, Bob can detect that his recovered secret key is wrong and simply start over with a new b_0.

When evil Bob chooses a random b_0 with two non-zero coefficients in $\{\pm 1\}$ and such that $(Ab_0)[0] = M$, the probability that in fact $(gab_0)[0] = M$ holds is just the probability that two Ψ_{16}-distributed values sum to zero:

$$\sum_{i=0}^{32} \binom{32}{i}^2 / 2^{64} \approx 9.9\%,$$

hence he can expect to find a good b_0 after about 10 tries. Since A can be approximated by a uniformly distributed sequence over $\mathbb{Z}_q$, the expected number of ± 1-combinations of two coefficients of A which equal M is

$$\binom{1024}{2} \cdot 4/q \approx 170.$$

Hence, the probability that evil Bob exhausts this pool of choices without finding a good b_0 is roughly 2^{-25}.

[2] Note that this also holds for some other "natural" choices of M as rounded fractions of q, but it is not automatically true for any conceivable M.

(If this ever happens, then evil Bob can still retry the attack with a larger interval, i.e., search for b_0 with $|(Ab_0)[0] - M| \leq K$ for some small K. This would in theory work for a wider range of keys, but the expected number of wrong guesses grows slightly. One could also choose three non-zero coefficients in b_0, although this increases the chance of unintended errors in Alice's shared secret. We have not had any problems with $K = 0$ in practice.)

Detecting bad guesses. After choosing b_0 based on Alice's public key as described above, evil Bob may just go ahead and try to recover Alice's secret key using that b_0. If it is correct, he will of course find a sequence that looks like it was sampled from the Ψ_{16} distribution. If b_0 is bad, say, $(gab_0)[0] = M + \gamma$ for some small $\gamma \neq 0$, then

$$(gab_\delta)[0] = M + \delta + \gamma - 8\delta\gamma,$$

hence typically $(gab_\delta)[0]$ is considerably smaller than M if $\delta > 0$ and considerably larger if $\delta < 0$; in both cases Alice's secret $(e'a)[0]$ is dominated by $\delta + \gamma - 8\delta\gamma$, which means the oracle output does not depend on the secret. This implies the binary search will always converge to 0 or -1 when b_0 is bad. (For $\delta = 0$, the behavior *does* depend on $(e'a)[0]$ since γ is small, so both cases really occur.) Evil Bob can detect this failure mode by determining a few coefficients and checking whether all of them are in $\{0, -1\}$. If this is the case, evil Bob simply starts over with a new b_0. The probability that an actual secret key starts with a sequence of k coefficients from $\{0, -1\}$ is about 0.27^k, hence setting $k = 8$ reduces the probability of a false negative to roughly 2^{-15}. There is a small probability of false positives if evil Bob uses only this heuristic (e. g., when $|\gamma| = 1$), but this can easily can be detected using statistical methods (the recovered sequence will not be Ψ_{16}-distributed) or by simply testing the obtained secret key in the end and running the attack again if it failed. In practice the heuristic works fine.

The number of queries. Assuming we already have a good b_0, the binary search needs an expected $5 + \varepsilon$ queries to the oracle to recover one coefficient.[3] Since evil Bob decides whether he has a good b_0 based on the first few coefficients that he obtains using that b_0, he usually wastes a few hundred queries on guesses for b_0 that turn out to be useless: If he looks at the first 8 coefficients obtained from each b_0 as suggested above, this adds expected ≈ 400 queries to the 5120 needed to recover all the coefficients. In summary, evil Bob will with overwhelming probability recover Alice's secret key in less than 6000 queries.

Evil Bob can trade computation for a smaller number of queries: retrieve some coefficients, and reduce the original lattice problem to low enough dimension to solve by computation.

10.4.3 – Implementation. We implemented a proof of concept of the attack in Python, reusing portions of the HILA5 reference implementation via the `ctypes` library. The only modifications we made to the reference implementation were making some functions non-`static` to be able to call them from within Python, and adding extra parameters to the encapsulation function (not used by Alice) such that evil Bob can override his private values b and e'. The complete attack script can be found at https://helaas.org/hila5-20171218.tar.gz. As expected, we have never observed the attack script failing to recover Alice's key. The empirical number of queries matches the theoretical prediction made above.

[3] The ε arises from the fact that Ψ_{16} samples from $33 > 2^5$ distinct values, but the extremal values occur so rarely that $\varepsilon \approx 2^{-27}$.

10.5 — HILA5 security claims

In this section, we discuss our interpretation of security claims made by both the paper and NIST submission of HILA5, which motivated this chapter.

NIST does not require IND-CCA2 security for KEM and PKE submissions. Instead it requires submissions to say whether they are aiming for IND-CCA2 security or merely for IND-CPA security.

IND-CPA security is adequate in the context of key exchange in TLS, if a new public key is generated for each TLS session. For example, New Hope [ADPS16] appears to be safe for use in TLS. New Hope does not aim for IND-CCA2 security, and specifically warns against using a key more than once: "No key caching … it is crucial that both parties use fresh secrets for each instantiation".

We emphasize that our attack does not break the IND-CPA security of HILA5. If HILA5 were clearly labeled as aiming merely for IND-CPA security then our attack would merely be a cautionary note, showing the importance of not reusing keys.

However, HILA5 went beyond claiming IND-CPA security. There are some undefined words in the HILA5 security claims, but the most natural interpretation of the security claims is that the HILA5 PKE provides IND-CCA2 security. There is certainly a high risk of the claims being interpreted in this way by potential users. Our attack shows that the HILA5 PKE does not provide IND-CCA2 security.

There is even a risk of users thinking that the HILA5 KEM is being claimed to provide IND-CCA2 security.[4] The HILA5 submission document does not say that the HILA5 KEM security target is merely IND-CPA. Our attack shows that the HILA5 KEM does not provide IND-CCA2 security.

We give four quotes from [Saa17a] to explain why the HILA5 security claims are most naturally interpreted as claiming IND-CCA2 security for the HILA5 PKE. We have not found anything in [Saa17a] or [Saa17b] indicating a different interpretation.

> [Saa17a, Section 1]: *The HILA5 KEM can be adopted for public key encryption in straightforward fashion. We recommend using the AES-256-GCM AEAD [FIP01, Dwo07] in conjunction with the KEM when public key encryption functionality is desired.*

The details of this "conjunction" are not formally defined. The most natural interpretation is that this is the HILA5 PKE, using the session key produced by the HILA5 KEM as the AES-GCM key.

> [Saa17a, Section 4.1]: *NIST requires at least IND-CPA [BDPR98] security from a KEM scheme (Section 1.6). … The design also provides IND-CCA secure KEM-DEM [CS03] public key encryption if used in conjunction with an appropriate AEAD [Rog02] such as NIST approved AES256-GCM [FIP01, Dwo07]. These properties are derived from [Pei14].*

This is a claim of IND-CCA security for a PKE. "IND-CCA" in the literature usually means IND-CCA2, although sometimes it means merely IND-CCA1. The PKE is not formally defined, but again the most natural interpretation is simply that the session key produced by the HILA5 KEM

[4] Adam Langley posted an online table of speeds for announced KEMs submitted to NIST. He wrote "I only want to list CCA-secure KEMs here". He listed HILA5, and accepted a correction from the HILA5 author regarding the speed of HILA5. After the correction, HILA5 had the fastest decapsulation in the entire table.

is the AES-GCM key used to encrypt a user-supplied plaintext. Our attack shows that this PKE does not even provide IND-CCA1 security, let alone IND-CCA2 security.

Our attack does not work against what we call the HILA5FO PKE (see Section 10.2.4), a more complicated PKE using the Fujisaki–Okamoto transformation. This transformation is also mentioned in "[Pei14]" as a way to achieve IND-CCA security. It is conceivable that the HILA5 submission was alluding to a PKE of this type. However, this interpretation does not appear to be compatible with the statement "Ciphertext size: 2012 Byte expansion (KEM) + payload + MAC" in [Saa17a, Section 6]; the HILA5FO ciphertext size is 32 bytes larger than this.

> [Saa17a, Section 4.9]: *For active security we suggest that* K *is used as keying material for an AEAD (Authenticated Encryption with Associated Data) [Rog02] scheme such as AES256-GCM [Dw007, FIP01] or Keyak [BDP+16] in order to protect message integrity.*

Here "K" is defined as the session key produced by the HILA5 KEM. In the context of KEMs and PKEs, "active security" is normally interpreted as IND-CCA2 security, although it might have other interpretations. The authentication in AES-GCM prevents modifications to the message encrypted by AES-GCM, but this is not enough to stop active attacks, since it does not protect the underlying KEM.

> [Saa17a, Section 6.1]: *HILA5 is essentially drop-in compatible with current public key encryption applications. There are no practical usage restrictions.*

Security against chosen-ciphertext attacks is essential for a wide range of current PKE applications, so this would appear to include a claim of CCA security for the HILA5 PKE. However, our attack retrieves the secret key from the HILA5 PKE.

Chapter 11

Recent developments

In this short chapter, we account for some developments that occurred after the works this thesis is based upon were first written, and hence may not have received sufficient appreciation in the earlier chapters.

11.1 — CSIDH is not an ideal group action

Chapter 3 focused on the application of CSIDH to Diffie–Hellman style (non-interactive) key exchange, which actually requires less than a full-fledged group action. More concretely, the protocol does not involve composing group elements; it merely acts by elements sampled from a convenient distribution. However, more advanced protocols based on group actions do require the full functionality of a group, including efficient composition of arbitrary elements. It may seem at first that this is trivial in CSIDH: The ideals in CSIDH are of the form $\prod \mathfrak{l}_i^{e_i}$ with exponent vectors $e \in \mathbb{Z}^n$, thus (cf. Chapter 4) the CSIDH group action can be viewed as an action of $(\mathbb{Z}^n, +)$ which happens to factor through $\mathrm{cl}(\mathcal{O})$, and composition in $\mathbb{Z}^n$ is simply addition. However, even for short vectors such as those sampled according to Section 3.4, composing just a few of these vector additions can make the size of the coefficients grow very quickly, and unfortunately, the cost of Algorithm 3.2 has a strong (at least linear) dependency on the 1-norm of the exponent vector. Thus, while it is obviously true that addition of group elements is efficient, the resulting exponent vector may no longer admit efficiently computing its action on the isogeny graph.

This issue can be solved at the cost of an exponential precomputation phase as suggested by Couveignes [Cou06]. Another, more recent take on the problem is based on ideas from [Bis12]: CSI-FiSh (see [BKV19] and below) performs only subexponential precomputation, leading to subexponential complexity for evaluating the action of ideals. While still inefficient in an asymptotic sense, this appears to be reasonably fast in practice [BKV19]. However, there are still no known techniques for evaluating the action of arbitrary ideals in overall polynomial time.

11.2 — CSI-FiSh: Canonical exponent vectors

In Remark 3.9, we stated as an open problem to find an algorithm for representing the product of two class-group elements given as CSIDH exponent vectors (cf. Section 11.1) in such a way that the product does not reveal information about the factors. For instance, if the exponents are all sampled independently from $\{-m, \ldots, +m\}$ as suggested in Section 3.4, then any coefficient of the sum greater than $+m$ reveals that both summands must have a positive entry at that index. Generalizing this simple example leads to much more statistically detectable leakage.

