Practical isogeny-based cryptography

Michael Meyer

Practical isogeny-based cryptography

A dissertation submitted to

Julius-Maximilians-Universität Würzburg
Fakultät für Mathematik und Informatik
Institut für Mathematik

for the degree of
doctor rerum naturalium (Dr. rer. nat.)

presented by
Michael Meyer

Referees:
Prof. Dr. Jörn Steuding
Prof. Dr. Steffen Reith
Dr. Luca De Feo

Würzburg, 2021

Acknowledgments

Although there is only a single name on the cover of this thesis, it certainly would not exist without the support and encouragement of many people. I want to take the opportunity to thank them here, and apologize for the fact that the following will certainly be incomplete.

First of all, thanks to my supervisors Steffen Reith and Jörn Steuding for their constant support, guidance, and encouragement during the last years. In particular, thanks to Steffen Reith for taking the trouble to set up the PQC research group at HSRM, and to Jörn Steuding for steering my research interest towards cryptography and elliptic curves already when I asked for potential topics for my bachelor thesis.

Thanks to Fabio Campos for the fruitful collaboration during the last years; thanks for the discussions, listening to (good and bad) ideas, joint trips to workshops and conferences, the introduction to brazilian food, and countless lessons on programming. I would have had much fewer things to write about in this thesis, if we would not have shared an office over the last three years.

Thanks to Thorsten Knoll for joining our research group, setting up our frequent *random oracles* meetings, and for (together with Fabio) making my more or less regular trips to Wiesbaden much more pleasant through the subsequent dinners and beers.

Thanks to Luca De Feo (sorry – *Captain* Luca De Feo) for joining the commitee for this thesis, for the fruitful collaboration on threshold schemes, and for the initiation and production of the blockbuster movie "The Pirates of the CSIDH".

Thanks to the "smooth twins" Craig Costello and Michael Naehrig for inviting me to a research internship at Microsoft Research, and even though the pandemic prevented the internship to take place, making my internship an "inofficial" and virtual one.

Thanks to Elektrobit Automotive GmbH, and in particular to Martin Böhner, for funding my PhD position through a research collaboration, and thereby bootstrapping the PQC research group at HSRM. Thanks also to Marc Stöttinger for the frequent discussions, advice, collaboration, and invitations to PQC workshops.

Thanks to Juliane Krämer for providing me with a temporary position in the QPC research group at TU Darmstadt, which closed the gap that resulted from the canceled internship at MSR, and which allowed me to finish this thesis without running into serious time issues.

Thanks to my coauthors Fabio Campos, Craig Costello, Luca De Feo, Matthias Kannwischer, Michael Naehrig, Hiroshi Onuki, Steffen Reith, and Marc Stöttinger for the fruitful collaboration and contribution to this thesis.

Thanks to Fabio Campos and Michael Naehrig for taking the time to read and comment on an earlier draft of parts of this thesis.

Last but not least, I want to thank my friends and family for their constant support and motivation, as well as distraction whenever needed, throughout my academic journey.

Contents

Chapter 1

Introduction

The term *cryptography* is derived from the greek words κρυπτός and γράφειν, and thus translates to "secret writing". Whenever people have been communicating in written form, confidential messages were required to remain secret; i.e., if the message got into the wrong hands, the non-intended recipient was not supposed to be able to extract the original content of the message. To this end, methods were invented that allowed for such messages to be encrypted and decrypted only by the use of a *secret* that had to be initially shared among the communicating parties. On the other hand, whoever received the encoded message, or *ciphertext*, was supposed to only be able to decode the message to its *plaintext* if in possession of the corresponding secret.

Among many others, examples for this approach can be found in the times of the ancient Greeks or Romans [141]. The Greeks used a *scytale* to construct a mechanical *transposition cipher*, which produces a permutation of the plaintext letters as ciphertext. It consists of a cylinder and a strip of parchment, which is wound around the cylinder. The message is written on the parchment, which is then removed from the cylinder and sent to the recipient. In order to read the message, the recipient had to wound the parchment around a cylinder of the same diameter as the original one, such that the letters aligned correctly. The shared secret is thus given by the cylinder, resp. its diameter (see [141]).

A different method, e.g. used by the Romans and attributed to Julius Caesar, is the *Caesar cipher*, a *substitution cipher*. In this cipher each plaintext letter is replaced by the letter a fixed number of steps to its left side in the alphabet, e.g. substituting 'D' by 'A' for a left shift of 3 steps. The resulting ciphertext can easily be decoded by reversing this shift, if the number of steps is known. Thus, this number is the shared secret involved in the Caesar cipher (see [141]).

Although these ciphers can be broken easily by today's standards, they offered a significant advantage over unencrypted plaintext messages in the corresponding times, especially when messages had to remain confidential only for a short time frame. Over the centuries, more secure ciphers have been invented, e.g. the *Vigenère cipher* or the *running key cipher*, resp. encryption machines like the *Enigma* have been built [141]. Despite many

differences between these ciphers and their security, all of these ciphers share the property of requiring a secret, or *key*, that the participants agree upon initially and secretly, and which must remain confidential. Today, such schemes are collected under the name *symmetric cryptography*.

Another commonality among these ciphers is that the common people rarely got in touch with them. Instead, they were mostly deployed in a political or military context, as e.g. the Greeks' usage of the scytale for communication during military campaigns or the usage of the Enigma during World War II [141]. However, the rise of the internet over the last decades rapidly changed this picture. Although often unnoticed, our modern digital society could hardly exist without secure cryptographic primitives such as the advanced encryption standard (AES) [56], the most popular and commonplace modern symmetric cipher. Thus, nowadays cryptography plays an important role in our everyday life, despite not being explicitly visible to the user most of the time.

However, the main driving force behind this development has not been symmetric cryptography, but the invention of *asymmetric* or *public-key cryptography* by Diffie and Hellman in 1976 [68]. They provide a *key exchange* protocol, which allows communicating parties to establish a shared secret over an insecure channel, thus eliminating the need to initially agree on such a secret in private, i.e., over a secure channel. Such a shared secret can then either directly be used to encrypt or decrypt messages, or can be used to derive a key for the deployment of a symmetric cipher. This can be achieved through the generation of a key pair for each participant, consisting of a secret *private key* and a publicly available *public key*, from which the communicating parties can derive their shared secret.

Apart from the secrecy that key exchanges provide, Diffie and Hellman's work builds the basis for digital signature schemes. Digital signatures consist of data that is attached to messages, and when verified by the recipient to match the sender's public key and the received message, guarantee the authenticity and integrity of the message. In other words, a valid signature guarantees that the message has indeed been sent by the expected sender and has not been manipulated. These two primitives, key exchange schemes and digital signatures, essentially form the whole foundation of modern communication over the internet. Apart from the original proposals by Diffie and Hellman [68], famous public-key encryption, key exchange, or signature schemes were given by RSA encryption and signatures by Rivest, Shamir, and Adleman [132], ElGamal encryption and signatures [73], Schnorr signatures [134], and elliptic curve cryptography (ECC) as proposed by Miller [116] and Koblitz [103].

Even though the most common requirements are covered by these primitives, there are many other applications that require different protocols; e.g., multiparty computation, electronic voting, blockchains and cryptocurrencies, or the very recent field of privacy-preserving contact tracing as a means of fighting the transmission of infectious diseases. Throughout this thesis, we will entirely focus on public-key cryptography, and mostly on key exchange schemes.

1.1 Post-quantum cryptography

The aforementioned public-key schemes rely on hard computational problems to justify their security. In particular, the hard problems corresponding to the mentioned schemes are the integer factorization problem and the discrete logarithm problem. Over the decades, we gained sufficient confidence in their hardness to allow for their widespread usage.

However, this only accounts for attackers using classical computers. If we consider an attacker with access to a large-scale quantum computer, Shor showed that both of these problems can be solved efficiently [137]. Thus, in a quantum era, where large-scale quantum computers exist, essentially all of the currently deployed public-key cryptography is broken. For symmetric cryptography, the situation is not as alarming; although Grover's algorithm [89] allows for a substantial speedup for key search attacks, it suffices to roughly double key sizes to compensate for the faster quantum key search.

Fortunately, such large-scale quantum computers are currently not known to exist. However, it is unclear if and when this could change. This puts us in a situation, where potentially all of our currently deployed public-key schemes could be broken in the coming years, yet without knowing an approximate time frame. It may thus be tempting to lean back and wait for further progress in the development of quantum computers, before starting to replace schemes like RSA or ECC by quantum-resistant alternatives. However, in the meantime adversaries could archive confidential encrypted messages, and decrypt them retroactively once a large enough quantum computer is available. Moreover, the transition to quantum-resistant schemes is a complex and tedious process, which requires the standardization of such schemes, as well as their implementation for real-world applications. Thus, such a transition cannot happen overnight once a large-scale quantum computer is available, but requires an immense amount of preparations within a time frame of years.

After considerable effort among the academic world, this development led the United States National Institute of Standards and Technology (NIST), the most important organization for cryptographic standards, to initiate a standardization process for *post-quantum cryptography* (PQC) in 2016 [121]. Researchers were asked to submit quantum-resistant key encapsulation mechanisms (KEMs, variants of key exchange schemes) and digital signature schemes, with the aim of selecting and standardizing the "best" schemes among them.[1] In the first round, 69 submissions took part in the process. Over the following years, in an effort of a transparent process, NIST narrowed down the number of potentially standardized schemes through several rounds. Beginning in 2020, 15 schemes have moved on to the final third round. NIST expects that by 2024 the standardization of a small

[1] The question of which scheme is the "best" mainly depends on the use case; it could for example be required to prioritize performance, small key sizes, or confidence in the security of the scheme.

number of these finalists will be finished [121], and deployment in practice then rapidly follows.

Isogeny-based cryptography. As explained above, schemes that rely on the computational hardness of integer factorization or the discrete logarithm problem are not quantum-resistant. Fortunately, other approaches have been explored over the last decades, and it turned out that some of them are conjectured to withstand quantum attacks. In particular, the first four families of well-known quantum-resistant schemes are given by *code-based cryptography*, *hash-based cryptography*, *lattice-based cryptography*, and *multivariate cryptography* (see [12]). However, over the last decade, a fifth family of PQC schemes gained much attention and experienced a fast-paced development: *isogeny-based cryptography*. It is based on the computation of isogenies, maps between elliptic curves, and can thus be seen as an extension of traditional ECC.

Isogeny-based cryptography has first been proposed by Couveignes in 1997 [51], but after being rejected at a conference, his proposal only circulated privately. In 2006, Rostovtsev and Stolbunov [133] independently rediscovered Couveignes' scheme, which is therefore often abbreviated as CRS. The major drawback of this scheme is its rather impractical performance, which prevented it to be considered as a serious PQC alternative.

A different line of work was initiated in 2006 by Charles, Goren, and Lauter through a hash function (CGL) based on isogeny graphs of supersingular elliptic curves [38]. Following this approach, in 2011 Jao and De Feo [94] proposed the *Supersingular Isogeny Diffie-Hellman* (SIDH) key exchange, which uses similar isogeny graphs and achieves a much better performance than CRS. After several improvements, e.g. [60, 47, 46], a KEM variant of SIDH was submitted to the NIST PQC standardization process under the name SIKE in 2017 [93], and advanced to the third round as an alternate candidate.

Many other isogeny-based schemes were only invented in the last three years, and therefore too late to participate in the NIST PQC process. The most prominent of these schemes is the *Commutative Supersingular Isogeny Diffie-Hellman* (CSIDH) key exchange scheme, published by Castryck, Lange, Martindale, Panny, and Renes [35] in 2018. It builds upon improvements of the CRS scheme by De Feo, Kieffer, and Smith [61], and spurred much follow-up work that exploits its commutative structure for digital signatures and other advanced protocols.

Other recent isogeny-based schemes are, e.g., the key exchange scheme B-SIDH [44] and the signature schemes SeaSign [59], CSI-FiSh [19], and SQISign [62]. We will expand upon some of those in Chapter 2.

1.2 Organization of this thesis

As the title suggests, this thesis is focused on practical aspects of isogeny-based cryptography, in particular, implementation aspects and practical applications. Moreover, large

parts of this thesis will focus on CSIDH. This may appear overly selective, but can be explained by the timeline of developments in isogeny-based cryptography. This work covers research done between 2018 and 2020, which can be placed after the numerous improvements in the realm of SIDH/SIKE that culminated in its submission to the NIST PQC standardization process in 2017. It is thus natural for research on implementations and practical applications to focus on less studied topics, such as CSIDH, which was published only in 2018, aligning with the time frame of research for this work. Three chapters of this thesis will cover efficient and side-channel resistant implementations of CSIDH. Thereafter, applications of general isogeny group actions and practical considerations for setting up the very recent schemes B-SIDH and SQISign will be covered.