One way to thwart this issue is to truncate the probability space to a subset where everything behaves uniformly; this is done using rejection sampling in SeaSign [DG19]. Unfortunately, despite our improvements in Chapter 4, this approach is not very efficient, and may not satisfy the requirements of many practical applications.

Another avenue is taken by the signature scheme CSI-FiSh: It simply accepts the idea of naïvely adding vectors together with no rejections, but rewrites the result into a canonical representative using the *relation lattice* of the class group with respect to the chosen generators $\mathfrak{l}_i$. Concretely, the relation lattice is the kernel Λ of the group homomorphism

$$\mathbb{Z}^n \longrightarrow \mathrm{cl}(\mathcal{O}), \quad (e_1, ..., e_n) \longmapsto \prod_{i=1}^{n} \mathfrak{l}_i^{e_i},$$

and it is clear that the action of two CSIDH exponent vectors $e, e' \in \mathbb{Z}^n$ on the isogeny graph is the same if and only if $e' \in e + \Lambda$. To compute a canonical representative of the coset $e + \Lambda$ for some input vector $e \in \mathbb{Z}^n$, CSI-FiSh solves a closest-vector problem to find a vector $k \in \Lambda$ close to e, and the result of the rewriting step is a short vector $e - k \in e + \Lambda$, which is used as a canonical representative of $e + \Lambda$.

Note that besides "canonicalizing" sums of exponent vectors, this approach also enables us to sample *uniformly* at random in $\mathrm{cl}(\mathcal{O})$: For simplicity (but by no means necessarily), assume that $\mathrm{cl}(\mathcal{O})$ is cyclic of order N with generator $\mathfrak{l}_j$; then we can sample from $\mathrm{cl}(\mathcal{O})$ uniformly at random by picking a uniform $u \in \{0, ..., N-1\}$, letting $e \in \mathbb{Z}^n$ be the exponent vector with all zeroes except for coefficient u at index j, and applying the rewriting algorithm to e.

The (perhaps only) downside is that some of these steps are computationally expensive: Computing the relation lattice means computing the class-group structure, which can be done in subexponential time using an algorithm of Hafner–McCurley [HM89], and solving the closest-vector problem during the rewriting step requires to precompute a short lattice basis which takes time exponential in the dimension n. CSI-FiSh has managed to solve all these problems in practice for the smallest CSIDH parameter set (CSIDH-512; see Section 11.4 for security considerations) and produced a proof-of-concept implementation taking less than 400 milliseconds to sign and verify with this instantiation [BKV19].

11.3 — Slow isogenies may be a good thing

Interestingly, the problems described in Sections 11.1 and 11.2 are not *just* an obstacle: They can also be used constructively as a hardness assumption for a cryptographic primitive known as *verifiable delay function* (VDF), which has blockchain[1] applications. Intuitively, it consists of a function which is slow to evaluate (even using arbitrary parallelism), but once the output is known, its correctness can be verified efficiently (potentially given a witness that is also generated by the evaluation algorithm). See [BBBF18] for a more detailed and formal description.

In the isogeny setting, such a construction can be obtained as follows [DMPS19]: Publish an isogeny $\varphi\colon E \to E'$ of large smooth degree, say ℓ^T for a small prime ℓ, and also publish a point $P \in E$ of large prime order N together with its image $\varphi(P) \in E'$ under the isogeny. The input to the VDF is a point $Q \in E'$ of order N, and evaluating the VDF means simply computing the image $\widehat{\varphi}(Q)$ of the input point Q under the dual isogeny $\widehat{\varphi}$. Doing so naïvely using repeated ℓ-isogeny steps clearly takes time $\Theta(T)$. To verify the output, anyone can compute the two pairings $e_N(P, \widehat{\varphi}(Q))$ and $e_N(\varphi(P), Q)$ and accept if and only if they match. Since taking duals is the adjoint with respect to the Weil pairing, this procedure works if $\widehat{\varphi}(Q)$ is correct. An attacker would like to obtain $\widehat{\varphi}(Q)$ without really going through the effort of evaluating $\widehat{\varphi}$ step by step: One way of doing this is to precompute $\widehat{\varphi}$ on a basis of the N-torsion, then solve a two-dimensional DLP on input Q, and simply return the appropriate linear combination of the precomputed images. Another attack avenue is to try and "shorten" the isogeny φ into an isogeny

[1] I said it!

with the same action on N-torsion that can be evaluated more efficiently. When using a CSIDH-style instantiation (which is advantageous since it allows for very compact representations of φ), finding such a shortcut essentially means going through the expensive computation decribed in Section 11.2, which is infeasible for sufficiently large parameters. (Note that this primitive is not post-quantum secure due to relying on the hardness of two-dimensional discrete logarithms, which can be computed in polynomial time using Shor's algorithm; see Section 2.6.2.)

11.4 — Quantum attacks on CSIDH

Starting with our publication of a preprint of the article [Cas+18] underlying Chapter 3, the quantum security level of CSIDH has been the subject of some controversy. At that time, all available sources stated only asymptotic security levels, which the first draft of [Cas+18] used in a naïve way to come up with preliminary parameter-size estimates. The Asiacrypt version of the paper (which Chapter 3 is based on) added an overview table covering the various estimates existing in the literature at that point. Moreover, the decision to focus on a 512-bit instantiation for the proof-of-concept implementation was in part made to ease comparison with earlier and parallel work [Kie17; DKS18], which had used the same field size.[2] The contemporary responses to these initial estimates are summarized in Remark 3.13.

In February 2019, Kuperberg gave a presentation at the AIM workshop *Quantum algorithms for analysis of public-key crypto* about his second — generalized and optimized — algorithm for the abelian hidden-shift problem (see Section 2.6.3). The second algorithm does not improve upon the complexity $2^{O(\sqrt{\log n})}$ of the first algorithm (see Theorem 2.84), but the hidden constants are smaller and the algorithm is much more configurable. While the article [Kup13] focuses primarily on asymptotic complexity, Kuperberg outlined several directions in which the algorithm could be optimized in terms of concrete cost, including in particular ways to trade off quantum for (presumably much cheaper) classical computation.

Peikert [Pei20] concretized these ideas and suggested example parametrizations of Kuperberg's algorithm together with detailed cost estimates in a particular model of quantum computation. His conclusion is that the number of queries to the CSIDH oracle for breaking CSIDH-512 is only about 2^{16}, and that the dominating cost factor for breaking this parameter set is incurred by the implementation of the group action in superposition, i.e., the component analyzed in Chapter 9. This leads to a total CSIDH-512 attack cost estimate of 2^{58} T-gate operations on approximately 2^{40} qubits,[3] while also using 2^{48} bits of quantum-accessible classical storage.

Bonnetain and Schrottenloher [BS20] considerably updated their original preprint [BS18] to rectify some of the shortcomings pointed out in [BLMP19], and the resulting conclusions are similar to those of [Pei20], albeit focusing on different tradeoffs that allow for much more expensive classical computation on the quantum computer's control hardware. Concretely, they claim an attack on CSIDH-512 that performs 2^{19} oracle queries using $2^{71.6}$ T-gate operations on $2^{15.3}$ qubits, and in addition requires 2^{86} classical operations.

In December 2020, Chávez-Saab, Chi-Domínguez, Jaques, and Rodríguez-Henríquez published a preprint [CCJR20] revisiting these cost estimates, including a more detailed analysis in the *depth* $\times$ *width* (DW) cost metric. For CSIDH-512, they state a *sieving* cost of 2^{21} oracle calls,

[2] Stolbunov's PhD thesis [Sto12] discusses sizes up to 428 bits.

[3] The number of qubits used by the oracle is not mentioned in [Pei20], but to achieve the number of T-gates cited from [BLMP19] in [Pei20], about the same number of qubits is required. Applying Bennett's depth–width tradeoff (halving the number of qubits multiplies the time by 1.5; cf. Section 9.11) yields, for instance, an estimate of $\approx 2^{73.8}$ T-gates on presumably much more realistic $\approx 2^{13}$ qubits.

using 2^{24} classical processors and 2^{13} qubits, resulting in a total cost of 2^{63} in the DW metric. Note that this does not include the cost of oracle calls in superposition, which (cf. Chapter 9) appear to be very significant.

Do these analyses imply that CSIDH-512 does not reach the attack costs required for the "level 1" security category used by NIST for its post-quantum cryptography standardization project [NIST16]? The cited sources affirm the answer is "yes". However, coming up with a fair comparison remains difficult as NIST's categories are phrased in terms of attacking AES-128 using generic attacks. The relevant algorithms for AES are very different in nature from the Kuperberg sieve, and (for lack of sufficiently advanced quantum hardware to run experiments on) it is at present unclear how to accurately model the costs of accessing different kinds of memory, and in particular the relative costs of quantum versus classical computation. (For example, [CCJR20] states that classical processors and qubits are given "equal weight" in the cost metric, which can only hold true in reality assuming *colossal* future improvements in quantum-computing technology in terms of both energy consumption and monetary investment.) In conclusion, while it is clearly possible to compare the costs of breaking CSIDH-512 and AES-128 in *some choice of cost model*, today's evidence is perfectly consistent with both the scenario that even a single CSIDH-512 oracle call in superposition incurs a higher real-world cost than breaking AES-128 using classical processors, as well as the scenario that CSIDH-512 will be broken within a minute on a pocket quantum calculator.

11.5 — The DDH problem for CM actions

We have established in Chapter 6 that being able to reliably break the group-action analogue of the computational Diffie–Hellman problem is sufficient to construct a quantum attack against the group-action analogue of the discrete logarithm problem with only polynomial overhead. (See Section 2.1.3 for some background.) However, this implies nothing about the hardness of the *decisional* Diffie–Hellman problem. This knowledge gap was partially filled in 2020 by [CSV20], which exhibits families of CM-action-based cryptosystems where one *can* efficiently solve DDH, with no indication that this affects the security of CDH or DLP in any way. Luckily, CSIDH is not a member of these (big) families where DDH is weak, and there are no known attacks against DDH or CDH short of simply recovering the secret key first.

The DDH attack is based on the centuries-old mathematical framework of *genus theory*, which relates to the two-torsion of ideal-class groups of imaginary quadratic number rings. It turns out that some of this two-torsion structure remains accessible even after passing from ideals to elliptic curves by applying the CM action: Concretely, [CSV20] shows how to evaluate the maps induced on $\mathscr{E}\ell\ell_k(\mathcal{O})$ by quadratic characters χ of $\mathrm{cl}(\mathcal{O})$ via the isomorphism from Theorem 2.55; that is, given nothing but the *curve* $[\mathfrak{a}]E_0$ it recovers the value of $\chi([\mathfrak{a}])$.[4] The computation requires polynomial time and breaks DDH with significant advantage, very similar to the classical example of computing Legendre symbols to solve DDH in $\mathbb{F}_p^*$.

The simple reason why the attack does not apply to CSIDH is that the class groups used in CSIDH always have odd order, hence there simply are no non-trivial quadratic characters. For systems where the attack does apply, a simple (but perhaps not computationally advantageous) fix is to restrict private keys to a subgroup where all quadratic characters are constant, much like DDH remains (pre-quantumly) unbroken in subgroups of $\mathbb{F}_p^*$ of large prime order.

[4] In the situation depicted in Figure 5.1, the unique nontrivial quadratic character essentially reveals whether we have jumped to the "opposite side" of the isogeny graph via $[\mathfrak{t}]$ or not.

11.6 — Faster isogeny evaluation: √élu

It seems that until recently, many isogeny researchers took the assumption for granted that Vélu's formulas (Proposition 2.31) are optimal in the sense that computing a (separable) isogeny from its kernel must inherently take time linear in the degree. This belief may be motivated in part by the fact that the object that is apparently being computed also has size linear in the degree: Explicit defining polynomials for the isogeny require linear space, hence writing them down will surely take at least linear time as well.

However, in most (if not all) known isogeny-based cryptographic protocols, the explicit defining polynomials are of little interest, and the core operations used in cryptography seem to be computing the image of a point under the isogeny and computing the coefficients of the codomain curve. Notice that the second task can be reduced to the first: Given a handful of points on an elliptic curve known to be of a specific form (e.g. a Weierstraß curve) easily permits interpolating the curve equation. Hence, the fundamental task is to compute the map

$$(E, P, \ell, Q) \mapsto \varphi_P(Q)\,,$$

where $P, Q \in E$ are points, P has finite order ℓ, and φ_P is a fixed isogeny with kernel $\langle P \rangle$, and there is no obvious lower bound for the complexity of this problem (except for the time it takes to read the input).

Vélu's original formulas [Vél71] and more modern and optimized versions following the same idea [Ren18; MS16] use $\Theta(\ell)$ base-field operations to compute an ℓ-isogeny. In 2020, the √élu algorithm [BDLS20] has finally improved the complexity to $\widetilde{O}(\sqrt{\ell})$ base-field operations, and the method is founded on an adaptation of known techniques to quickly evaluate polynomials whose roots are powers to the elliptic-curve setting. This is not just an asymptotic improvement: The speedup begins to kick in at isogeny degrees relevant for CSIDH, and the proof-of-concept implementation of [BDLS20] achieves a 1% speedup for CSIDH-512 with growing speedup factors the larger the parameter sizes become, already reaching an 8% speedup for CSIDH-1024. This is improved further in follow-up work [ACR20].

We remark that in principle, there is again no apparent reason for the complexity $\widetilde{O}(\sqrt{\ell})$ to be optimal. It does however seem that fundamental breakthroughs are needed to achieve isogeny evaluation in (for instance) a logarithmic or even constant number of base-field operations.

11.7 — Hardened CSIDH implementations

Section 3.8 described a simple proof-of-concept implementation of CSIDH without any attempt being made to resist side-channel attacks. Since then, several works on constant-time and/or optimized implementations of CSIDH have appeared. We note that turning Algorithm 3.2 into a constant-time algorithm in an efficient manner is non-trivial due to possible failures when sampling rational points of small order ℓ, and due to a potentially large (secret-dependent) search space of evaluation strategies for chained isogenies.

We briefly survey some works on constant-time CSIDH. All numbers given are for CSIDH-512, but note that the more recent works on the topic include bigger sizes in light of Section 11.4. In 2018, [MCR19] gave the first complete constant-time implementation of CSIDH by altering the proof-of-concept implementation of Chapter 3. Constant-timeness caused an almost threefold increase in runtime, and was achieved using generic branch-elimination techniques together with some optimizations specific to CSIDH: Restricting private keys to non-negative exponents to avoid case distinctions; "dummy isogenies", which exploit the fact that Vélu's formulas can be repurposed to (almost) compute a scalar multiplication; and "SIMBA", which reduces the time

spent on scalar multiplications by processing isogenies in smaller batches when running (a variant of) Algorithm 3.2. (The latter optimization is also relevant for variable-time CSIDH implementations.) These ideas found continued use in [OAYT19], which however reintroduced negative exponents, yielding a speedup of close to 30%. Both of these algorithms were revisited in [Cer+19], which improved the side-channel protections at the expense of some performance. All the works mentioned so far rely on seemingly rather ad-hoc optimizations. The underlying parameter spaces were formalized (in different ways) in [HLKA20] and [CR20], which phrase finding good strategies formally as an optimization problem and give improved solutions with speedups between 3% and 12%. Finally, the currently fastest constant-time CSIDH implementation was obtained in [ACR20] based on the $\sqrt{}$élu isogeny-evaluation algorithm (see Section 11.6). It saves between 3% and 5% for CSIDH-512, but as mentioned in Section 11.6, the speedup grows for bigger parameters (reaching around 25% for CSIDH-1792).

Given that the main disadvantage of CSIDH is its relatively bad performance, more work on optimized and side-channel resistant implementations is definitely necessary and important.

11.8 — Repeated isogenies from radicals

All the algorithms mentioned in Section 11.7 still follow the general layout of Algorithm 3.2, in the sense that they repeatedly sample points in the $+1$ or -1 eigenspace of Frobenius, project them to ℓ_i-torsion subgroups, and push points through isogenies to compute several different ℓ_i-isogenies using only one random point. Castryck, Decru, and Vercauteren [CDV20] propose a radically different approach, which (specialized to the CSIDH setting) allows computing the action of $[\mathfrak{l}]^e$ faster than computing the action of $[\mathfrak{l}]$ independently e times when ℓ is small. They achieve this by giving explicit formulas (involving polynomials and an ℓth root) for a rational point of order ℓ on the curve $E/\langle P\rangle$ when $P \in E$ is itself a rational point of order ℓ, which allows immediately applying $[\mathfrak{l}]$ again using Vélu's or the $\sqrt{}$élu formulas, without going through the effort of random sampling and cofactor multiplication (with a failure chance) again.

The net gain of this approach is a speedup of 19% over the plain variable-time $\sqrt{}$élu-based implementation of CSURF-512.[5]

[5] CSURF [CD20] is a slight variant of CSIDH that can use 2-isogenies in addition to odd ℓ_i by working with supersingular elliptic curves on the *surf*ace of the 2-isogeny volcano.

Summary

Cryptography on Isogeny Graphs

This thesis contains a variety of results on *isogeny-based cryptography*.

Conventional algorithms for asymmetric operations like public-key encryption or digital signatures rely on the (presumed) hardness of certain mathematical problems such as the factorization of integers into primes or the discrete-logarithm problem (DLP). However, once sufficiently powerful quantum computers become reality, some of the most commonly used underlying problems will be solvable in polynomial time using variants of a 1994 quantum algorithm by Shor, which in turn breaks a large fraction of the cryptography used on the internet today.

One particular, relatively new approach to replace quantum-vulnerable algorithms is based on *isogenies* between elliptic curves over finite fields. Isogenies are non-constant rational maps which are also group homomorphisms. Their very rich mathematical structure gives rise to new cryptosystems suitable as replacements for endangered algorithms, while (to the best of our knowledge) offering post-quantum security.

This work spans both facets of cryptologic research, focusing on isogeny-based cryptography in particular: It covers constructive contributions — building completely new functionality or making cryptosystems more efficient to use in practice — as well as cryptanalysis, which helps us understand the (in)security of these systems against powerful adversaries.

On the constructive side, we show how to construct an efficient one-way group action using isogeny graphs of supersingular elliptic curves defined over $\mathbb{F}_p$, now known as *CSIDH* /ˈsiːˌsaɪd/. We also describe an improved, more efficient version of *SeaSign*, a signature scheme using such group actions as a building block. Moreover, we give an algorithm to locate a supersingular curve in the $\mathbb{F}_p$-isogeny graph when given additional information about its ring of endomorphisms, which can be seen as an $\mathbb{F}_p$-analogue of an existing, very important algorithm for the $\mathbb{F}_{p^2}$ case and may have both constructive and destructive applications. Lastly, we give a short proof that the natural group-action analogues of the Diffie–Hellman and discrete-logarithm problems are polynomial-time equivalent under *quantum* reductions.

On the cryptanalysis side, we extend and improve so-called *torsion-point attacks* that apply to certain families of SIDH-like cryptosystems. We also give a summary of several other natural ideas to attack SIDH and explain why each of them appears to fail. Finally, we analyze the cost of evaluating the CSIDH group action on a quantum computer, an essential subroutine in a known subexponential-time attack whose cost has a significant impact on the total attack complexity.

Curriculum Vitae

Lorenz Panny (or, in other words: I) was born in 1994 in Eggenfelden, Bavaria, Germany.

After graduating from König-Karlmann-Gymnasium Altötting in 2011 with a programming project, he found himself struggling with the decision whether to study mathematics or computer science, therefore he opted for the natural choice — both, at TU München. His computer science studies ended in 2014 with a B.Sc. thesis on automated theorem proving at the Chair for Logic and Verification. In mathematics, his focus converged towards (algorithmic) algebra, culminating in a B.Sc. thesis on ℓ-adic point counting for elliptic curves in 2015, and a M.Sc. thesis on p-adic point counting for hyperelliptic curves in 2017. Just a few weeks later, he started his PhD studies in cryptology as an ECRYPT-NET fellow at TU Eindhoven, which have since led to the existence of this very thesis. During his PhD, he undertook research visits to the COSIC group at KU Leuven (yielding some of the work contained in this thesis), as well as to Simons Institute programs on lattices and quantum computing in Berkeley.

Outside academia, Lorenz frequently participates in and co-organizes hacking competitions known as "CTF" together with his team *hxp*, achieving fairly decent success in international championships. In a non-empty subset of the remaining time, he likes to play the drums.

Bibliography

[ACR20] Gora Adj, Jesús-Javier Chi-Domínguez and Francisco Rodríguez-Henríquez. *On new Vélu's formulae and their applications to CSIDH and B-SIDH constant-time implementations*. IACR Cryptology ePrint Archive 2020/1109. 2020. url: https://ia.cr/2020/1109.

[AD97] Miklós Ajtai and Cynthia Dwork. "A Public-Key Cryptosystem with Worst-Case/Average-Case Equivalence". In: *STOC*. ACM, 1997, pp. 284–293.

[Adj+18] 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: *Selected Areas in Cryptography – SAC 2018*. Vol. 11349. Lecture Notes in Computer Science. Springer, 2018, pp. 322–343. url: https://ia.cr/2018/313.

[ADPS16] Erdem Alkim, Léo Ducas, Thomas Pöppelmann and Peter Schwabe. "Post-quantum Key Exchange — A New Hope". In: *USENIX Security Symposium*. USENIX Association, 2016, pp. 327–343. url: https://ia.cr/2016/758.

[AJJS19] Reza Azarderakhsh, Amir Jalali, David Jao and Vladimir Soukharev. *Practical Supersingular Isogeny Group Key Agreement*. IACR Cryptology ePrint Archive 2019/330. 2019. url: https://ia.cr/2019/330.

[AJL17] Reza Azarderakhsh, David Jao and Christopher Leonardi. "Post-Quantum Static-Static Key Agreement Using Multiple Protocol Instances". In: *Selected Areas in Cryptography – SAC 2017*. Vol. 10719. Lecture Notes in Computer Science. Springer, 2017, pp. 45–63.

[Arp+19] Sarah Arpin, Catalina Camacho-Navarro, Kristin Lauter, Joelle Lim, Kristina Nelson, Travis Scholl and Jana Sotáková. *Adventures in Supersingularland*. IACR Cryptology ePrint Archive 2019/1056. 2019. url: https://ia.cr/2019/1056.