In Chapter 2 we introduce the mathematical background of isogeny-based cryptography. We describe the necessary theory of elliptic curves and isogenies, explain their arithmetic in implementations, and describe three isogeny-based key exchange schemes, namely SIDH, B-SIDH, and CSIDH. The remainder of this thesis is organized in chapters that correspond to one academic paper, respectively. We briefly describe each chapter's content, and refer to the corresponding publications.

Chapter 3 is based on [114]. It reviews the variable-time implementation of CSIDH accompanying the original CSIDH paper [35], and introduces several speedups. Firstly, we describe how the algorithm can be reorganized in order to save a substantial amount of computational effort for scalar multiplications. Moreover, we present a method to compute isogenies exploiting the correspondence between Montgomery and twisted Edwards curves, resulting in a significant speedup over the prior methods using only Montgomery curves. In total, our improvements gain a speedup of 25% when plugged into the implementation of [35].

Chapter 4 is based on [113]. It analyzes how variable-time implementations of CSIDH provide side-channel leakage, and introduces the first efficient constant-time implementation of CSIDH. This is achieved through the usage of dummy isogenies, in order to allow for a constant total number of isogenies, and through the usage of only non-negative key elements, which prevents leakage on the sign distribution of the corresponding private key. When implemented in a straightforward fashion, these countermeasures roughly lead to a slowdown by a factor of 6 compared to [114]. However, we present several speedups, such as a method that processes the required isogenies in batches, and achieve an overall slowdown by a factor of 3.03, which is considerably faster than the straightforward implementation.

Chapter 5 is based on [31]. It analyzes practical fault injection attacks on CSIDH implementations with dummy isogenies in different attacker models. In particular, an attacker can try to determine the ratio of real vs. dummy isogenies through fault injections, or, in a stronger attacker model, specifically aim at faulting an isogeny in order to determine whether it is a dummy computation. We simulate these attacks in order to analyze their impact, and demonstrate the practical feasibility using only low-budget equipment.

Furthermore, we provide countermeasures against the considered attacks, which detect any fault injection covered by our models, and only induce a computational overhead by a factor of 1.07. Thus, our protected algorithm is significantly faster than a variant that avoids using dummy isogenies in order to protect against fault injection attacks.

Chapter 6 is based on [63]. It initiates the study of threshold schemes from hard homogeneous spaces (HHS). HHS have first appeared in the work of Couveignes [51] and generalize the concept of discrete logarithm groups to cryptographic group actions. In this chapter we analyze how threshold schemes from discrete logarithm groups can be adapted to the HHS setting. We describe a threshold KEM and signature scheme based on HHS and Shamir secret sharing [136], and prove their security in the honest-but-curious adversary model. Although CSIDH uses an isogeny group action, it is not a HHS in the strictest sense, since e.g. the group structure of the involved class group is not known in general. However, with the tools provided by the CSI-FiSh signature scheme [19], our HHS threshold schemes can be instantiated based on isogenies.

Chapter 7 is based on [49]. It provides a sieving algorithm for searching for prime numbers that allow for an efficient instantiation of B-SIDH [44] or SQISign [62]. These schemes require as parameter a prime of a certain size that lies between two integers that are as smooth as possible. In order to efficiently search for such primes, we utilize solutions to the Prouhet-Tarry-Escott (PTE) problem, which then provide pairs of polynomials that completely split into linear factors in $\mathbb{Z}[x]$ and have constant differences. We explain how searching with such polynomials improves upon the methods used in [44, 62], and implement an efficient sieving algorithm that combines many PTE solutions into a single search with minimal overhead. In practice, our algorithm found cryptographically sized primes p with $2^{240} \leq p \leq 2^{256}$, where $p-1$ and $p+1$ are both 2^{15}-smooth. This significantly improves upon the best previously known examples, where the best smoothness bound was given by 2^{19}.

Remark 1. Throughout the last few years, isogeny-based cryptography, and specifically isogeny-based group actions as in CSIDH, have gained an increased amount of attention. This led to an especially fast-paced progress in its research, and an increasing amount of published papers. Thus, we will comment on follow-up work referring to the content of this thesis; in particular, some of the chapters of this thesis are concluded by such an overview, in order to help the reader in assessing the contributions of this thesis and the current state-of-the-art.

Chapter 2

Preliminaries

This chapter gives an introduction to isogeny-based cryptography, and as such does not contain any novel results, but presents the necessary background for the following chapters. It is loosely inspired by introductory texts by Costello [42, 43] and De Feo [57]. To fully understand the mathematics and cryptographic applications of isogenies, a vast background in mathematics is required. Therefore, this chapter certainly cannot have the ambition to be a complete and standalone introduction to the topic. Instead, since efficient implementations play an important role in large parts of this thesis, this introduction aims towards being an accessible entry point also for implementers and newcomers to isogeny-based cryptography. For more mathematical background, we refer to classical textbooks, e.g. by Silverman [140] or Washington [153], to Galbraith [79], or to De Feo's lecture notes [57].

The following sections briefly introduce elliptic curves and isogenies, and provide important results for cryptographic applications, mostly using definitions and results from [140]. We review the related elliptic curve and isogeny arithmetic, and describe cryptographic protocols such as SIDH, B-SIDH, and CSIDH.

2.1 Elliptic curves

Elliptic curves possess the cryptographically useful property of behaving like an abstract type of group. In particular, for a general field $\mathbb{K}$ with algebraic closure $\overline{\mathbb{K}}$, an elliptic curve contains points (x, y) with $x, y \in \overline{\mathbb{K}}$ that satisfy the affine curve equation

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6,$$

where $a_1, a_2, a_3, a_4, a_6 \in \overline{\mathbb{K}}$, and the distinguished point ∞. An elliptic curve in this form is called a *general Weierstraß curve*. The set of points on such an elliptic curve can thus be written as

$$E = \{(x, y) \in \overline{\mathbb{K}}^2 \mid y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6\} \cup \{\infty\},$$

where ∞ denotes the so-called *point at infinity*.[2] Although the existence of such a point cannot be explained from the affine curve equation, it will be required to define the elliptic curve group law.

If the characteristic of the underlying field is not 2 or 3, there exist simple coordinate transformations of a general Weierstraß curve that set $a_1, a_2, a_3 = 0$, and thus yield an equation of the following form [140, §III.1].

Definition 1 (Short Weierstraß curve)**.** Over a field $\overline{\mathbb{K}}$ of characteristic not equal to 2 or 3, an elliptic curve can be written in the form

$$E : y^2 = x^3 + ax + b,$$

where $4a^3 + 27b^2 \neq 0$ in $\overline{\mathbb{K}}$. An elliptic curve of this form is called a *short Weierstraß curve*.

It is important to note that coefficients a and b only define an elliptic curve if $4a^3 + 27b^2 \neq 0$ in $\overline{\mathbb{K}}$, since otherwise singularities are contained in the arising curve [140, §III.1]. Throughout this thesis, only fields with large prime characteristics will appear, which means that we can always think of elliptic curves as being representable by a short Weierstraß equation.

For $a_1, a_2, a_3, a_4, a_6 \in \mathbb{K}$, resp. $a, b \in \mathbb{K}$, we can further define the elliptic curve over $\mathbb{K}$ instead of the algebraic closure $\overline{\mathbb{K}}$, such that only $\mathbb{K}$-rational points $(x, y) \in \mathbb{K}^2$ will be considered. We write $E(\mathbb{K})$ in this case, unless the field of definition is clear from the context and we can abbreviate this by only writing E.

For a given elliptic curve $E(\mathbb{K})$, there exists an addition operation "+", such that the following holds [140, Proposition III.2.2].

- For any not necessarily distinct points $P, Q \in E(\mathbb{K})$, we have $P + Q = R \in E(\mathbb{K})$, i.e., the operation is closed.
- For any not necessarily distinct points $P, Q, R \in E(\mathbb{K})$, we have $(P + Q) + R = P + (Q + R) \in E(\mathbb{K})$, i.e., the operation is associative.
- The point $\infty \in E(\mathbb{K})$ is the neutral element; i.e., for any $P \in E(\mathbb{K})$, we have $P + \infty = \infty + P = P$.
- For any $P \in E(\mathbb{K})$ there is an inverse element $Q \in E(\mathbb{K})$ such that $P + Q = Q + P = \infty$.

Hence, the set of points on an elliptic curve forms a group, and the addition operation forms the group law. Since this group is written additively, we refer to the inverse point of $P \in E$ as the negative point $-P$. In particular, $-(x, y) = (x, -y)$ for short Weierstraß curves. Note that adding points on a Weierstraß curve with distinct x-coordinates requires

[2] In the literature this point is often denoted as $\mathcal{O}$ or O. Throughout this thesis we use ∞ in order to avoid notational collisions with other occurrences of $\mathcal{O}$ and O.

different formulas than adding a point to itself. For explicit formulas, we refer to [140, Algorithm III.2.3]. When looking at elliptic curves over $\mathbb{R}$, the addition and doubling of points can be represented graphically via the *chord-and-tangent-rule*, see e.g. [140, §III.2]. From this, it is immediately clear that for $P, Q \in E$, it must hold that $P + Q = Q + P$. This is indeed the case for any elliptic curve, which means that the group of points is commutative, or an *abelian group* [140, Proposition III.2.2].

Following the group law, we can therefore add a point multiple times to itself. For any integer $m > 0$, this justifies the introduction of the multiplication-by-m map $[m] : E \to E$. For integers $m < 0$, one defines $[m]P = -[|m|]P$, and for $m = 0$, we set $[0]P = \infty$.

If we now fix a curve E over $\overline{\mathbb{K}}$, we can look at the kernels of these scalar multiplication maps, i.e., the set of points P such that $[m]P = \infty$.

Definition 2 (Torsion group)**.** Let $E(\overline{\mathbb{K}})$ be an elliptic curve and m an integer. Then the subgroup

$$E[m] = \{P \in E(\overline{\mathbb{K}}) \mid [m]P = \infty\}$$

is called the *m-torsion group* or *m-torsion* of E.

Note that the case $m < 0$ is equivalent to the $|m|$-torsion, and the case $m = 0$ is not particularly interesting, hence in the following we only consider positive integers m without imposing restrictions. The occurring structures of torsion groups are presented in the following result.

Proposition 1 ([140, Corollary III.6.4])**.** *Let $E(\overline{\mathbb{K}})$ be an elliptic curve, and $m \in \mathbb{Z}$ with $m > 0$.*

(a) If $\mathrm{char}(\mathbb{K}) = 0$, or $p = \mathrm{char}(\mathbb{K}) > 0$ and $p \nmid m$, then we have

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

(b) If $p = \mathrm{char}(\mathbb{K}) > 0$, then one of the following is true:

(i) $E[p^e] = \{\infty\}$ for all $e = 1, 2, 3, \ldots$.

(ii) $E[p^e] = \mathbb{Z}/p^e\mathbb{Z}$ for all $e = 1, 2, 3, \ldots$.

A related concept is the *order* of a point $P \in E$. This denotes the smallest positive integer r such that $[r]P = \infty$. If no such integer exists, then we say that P has infinite order, otherwise the point is called a *torsion point*. It is important to note that for positive integers m with $\mathrm{char}(\mathbb{K}) \nmid m$, the m-torsion group $E[m]$ is not equal to the set of points of order m. In particular, $E[m]$ contains the point ∞ and all points of orders $r \neq m$ with $r \mid m$, as well as the points of order m.

For elliptic curves over $\mathbb{Q}$, a theorem by Mazur implies that $\mathbb{Q}$-rational torsion points are rare [140, Theorem VIII.7.5]. On the other hand, the picture looks completely different when looking at elliptic curves over finite fields. Let p be a prime and q a power of p. Then we denote by $\mathbb{F}_q$ a finite field with q elements and characteristic p. Naturally, in this

case there are only finitely many $\mathbb{F}_q$-rational points on a curve $E(\mathbb{F}_q)$, and any such point necessarily is a torsion point. This however does not mean that for fixed q and any integer m with $p \nmid m$ the torsion group $E[m]$ is completely contained in $E(\mathbb{F}_q)$. This surely is the case when looking at $E(\overline{\mathbb{F}_q})$, but for the subfield curve over $\mathbb{F}_q$, not all of these torsion points must be $\mathbb{F}_q$-rational. The following theory helps to make precise statements about $\mathbb{F}_q$-rational torsion points.