[Aza+16] Reza Azarderakhsh, David Jao, Kassem Kalach, Brian Koziel and Christopher Leonardi. "Key Compression for Isogeny-Based Cryptosystems". In: *AsiaPKC@AsiaCCS*. ACM, 2016, pp. 1–10. url: https://ia.cr/2016/229.

[Bac90] Eric Bach. "Explicit bounds for primality testing and related problems". In: *Mathematics of Computation* 55.191 (1990), pp. 355–380.

[Bar86] Paul Barrett. "Implementing the Rivest Shamir and Adleman Public Key Encryption Algorithm on a Standard Digital Signal Processor". In: *CRYPTO*. Vol. 263. Lecture Notes in Computer Science. Springer, 1986, pp. 311–323.

[BBBF18] Dan Boneh, Joseph Bonneau, Benedikt Bünz and Ben Fisch. "Verifiable Delay Functions". In: *CRYPTO (1)*. Vol. 10991. Lecture Notes in Computer Science. Springer, 2018, pp. 757–788. url: https://ia.cr/2018/601.

[BDLS20] Daniel J. Bernstein, Luca De Feo, Antonin Leroux and Benjamin Smith. "Faster computation of isogenies of large prime degree". In: *ANTS XIV: Proceedings of the fourteenth algorithmic number theory symposium*. Ed. by Steven Galbraith. Auckland, 2020. url: https://iac.r/2020/341.

[Ben73] Charles H. Bennett. "Logical Reversibility of Computation". In: *IBM Journal of Research and Development* 17 (1973), pp. 525–532.

[Ben89] Charles H. Bennett. "Time/Space Trade-Offs for Reversible Computation". In: *SIAM Journal on Computing* 18.4 (1989), pp. 766–776.

[Ber+14] Daniel J. Bernstein, Bernard van Gastel, Wesley Janssen, Tanja Lange, Peter Schwabe and Sjaak Smetsers. "TweetNaCl: A Crypto Library in 100 Tweets". In: *LATINCRYPT*. Vol. 8895. Lecture Notes in Computer Science. Springer, 2014, pp. 64–83. url: https://tweetnacl.cr.yp.to/.

[Ber06] Daniel J. Bernstein. "Curve25519: New Diffie–Hellman Speed Records". In: *Public Key Cryptography*. Vol. 3958. Lecture Notes in Computer Science. Springer, 2006, pp. 207–228. url: https://cr.yp.to/papers.html#curve25519.

[Ber09a] Daniel J. Bernstein. "Batch Binary Edwards". In: *CRYPTO*. Vol. 5677. Lecture Notes in Computer Science. Springer, 2009, pp. 317–336. url: https://cr.yp.to/papers.html#bbe.

[Ber09b] Daniel J. Bernstein. "Introduction to post-quantum cryptography". In: *Post-Quantum Cryptography*. Ed. by Daniel J. Bernstein, Johannes Buchmann and Erik Dahmen. 1st ed. Springer, 2009. isbn: 978-3-540-88702-7.

[BGLP18] Daniel J. Bernstein, Leon Groot Bruinderink, Tanja Lange and Lorenz Panny. "HILA5 Pindakaas: On the CCA Security of Lattice-Based Encryption with Error Correction". In: *AFRICACRYPT*. Vol. 10831. Lecture Notes in Computer Science. Springer, 2018, pp. 203–216. url: https://ia.cr/2017/1214.

[BHKL13] Daniel J. Bernstein, Mike Hamburg, Anna Krasnova and Tanja Lange. "Elligator: elliptic-curve points indistinguishable from uniform random strings". In: *ACM Conference on Computer and Communications Security*. ACM, 2013, pp. 967–980. url: https://ia.cr/2013/325.

[BIJ18] Jean-François Biasse, Annamaria Iezzi and Michael J. Jacobson, Jr. "A note on the security of CSIDH". In: *INDOCRYPT*. Vol. 11356. Lecture Notes in Computer Science. Springer, 2018, pp. 153–168. url: https://arxiv.org/abs/1806.03656.

[Bis12] Gaetan Bisson. "Computing endomorphism rings of elliptic curves under the GRH". In: *Journal of Mathematical Cryptology* 5.2 (2012), pp. 101–114. url: https://ia.cr/2011/042.

[BJS14] Jean-François Biasse, David Jao and Anirudh Sankar. "A Quantum Algorithm for Computing Isogenies between Supersingular Elliptic Curves". In: *INDOCRYPT*. Vol. 8885. Lecture Notes in Computer Science. Springer, 2014, pp. 428–442.

[BK81] Richard P. Brent and Hsiang-Tsung Kung. "The Area-Time Complexity of Binary Multiplication". In: *Journal of the ACM* 28.3 (1981), pp. 521–534. url: https://maths-people.anu.edu.au/~brent/pd/rpb055.pdf.

[BKV19] Ward Beullens, Thorsten Kleinjung and Frederik Vercauteren. "CSI-FiSh: Efficient Isogeny based Signatures through Class Group Computations". In: *ASIACRYPT (1)*. Vol. 11921. Lecture Notes in Computer Science. Springer, 2019, pp. 227–247. url: https://ia.cr/2019/498.

[BL07] Daniel J. Bernstein and Tanja Lange. "Faster Addition and Doubling on Elliptic Curves". In: *ASIACRYPT*. Vol. 4833. Lecture Notes in Computer Science. Springer, 2007, pp. 29–50. url: https://ia.cr/2007/286.

[BL08] Daniel J. Bernstein and Tanja Lange. "Analysis and optimization of elliptic-curve single-scalar multiplication". In: *Finite fields and applications 2007*. American Mathematical Society, 2008, pp. 1–19. isbn: 978-0-8218-4309-3/pbk. url: https://ia.cr/2007/455.

[BL13] Daniel J. Bernstein and Tanja Lange. "Non-uniform Cracks in the Concrete: The Power of Free Precomputation". In: *ASIACRYPT (2)*. Vol. 8270. Lecture Notes in Computer Science. Springer, 2013, pp. 321–340. url: https://ia.cr/2012/318.

[BL17] Daniel J. Bernstein and Tanja Lange. "Montgomery curves and the Montgomery ladder". In: *Topics in computational number theory inspired by Peter L. Montgomery*. Ed. by Joppe W. Bos and Arjen K. Lenstra. Cambridge University Press, 2017, pp. 82–115. url: https://ia.cr/2017/293.

[BL95] Dan Boneh and Richard J. Lipton. "Quantum Cryptanalysis of Hidden Linear Functions (Extended Abstract)". In: *CRYPTO*. Vol. 963. Lecture Notes in Computer Science. Springer, 1995, pp. 424–437. url: https://crypto.stanford.edu/~dabo/pubs/papers/quantum.pdf.

[BLMP19] Daniel J. Bernstein, Tanja Lange, Chloe Martindale and Lorenz Panny. "Quantum Circuits for the CSIDH: Optimizing Quantum Evaluation of Isogenies". In: *EUROCRYPT (2)*. Vol. 11477. Lecture Notes in Computer Science. Springer, 2019, pp. 409–441. url: https://ia.cr/2018/1059.

[BMSS08] Alin Bostan, François Morain, Bruno Salvy and Éric Schost. "Fast algorithms for computing isogenies between elliptic curves". In: *Mathematics of Computation* 77.263 (2008), pp. 1755–1778. url: https://www.ams.org/journals/mcom/2008-77-263/S0025-5718-08-02066-8/S0025-5718-08-02066-8.pdf.

[BN18] Xavier Bonnetain and María Naya-Plasencia. "Hidden Shift Quantum Cryptanalysis and Implications". In: *ASIACRYPT*. Vol. 11274. Lecture Notes in Computer Science. Springer, 2018, pp. 560–592. url: https://ia.cr/2018/432.

[Bon98] Dan Boneh. "The Decision Diffie–Hellman Problem". In: *ANTS*. Vol. 1423. Lecture Notes in Computer Science. Springer, 1998, pp. 48–63. url: https://crypto.stanford.edu/~dabo/abstracts/DDH.html.

[Bos14] Joppe W. Bos. "Constant time modular inversion". In: *Journal of Cryptographic Engineering* 4.4 (2014), pp. 275–281. url: http://joppebos.com/files/CTInversion.pdf.

[Bot+19] Paul Bottinelli, Victoria de Quehen, Chris Leonardi, Anton Mosunov, Filip Pawlega and Milap Sheth. *The Dark SIDH of Isogenies*. IACR Cryptology ePrint Archive 2019/1333. 2019. url: https://ia.cr/2019/1333.

[Brö08] Reinier Bröker. "A p-adic algorithm to compute the Hilbert class polynomial". In: *Mathematics of Computation* 77.264 (2008), pp. 2417–2435.

[Brö09] Reinier Bröker. "Constructing supersingular elliptic curves". In: *Journal of Combinatorics and Number Theory* 1.3 (2009), pp. 469–273.

[BS07] Reinier Bröker and Peter Stevenhagen. "Efficient CM-constructions of elliptic curves over finite fields". In: *Mathematics of Computation* 76.260 (2007), pp. 2161–2179.

[BS18] Xavier Bonnetain and André Schrottenloher. *Quantum Security Analysis of CSIDH and Ordinary Isogeny-based Schemes*. IACR Cryptology ePrint Archive 2018/537; version 20180621:135910. Newer version: [BS20]. 2018. url: https://eprint.iacr.org/2018/537/20180621:135910.

[BS20] Xavier Bonnetain and André Schrottenloher. "Quantum Security Analysis of CSIDH". In: *EUROCRYPT (2)*. Vol. 12106. Lecture Notes in Computer Science. Springer, 2020, pp. 493–522. url: https://ia.cr/2018/537.

[BS96] Wieb Bosma and Peter Stevenhagen. "On the computation of quadratic 2-class groups". In: *Journal de Théorie des Nombres de Bordeaux* 8.2 (1996), pp. 283–313.

[BV07] Johannes Buchmann and Ulrich Vollmer. *Binary quadratic forms: an algorithmic approach*. Vol. 20. Algorithms and Computation in Mathematics. Springer, 2007. isbn: 978-3-540-46367-2.

[BY90] Gilles Brassard and Moti Yung. "One-Way Group Actions". In: *CRYPTO*. Vol. 537. Lecture Notes in Computer Science. Springer, 1990, pp. 94–107.

[Cas+18] Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny and Joost Renes. "CSIDH: An Efficient Post-Quantum Commutative Group Action". In: *ASIACRYPT (3)*. Vol. 11274. Lecture Notes in Computer Science. Springer, 2018, pp. 395–427. url: https://ia.cr/2018/383.

[CCJR20] Jorge Chávez-Saab, Jesús-Javier Chi-Domínguez, Samuel Jaques and Francisco Rodríguez-Henríquez. *The SQALE of CSIDH: Square-root vélu Quantum-resistant isogeny Action with Low Exponents*. IACR Cryptology ePrint Archive 2020/1520. 2020. url: https://ia.cr/2020/1520.

[CD20] Wouter Castryck and Thomas Decru. "CSIDH on the Surface". In: *PQCrypto*. Vol. 12100. Lecture Notes in Computer Science. Springer, 2020, pp. 111–129. url: https://ia.cr/2019/1404.

[CDV20] Wouter Castryck, Thomas Decru and Frederik Vercauteren. "Radical Isogenies". In: *ASIACRYPT (2)*. Vol. 12492. Lecture Notes in Computer Science. Springer, 2020, pp. 493–519. url: https://ia.cr/2020/1108.