A very important property of an elliptic curve $E(\mathbb{F}_q)$ is its number of points $\#E(\mathbb{F}_q)$, and therefore its group order. For fixed $\mathbb{F}_q$, there are relatively few possible group orders, as a theorem by Hasse suggests.

Theorem 1 (Hasse, [140, Theorem V.1.1])**.** *Let $E(\mathbb{F}_q)$ be an elliptic curve over a finite field. Then*

$$q + 1 - 2\sqrt{q} \leq \#E(\mathbb{F}_q) \leq q + 1 + 2\sqrt{q}.$$

In other words, we have $\#E(\mathbb{F}_q) = q + 1 - t$, where $|t| \leq 2\sqrt{q}$ is the so-called *trace of Frobenius* (see Section 2.2 for more details). An efficient algorithm to identify the group order of a given curve $E(\mathbb{F}_q)$ is Schoof's algorithm [135], resp. its extension, the Schoof-Elkies-Atkin algorithm [23, Chapter VII].

Since we are now able to efficiently compute the group order of a given curve $E(\mathbb{F}_q)$, this immediately yields results for the occurring point orders. For a point $P \in E(\mathbb{F}_q)$, Lagrange's Theorem [99, §3.3.2] implies that its order divides $\#E(\mathbb{F}_q)$. However, this still does not directly lead to statements about torsion groups. In particular, for a fixed $m > 0$ that is coprime to $\mathrm{char}(\mathbb{F}_q) = p$, points of order m on $E(\mathbb{F}_q)$ only possibly exist if $m \mid \#E(\mathbb{F}_q)$, but this does not mean that the whole torsion group $E[m]$ is contained in $E(\mathbb{F}_q)$. We have already seen that the structure of the torsion group is $E[m] \cong \mathbb{Z}/m\mathbb{Z} \times \mathbb{Z}/m\mathbb{Z}$ in this case, which means that $\#E[m] = m^2$. Further, since the point ∞ must be contained in any subgroup of $E[m]$, this means that for primes m, which will be the most interesting case for our purposes, $E[m]$ contains $m + 1$ cyclic subgroups of order m. It is easy to see that $E[m]$ can only be fully contained in $E(\mathbb{F}_q)$ if $m^2 \mid \#E(\mathbb{F}_q)$. On the other hand, if $m \mid \#E(\mathbb{F}_q)$ and $m^2 \nmid \#E(\mathbb{F}_q)$, then there is one unique subgroup of $E[m]$ that is contained in $E(\mathbb{F}_q)$ [42, §4.1].

In the latter case, in order to see where $E[m]$ is defined in its entirety, we can look at the curve E over extensions of $\mathbb{F}_q$. In particular, we call the smallest positive integer k such that $E[m] \subset E(\mathbb{F}_{q^k})$ the *embedding degree*, which in general depends on q and m. In case that $k > 1$, it is interesting to note that as soon as we reach a field extension $\mathbb{F}_{q^k}$, over which one point of order m is found that is not contained in $E(\mathbb{F}_q)$, then already the whole torsion group $E[m]$ must be contained in $E(\mathbb{F}_{q^k})$. We refer to [42, §4.1] for more details.

Remark 2. Since in the remaining chapters of this thesis, and for cryptographic applications in general, we are only interested in elliptic curves over finite fields, the following sections will mainly focus on results for this special case.

Remark 3. We can generalize short Weierstraß elliptic curve equations to $y^2 = f(x)$, with $\deg(f) > 4$, and obtain *hyperelliptic curves*. Also in this case, we can define a group law for applications in cryptography. However, we cannot simply work with points here, but have to use divisors and elements of the *divisor class group* or *Picard group* of the curve. In principle, one could also define the elliptic curve group law in terms of divisors; in this fortunate case however, there is a one-to-one correspondence between the divisor class group and the points on the curve. Due to this, we can resort to describing the elliptic curve group law entirely in terms of curve points, without using the language of divisors and examining the related theory. Thus, compared with the hyperelliptic case, elliptic curve arithmetic and the related theory as presented in this section appears to be particularly simple. We refer to [140, 79] for more details, and to [42, Chapter 3] for a gentle introduction to the topic.

2.2 Maps between elliptic curves

Up to this point, we have fixed elliptic curves, and analyzed their group structures and multiplication maps. Moving to a higher level, we can now look at maps between elliptic curves. In the following, we assume the occurring curves to be defined over $\overline{\mathbb{K}}$.

Definition 3 (Isogeny)**.** An *isogeny* between two elliptic curves E_1, E_2 is a non-constant morphism $\varphi : E_1 \to E_2$ that satisfies $\varphi(\infty) = \infty$. Two curves E_1 and E_2 are called *isogenous* if there is an isogeny from E_1 to E_2.

In this thesis, we will only consider *separable* isogenies (see [140, §III.4] for details), and oftentimes simply refer to separable isogenies as isogenies. A consequence of this is that the degree of a separable isogeny φ, which is its degree as a rational map, equals the cardinality of its kernel, i.e., $\#\ker\varphi = \deg\varphi$ [140, Theorem III.4.10]. Isogenies are group homomorphisms, and the following result holds.

Proposition 2 ([140, Proposition III.4.12])**.** *For any finite subgroup $G \subset E_1$, there is, up to post-composition with isomorphisms, a unique elliptic curve E_2 and a separable isogeny $\varphi : E_1 \to E_2$, such that* $\ker\varphi = G$*. The codomain curve is often written as E_1/G.*

Given $G \subset E_1$, explicit formulas for the computation of φ and the codomain curve $E_2 = E_1/G$ are given by Vélu's formulas [152].

For every isogeny $\varphi : E_1 \to E_2$, there exists a unique (up to composition with isomorphisms) *dual isogeny* $\widehat{\varphi} : E_2 \to E_1$, such that $\widehat{\varphi} \circ \varphi = [\deg\varphi]$, i.e., the multiplication-by-$\deg\varphi$ map on E_1 [140, Theorem III.6.1]. In fact, the isogenous property is an equivalence relation, and we call the equivalence class of the set of elliptic curves isogenous to a given curve E the *isogeny class* of E. The above statements further motivate that we do not distinguish between isomorphic elliptic curves in an isogeny class. Over $\overline{\mathbb{K}}$, two elliptic curves

are isomorphic if and only if their *j-invariants* are equal [140, Proposition III.1.4]. The *j*-invariant of a short Weierstraß curve can be computed via

$$j(E) = j(a,b) = \frac{1728(4a)^3}{16(4a^3 + 27b^2)}.$$

Note that $j(E)$ is well-defined, since as described above, the parameters a, b are required to satisfy $4a^3 + 27b^2 \neq 0$ in $\overline{\mathbb{K}}$ in order to give rise to an elliptic curve.

If we now look at elliptic curves over a field $\mathbb{K} \subsetneq \overline{\mathbb{K}}$, then curves can have equal *j*-invariants, although not being isomorphic over $\mathbb{K}$.

Definition 4 (Twist)**.** For an elliptic curve $E(\mathbb{K})$, the curve $E'(\mathbb{K})$ is called a *twist* of E, if E' is isomorphic to E over $\overline{\mathbb{K}}$, but not over $\mathbb{K}$.

To every twisting isomorphism of a curve $E(\mathbb{K})$ corresponds a specific degree, which defines over which field extension of $\mathbb{K}$ the isomorphism is defined. In particular, every curve has a *quadratic* twist. Some special curves further have *cubic*, *quartic*, or *sextic* twists, see [57, Section 2] or [42, §4.3].

In the following we restrict to our main area of interest, namely elliptic curves over finite fields. Again we denote by $\mathbb{F}_q$ a finite field with q elements of prime characteristic p. It is worth pointing out that isogenies are surjective, but do not appear so when only looking at curves over $\mathbb{F}_q$. In particular, if $\varphi : E_1(\mathbb{F}_q) \to E_2(\mathbb{F}_q)$ is an isogeny of degree $m = \deg \varphi$, then it is a m-to-1 map between points of $E_1(\mathbb{F}_q)$ and points of $E_2(\mathbb{F}_q)$. However, the "missing points" of $E_2(\mathbb{F}_q)$ in the image $\varphi(E_1(\mathbb{F}_q))$ eventually appear over field extensions of $\mathbb{F}_q$, so φ is surjective for the corresponding curves over $\overline{\mathbb{F}_q}$, see e.g. [82, Example 3].

Apart from isogenies, there is another type of maps of interest to us.

Definition 5 (Endomorphism)**.** An isogeny $\psi : E \to E$ from a curve E to itself is called *endomorphism*.

We have already encountered endomorphisms in Section 2.1; in particular, the multiplication-by-m map $[m]$ clearly is an endomorphism for any integer m, regardless of the field the curve is defined over. For most curves over fields of characteristic zero, there are no other endomorphisms. For elliptic curves over finite fields $\mathbb{F}_q$ however, we can define the Frobenius endomorphism, which is non-scalar in most situations.

Definition 6 (Frobenius endomorphism, [140, §III.4])**.** Let $E(\mathbb{F}_q)$ be an elliptic curve. Then the map

$$\pi : E \to E, \quad (x,y) \mapsto (x^q, y^q)$$

is called *Frobenius endomorphism*.

The Frobenius endomorphism has a number of interesting properties. For example, π acts trivially for all $P \in E(\mathbb{F}_q)$, i.e., $\pi(P) = P$ for these points, but acts non-trivially on any point defined over an extension of $\mathbb{F}_q$. This further generalizes to $\pi^i(x,y) \mapsto (x^{q^i}, y^{q^i})$,

which only acts non-trivially on points $P \in E(\overline{\mathbb{F}_q}) \backslash E(\mathbb{F}_{q^i})$ [42, §2.2]. For a curve $E(\mathbb{F}_q)$, this implies that $([1] - \pi)P = P - \pi(P) = \infty$ if and only if $P \in E(\mathbb{F}_q)$. Thus, we can derive that $\#E(\mathbb{F}_q) = \#\ker([1] - \pi)$, which links the Frobenius endomorphism to Hasse's theorem (Theorem 1), and gives a hint on the naming of t, the trace of Frobenius (see [42, §2.2] for details).

A theorem by Sato and Tate links the isogeny class of curves with their cardinality, and therefore also their Frobenius endomorphisms.

Theorem 2 (Sato-Tate, [57, Theorem 12], [153, §12.5])**.** *Two elliptic curves E_1, E_2 over $\mathbb{F}_q$ are isogenous over $\mathbb{F}_q$ if and only if $\#E_1(\mathbb{F}_q) = \#E_2(\mathbb{F}_q)$.*

At this point, it is worth noting that formally, an isogeny (resp. an endomorphism) φ between two curves E_1 and E_2 (where $E_1 = E_2$ in the case of endomorphisms) over $\mathbb{F}_q$ is defined over $\mathbb{F}_q$, if the Frobenius endomorphism on E_1 stabilizes the kernel $\ker \varphi$. In particular, the points of $\ker \varphi$ are not necessarily defined over $\mathbb{F}_q$ for this to hold, see [61].

When looking at the set of endomorphisms of a given curve E over any field, one finds that together with the zero map, it forms a ring with addition and composition, the *endomorphism ring* $\mathrm{End}(E)$ [140, §III.4]. It is easy to see that $\mathrm{End}(E)$ always contains $\mathbb{Z}$. For elliptic curves over finite fields however, the endomorphism ring is strictly larger [140, Theorem V.3.1], e.g. containing $\mathbb{Z}[\pi]$ if the Frobenius endomorphism π is non-scalar. The possible structures of endomorphism rings are collected in the following result.

Proposition 3 ([140, Corollary III.9.4])**.** *Let $E(\mathbb{K})$ be an elliptic curve over a field $\mathbb{K}$. Then the endomorphism ring $\mathrm{End}(E)$ is either $\mathbb{Z}$, an order in an imaginary quadratic field, or an order in a quaternion algebra.*

For elliptic curves over finite fields, we already noted that the first case cannot occur. The other two cases define two different types of elliptic curves [140, §V.3].

Definition 7 (ordinary, supersingular)**.** Let E be an elliptic curve over a finite field. If $\mathrm{End}(E)$ is an order in an imaginary quadratic field, then E is called *ordinary*. If $\mathrm{End}(E)$ is an order in a quaternion algebra, then E is called *supersingular*.

There are several different characterizations of supersingular resp. ordinary curves, as the following result suggests, thereby linking Proposition 3 with Proposition 1 and Theorem 1.

Theorem 3 ([79, Theorem 9.11.2], [140, Theorem V.3.1])**.** *Let $E(\mathbb{F}_q)$ be an elliptic curve over a finite field of characteristic p. Then the following are equivalent.*

(a) $\mathrm{End}(E)$ *is not commutative, i.e., following Proposition 3, an order in a quaternion algebra.*

(b) $E[p] = \{\infty\}$.

(c) $\#E(\mathbb{F}_q) = q + 1 - t$, *where* $p \mid t$.

It is particularly important to note that this implies that a curve $E(\mathbb{F}_q)$ is supersingular if and only if $\#E(\mathbb{F}_q) \equiv 1 \bmod p$. Together with Theorem 2, this further means that the isogeny class of a supersingular elliptic curve only contains supersingular curves. Similarly, the isogeny class of an ordinary elliptic curve only contains ordinary curves.

In the remainder of this section, we will analyze the structure of the occurring isogeny classes, mostly following the exposition of Galbraith [79, §25.3]. We recall that as defined above, the isogeny class of a given curve $E(\mathbb{F}_q)$ considers both elliptic curves and isogenies over $\overline{\mathbb{F}_q}$. Consequently, we define the $\mathbb{F}_q$-*isogeny class* of $E(\mathbb{F}_q)$ as the set of $\mathbb{F}_q$-isomorphism classes[3] of elliptic curves over $\mathbb{F}_q$, which are isogenous to E over $\mathbb{F}_q$. Such an isogeny class can be represented in a graph.

Definition 8 (Isogeny graph)**.** Let $E(\mathbb{F}_q)$ be an elliptic curve over a finite field of characteristic p, and let ℓ be a prime. Then we define the ℓ-*isogeny graph* $G_{E,\mathbb{F}_q,\ell}$ to be the directed multi-graph where the vertex set is the $\mathbb{F}_q$-isogeny class of E, usually represented by j-invariants. Directed edges represent (equivalence classes of) degree ℓ isogenies between curves from the corresponding vertices.

One can essentially treat the edges as being undirected, since every isogeny gives rise to a dual isogeny, and therefore to a directed edge connecting the same vertices, but pointing in the opposite direction. There is a special case however, when a curve E with $j(E) \in \{0, 1728\}$ is involved. In this case, the numbers of outgoing and incoming edges differ, see [79, Remark 25.3.2]. Now fix a finite field $\mathbb{F}_q$ of characteristic p, and a prime ℓ with $p \nmid \ell$. Then Proposition 1 and Proposition 2 imply that when looking at curves and isogenies over $\overline{\mathbb{F}_q}$, each vertex must have $\ell + 1$ outgoing edges, and therefore, except for the described special cases, $G_{E,\overline{\mathbb{F}_q},\ell}$ is $(\ell+1)$-regular[4] for every curve E. When looking at ℓ-isogeny graphs over $\mathbb{F}_q$, then each vertex has either 0, 1, 2, or $\ell + 1$ outgoing edges [57, Proposition 35].

As suggested by the results above, we will look at isogeny graphs of ordinary and supersingular elliptic curves separately, and call the respective graphs ordinary resp. supersingular isogeny graphs. Note that we will focus on the supersingular case, since the cryptographic applications in this thesis utilize supersingular curves.

In the ordinary case, the involved curves can be placed on different layers, depending on their endomorphism rings. Isogenies between those curves then are either *horizontal*, *ascending*, or *descending* isogenies, depending on which layers the involved curves lie on, see [105, Proposition 21], [57, Proposition 36]. When putting together all the results about these cases, one finds that there is a cycle of horizontal isogenies at the top level, the *surface*, as well as trees of descending isogenies of a certain height [57, Section 9]. Due to this structure, ordinary isogeny graphs are called *isogeny volcanoes* [77]. An example of such an isogeny volcano is given in Figure 2.1.

[3] Two curves being in the same $\mathbb{F}_q$-isomorphism class means that they are isomorphic over $\mathbb{F}_q$.

[4] In an m-regular graph, each vertex has degree m, i.e., in our case m outgoing and incoming edges.

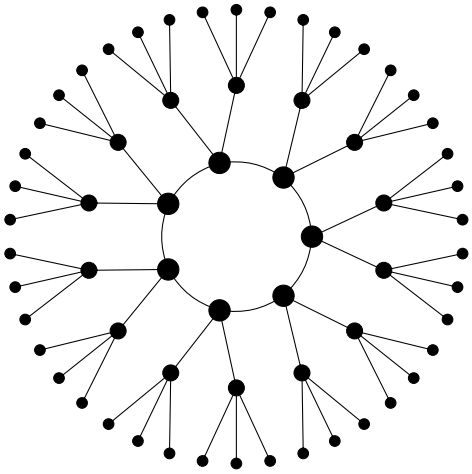


Figure 2.1: The 3-isogeny graph representing the isogeny class of the curve E with $j(E) = 607$ over $\mathbb{F}_{6007}$, see [61, Fig. 3]. The different sizes of vertices represent different sizes of endomorphism rings.

In the supersingular case, the arising ℓ-isogeny graphs look substantially different. Let $E(\overline{\mathbb{F}_q})$ be a supersingular elliptic curve and $\text{char}(\mathbb{F}_q) = p > 3$. The first thing to note here is that j-invariants of supersingular elliptic curves are always contained in $\mathbb{F}_{p^2}$ [140, Theorem V.3.1], which means that it suffices to consider supersingular curves over $\mathbb{F}_{p^2}$ for the vertices of isogeny graphs. This further implies that all corresponding isogenies are defined over $\mathbb{F}_{p^2}$ too; thus, we can choose appropriate scenarios such that the whole required ℓ-isogeny graph will be set up by curves and isogenies over $\mathbb{F}_{p^2}$, see Section 2.4.

The fact that $j(E) \in \mathbb{F}_{p^2}$ for supersingular elliptic curves E now implies that there are only finitely many isomorphism classes of curves, and therefore vertices in the respective isogeny graphs. The exact number can be obtained from the following result.

Theorem 4 ([140, Theorem V.4.1]). *Let $\mathbb{F}_q$ be a finite field of characteristic $p > 3$. Then the number of $\overline{\mathbb{F}_q}$-isomorphism classes of supersingular elliptic curves is given by*

$$\lfloor p/12 \rfloor + \begin{cases} 0 & \text{if } p \equiv 1 \bmod 12, \\ 1 & \text{if } p \equiv 5 \bmod 12, \\ 1 & \text{if } p \equiv 7 \bmod 12, \\ 2 & \text{if } p \equiv 11 \bmod 12. \end{cases}$$

Together with the results from above, we now know the number of vertices of $G_{E,\overline{\mathbb{F}_q},\ell}$ for a supersingular elliptic curve E. For a prime ℓ such that $p \nmid \ell$, we further know that the graph is $(\ell+1)$-regular (apart from the nodes of j-invariant 0 or 1728). In addition, one can prove that $G_{E,\overline{\mathbb{F}_q},\ell}$ is connected [79, §25.3.3], an expander graph, and has the *Ramanujan property* [126]. We will not expand upon these properties here; however, we can make the informal statement that such a graph has excellent mixing properties. In particular, a

random walk of length close to the diameter of the graph[5] leads to any vertex with close to uniform probabilities. We refer to [57, Section 11] or [79, §25.3] for more details.

These properties give an intuition on why such graphs are suitable for cryptographic applications through the usage of random walks. Section 2.4.1 will present such an application, as well as examples for supersingular isogeny graphs.

Instead of looking at the full supersingular isogeny graph, one can restrict to supersingular elliptic curves and isogenies over $\mathbb{F}_p$, where $p > 3$ is prime. While the full endomorphism ring of such a curve $E(\mathbb{F}_p)$ is an order in a quaternion algebra (see Theorem 3), when restricting to endomorphisms over $\mathbb{F}_p$, we find that this subring $\mathrm{End}_{\mathbb{F}_p}(E)$ is an order in the imaginary quadratic field $\mathbb{Q}(\sqrt{-p})$ [65]. Related to this, the structure of isogeny graphs $G_{E,\mathbb{F}_p,\ell}$, where ℓ is prime and $p \nmid \ell$, is similar to the volcano structure of ordinary isogeny graphs. In particular, these graphs mostly consist of isogeny cycles, similar to the surface layer of ordinary isogeny volcanoes, see [65]. We expand upon this case, give examples, and describe a cryptographic application of such isogeny graphs in Section 2.4.3.

2.3 Elliptic curve models and arithmetic

An important point when it comes to cryptographic applications of elliptic curves and isogenies is the efficiency of the involved computational operations. In particular, we eventually want to use as few field operations as possible for implementations. When working over a finite field $\mathbb{F}_q$, the basic operations are additions, subtractions, multiplications, squarings, and inversions over $\mathbb{F}_q$. As usual, we will use the abbreviations **a** for both field additions and subtractions, **M** for multiplications, **S** for squarings, and **I** for inversions over $\mathbb{F}_q$. Usually multiplications and squarings have a similar computational cost, while additions are significantly faster, and inversions are significantly slower. For this reason, optimized elliptic curve and isogeny arithmetic tries to use as few inversions as possible.

When looking at explicit formulas for point additions or doublings in the general or short Weierstraß model with affine points as described in Section 2.1 (see, e.g. [140, Algorithm III.2.3]), one finds that inversions are involved in each such operation. In practice, it is thus more efficient to work with *projective coordinates*.

For an elliptic curve $E(\mathbb{K})$, in Section 2.1 we have defined points in *affine space*, i.e., $P \in E(\mathbb{K})$ with $P = (x, y) \in \mathbb{A}^2(\mathbb{K})$, together with the abstract point $\infty \in E(\mathbb{K})$. However, following [42, §2.1.1], points in the affine space $\mathbb{A}^2$ can be identified with lines in the the affine space $\mathbb{A}^3$ passing through the origin, resp. points in the *projective space* $\mathbb{P}^2$. In particular, a line corresponding to an affine point $P = (x, y) \in \mathbb{A}^2(\mathbb{K})$ consists of the points $(\lambda x, \lambda y, \lambda) \in \mathbb{K}^3$ with $\lambda \in \mathbb{K}\backslash\{0\}$. We thus have a congruence relation, i.e., (x, y, z) and (x', y', z') are congruent if there exists $\lambda \in \mathbb{K}\backslash\{0\}$, such that $(\lambda x, \lambda y, \lambda z) = (x', y', z')$. This

[5] The diameter of a connected graph is defined as the largest *distance* between any two vertices, where the distance between two given vertices is the length of the shortest path between them.

means that $\mathbb{P}^2(\mathbb{K})$ corresponds to $\mathbb{A}^3(\mathbb{K})\backslash(0,0,0)$ modulo this equivalence relation [42, §2.1.1].

Instead of using affine coordinates $(x, y) \in E$, we can thus use projective coordinates, which are written as $(X : Y : Z) \in E$, where the simple transformation $x = X/Z$ and $y = Y/Z$ is used for converting the coordinates. Further, this substitution and multiplication by Z^3 translates the affine curve equation of a short Weierstraß curve $E : y^2 = x^3 + ax + b$ into the projective curve equation

$$E : Y^2Z = X^3 + aXZ^2 + bZ^3.$$

The point at infinity is given by $\infty = (0 : 1 : 0)$, where we note that the above transformation is not defined if $Z = 0$, and thus this point does not have a representation in the usual affine coordinates.

The reason for projective coordinates being preferred in practice is that the point addition and doubling formulas can be rewritten projectively without field inversions, see e.g. [16] for explicit formulas. In particular, roughly speaking inversions are replaced by multiplications in the Z-coordinate. We only have to use inversions and map back to affine coordinates whenever a unique representation of points is required.

Remark 4. The projective coordinates as described above more precisely are *homogeneous* projective coordinates. There are other types of projective coordinates using transformations of the form $x = X/Z^i$ and $y = Y/Z^j$, see [16]. However, in the context of isogeny-based cryptography, we will only use the coordinates described above and continue to simply call them projective coordinates.

The other main types of operations that we will encounter in this thesis, apart from the elliptic curve group law, naturally are given by isogeny computations. In particular, given a domain curve $E_1(\mathbb{F}_q)$ in short Weierstraß form and a subgroup $G \subset E_1$ of order ℓ, according to Proposition 2 we want to compute the degree ℓ isogeny $\varphi : E_1 \to E_2$; i.e., we want to compute the codomain curve E_2 and be able to map points $P \in E_1$ to $P' = \varphi(P) \in E_2$.