[Cer+19] Daniel Cervantes-Vázquez, Mathilde Chenu, Jesús-Javier Chi-Domínguez, Luca De Feo, Francisco Rodríguez-Henríquez and Benjamin Smith. "Stronger and Faster Side-Channel Protections for CSIDH". In: *LATINCRYPT*. Vol. 11774. Lecture Notes in Computer Science. Springer, 2019, pp. 173–193. url: https://ia.cr/2019/837.

[CH17] Craig Costello and Hüseyin Hişil. "A Simple and Compact Algorithm for SIDH with Arbitrary Degree Isogenies". In: *ASIACRYPT (2)*. Vol. 10625. Lecture Notes in Computer Science. Springer, 2017, pp. 303–329. url: https://ia.cr/2017/504.

[Che16] Yiping Cheng. *Space-Efficient Karatsuba Multiplication for Multi-Precision Integers*. 2016. arXiv: 1605.06760. url: https://arxiv.org/abs/1605.06760.

[CJS14] Andrew M. Childs, David Jao and Vladimir Soukharev. "Constructing elliptic curve isogenies in quantum subexponential time". In: *Journal of Mathematical Cryptology* 8.1 (2014), pp. 1–29. url: https://arxiv.org/abs/1012.4019.

[CK19] Leonardo Colò and David Kohel. "Orienting supersingular isogeny graphs". In: *NutMiC 2019*. 2019. url: https://ia.cr/2020/985.

[CL84] Henri Cohen and Hendrik W. Lenstra, Jr. "Heuristics on class groups of number fields". In: *Number Theory Noordwijkerhout 1983*. Ed. by Hendrik Jager. Springer, 1984, pp. 33–62. isbn: 978-3-540-38906-4.

[CLG09] Denis X. Charles, Kristin E. Lauter and Eyal Z. Goren. "Cryptographic Hash Functions from Expander Graphs". In: *Journal of Cryptology* 22.1 (2009), pp. 93–113. url: https://ia.cr/2006/021.

[CLN16] Craig Costello, Patrick Longa and Michael Naehrig. "Efficient Algorithms for Supersingular Isogeny Diffie–Hellman". In: *CRYPTO (1)*. Vol. 9814. Lecture Notes in Computer Science. Springer, 2016, pp. 572–601. url: https://ia.cr/2016/413.

[Con] Keith Conrad. *The conductor ideal*. Expository paper. url: https://kconrad.math.uconn.edu/blurbs/gradnumthy/conductor.pdf.

[Cos+17] Craig Costello, David Jao, Patrick Longa, Michael Naehrig, Joost Renes and David Urbanik. "Efficient Compression of SIDH Public Keys". In: *EUROCRYPT (1)*. Vol. 10210. Lecture Notes in Computer Science. 2017, pp. 679–706.

[Cos+20] Craig Costello, Patrick Longa, Michael Naehrig, Joost Renes and Fernando Virdia. "Improved Classical Cryptanalysis of SIKE in Practice". In: *Public Key Cryptography (2)*. Vol. 12111. Lecture Notes in Computer Science. Springer, 2020, pp. 505–534. url: https://ia.cr/2019/298.

[Cos18] Craig Costello. "Computing Supersingular Isogenies on Kummer Surfaces". In: *ASIACRYPT (3)*. Vol. 11274. Lecture Notes in Computer Science. Springer, 2018, pp. 428–456. url: https://ia.cr/2018/850.

[Cos20] Craig Costello. "B-SIDH: Supersingular Isogeny Diffie-Hellman Using Twisted Torsion". In: *ASIACRYPT (2)*. Vol. 12492. Lecture Notes in Computer Science. Springer, 2020, pp. 440–463. url: https://ia.cr/2019/1145.

[Cou06] Jean-Marc Couveignes. *Hard Homogeneous Spaces*. IACR Cryptology ePrint Archive 2006/291. 2006. url: https://ia.cr/2006/291.

[Cox13] David A. Cox. *Primes of the form $x^2 + ny^2$: Fermat, class field theory, and complex multiplication*. 2nd ed. Pure and Applied Mathematics. Wiley, 2013, pp. xviii+356. isbn: 978-1-118-39018-4.

[CPV20] Wouter Castryck, Lorenz Panny and Frederik Vercauteren. "Rational Isogenies from Irrational Endomorphisms". In: *EUROCRYPT (2)*. Vol. 12106. Lecture Notes in Computer Science. Springer, 2020, pp. 523–548. url: https://ia.cr/2019/1202.

[CR03] John Cremona and David Rusin. "Efficient solution of rational conics". In: *Mathematics of Computation* 72.243 (2003), pp. 1417–1441.

[CR15] Romain Cosset and Damien Robert. "Computing (ℓ, ℓ)-isogenies in polynomial time on Jacobians of genus 2 curves". In: *Mathematics of Computation* 84.294 (2015), pp. 1953–1975.

[CR20] Jesús-Javier Chi-Domínguez and Francisco Rodríguez-Henríquez. *Optimal strategies for CSIDH*. IACR Cryptology ePrint Archive 2020/417. 2020. url: https://ia.cr/2020/417.

[CS03] Ronald Cramer and Victor Shoup. "Design and Analysis of Practical Public-Key Encryption Schemes Secure against Adaptive Chosen Ciphertext Attack". In: *SIAM Journal on Computing* 33.1 (2003), pp. 167–226. url: https://ia.cr/2001/108.

[CS18] Craig Costello and Benjamin Smith. "Montgomery curves and their arithmetic: The case of large characteristic fields". In: *Journal of Cryptographic Engineering* 8.3 (2018), pp. 227–240. url: https://ia.cr/2017/212.

[CSV20] Wouter Castryck, Jana Sotáková and Frederik Vercauteren. "Breaking the Decisional Diffie–Hellman Problem for Class Group Actions Using Genus Theory". In: *CRYPTO (2)*. Vol. 12171. Lecture Notes in Computer Science. Springer, 2020, pp. 92–120. url: https://ia.cr/2020/151.

[Dad65] Luigi Dadda. "Some schemes for parallel multipliers". In: *Alta frequenza* 34.5 (1965), pp. 349–356.

[dB88] Bert den Boer. "Diffie–Hellman is as Strong as Discrete Log for Certain Primes". In: *CRYPTO*. Vol. 403. Lecture Notes in Computer Science. Springer, 1988, pp. 530–539.

[DeF+20] Luca De Feo, David Kohel, Antonin Leroux, Christophe Petit and Benjamin Wesolowski. "SQISign: Compact Post-quantum Signatures from Quaternions and Isogenies". In: *ASIACRYPT (1)*. Vol. 12491. Lecture Notes in Computer Science. Springer, 2020, pp. 64–93. url: https://ia.cr/2020/1240.

[DeF17] Luca De Feo. *Mathematics of Isogeny Based Cryptography*. Lecture notes for a summer school on mathematics for post-quantum cryptography. Thiès, Senegal, 2017. url: https://defeo.lu/ema2017/poly.pdf.

[Deu41] Max Deuring. "Die Typen der Multiplikatorenringe elliptischer Funktionenkörper". In: *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg* 14 (1941), pp. 197–272.

[DG16] Christina Delfs and Steven D. Galbraith. "Computing isogenies between supersingular elliptic curves over $\mathbb{F}_p$". In: *Designs, Codes and Cryptography* 78.2 (2016), pp. 425–440. url: https://arxiv.org/abs/1310.7789.

[DG19] Luca De Feo and Steven D. Galbraith. "SeaSign: Compact Isogeny Signatures from Class Group Actions". In: *EUROCRYPT (3)*. Vol. 11478. Lecture Notes in Computer Science. Springer, 2019, pp. 759–789. url: https://ia.cr/2018/824.

[DH76] Whitfield Diffie and Martin E. Hellman. "New directions in cryptography". In: *IEEE Trans. Information Theory* 22.6 (Nov. 1976), pp. 644–654.

[Din+17] Jintai Ding, Saed Alsayigh, R. V. Saraswathy, Scott R. Fluhrer and Xiaodong Lin. "Leakage of signal function with reused keys in RLWE key exchange". In: *ICC*. IEEE, 2017, pp. 1–6.

[DIZ07] Vassil S. Dimitrov, Laurent Imbert and Andrew Zakaluzny. "Multiplication by a Constant is Sublinear". In: *ARITH-18 2007*. 2007, pp. 261–268. url: http://www.lirmm.fr/~imbert/pdfs/constmult_arith18.pdf.

[DJP14] Luca De Feo, David Jao and Jérôme Plût. "Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies". In: *Journal of Mathematical Cryptology* 8.3 (2014), pp. 209–247. url: https://ia.cr/2011/506.

[DKS18] Luca De Feo, Jean Kieffer and Benjamin Smith. "Towards Practical Key Exchange from Ordinary Isogeny Graphs". In: *ASIACRYPT (3)*. Vol. 11274. Lecture Notes in Computer Science. Springer, 2018, pp. 365–394. url: https://ia.cr/2018/485.

[DMPS19] Luca De Feo, Simon Masson, Christophe Petit and Antonio Sanso. "Verifiable Delay Functions from Supersingular Isogenies and Pairings". In: *ASIACRYPT (1)*. Vol. 11921. Lecture Notes in Computer Science. Springer, 2019, pp. 248–277. url: https://ia.cr/2019/166.

[DN03] Claus Diem and Niko Naumann. "On the structure of Weil restrictions of abelian varieties". In: *Journal of the Ramanujan Mathematical Society* 18.2 (2003), pp. 153–174. url: https://arxiv.org/abs/math/0504359.

[DPV19] Thomas Decru, Lorenz Panny and Frederik Vercauteren. "Faster SeaSign Signatures Through Improved Rejection Sampling". In: *PQCrypto*. Vol. 11505. Lecture Notes in Computer Science. Springer, 2019, pp. 271–285. url: https://ia.cr/2018/1109.

[Eic38] Martin Eichler. "Über die Idealklassenzahl total definiter Quaternionenalgebren". In: *Mathematische Zeitschrift* 43.1 (Dec. 1938), pp. 102–109.

[Eis+18] Kirsten Eisenträger, Sean Hallgren, Kristin E. Lauter, Travis Morrison and Christophe Petit. "Supersingular Isogeny Graphs and Endomorphism Rings: Reductions and Solutions". In: *EUROCRYPT (3)*. Vol. 10822. Lecture Notes in Computer Science. Springer, 2018, pp. 329–368. url: https://ia.cr/2018/371.

[Eis+20] Kirsten Eisenträger, Sean Hallgren, Chris Leonardi, Travis Morrison and Jennifer Park. *Computing endomorphism rings of supersingular elliptic curves and connections to pathfinding in isogeny graphs*. 2020. arXiv: 2004.11495. url: https://arxiv.org/abs/2004.11495.

[EvM07] Bas Edixhoven, Gerard van der Geer and Ben Moonen. *Abelian varieties*. Book in preparation. 2007. url: https://www.math.ru.nl/personal/bmoonen/research.html#bookabvar.

[FHKP13] Eduarda S. V. Freire, Dennis Hofheinz, Eike Kiltz and Kenneth G. Paterson. "Non-Interactive Key Exchange". In: *Public Key Cryptography*. Vol. 7778. Lecture Notes in Computer Science. Springer, 2013, pp. 254–271. url: https://ia.cr/2012/732.