Vélu's formulas [152] suggest that we have to compute ratios of polynomials of degree ℓ for this, where the computational effort is linear in the degree ℓ. For point evaluations, projective coordinates again help to avoid inversions. On the other hand, we also have to compute the parameters a' and b' of the codomain curve E_2. In order to also avoid inversions here, Costello, Longa, and Naehrig proposed to use a projective representation of the curve parameters in the context of SIDH [47]. In particular, for a short Weierstraß curve $E : y^2 = x^3 + ax + b$, we can use $(A : B : C)$ as projective curve parameters instead of the affine parameters (a, b), where $a = A/C$ and $b = B/C$. The projective parameters of codomain curves can then be computed without inversions, and analogously to the projective point coordinates, we only have to switch back to affine parameters if a unique curve representation is required. Note that the use of projective curve parameters requires minor

adjustments to the point addition and doubling formulas, as detailed in [47] (although for a different curve model).

In traditional elliptic curve cryptography however, short Weierstraß curves are not the preferred curve model. Instead, Montgomery curves or (twisted) Edwards curves are used due to their more efficient group law operations. The same is true for isogeny-based cryptography, where implementations usually utilize these two curve models. The remainder of this section briefly introduces Montgomery [117] and twisted Edwards curves [10].

Definition 9 (Montgomery curve)**.** Let $\mathbb{K}$ be a field with $\mathrm{char}(\mathbb{K}) > 2$. Then

$$E_{a,b} : by^2 = x^3 + ax^2 + x$$

is an elliptic curve in *Montgomery form*, where $a \in \mathbb{K}\backslash\{-2, 2\}$ and $b \in \mathbb{K}\backslash\{0\}$.

Following [50, §2.4], every Montgomery curve over a finite field $\mathbb{F}_q$ can easily be transformed into a short Weierstraß curve whenever $\mathrm{char}(\mathbb{F}_q) > 3$. The converse however is not true; for this to be possible, a short Weierstraß curve over $\mathbb{F}_q$ (or its twist) must have group order divisible by 4. The j-invariant of a Montgomery curve is given by

$$j(E_{a,b}) = \frac{256(a^2 - 3)^3}{a^2 - 4}.$$

The above statements about the (non-)equivalence with Weierstraß curves imply that there exist j-invariants over $\mathbb{F}_q$, for which no Montgomery curve over $\mathbb{F}_q$ with curve parameters $(a, b) \in \mathbb{F}_q^2$ exists. Note that the curve parameter b does not play a role for the value of the j-invariant. Indeed, this parameter can be seen as a "twisting factor", and can be disregarded in implementations [50, §2.1].

Again we can use projective point coordinates $(X : Y : Z)$ to avoid inversions in the group law formulas, where the point at infinity is given by $\infty = (0 : 1 : 0)$. Moreover, the efficient arithmetic given by Montgomery in [117] allows for disregarding the Y-coordinate and still performing XZ-only point doublings and differential additions, which require the knowledge of the XZ-coordinates of P, Q, and $P - Q$ in order to compute $P + Q$.

As described above, we can projectivize point coordinates and curve parameters. Instead of a Montgomery curve of the form given above, we then work with an equation of the form

$$E_{(A:B:C)} : By^2 = Cx^3 + Ax^2 + Cx,$$

where $(A : B : C) \in \mathbb{P}^2(\mathbb{K})$, such that $a = A/C$ and $b = B/C$ for the corresponding curve $E_{a,b}$. As hinted above, it furthermore suffices to work with $(A : C) \in \mathbb{P}^1(\mathbb{K})$ in the projective model, since neither the Montgomery curve arithmetic, nor the isogeny computations require the coefficients b or B, respectively.

Following [37], doublings and differential additions in our situation can be computed by the following formulas. Let $E_{a,b}$ be a Montgomery curve over $\mathbb{F}_q$ with A, C satisfying $a = A/C$, and $P, Q \in E_{a,b}$ with $P = (X_P : Z_P)$, $Q = (X_Q : Z_Q)$, and $P - Q = (X_{P-Q} :$

Z_{P-Q}). Then upon precomputation of $A_{24} = A + 2C$ and $C_{24} = 4C$, we can compute $[2]P = (X_{[2]P} : Z_{[2]P})$ via

$$X_{[2]P} = C_{24}(X_P + Z_P)^2(X_P - Z_P)^2,$$
$$Z_{[2]P} = [(X_P + Z_P)^2 - (X_P - Z_P)^2] \cdot [C_{24}(X_P - Z_P)^2 + A_{24}((X_P + Z_P)^2 - (X_P - Z_P)^2)]$$

at a cost of $4\mathbf{M} + 2\mathbf{S} + 4\mathbf{a}$. The sum $P + Q = (X_{P+Q} : Z_{P+Q})$ can be computed via the differential addition formulas

$$X_{P+Q} = Z_{P-Q}[(X_P - Z_P)(X_Q + Z_Q) + (X_P + Z_P)(X_Q - Z_Q)]^2,$$
$$Z_{P+Q} = X_{P-Q}[(X_P - Z_P)(X_Q + Z_Q) - (X_P + Z_P)(X_Q - Z_Q)]^2$$

at a cost of $4\mathbf{M} + 2\mathbf{S} + 6\mathbf{a}$. Scalar point multiplications are then done via the *Montgomery ladder*, which generalizes the usual square-and-multiply approach for exponentiations to scalar multiplications of points on Montgomery curves (see e.g. [117, 50]), and requires one combined doubling and differential addition step per bit of the involved scalar. The computational effort for isogeny computations and efficient explicit formulas will be detailed in Chapter 3.

Another form of elliptic curves was introduced by Edwards [72], resp. Bernstein, Birkner, Joye, Lange, and Peters [10].

Definition 10 (Twisted Edwards curve)**.** Let $\mathbb{K}$ be a field with $\mathrm{char}(\mathbb{K}) > 2$. Then

$$E_{a,d} : ax^2 + y^2 = 1 + dx^2y^2,$$

is an elliptic curve in *twisted Edwards form*, where $ad \neq 0$, $d \neq 1$, and $a \neq d$. For $a = 1$ the twisted Edwards curve $E_{1,d} = E_d$ is called *Edwards curve*.

Again we first look at transformations between curve models.

Proposition 4 ([10, Theorem 3.2])**.** *Let $\mathbb{K}$ be a field with $\mathrm{char}(\mathbb{K}) > 2$. Then every twisted Edwards curve over $\mathbb{K}$ is birationally equivalent over $\mathbb{K}$ to a Montgomery curve. Conversely, every Montgomery curve over $\mathbb{K}$ is birationally equivalent over $\mathbb{K}$ to a twisted Edwards curve.*

The corresponding curve and point transformations are particularly simple, and will play a major role in Chapter 3. Note that the above statement does not hold in general for the equivalence of Montgomery curves and Edwards curves, i.e., twisted Edwards curves with $a = 1$. In this case, there are some restrictions, as detailed in [10, Section 3]. The potential equivalence to Weierstraß curves follows from the corresponding results for Montgomery curves. The j-invariant of a twisted Edwards curve is given by

$$j(E_{a,d}) = \frac{16(a^2 + 14ad + d^2)^3}{ad(a-d)^4}.$$

As in the above cases, the elliptic curve arithmetic with twisted Edwards curves benefits from using projective coordinates. Besides the usual projective representation, there

are certain different types of projective twisted Edwards coordinates, which can be beneficial depending on the concrete application. In particular, there are *inverted*, *extended*, and *completed* coordinates, as detailed in [11]. We will however only encounter the standard projective coordinates, i.e., of the form $(X : Y : Z)$, where $x = X/Z$ and $y = Y/Z$, with points at infinity $(1 : 0 : 0)$ and $(0 : 1 : 0)$, see [11]. Furthermore, we can drop the X-coordinate altogether in this case, and only work with YZ-coordinates here. Due to the simple transformations between twisted Edwards curves and Montgomery curves, one can reformulate the Montgomery XZ-only doubling and differential addition formulas for twisted Edwards YZ-coordinates, and ends up with the same costs of $4\mathbf{M} + 2\mathbf{S} + 4\mathbf{a}$ for point doublings, and $4\mathbf{M} + 2\mathbf{S} + 6\mathbf{a}$ for differential additions. In particular, given a twisted Edwards curve $E_{a,d}$ over $\mathbb{F}_q$, and points $P, Q \in E_{a,d}$ with $P = (Y_P : Z_P)$, $Q = (Y_Q : Z_Q)$, and $P - Q = (Y_{P-Q} : Z_{P-Q})$, we precompute $e = a - d$ and obtain $[2]P = (Y_{[2]P} : Z_{[2]P})$ via

$$\begin{aligned} Y_{[2]P} &= eY_P^2Z_P^2 - (Z_P^2 - Y_P^2) \cdot [eY_P^2 + a(Z_P^2 - Y_P^2)], \\ Z_{[2]P} &= eY_P^2Z_P^2 + (Z_P^2 - Y_P^2) \cdot [eY_P^2 + a(Z_P^2 - Y_P^2)]. \end{aligned}$$

The sum $P + Q = (Y_{P+Q} : Z_{P+Q})$ can be computed via

$$\begin{aligned} Y_{P+Q} &= (Z_{P-Q} - Y_{P-Q})(Y_PZ_Q + Y_QZ_P)^2 - (Z_{P-Q} + Y_{P-Q})(Y_PZ_Q + Y_QZ_P)^2, \\ Z_{P+Q} &= (Z_{P-Q} - Y_{P-Q})(Y_PZ_Q + Y_QZ_P)^2 + (Z_{P-Q} + Y_{P-Q})(Y_PZ_Q + Y_QZ_P)^2. \end{aligned}$$

The efficient computation of isogenies in this case will again be detailed in Chapter 3.

2.4 Cryptographic protocols

In this section we describe several popular cryptographic protocols based on supersingular isogenies. In particular, we will look at the key exchange protocols SIDH, B-SIDH, and CSIDH, which are the main topics of interest throughout this thesis. In addition to this, we give a short overview of other isogeny-based schemes such as digital signature schemes in Section 2.4.4. Before going into the details, we give a high-level view of isogeny-based key exchange protocols.

The basis for these schemes is given by the famous Diffie-Hellman key exchange [68], which uses the discrete logarithm problem in finite fields as computationally hard underlying problem. Miller [116] and Koblitz [103] later introduced an analogue scheme based on elliptic curve discrete logarithms, usually called Elliptic Curve Diffie-Hellman (ECDH).

From a high-level point of view, this scheme chooses an elliptic curve E over a finite field $\mathbb{F}_p$ with a prime p, and a point $P \in E$ of prime order ℓ as public parameters. The participants of the key exchange, as usual called Alice and Bob, then choose an integer a resp. b with $a, b \in (\mathbb{Z}/\ell\mathbb{Z})^*$, which is used as *private key*. Alice computes and publishes $P_A = [a]P$ as her *public key*, and Bob proceeds similarly, and computes his public key

$P_B = [b]P$. Because of the commutativity of the elliptic curve group law, both parties can now compute $[ab]P = [a]P_B = [b]P_A$, which then is used as the *shared secret*. The flow of this protocol can be summarized in the following diagram.

$$\begin{array}{ccc} P & \xrightarrow{\quad a \quad} & [a]P \\ {\scriptstyle b}\downarrow & & \downarrow{\scriptstyle b} \\ [b]P & \xrightarrow[\quad a \quad]{} & [ab]P \end{array}$$

The security of this key exchange protocol relies on the hardness of two computational problems. The *discrete logarithm problem* asks for finding the integer a when P and $[a]P$ are given. The *computational Diffie-Hellman problem* asks for finding $[ab]P$ when P, $[a]P$, and $[b]P$ are given. Both problems are believed to be hard in general in the classical setting. However, if large-scale quantum computers are available, these problems can be solved in polynomial time on such a quantum computer via Shor's algorithm [137].

Remark 5. The elliptic curve Diffie-Hellman key exchange protocol appears to be particularly simple, since only the basic group operations are involved. However, the choice of elliptic curves for this scheme is non-trivial, since it influences both the security and performance. E.g., supersingular elliptic curves, which are commonly used in isogeny-based schemes, are not suitable for ECDH, since their discrete logarithm problem is easier to solve than in the general case due to their small embedding degrees, see [112]. One of the most popular curves that is secure and allows for efficient implementations is Bernstein's Curve25519 [9].

The main idea behind isogeny-based cryptography significantly differs from the approach in ECDH. In particular, the hard underlying problem shifts to the problem of finding an isogeny between two given elliptic curves, instead of the discrete logarithm problem. The high-level view of such a protocol is given in the following diagram, where isogenies are denoted by φ.

$$\begin{array}{ccc} E & \xrightarrow{\quad \varphi_A \quad} & E_A \\ {\scriptstyle \varphi_B}\downarrow & & \downarrow{\scriptstyle \varphi'_B} \\ E_B & \xrightarrow[\quad \varphi'_A \quad]{} & E_{AB} \end{array}$$

This means that we are no longer working with a group of points on an elliptic curve, but with the set of elliptic curves that are isogenous to a given starting curve E. This

gives a hint on why the underlying problem could potentially be quantum resistant; Shor's algorithm explicitly uses the homomorphic property of the group operation to solve the discrete logarithm problem. Resorting to a problem that does not provide this structure thus also means that Shor's algorithm is not applicable.