[Flu16] Scott R. Fluhrer. *Cryptanalysis of ring-LWE based key exchange with key share reuse*. IACR Cryptology ePrint Archive 2016/085. 2016. url: https://ia.cr/2016/085.

[FMMC12] Austin G. Fowler, Matteo Mariantoni, John M. Martinis and Andrew N. Cleland. "Surface codes: Towards practical large-scale quantum computation". In: *Physical Review A* 86.032324 (2012). url: https://arxiv.org/abs/1208.0928.

[FO99] Eiichiro Fujisaki and Tatsuaki Okamoto. “Secure Integration of Asymmetric and Symmetric Encryption Schemes”. In: *CRYPTO*. Vol. 1666. Lecture Notes in Computer Science. Springer, 1999, pp. 537–554.

[FS86] Amos Fiat and Adi Shamir. “How to Prove Yourself: Practical Solutions to Identification and Signature Problems”. In: *CRYPTO*. Vol. 263. Lecture Notes in Computer Science. Springer, 1986, pp. 186–194.

[Für07] Martin Fürer. “Faster integer multiplication”. In: *STOC*. ACM, 2007, pp. 57–66.

[Gal12] Steven D. Galbraith. *Mathematics of Public-Key Cryptography*. Cambridge University Press, 2012. isbn: 978-1-107-01392-6.

[Gal99] Steven D. Galbraith. “Constructing Isogenies between Elliptic Curves Over Finite Fields”. In: *LMS Journal of Computation and Mathematics* 2 (1999), pp. 118–138.

[GG13] Joachim von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra*. 3rd ed. Cambridge University Press, 2013. isbn: 978-1-139-85606-5.

[Gid17] Craig Gidney. “Halving the cost of quantum addition”. In: *Quantum* 2.74 (2017). url: https://quantum-journal.org/papers/q-2018-06-18-74/.

[GLRS16] Markus Grassl, Brandon Langenberg, Martin Roetteler and Rainer Steinwandt. “Applying Grover’s Algorithm to AES: Quantum Resource Estimates”. In: *PQCrypto*. Vol. 9606. Lecture Notes in Computer Science. Springer, 2016, pp. 29–43. url: https://arxiv.org/abs/1512.04965.

[GPS17] Steven D. Galbraith, Christophe Petit and Javier Silva. “Identification Protocols and Signature Schemes Based on Supersingular Isogeny Problems”. In: *ASIACRYPT (1)*. Vol. 10624. Lecture Notes in Computer Science. Springer, 2017, pp. 3–33. url: https://ia.cr/2016/1154.

[GPST16] Steven D. Galbraith, Christophe Petit, Barak Shani and Yan Bo Ti. “On the Security of Supersingular Isogeny Cryptosystems”. In: *ASIACRYPT (1)*. Vol. 10031. Lecture Notes in Computer Science. Springer, 2016, pp. 63–91. url: https://ia.cr/2016/859.

[GPSV18] Steven D. Galbraith, Lorenz Panny, Benjamin Smith and Frederik Vercauteren. *Quantum Equivalence of the DLP and CDHP for Group Actions*. IACR Cryptology ePrint Archive 2018/1199. 2018. url: https://ia.cr/2018/1199.

[GR04] Steven Galbraith and Victor Rotger. “Easy decision Diffie–Hellman groups”. In: *LMS Journal of Computation and Mathematics* 7 (2004), pp. 201–218. url: https://ia.cr/2004/070.

[Gro96] Lov K. Grover. “A Fast Quantum Mechanical Algorithm for Database Search”. In: *STOC*. ACM, 1996, pp. 212–219. url: https://arxiv.org/abs/quant-ph/9605043.

[GV18] Steven D. Galbraith and Frederik Vercauteren. “Computational problems in supersingular elliptic curve isogenies”. In: *Quantum Information Processing* 17 (2018). url: https://ia.cr/2017/774.

[GZ85] Benedict H. Gross and Don B. Zagier. “On singular moduli”. In: *Journal für die Reine und Angewandte Mathematik*. 355 (1985), pp. 191–220.

[Hal05] Sean Hallgren. “Fast quantum algorithms for computing the unit group and class group of a number field”. In: *STOC*. ACM, 2005, pp. 468–474. url: http://cse.psu.edu/~sjh26/unitgroup.pdf.

[Har77] Robin Hartshorne. *Algebraic Geometry*. 1st ed. Graduate Texts in Mathematics 52. Springer, 1977. isbn: 978-1-4419-2807-8.

[Has36] Helmut Hasse. "Zur Theorie der abstrakten elliptischen Funktionenkörper III. Die Struktur des Meromorphismenrings. Die Riemannsche Vermutung." In: *Journal für die reine und angewandte Mathematik* 175 (1936), pp. 193–208.

[HGS99] Chris Hall, Ian Goldberg and Bruce Schneier. "Reaction Attacks against several Public-Key Cryptosystems". In: *ICICS*. Vol. 1726. Lecture Notes in Computer Science. Springer, 1999, pp. 2–12.

[HH18] David Harvey and Joris van der Hoeven. *Faster integer multiplication using short lattice vectors*. 2018. arXiv: 1802.07932. url: https://arxiv.org/abs/1802.07932.

[HHK17] Dennis Hofheinz, Kathrin Hövelmanns and Eike Kiltz. "A Modular Analysis of the Fujisaki–Okamoto Transformation". In: *TCC (1)*. Vol. 10677. Lecture Notes in Computer Science. Springer, 2017, pp. 341–371. url: https://ia.cr/2017/604.

[HHL16] David Harvey, Joris van der Hoeven and Grégoire Lecerf. "Even faster integer multiplication". In: *Journal of Complexity* 36 (2016), pp. 1–30. url: https://arxiv.org/abs/1407.3360.

[HHL17] David Harvey, Joris van der Hoeven and Grégoire Lecerf. "Faster Polynomial Multiplication over Finite Fields". In: *Journal of the ACM* 63.6 (2017), 52:1–52:23. url: https://arxiv.org/abs/1407.3361.

[Hiş10] Hüseyin Hişil. "Elliptic curves, group law, and efficient computation". PhD thesis. Queensland University of Technology, 2010. url: https://eprints.qut.edu.au/33233/.

[HLKA20] Aaron Hutchinson, Jason T. LeGrow, Brian Koziel and Reza Azarderakhsh. "Further Optimizations of CSIDH: A Systematic Approach to Efficient Strategies, Permutations, and Bound Vectors". In: *ACNS (1)*. Vol. 12146. Lecture Notes in Computer Science. Springer, 2020, pp. 481–501. url: https://ia.cr/2019/1121.

[HM89] James L. Hafner and Kevin S. McCurley. "A rigorous subexponential algorithm for computation of class groups". In: *Journal of the American Mathematical Society* 2.4 (1989), pp. 837–850. issn: 0894–0347.

[How+03] Nick Howgrave-Graham, Phong Q. Nguyen, David Pointcheval, John Proos, Joseph H. Silverman, Ari Singer and William Whyte. "The Impact of Decryption Failures on the Security of NTRU Encryption". In: *CRYPTO*. Vol. 2729. Lecture Notes in Computer Science. Springer, 2003, pp. 226–246.

[HPS98] Jeffrey Hoffstein, Jill Pipher and Joseph H. Silverman. "NTRU: A Ring-Based Public Key Cryptosystem". In: *ANTS*. Vol. 1423. Lecture Notes in Computer Science. Springer, 1998, pp. 267–288.

[HRS17] Thomas Häner, Martin Roetteler and Krysta M. Svore. "Factoring using $2n+2$ qubits with Toffoli based modular multiplication". In: *Quantum Information & Computation* 17.7&8 (2017), pp. 673–684. url: https://arxiv.org/abs/1611.07995.

[HS00] Jeffrey Hoffstein and Joseph H. Silverman. *Reaction Attacks Against the NTRU Public Key Cryptosystem*. NTRU Cryptosystems Technical Report 015, version 2. 2000. url: https://web.archive.org/web/http://www.ntru.com/NTRUFTPDocsFolder/NTRUTech015.pdf.

[HS15] Michael Hutter and Peter Schwabe. "Multiprecision multiplication on AVR revisited". In: *Journal of Cryptographic Engineering* 5.3 (2015), pp. 201–214. url: https://ia.cr/2014/592.

[IJ13] Sorina Ionica and Antoine Joux. "Pairing the volcano". In: *Mathematics of Computation* 82.281 (2013), pp. 581–603.

[IR93] Gábor Ivanyos and Lajos Rónyai. "Finding maximal orders in semisimple algebras over $\mathbb{Q}$". In: *Computational Complexity* 3.3 (1993), pp. 245–261.

[Jao+17] David Jao, Reza Azarderakhsh, Matthew Campagna, Craig Costello, Luca De Feo, Basil Hess, Amir Jalali, Brian Koziel, Brian LaMacchia, Patrick Longa, Michael Naehrig, Joost Renes, Vladimir Soukharev and David Urbanik. *Supersingular Isogeny Key Encapsulation*. Submission to [NIST16]. 2017. url: https://sike.org.

[Jao+19] David Jao, Reza Azarderakhsh, Matthew Campagna, Craig Costello, Luca De Feo, Basil Hess, Amir Jalali, Brian Koziel, Brian LaMacchia, Patrick Longa, Michael Naehrig, Joost Renes, Vladimir Soukharev and David Urbanik. *Supersingular Isogeny Key Encapsulation*. Update of [Jao+17] for round 2 of [NIST16]. 2019. url: https://sike.org.

[JD11] David Jao and Luca De Feo. "Towards Quantum-Resistant Cryptosystems from Supersingular Elliptic Curve Isogenies". In: *PQCrypto*. Vol. 7071. Lecture Notes in Computer Science. Springer, 2011, pp. 19–34. url: https://ia.cr/2011/506.

[JLLR18] David Jao, Jason LeGrow, Christopher Leonardi and Luis Ruiz-Lopez. "A subexponential-time, polynomial quantum space algorithm for inverting the CM group action". In: *MathCrypt 2018* (2018). To appear.

[JMV09] David Jao, Stephen D. Miller and Ramarathnam Venkatesan. "Expander graphs based on GRH with an application to elliptic curve cryptography". In: *Journal of Number Theory* 129.6 (2009), pp. 1491–1504. url: https://arxiv.org/abs/0811.0647.

[Jon12] Cody Jones. "Low-overhead constructions for the fault-tolerant Toffoli gate". In: *Physical Review A* 87.022328 (2012).

[JS19] Samuel Jaques and John M. Schanck. "Quantum Cryptanalysis in the RAM Model: Claw-Finding Attacks on SIKE". In: *CRYPTO 2019*. Ed. by Alexandra Boldyreva and Daniele Micciancio. Vol. 11692. Lecture Notes in Computer Science. Springer, 2019, pp. 32–61. url: https://ia.cr/2019/103.

[JS20] Samuel Jaques and André Schrottenloher. "Low-gate Quantum Golden Collision Finding". In: *Selected Areas in Cryptography – SAC 2020*. 2020. url: https://ia.cr/2020/424.

[Kan89] Masanobu Kaneko. "Supersingular j-invariants as singular moduli mod p". In: *Osaka Journal of Mathematics* 26.4 (Jan. 1989), pp. 849–855. issn: 0030-6126.