In the remainder of this section, we will describe three different approaches on how isogeny-based key exchange schemes can be set up.

2.4.1 SIDH

The *Supersingular Isogeny Diffie-Hellman* (SIDH) protocol, introduced in 2011 by Jao and De Feo [94], uses supersingular elliptic curves over $\mathbb{F}_{p^2}$, where p is a large prime. As discussed in Section 2.2, the arising ℓ-isogeny graphs are (almost) $(\ell+1)$-regular expander graphs. This means that a series of random steps in such a graph has excellent mixing properties. Due to this, Charles, Goren, and Lauter first used such isogeny graphs to define a hash function, which exploits this rapid mixing property [38].

For the SIDH setting, we choose two distinct small primes ℓ_A and ℓ_B, and exponents e_A and e_B such that $\ell_A^{e_A} \approx \ell_B^{e_B}$ and $p = \ell_A^{e_A}\ell_B^{e_B} - 1$ is prime. It is then easy to construct a supersingular elliptic curve E over $\mathbb{F}_{p^2}$ via an efficient algorithm by Bröker [29]. In particular, we will always use a curve E where $\#E(\mathbb{F}_{p^2}) = (p+1)^2$, which means that $E(\mathbb{F}_{p^2}) \cong \mathbb{Z}/(p+1)\mathbb{Z} \times \mathbb{Z}/(p+1)\mathbb{Z}$. Thus, the torsion groups $E[\ell_A^{e_A}]$ and $E[\ell_B^{e_B}]$ are completely contained in $E(\mathbb{F}_{p^2})$. Moreover, we can choose points $P_A, Q_A \in E(\mathbb{F}_{p^2})$ of order $\ell_A^{e_A}$ that form a basis of $E[\ell_A^{e_A}]$, and analogously points P_B, Q_B that form a basis of $E[\ell_B^{e_B}]$.

This setup can then be used to perform random walks of length e_i in the ℓ_i-isogeny graph, where $i \in \{A, B\}$. We choose a random positive integer $s_i \leq \ell_i^{e_i}$, and compute $R_i = P_i + [s_i]Q_i$, which is a point of order $\ell_i^{e_i}$. Thus, Proposition 2 implies that we can compute a unique isogeny φ_i (up to isomorphisms) of degree $\ell_i^{e_i}$ and codomain curve E_i, such that $\varphi_i : E \to E_i$ has $\ker \varphi = \langle R_i \rangle$, and $E_i = E/\langle R_i \rangle$. However, since the currently available degree ℓ isogeny formulas have complexity $O(\ell)$ resp. $\widetilde{O}(\sqrt{\ell})$ (see Chapter 3), it is beneficial to compute a series of e_i isogenies of degree ℓ_i, instead of directly computing the full $\ell_i^{e_i}$-isogeny.

To this end, we start by computing $K_{i,1} = [\ell_i^{e_i-1}]R_i$, which is a point of order ℓ_i. In particular, the integer s_i determines in which of the $\ell_i + 1$ subgroups of the ℓ_i-torsion group $K_{i,1}$ lies. We then compute the ℓ_i-isogeny $\varphi_{i,1} : E \to E_2 = E/\langle K_{i,1} \rangle$ and $R_{i,2} = \varphi_{i,1}(R_i)$ of order $\ell_i^{e_i-1}$. In the next step, we compute $K_{i,2} = [\ell_i^{e_i-2}]R_{i,2}$ of order ℓ_i, and the corresponding ℓ_i-isogeny $\varphi_{i,2} : E_2 \to E_3 = E_2/\langle K_{i,2} \rangle$ and $R_{i,3} = \varphi_{i,2}(R_{i,2})$ of order $\ell_i^{e_i-2}$. This can be repeated until φ_{i,e_i} is computed, and the $\ell_i^{e_i}$-isogeny φ_i is the composition of the e_i degree ℓ_i isogenies. Note that all steps in the isogeny graph during this procedure are completely determined by the integer s_i.

With the described procedure, Alice and Bob can compute secret isogenies φ_A resp. φ_B using their secret keys s_A resp. s_B, the public starting curve, and torsion basis points. The resulting curves E_A and E_B then are exchanged as public keys. However, in order to

Alice		Bob
$R_A = P_A + [s_A]Q_A$		$R_B = P_B + [s_B]Q_B$
$\ker(\varphi_A) := \langle R_A \rangle$		$\ker(\varphi_B) := \langle R_B \rangle$
$\varphi_A : E \to E_A$		$\varphi_B : E \to E_B$
	$E_A, \varphi_A(P_B), \varphi_A(Q_B)$	
	$\longrightarrow$	
	$E_B, \varphi_B(P_A), \varphi_B(Q_A)$	
	$\longleftarrow$	
$R'_A = \varphi_B(P_A) + [s_A]\varphi_B(Q_A)$		$R'_B = \varphi_A(P_B) + [s_B]\varphi_A(Q_B)$
$\ker(\varphi'_A) := \langle R'_A \rangle$		$\ker(\varphi'_B) := \langle R'_B \rangle$
$\varphi'_A : E_B \to E_{BA}$		$\varphi'_B : E_A \to E_{AB}$
	$j(E_{BA}) = j(E_{AB})$	

Figure 2.2: SIDH key exchange protocol with public parameters given by a supersingular elliptic curve $E(\mathbb{F}_{p^2})$, and torsion group basis points P_A, Q_A resp. P_B, Q_B.

determine a shared secret, Alice has to repeat an analogous walk through the ℓ_A-isogeny graph starting from E_B, but there are no torsion basis points on E_B publicly available that would allow for Alice to achieve this. Thus, Bob must further send the image of Alice's torsion basis points under his secret isogeny; i.e., Bob's public key is $(E_B, P'_A = \varphi_B(P_A), Q'_A = \varphi_B(Q_A))$, and similarly, Alice's public key is $(E_A, P'_B = \varphi_A(P_B), Q'_B = \varphi_A(Q_B))$. In the shared secret phase of the protocol, both parties proceed analogously; they compute $R'_i = P'_i + [s_i]Q'_i$, and the corresponding sequence of ℓ_i-isogenies to obtain the $\ell_i^{e_i}$-isogeny φ'_i and codomain curves $E_{AB} = E_A/\langle R'_B \rangle$ resp. $E_{BA} = E_B/\langle R'_A \rangle$. The resulting curves E_{AB} and E_{BA} then are isomorphic to $E/\langle R_A, R_B \rangle$, and thus the j-invariant $j(E_{AB}) = j(E_{BA}) \in \mathbb{F}_{p^2}$ can be used as the shared secret (see [94]). The resulting SIDH key exchange is summarized in Figure 2.2.

Security aspects. The hard problems underlying SIDH differ from a pure isogeny problem, which would ask for finding an isogeny of smooth degree between two given curves in our case. Instead, the images of public torsion points are included in public keys; the underlying problems thus are the *supersingular isogeny problem* (SSI), which asks for recovering a secret key from a public key and the public parameters, and the *supersingular computational Diffie-Hellman problem*, which asks for computing the shared secret from the public keys and public parameters [94]. Cryptanalysis on SIDH usually focuses on the SSI problem, although it is not clear whether these problems are equivalent.

The currently best known classical attacks against SSI are generic *meet-in-the-middle* attacks with complexity $O(\sqrt[4]{p})$. However, when considering concrete memory limitations, [1] suggests that the van Oorschot-Wiener parallel collision finding algorithm [151]

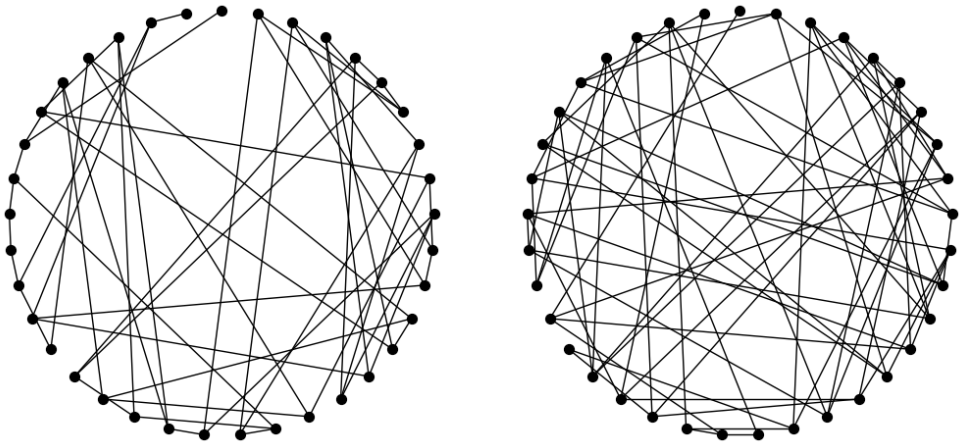


Figure 2.3: Isogeny graphs for degrees 2 (left) and 3 (right) of the isogeny class of the supersingular curve $E : y^2 = x^3 + x$ over $\mathbb{F}_{431^2}$.

is the best known attack, leading to a reduction of the parameter sizes initially proposed in [94]. The classical cost for this is analyzed in detail in [48]. Similarly, it was argued in [94] that Tani's claw finding algorithm [149] of complexity $O(\sqrt[6]{p})$ is the optimal choice for solving this problem on a quantum computer. However, the concrete security analysis in [96] suggests that, as in the classical case, the original parameters from [94, 47] reach a higher security level than initially considered.

We remark that these cryptanalytic efforts focus on a pure isogeny finding problem, and do not make use of the additional information provided by the published point evaluations, which may raise concerns about SIDH's security. Indeed, variants of SIDH, e.g. with highly unbalanced parameters $\ell_A^{e_A} \ll \ell_B^{e_B}$, can be attacked efficiently when exploiting this additional information [125, 108]. However, if SIDH is set up as e.g. described in [94, 47], these attacks do not apply.

On the other hand, there is an active attack on SIDH, which exploits the fact that torsion points are included in the public keys. In particular, malicious torsion points can be sent as part of a public key, which then allow the attacker to learn one bit of the secret key per run, see [80]. Thus, SIDH is only secure if ephemeral keys are used.

Implementation aspects. For efficiency of the isogeny computations, implementations of SIDH usually choose $\ell_A = 2$ and $\ell_B = 3$, which also means that $p \equiv 3 \bmod 4$, and the supersingular Montgomery curve $E(\mathbb{F}_{p^2}) : y^2 = x^3 + x$ can be chosen as starting curve. The isogeny and elliptic curve point operations can entirely be computed using projective coordinates and curve parameters, see [47]. A toy example of 2- and 3-isogeny graphs as used in SIDH is given in Figure 2.3.

The strategy of computing a chain of isogenies as described above is a *multiplication-based* strategy. In contrast to this, one can store an intermediate point of the involved large

scalar multiplications, evaluate this point under the next ℓ_i-isogeny, and thus have a smaller scalar multiplication in the next step at the cost of an additional isogeny point evaluation. In [60], *optimal strategies* are described, which minimize the total computational effort by pushing such points through isogenies.

A variant of SIDH participates in the NIST standardization process, which requires key encapsulation mechanisms (KEM) instead of key exchanges. Thus, in the *Supersingular Isogeny Key Encapsulation* (SIKE) [93], a variant of the Fujisaki-Okamoto transform [78] is applied to SIDH, and allows one of the two involved parties to reuse a long-term key pair.

SIDH and SIKE public keys contain a curve parameter and two x-coordinates of points (resp. only three x-coordinates, see [47]), and thus three $\mathbb{F}_{p^2}$-elements or six $\mathbb{F}_p$-elements, which amounts to $6\log_2 p$ bits. However, this key size can be further compressed to $\frac{7}{2}\log_2 p$ bits as explained in [4, 46]. The computational overhead for this results in less than a twofold slowdown, see [157, 120].

2.4.2 B-SIDH

B-SIDH is a variant of SIDH, published by Costello [44], where the naming hints on the usage of the "twisting factor" b (resp. B) of Montgomery curves. Recall from Section 2.4.1 that the isogeny problem in SIDH significantly differs from a generic isogeny problem. In particular, a random walk in the ℓ_i-isogeny graph as in SIDH can potentially only reach a maximum of $\ell_i^{e_i} \approx \sqrt{p}$ different vertices. However, according to Theorem 4 the full isogeny graph contains approximately $p/12$ vertices, which means that only a small fraction of nodes can be reached through an SIDH random walk. This also implies that the SIDH isogeny problem is significantly easier to solve than a generic isogeny problem, which asks for finding an isogeny between any two given vertices of the isogeny graph. On the other hand, SIDH requires to define the underlying prime in a way that guarantees the whole $\ell_i^{e_i}$-torsion of the curves in the respective isogeny class to be $\mathbb{F}_{p^2}$-rational for $i \in \{A, B\}$, e.g. by fixing $p = \ell_A^{e_A}\ell_B^{e_B} - 1$. In a way, this makes the parameter choices of SIDH seem to be non-optimal, since most of the existing graph structure is not even used for the involved isogeny walks.

B-SIDH uses the observation that each curve that appears in SIDH, i.e., satisfying $\#E(\mathbb{F}_{p^2}) = (p+1)^2$, has a quadratic twist E^t with $\#E^t(\mathbb{F}_{p^2}) = (p-1)^2$ [44]. This means that although both curves are supersingular due to Theorem 3, and E and E^t are isomorphic over $\mathbb{F}_{p^4}$, they are neither isomorphic, nor isogenous over $\mathbb{F}_{p^2}$ (see Theorem 2). However, $j(E) = j(E^t)$, and thus the nodes of the two corresponding isogeny graphs over $\mathbb{F}_{p^2}$ containing E and E^t have the same j-invariants. This further has to hold, since over $\mathbb{F}_{p^4}$, these quadratic twists are always contained in the same graph and vertex.

The key observation in B-SIDH is that we can in fact work with *both* sets of curves over $\mathbb{F}_{p^2}$. In particular, let b be a square and γ a non-square over $\mathbb{F}_{p^2}$, and let $\mathbb{F}_{p^4} = \mathbb{F}_{p^2}(\delta)$,

where $\delta^2 = \gamma$. For a Montgomery curve

$$E_{a,b}(\mathbb{F}_{p^2}) : by^2 = x^3 + ax^2 + x$$

such a quadratic twist is then given by

$$E^t_{a,\gamma b}(\mathbb{F}_{p^2}) : \gamma by^2 = x^3 + ax^2 + x,$$

and the twisting isomorphism $\sigma : E_{a,b}(\mathbb{F}_{p^4}) \to E^t_{a,\gamma b}(\mathbb{F}_{p^4})$ is given by $\sigma(x, y) = (x, \delta y)$. However, since isogeny-based protocols use x-only and a-only arithmetic, twisting a curve and computing the isomorphism become trivial. In fact, they would only be noticeable in the y-coordinates and b-parameters. Given a parameter $a \in \mathbb{F}_{p^2}$ and a coordinate $x \in \mathbb{F}_{p^2}$ such that the corresponding $y \neq 0$, this arithmetic does not notice whether the corresponding point lies on E or E^t, but we can still perform point operations and isogeny computations. Thus, the SIDH setting is said to be *twist agnostic* [44].

In the B-SIDH protocol we now choose a prime p, and analogous to the SIDH setting, a supersingular starting curve $E(\mathbb{F}_{p^2})$ such that $E(\mathbb{F}_{p^2}) \cong \mathbb{Z}/(p+1)\mathbb{Z} \times \mathbb{Z}/(p+1)\mathbb{Z}$. Alice can now choose any $M \mid (p+1)$, and given some public M-torsion basis points, compute a secret M-isogeny, similar to the SIDH case.[6] On the other hand we have $E^t(\mathbb{F}_{p^2}) \cong \mathbb{Z}/(p-1)\mathbb{Z} \times \mathbb{Z}/(p-1)\mathbb{Z}$. Bob can thus choose any $N \mid (p-1)$ coprime to M, and we are guaranteed to be able to find public N-torsion basis point, which Bob can use to compute a secret N-isogeny.

However, in addition to the isogeny computations, both parties must evaluate the other party's public torsion points. For Alice, this would mean to evaluate points on $E^t(\mathbb{F}_{p^2})$ under her secret isogeny, which has $E(\mathbb{F}_{p^2})$ as domain curve. Thus, for this operation to make sense, we have to lift the protocol to $\mathbb{F}_{p^4}$, and compose the isogeny evaluations with twisting isomorphisms in order to move Bob's points to Alice's curve. Fortunately, the twist agnostic isogeny arithmetic discussed above allows Alice to simply push Bob's points through her isogeny, since the involved twisting operations become trivial. Lifting the protocol to $\mathbb{F}_{p^4}$ thus becomes merely a theoretical tool that is required to define the protocol; in practice however, this is not noticeable. Like in SIDH, the resulting evaluated points are included in the respective public key in order to allow both parties to compute the shared secret.

The main advantage of B-SIDH is the possibility of working both in the $(p+1)$- and $(p-1)$-torsion, and thus of choosing smaller primes p, which in turn allow for a more efficient field arithmetic over $\mathbb{F}_{p^2}$. However, for the isogeny computations to be efficient, we require both $p+1$ and $p-1$ to contain enough small prime factors.

Remark 6. The potential usage of quadratic twists in the context of SIDH was first mentioned by De Feo [58], although without the construction of an explicit scheme. Prior to

[6] In particular, for any prime factor ℓ of M, we compute an ℓ-isogeny, instead of directly computing the full M-isogeny.

the publication of B-SIDH, Matsuo proposed a variant of SIDH that exploits quadratic twists [111]. However, the proposed instantiations significantly differ from Costello's proposal, see [44, §1.3].

Security aspects. Due to the above discussion, p, M, and N have to be chosen appropriately to achieve both security and efficiency. Analogous to the case of SIDH, collision finding attacks still work in the same way classically and quantumly, thus forcing $M \approx N$ to be of the same size as the SIDH parameters $\ell_A^{e_A} \approx \ell_B^{e_B}$ for reaching the same security level.

However, when significantly reducing p compared to SIDH, generic isogeny finding algorithms become competitive. In particular, the Delfs-Galbraith algorithm [65] tries to find paths from the public key resp. starting curve to subfield curves with j-invariants in $\mathbb{F}_p$. When this is done, it solves the much easier isogeny problem in the isogeny graph over $\mathbb{F}_p$. The complexity of this classical algorithm is $O(\sqrt{p})$. The best quantum algorithm is given in [22] and applies Grover's algorithm [89] to Delfs-Galbraith, with a complexity of $O(\sqrt[4]{p})$. Thus it can be deduced that p must be slightly larger than M and N, but can be chosen much smaller than in SIDH, see [44] for details.

Apart from this, B-SIDH inherits other security aspects from SIDH; the active attack from [80] also applies to B-SIDH, meaning that only ephemeral keys can be used. In addition, the torsion point attacks for unbalanced parameters $M \ll N$ from [125, 108] apply.

Implementation aspects. From an implementation point of view, B-SIDH works completely analogous to SIDH, except for the deviating isogeny degrees. The main problem for efficient implementations is to find suitable parameters. Current cryptanalytic results suggest that B-SIDH requires primes of roughly 256 bits, and $M \approx N \approx 2^{216}$, in order to achieve NIST level 1 security [44]. Thus, we have to find such primes, where both $p+1$ and $p-1$ contain smooth factors of size at least 2^{216}. Chapter 7 will treat the question of how to find such primes in detail.

An advantage of B-SIDH is its small public keys, which are slightly smaller than the compressed SIDH public keys [44]. The current literature explores different scenarios for applications of B-SIDH. If both parties must be equally fast, one can choose parameters such that the largest prime factors of M and N are roughly of the same size, where we recall that these prime factors will have the most influence on the total runtime due to the isogeny computation complexity of $\widetilde{O}(\sqrt{\ell})$ [13]. If the performance on one side of the protocol is much more important than on the other side, one can e.g. find M with much smaller prime factors, while the prime factors of N considerably grow, however without making the runtime impractical. This scenario could be of interest if the protocol involves communication between a server and clients, or when constrained devices are involved (see [44]).

2.4.3 CSIDH

Commutative Supersingular Isogeny Diffie-Hellman (CSIDH) is a key exchange protocol that was introduced by Castryck, Lange, Martindale, Panny, and Renes [35]. Although it follows SIDH in working with supersingular elliptic curves, its mathematical background considerably differs. In fact, CSIDH much more closely resembles the schemes using ordinary elliptic curves by Couveignes [51], and Rostovtsev and Stolbunov [133] (CRS). In this section, we only give a brief introduction on the mathematical background that sets the scene for the following chapters. For algorithmic and implementation details, we refer to Chapter 3. A similar, but more detailed description of the mathematical background of CSIDH in the context of hard homogeneous spaces will be given in Chapter 6. We further refer to [52, 61, 35] for more details.

In this section, we restrict to elliptic curves over $\mathbb{F}_p$, where p is prime. If E is a supersingular curve over $\mathbb{F}_p$, then necessarily $\#E(\mathbb{F}_p) \equiv 1 \bmod p$ (cf. Theorem 3), and Hasse's theorem (cf. Theorem 1) implies that $\#E(\mathbb{F}_p) = p + 1$. Further recall that there are approximately $p/12$ isomorphism classes of supersingular elliptic curves over $\mathbb{F}_{p^2}$ (see Theorem 4); [35] heuristically argues that there are roughly $\sqrt{p}$ such isomorphism classes of curves over $\mathbb{F}_p$.

As noted in Section 2.2, the ring of $\mathbb{F}_p$-rational endomorphisms $\mathrm{End}_{\mathbb{F}_p}(E)$ of a supersingular curve E is isomorphic to an order $\mathcal{O}$ of the imaginary quadratic field $\mathbb{Q}(\sqrt{-p})$. For such an order, the *ideal class group* $\mathrm{cl}(\mathcal{O})$ is a finite abelian group, given by the quotient of the group of invertible ideals of $\mathcal{O}$ by the group of its principal ideals.

Now let $\mathcal{E}\ell\ell_p(\mathcal{O}, \pi)$ be the set of supersingular elliptic curves over $\mathbb{F}_p$ with $\mathrm{End}_{\mathbb{F}_p}(E) \cong \mathcal{O} \subset \mathbb{Q}(\sqrt{-p})$. Then $\mathrm{cl}(\mathcal{O})$ acts freely and transitively on $\mathcal{E}\ell\ell_p(\mathcal{O}, \pi)$ via isogenies in the following way [35, Theorem 7]. For a given invertible ideal $\mathfrak{a} \in \mathcal{O}$, define the $\mathfrak{a}$-*torsion* subgroup of E as

$$E[\mathfrak{a}] = \{P \in E(\overline{\mathbb{F}_p}) \mid \alpha(P) = \infty \text{ for all } \alpha \in \mathfrak{a}\}.$$

This is a finite subgroup of E, and we can compute a unique isogeny $\varphi_\mathfrak{a} : E \to E/E[\mathfrak{a}]$ with kernel $E[\mathfrak{a}]$, and we write $\mathfrak{a} * E = E/E[\mathfrak{a}]$. Further, if $\mathfrak{b}$ is an ideal in the same class as $\mathfrak{a}$, then $E/E[\mathfrak{a}] \cong E/E[\mathfrak{b}]$. The degree of $\varphi_\mathfrak{a}$ is the norm of $\mathfrak{a}$, and thus the efficiency of the isogeny computation hinges on the ability to find a representative of $\mathfrak{a}$ of small norm.

To this end, let ℓ be a prime such that the ideal $(\ell) \in \mathcal{O}$ splits into a product $(\ell) = \mathfrak{l}\bar{\mathfrak{l}}$, where $\mathfrak{l}$ and $\bar{\mathfrak{l}}$ are prime ideals of norm ℓ, and $\mathfrak{l}^{-1} = \bar{\mathfrak{l}}$ in $\mathrm{cl}(\mathcal{O})$. Then we aim at writing any element $\mathfrak{a} \in \mathcal{O}$ as a product

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

with such ideals $\mathfrak{l}_i$ of prime norm ℓ_i, where the ℓ_i and exponents e_i are sufficiently small to make the involved isogeny computations efficient.

The CSIDH design choices allow for this; in particular, we choose a prime of the form

$$p = 4\prod_{i=1}^{n} \ell_i - 1,$$

where the factors ℓ_i are small odd primes. Then we can work with the isomorphism class of the Montgomery curve $E : y^2 = x^3 + x$ over $\mathbb{F}_p$ which has $\mathrm{End}_{\mathbb{F}_p}(E) \cong \mathbb{Z}[\pi]$, where the Frobenius endomorphism π can be identified with $\sqrt{-p}$. All $\mathbb{F}_p$-isomorphism classes of this isogeny class can then be uniquely characterized by a Montgomery curve of the form $E : y^2 = x^3 + ax^2 + x$, and thus by a parameter $a \in \mathbb{F}_p$.