[Kie17] Jean Kieffer. "Étude et accélération du protocole d'échange de clés de Couveignes–Rostovtsev–Stolbunov". Mémoire du Master 2. Université Paris VI, 2017. url: https://arxiv.org/abs/1804.10128.

[Kit96] Alexei Y. Kitaev. "Quantum measurements and the Abelian Stabilizer Problem". In: *Electronic Colloquium on Computational Complexity (ECCC)* 3.3 (1996). url: https://eccc.hpi-web.de/eccc-reports/1996/TR96-003.

[KLPT14] David Kohel, Kristin Lauter, Christophe Petit and Jean-Pierre Tignol. "On the quaternion ℓ-isogeny path problem". In: *LMS Journal of Computation and Mathematics* 17 (2014), pp. 418–432. url: https://ia.cr/2014/505.

[Kni95] Emanuel Knill. *An analysis of Bennett's pebble game*. 1995. arXiv: math/9508218. url: https://arxiv.org/abs/math/9508218.

[Knu81] Donald E. Knuth. *The Art of Computer Programming, Volume II: Seminumerical Algorithms*. 2nd ed. Addison-Wesley, 1981. isbn: 0-201-03822-6.

[KO63] Anatoly A. Karatsuba and Y. Ofman. "Multiplication of multidigit numbers on automata". In: *Soviet Physics Doklady* 7 (1963), pp. 595–596.

[Koh96] David Kohel. "Endomorphism rings of elliptic curves over finite fields". PhD thesis. University of California at Berkeley, 1996.
url: http://iml.univ-mrs.fr/~kohel/pub/thesis.pdf.

[KS15] Shane Kepley and Rainer Steinwandt. "Quantum circuits for $\mathbb{F}_{2^n}$-multiplication with subquadratic gate count". In: *Quantum Information Processing* 14.7 (2015), pp. 2373–2386.

[Kup05] Greg Kuperberg. "A Subexponential-Time Quantum Algorithm for the Dihedral Hidden Subgroup Problem". In: *SIAM Journal on Computing* 35.1 (2005), pp. 170–188. url: https://arxiv.org/abs/quant-ph/0302112.

[Kup13] Greg Kuperberg. "Another Subexponential-time Quantum Algorithm for the Dihedral Hidden Subgroup Problem". In: *TQC*. Vol. 22. LIPIcs. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2013, pp. 20–34.
url: https://arxiv.org/abs/1112.3333.

[Kut+20] Péter Kutas, Chloe Martindale, Lorenz Panny, Christophe Petit and Katherine E. Stange. *Weak instances of SIDH variants under improved torsion-point attacks*. IACR Cryptology ePrint Archive 2020/633. 2020. url: https://ia.cr/2020/633.

[Lan87] Serge Lang. *Elliptic functions*. 2nd ed. Vol. 112. Graduate Texts in Mathematics. With an appendix by John Tate. Springer, 1987, pp. xii+326.

[LB20] Jonathan Love and Dan Boneh. "Supersingular Curves With Small Non-integer Endomorphisms". In: *ANTS XIV: Proceedings of the fourteenth algorithmic number theory symposium*. Ed. by Steven Galbraith. Auckland, 2020.
url: https://arxiv.org/abs/1910.03180.

[Lef03] Vincent Lefèvre. "Multiplication by an Integer Constant: Lower Bounds on the Code Length". In: *5th Conference on Real Numbers and Computers 2003 – RNC5*. Lyon, France, 2003, pp. 131–146. url: https://hal.inria.fr/inria-00099684.

[Len96] Hendrik W. Lenstra, Jr. "Complex Multiplication Structure of Elliptic Curves". In: *Journal of Number Theory* 56.2 (1996), pp. 227–241. issn: 0022-314X.

[LLL82] Hendrik W. Lenstra, Jr., Arjen K. Lenstra and László Lovász. "Factoring Polynomials with Rational Coefficients." In: *Mathematische Annalen* 261 (1982), pp. 515–534. url: http://eudml.org/doc/182903.

[LST64] Jonathan Lubin, Jean-Pierre Serre and John Tate. *Elliptic Curves and Formal Groups*. Lecture notes. 1964. url: https://ma.utexas.edu/users/voloch/lst.html.

[Lyu09] Vadim Lyubashevsky. "Fiat-Shamir with aborts: Applications to lattice and factoring-based signatures". In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2009, pp. 598–616.

[Magma] Wieb Bosma, John Cannon and Catherine Playoust. "The Magma algebra system I: The user language". In: *Journal of Symbolic Computation* 24.3-4 (1997), pp. 235–265.

[Mar18a] Daniel A. Marcus. *Number fields*. 2nd ed. Universitext. With a foreword by Barry Mazur. Springer, 2018, pp. xviii+203.

[Mar18b] Chloe Martindale. "Isogeny graphs, modular polynomials, and applications". PhD thesis. Universiteit Leiden and Université de Bordeaux, 2018. url: https://martindale.info/research/Thesis.pdf.

[Mau94] Ueli M. Maurer. "Towards the Equivalence of Breaking the Diffie–Hellman Protocol and Computing Discrete Logarithms". In: *CRYPTO*. Vol. 839. Lecture Notes in Computer Science. Springer, 1994, pp. 271–281.

[McM14] Ken McMurdy. *Explicit representation of the endomorphism rings of supersingular elliptic curves*. Preprint. 2014. url: https://phobos.ramapo.edu/~kmcmurdy/research/McMurdy-ssEndoRings.pdf.

[MCR19] Michael Meyer, Fabio Campos and Steffen Reith. "On Lions and Elligators: An Efficient Constant-Time Implementation of CSIDH". In: *PQCrypto*. Vol. 11505. Lecture Notes in Computer Science. Springer, 2019, pp. 307–325. url: https://ia.cr/2018/1198.

[Mes72] William Messing. *The crystals associated to Barsotti–Tate groups: with applications to abelian schemes*. Vol. 264. Lecture Notes in Mathematics. Springer, 1972, pp. iii+190.

[Mic01] Daniele Micciancio. "Improving Lattice Based Cryptosystems Using the Hermite Normal Form". In: *CaLC*. Vol. 2146. Lecture Notes in Computer Science. Springer, 2001, pp. 126–145. url: https://cseweb.ucsd.edu/~daniele/papers/HNFcrypt.html.

[Mil85] Victor S. Miller. "Use of Elliptic Curves in Cryptography". In: *CRYPTO*. Vol. 218. Lecture Notes in Computer Science. Springer, 1985, pp. 417–426.

[Mon85] Peter L. Montgomery. "Modular multiplication without trial division". In: *Mathematics of Computation* 44 (1985), pp. 519–521. url: http://www.ams.org/journals/mcom/1985-44-170/S0025-5718-1985-0777282-X/home.html.

[Mon87] Peter L. Montgomery. "Speeding the Pollard and elliptic curve methods of factorization". In: *Mathematics of Computation* 48.177 (1987), pp. 243–264.

[Mor61] Louis J. Mordell. "The congruence $(p-1/2)! \equiv \pm 1 \bmod p$". In: *American Mathematical Monthly* 68.2 (1961), pp. 145–146.

[MP19] Chloe Martindale and Lorenz Panny. "How to not break SIDH". In: *CFAIL 2019*. New York, 2019. url: https://ia.cr/2019/558.

[MR18] Michael Meyer and Steffen Reith. "A Faster Way to the CSIDH". In: *INDOCRYPT*. Vol. 11356. Lecture Notes in Computer Science. Springer, 2018, pp. 137–152. url: https://ia.cr/2018/782.

[MS16] Dustin Moody and Daniel Shumow. "Analogues of Vélu's formulas for isogenies on alternate models of elliptic curves". In: *Mathematics of Computation* 85.300 (2016), pp. 1929–1951. url: https://ia.cr/2011/430.

[MW96] Ueli M. Maurer and Stefan Wolf. "Diffie–Hellman Oracles". In: *CRYPTO*. Vol. 1109. Lecture Notes in Computer Science. Springer, 1996, pp. 268–282.

[NC11] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. 10th ed. USA: Cambridge University Press, 2011. isbn: 1107002176.

[Nic71] Peter J. Nicholson. "Algebraic Theory of Finite Fourier Transforms". In: *Journal of Computer and System Sciences* 5.5 (1971), pp. 524–547.

[NIST16] National Institute of Standards and Technology. *Post-Quantum Cryptography Standardization*. Dec. 2016. url: https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Post-Quantum-Cryptography-Standardization.

[NV10] Phong Q. Nguyen and Brigitte Vallée, eds. *The LLL Algorithm. Survey and Applications*. Springer, 2010. isbn: 978-3-642-02295-1.

[OAYT19] Hiroshi Onuki, Yusuke Aikawa, Tsutomu Yamazaki and Tsuyoshi Takagi. "(Short Paper) A Faster Constant-Time Algorithm of CSIDH Keeping Two Points". In: *IWSEC*. Vol. 11689. Lecture Notes in Computer Science. Springer, 2019, pp. 23–33. url: https://ia.cr/2019/353.

[Onu20] Hiroshi Onuki. *On oriented supersingular elliptic curves*. 2020. arXiv: 2002.09894. url: https://arxiv.org/abs/2002.09894.

[Oor74] Frans Oort. "Subvarieties of moduli spaces". In: *Inventiones Mathematicae* 24 (1974), pp. 95–119.

[OT20] Hiroshi Onuki and Tsuyoshi Takagi. *On Collisions Related to an Ideal Class of Order 3 in CSIDH*. 2020. url: https://ia.cr/2019/1209.

[Pan20] Lorenz Panny. "Guess what?! On the impossibility of unconditionally secure public-key encryption". In: *Mathematical Cryptology* 1 (2020), pp. 1–7. url: https://ia.cr/2019/1228.

[Pari] Christian Batut, Karim Belabas, Dominique Bernardi, Henri Cohen and Michel Olivier. *User's Guide to PARI-GP*. Université de Bordeaux I.

[Pei14] Chris Peikert. "Lattice Cryptography for the Internet". In: *PQCrypto*. Vol. 8772. Lecture Notes in Computer Science. Springer, 2014, pp. 197–219.

[Pei20] Chris Peikert. "He Gives C-Sieves on the CSIDH". In: *EUROCRYPT (2)*. Vol. 12106. Lecture Notes in Computer Science. Springer, 2020, pp. 463–492. url: https://ia.cr/2019/725.

[Pet17] Christophe Petit. "Faster Algorithms for Isogeny Problems Using Torsion Point Images". In: *ASIACRYPT (2)*. Vol. 10625. Lecture Notes in Computer Science. Springer, 2017, pp. 330–353. url: https://ia.cr/2017/571.

[Piz90] Arnold K. Pizer. "Ramanujan graphs and Hecke operators". In: *Bulletin of the American Mathematical Society* 23.1 (1990), pp. 127–137. url: https://projecteuclid.org/euclid.bams/1183555725.

[PL17] Christophe Petit and Kristin E. Lauter. *Hard and Easy Problems for Supersingular Isogeny Graphs*. IACR Cryptology ePrint Archive 2017/962. 2017. url: https://ia.cr/2017/962.

[Pol71] John M. Pollard. "The fast Fourier transform in a finite field". In: *Mathematics of Computation* 25 (1971), pp. 365–374. url: https://www.ams.org/journals/mcom/1971-25-114/S0025-5718-1971-0301966-0/.

[PR15] Julia Pieltant and Hugues Randriam. "New uniform and asymptotic upper bounds on the tensor rank of multiplication in extensions of finite fields". In: *Mathematics of Computation* 84.294 (2015), pp. 2023–2045. url: https://arxiv.org/abs/1305.5166.

[PRM17] Alex Parent, Martin Roetteler and Michele Mosca. "Improved reversible and quantum circuits for Karatsuba-based integer multiplication". In: *TQC*. Vol. 73. LIPIcs. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017, 7:1–7:15. url: https://arxiv.org/abs/1706.03419.

[Reg04] Oded Regev. *A Subexponential Time Algorithm for the Dihedral Hidden Subgroup Problem with Polynomial Space*. 2004. arXiv: quant-ph/0406151. url: https://arxiv.org/abs/quant-ph/0406151.

[Ren18] Joost Renes. "Computing Isogenies Between Montgomery Curves Using the Action of $(0, 0)$". In: *PQCrypto*. Vol. 10786. Lecture Notes in Computer Science. Springer, 2018, pp. 229–247. url: https://ia.cr/2017/1198.

[RNSL17] Martin Roetteler, Michael Naehrig, Krysta M. Svore and Kristin E. Lauter. "Quantum Resource Estimates for Computing Elliptic Curve Discrete Logarithms". In: *ASIACRYPT (2)*. Vol. 10625. Lecture Notes in Computer Science. Springer, 2017, pp. 241–270. url: https://ia.cr/2017/598.

[RS06] Alexander Rostovtsev and Anton Stolbunov. *Public-Key Cryptosystem Based on Isogenies*. IACR Cryptology ePrint Archive 2006/145. 2006. url: https://ia.cr/2006/145.

[Saa17a] Markku-Juhani O. Saarinen. *HILA5: Key Encapsulation Mechanism (KEM) and Public Key Encryption Algorithm*. Submission to [NIST16]. 2017. url: https://github.com/mjosaarinen/hila5/blob/master/Supporting_Documentation/hila5spec.pdf.

[Saa17b] Markku-Juhani O. Saarinen. "HILA5: On Reliability, Reconciliation, and Error Correction for Ring-LWE Encryption". In: *Selected Areas in Cryptography – SAC 2017*. Ed. by Carlisle Adams and Jan Camenisch. Vol. 10719. Lecture Notes in Computer Science. Ottawa: Springer, 2017, pp. 192–212. isbn: 978-3-319-72564-2.

[Sage] The Sage Developers. *SageMath, the Sage Mathematics Software System*. url: https://sagemath.org.

[Sat00] Takakazu Satoh. "The canonical lift of an ordinary elliptic curve over a finite field and its point counting". In: *Journal of the Ramanujan Mathematical Society* 15.4 (2000), pp. 247–270.

[Sch77] Arnold Schönhage. "Schnelle Multiplikation von Polynomen über Körpern der Charakteristik 2". In: *Acta Informatica* 7 (1977), pp. 395–398.

[Sch85] René Schoof. "Elliptic curves over finite fields and the computation of square roots mod p". In: *Mathematics of Computation* 44.170 (May 1985), pp. 483–483.

[Sch87] René Schoof. "Nonsingular plane cubic curves over finite fields". In: *Journal of Combinatorial Theory, Series A* 46.2 (1987), pp. 183–211.

[SE94] Claus-Peter Schnorr and M. Euchner. "Lattice basis reduction: Improved practical algorithms and solving subset sum problems". In: *Mathematical Programming* 66 (1994), pp. 181–199.

[SGP19] Rajeev Anand Sahu, Agnese Gini and Ankan Pal. *Supersingular Isogeny-Based Designated Verifier Blind Signature*. IACR Cryptology ePrint Archive 2019/1498. 2019. url: https://ia.cr/2019/1498.

[Sha71] Daniel Shanks. "Class number, a theory of factorization, and genera". In: *Proceedings of Symposia in Pure Mathematics*. Vol. 20. 1971, pp. 415–440.

[Sho94] Peter W. Shor. "Algorithms for Quantum Computation: Discrete Logarithms and Factoring". In: *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*. SFCS '94. USA: IEEE Computer Society, 1994, pp. 124–134. isbn: 0818665807.

[Sho97a] Peter W. Shor. "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer". In: *SIAM Journal on Computing* 26.5 (1997), pp. 1484–1509. url: https://arxiv.org/abs/quant-ph/9508027.

[Sho97b] Victor Shoup. "Lower Bounds for Discrete Logarithms and Related Problems". In: *EUROCRYPT*. Vol. 1233. Lecture Notes in Computer Science. Springer, 1997, pp. 256–266.

[Sie35] Carl Siegel. "Über die Classenzahl quadratischer Zahlkörper". In: *Acta Arithmetica* 1.1 (1935), pp. 83–86.

[Sil09] Joseph H. Silverman. *The arithmetic of elliptic curves*. 2nd ed. Graduate Texts in Mathematics 106. Errata: [Sil15]. Springer, 2009. isbn: 978-0-387-09493-9.

[Sil15] Joseph H. Silverman. *Errata and Corrections to The Arithmetic of Elliptic Curves, 2nd Edition*. Apr. 2015. url: https://www.math.brown.edu/~jhs/AEC/AECErrata.pdf.

[Sim05] Denis Simon. *Quadratic equations in dimensions 4, 5 and more*. Preprint. 2005.

[SKPS19] Cyprien Delpech de Saint Guilhem, Péter Kutas, Christophe Petit and Javier Silva. *SÉTA: Supersingular Encryption from Torsion Attacks*. IACR Cryptology ePrint Archive 2019/1291. 2019. url: https://ia.cr/2019/1291.

[Smi18] Benjamin Smith. "Pre- and Post-quantum Diffie–Hellman from Groups, Actions, and Isogenies". In: *WAIFI*. Vol. 11321. Lecture Notes in Computer Science. Springer, 2018, pp. 3–40. url: https://ia.cr/2018/882.

[Smi20] Benjamin Smith. *Isogenies: what now, and what next?* Invited talk at PQCrypto 2020. Sept. 2020. url: https://youtu.be/HfmvNenGyok?t=3190.

[SS71] Arnold Schönhage and Volker Strassen. "Schnelle Multiplikation großer Zahlen". In: *Computing* 7.3-4 (1971), pp. 281–292.

[Sto04] Anton Stolbunov. "Public-key encryption based on cycles of isogenous elliptic curves". In Russian. MA thesis. Saint-Petersburg State Polytechnical University, 2004.

[Sto10] Anton Stolbunov. "Constructing public-key cryptographic schemes based on class group action on a set of isogenous elliptic curves". In: *Advances in Mathematics of Communications* 4.2 (2010), pp. 215–235.

[Sto12] Anton Stolbunov. "Cryptographic Schemes Based on Isogenies". PhD thesis. Norwegian University of Science and Technology, 2012.

[Str83] Volker Strassen. "The computational complexity of continued fractions". In: *SIAM Journal on Computing* 12 (1983), pp. 1–27.

[Suto7] Andrew V. Sutherland. "Order computations in generic groups". PhD thesis. Massachusetts Institute of Technology, 2007.

[Sut12a] Andrew V. Sutherland. "Identifying supersingular elliptic curves". In: *LMS Journal of Computation and Mathematics* 15 (2012), pp. 317–325.
url: https://arxiv.org/abs/1107.1140.

[Sut12b] Andrew V. Sutherland. "Isogeny volcanoes". In: *ANTS X*. Vol. 1. The Open Book Series. Mathematical Sciences Publishers, 2012, pp. 507–530.
url: https://arxiv.org/abs/1208.5370.

[Tano7] Seiichiro Tani. "An Improved Claw Finding Algorithm Using Quantum Walk". In: *MFCS*. Vol. 4708. Lecture Notes in Computer Science. Springer, 2007, pp. 536–547.
url: https://arxiv.org/abs/0708.2584.

[Tat66] John Tate. "Endomorphisms of abelian varieties over finite fields". In: *Inventiones mathematicae* 2.2 (1966), pp. 134–144.

[Tib14] Mehdi Tibouchi. "Elligator Squared: Uniform Points on Elliptic Curves of Prime Order as Uniform Random Strings". In: *Financial Cryptography*. Vol. 8437. Lecture Notes in Computer Science. Springer, 2014, pp. 139–156.
url: https://ia.cr/2014/043.

[Too63] Andrei L. Toom. "The complexity of a scheme of functional elements realizing the multiplication of integers". In: *Soviet Mathematics Doklady* 3 (1963), pp. 714–716. url: http://toomandre.com/my-articles/engmat/MULT-E.PDF.

[Unr12] Dominique Unruh. "Quantum Proofs of Knowledge". In: *EUROCRYPT*. Vol. 7237. Lecture Notes in Computer Science. Springer, 2012, pp. 135–152.
url: https://ia.cr/2010/212.

[VDTo2] Eric R. Verheul, Jeroen M. Doumen and Henk C. A. van Tilborg. "Sloppy Alice attacks! Adaptive chosen ciphertext attacks on the McEliece public-key cryptosystem". In: *Information, Coding and Mathematics: Proceedings of Workshop honoring Prof. Bob McEliece on his 60th birthday*. Springer, 2002, pp. 99–119.

[Vél71] Jacques Vélu. "Isogénies entre courbes elliptiques". In: *Comptes Rendus de l'Académie des Sciences de Paris* 273 (1971), pp. 238–241.

[Voi13] John Voight. "Identifying the matrix ring: algorithms for quaternion algebras and quadratic forms". In: *Quadratic and higher degree forms*. Springer, 2013, pp. 255–298.

[Voi18] John Voight. *Quaternion algebras*. Book in preparation; version vo.9.14. July 2018. url: https://math.dartmouth.edu/~jvoight/quat-book.pdf.

[Wal64] Christopher S. Wallace. "A suggestion for a fast multiplier". In: *IEEE Transactions on electronic Computers* 1 (1964), pp. 14–17.

[Was08] Lawrence C. Washington. *Elliptic curves: Number theory and cryptography*. 2nd ed. Discrete Mathematics and its Applications. Chapman & Hall/CRC, 2008. isbn: 978-1-4200-7146-7.

[Wat69] William C. Waterhouse. "Abelian varieties over finite fields". In: *Annales scientifiques de l'École Normale Supérieure* 2 (4 1969), pp. 521–560.

[Wil94] Herbert S. Wilf. *generatingfunctionology*. Academic Press, 1994. url: https://www.math.upenn.edu/~wilf/DownldGF.html.

[Yoo+17] Youngho Yoo, Reza Azarderakhsh, Amir Jalali, David Jao and Vladimir Soukharev. "A Post-quantum Digital Signature Scheme Based on Supersingular Isogenies". In: *Financial Cryptography and Data Security*. Vol. 10322. Lecture Notes in Computer Science. Springer, 2017, pp. 163–181. url: https://ia.cr/2017/186.

[Zan+18] Gustavo Zanon, Marcos A. Simplício Jr., Geovandro C. C. F. Pereira, Javad Doliskani and Paulo S. L. M. Barreto. "Faster Isogeny-Based Compressed Key Agreement". In: *PQCrypto*. Vol. 10786. Lecture Notes in Computer Science. Springer, 2018, pp. 248–268. url: https://ia.cr/2017/1143.