Moreover, we are guaranteed that all of the ℓ_i split in $\mathbb{Z}[\pi]$ as described above, i.e., $(\ell_i) = \mathfrak{l}_i\bar{\mathfrak{l}}_i = \langle \ell_i, \pi - 1\rangle \cdot \langle \ell_i, \pi + 1\rangle$. Computing the action of $\mathfrak{l}_i$ on any curve E in the isogeny class is easy, since we have $E[\mathfrak{l}_i] = E[\ell_i] \cap \ker(\pi - 1) = E[\ell_i] \cap E(\mathbb{F}_p)$. Similarly, we have $E[\mathfrak{l}_i^{-1}] = E[\ell_i] \cap \ker(\pi + 1)$. Apart from the point ∞, this amounts to $\mathbb{F}_{p^2}$-rational points $P = (x, y)$ of $E[\ell_i]$, where $x \in \mathbb{F}_p$ and $y \in \mathbb{F}_{p^2} \backslash \mathbb{F}_p$. However, due to the x-only arithmetic, this will not be noticeable in practice, and we can entirely work over $\mathbb{F}_p$. Computing the action of such an $\mathfrak{a} = \prod \mathfrak{l}_i^{e_i}$ now simply amounts to computing a sequence of isogenies of small degree, and thus of finding $E[\mathfrak{l}_i]$ resp $E[\mathfrak{l}_i^{-1}]$ on the current curve E, and computing $E/E[\mathfrak{l}_i]$ resp. $E/E[\mathfrak{l}_i^{-1}]$ through the usual isogeny formulas.

In spite of these desirable properties, we cannot efficiently compute the action of any given $\mathfrak{a} \in \mathrm{cl}(\mathcal{O})$, since in general the cardinality and structure of $\mathrm{cl}(\mathcal{O})$ is not known, and we cannot find a decomposition of the form $\mathfrak{a} = \prod \mathfrak{l}_i^{e_i}$. In CSIDH, this is solved by instead directly sampling the integer exponents e_i from small intervals $[-B_i, B_i]$, such that the number of possible exponent vectors is approximately $\#\mathrm{cl}(\mathcal{O}) \approx \sqrt{p}$. This approach uses the heuristic that the number of exponent vectors that are equivalent is small among these small norm vectors, and therefore almost the whole class group is represented in this way [35].

The CSIDH key exchange protocol now is straightforward due to the commutative group action. Alice and Bob choose a secret $\mathfrak{a}$ resp. $\mathfrak{b}$ through sampling such exponent vectors. They compute the public keys $E_A = \mathfrak{a} * E$ resp. $E_B = \mathfrak{b} * E$, and both parties can compute the shared secret $\mathfrak{ab} * E = \mathfrak{a} * E_B = \mathfrak{b} * E_A$.

Remark 7. As mentioned in Section 2.2, the isogeny graphs in the CSIDH case resemble the isogeny volcanoes of ordinary elliptic curves. The choices made in CSIDH imply that we work in a horizontal isogeny class, i.e., all involved isogenies are horizontal. The arising ℓ_i-isogeny graphs then consist of isogeny cycles, resembling the top-level craters of the ordinary case. The CSIDH specification implies that we move in a union of ℓ_i-isogeny graphs, as depicted in Figure 2.4. Note that using j-invariants does not uniquely determine the involved $\mathbb{F}_p$-isomorphism classes here. This is due to quadratic twists, which have the same j-invariant, but are not isomorphic over $\mathbb{F}_p$, and thus are contained in distinct vertices of the $\mathbb{F}_p$-isogeny graph. We refer to [36] for more details.

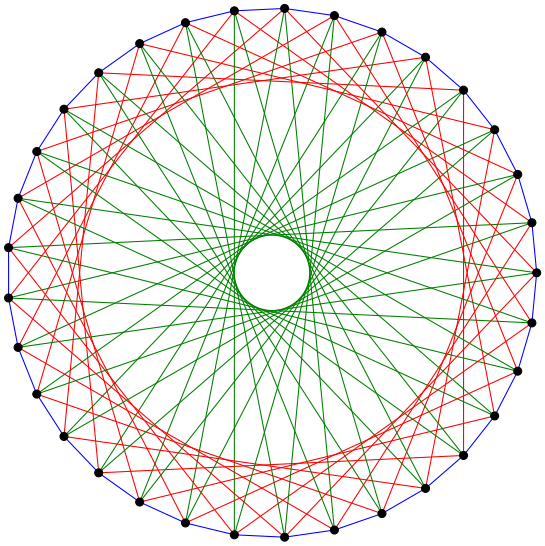


Figure 2.4: $\mathbb{F}_p$-isogeny graph for degrees 3 (blue), 5 (green), and 11 (red) of the isogeny class of the supersingular curve $E : y^2 = x^3 + x$ over $\mathbb{F}_p$ with $p = 659$.

Remark 8. The Couveignes-Rostovtsev-Stolbunov key exchange scheme [51, 133] follows the same theory, but uses ordinary elliptic curves and is much slower, if not impractical. De Feo, Kieffer, and Smith improved this by fixing a set of small odd primes ℓ_i, searching for primes p such that $p \equiv -1 \bmod \ell_i$ for all these ℓ_i, and finally searching ordinary curves E such that $\#E(\mathbb{F}_p) \equiv 0 \bmod \ell_i$ for all the ℓ_i [61]. As in CSIDH, this guarantees these ℓ_i to split, and points of orders ℓ_i are defined over $\mathbb{F}_p$ and can be used for efficient isogeny computations. Thus, for these ℓ_i, the same strategy as above can be applied. However, this search proved to be successful only for a very small number of primes ℓ_i, and the resulting implementation performance is much worse than in CSIDH. In fact, for primes that do not fulfill the properties above, isogenies can be computed through other means, which are however much more expensive, and thus partly explain the slow performance of the CRS scheme. The key observation of CSIDH is that when working with supersingular curves, finding such desirable curves is straightforward, as described above.

Security aspects. Analogously to SIDH and B-SIDH, the same meet-in-the-middle and claw finding attacks apply to CSIDH, both classically and quantumly.

In contrast to SIDH and B-SIDH, the underlying problem in CSIDH is a pure isogeny finding problem, since no evaluated torsion points are included in public keys. Thus, concerns about possible attack avenues using these points like in SIDH do not apply here. On the other hand, the commutative class group action induces much more structure than in the case of SIDH, and indeed, this allows for a subexponential quantum attack. It was first noted by Childs, Jao, and Soukharev for the case of ordinary curves [40], and analyzed in [35] for the case of supersingular curves, that the underlying problem can be expressed as

an abelian hidden-shift problem. This implies that Kuperberg's subexponential quantum algorithm [107], as well as its variants by Regev [130] resp. Kuperberg [106], are applicable to CSIDH.

This however does not mean that CSIDH is broken; Castryck et al. estimate that using the CSIDH-512 parameter set with an underlying 511-bit prime suffices to reach NIST level 1 [35]. In follow-up work the costs of the quantum attack [21] and the corresponding quantum oracle [17] have been analyzed in more detail. Recently, Bonnetain and Schrottenloher [25] and Peikert [124] concluded that CSIDH-512 falls short of the desired quantum security levels, and thus the underlying prime sizes have to be significantly increased. However, these results are currently debated, mainly due to potential ambiguities in NIST's definition of security levels. Essentially, this means that the exact quantum security of CSIDH is not known yet.

A significant advantage of CSIDH compared to SIDH and B-SIDH is its non-interactive structure and efficient key validation. In particular, active attacks as described for SIDH do not apply since CSIDH public keys can be efficiently validated, which means that CSIDH allows for the usage of static keys for all involved parties.

Remark 9. Since CSIDH-512 was the first and only implemented instantiation of CSIDH from [35], follow-up work on efficient implementations mainly focused on CSIDH-512. Thus, also the following chapters only consider implementations of CSIDH-512 in order to allow for comparability. However, even if a different underlying prime is chosen, it is straightforward to apply the corresponding approaches and optimizations to a different instantiation of CSIDH.

Implementation aspects. For details on efficient implementations, we refer to Chapter 3 and Chapter 4, where fast isogeny computations in the context of CSIDH and efficient constant-time implementations are treated.

It is worth to note that CSIDH's public keys only contain a supersingular Montgomery curve over $\mathbb{F}_p$, which can be represented by a single element of $\mathbb{F}_p$. Thus, when considering the security analysis from [35], the key sizes are even smaller than in the case of SIDH or B-SIDH. On the other hand, currently CSIDH can be viewed as being roughly an order of magnitude slower than SIDH.

2.4.4 Other protocols

Apart from the described key exchange schemes, there are various other isogeny-based protocols. We briefly comment on some of these schemes, mainly limiting to schemes that will become relevant throughout this thesis.

In the realm of digital signatures, SIDH-based protocols have proven to yield non-practical signature sizes and performance [156, 81]. The CSIDH-based signature scheme SeaSign by De Feo and Galbraith [59, 64] is similarly slow, while pointing out that a much

more efficient signature scheme could be defined if CSIDH class group structures were known. To this end, Beullens, Kleinjung, and Vercauteren improved the state-of-the-art of such computations by successfully determining the CSIDH-512 class group structure, and published CSI-FiSh [19], which is an efficient and practical signature scheme, although being limited to the CSIDH-512 parameters as long as larger class group computations are out of reach.

A different and promising signature scheme based on isogenies, called SQISign, has recently been published by De Feo, Kohel, Leroux, Petit, and Wesolowski [62]. Remarkably, the combined public key and signature size is much smaller than for any other signature scheme in the NIST PQC standardization process. For efficient instantiations, SQISign requires similar primes as B-SIDH, as discussed in Chapter 7.

Especially the non-interactive structure of CSIDH, and its similarity to traditional discrete logarithm-based protocols, allows for many more advanced applications of isogeny-based cryptography. As an example, Chapter 6 presents CSIDH- and CSI-FiSh-based threshold schemes.

Chapter 3

A fast variable-time implementation of CSIDH

3.1 Introduction

The Commutative Supersingular Isogeny Diffie-Hellman (CSIDH) scheme by Castryck, Lange, Martindale, Panny, and Renes [35] is a promising primitive due to its unique features among quantum-resistant schemes. In particular, as described in Section 2.4, its non-interactive structure and the possibility of using long-term static keys mean that CSIDH could potentially be used as a drop-in replacement for currently used pre-quantum Diffie-Hellman schemes. However, its exact quantum security is not known, and its performance is rather slow, i.e., an order of magnitude slower than SIDH or SIKE at similar security levels.

In this chapter we review the implementation by Castryck et al. from [35], and present several optimizations. In particular, we explain a simple way to significantly reduce the computational cost of point multiplications, and improve the involved isogeny computation by exploiting the well-known correspondence between Montgomery and twisted Edwards curves. We note that the implementation from [35] is a variable-time proof-of-concept implementation. Our improvements do not change the variable-time character of the implementation, but naturally also apply to constant-time implementations.

Organization. In the following section we give an introduction to the algorithmic features of CSIDH, mainly focusing on an implementer's point of view. Section 3.3 introduces a way to restructure the class group action algorithm, which allows for a reduction of the computational effort of scalar point multiplications. In Section 3.4 we review some methods to compute isogenies, i.e., point evaluations and computations of the image curves. In the first case, we employ an observation of Costello and Hisil [45] for a speed-up to the implementation of [35], whereas in the latter case, we exploit more efficient isogeny computations for twisted Edwards curves, in order to compute the Montgomery image curves

Algorithm 1: Evaluating the class group action.

Input : $A \in \mathbb{F}_p$ and a list of integers $(e_1, \ldots, e_n)$.
Output: A' such that $[\mathfrak{l}_1^{e_1} \cdot \cdots \cdot \mathfrak{l}_n^{e_n}] * E_A = E_{A'}$.

```
1  while some e_i ≠ 0 do
2      Sample a random x ∈ F_p.
3      Set s ← +1 if x^3 + Ax^2 + x is a square in F_p, else s ← −1.
4      Let S = {i | sign(e_i) = s}.
5      if S = ∅ then
6          Go to line 2.
7      P = (x : 1), k ← ∏_{i∈S} ℓ_i, P ← [(p + 1)/k]P.
8      foreach i ∈ S do
9          K ← [k/ℓ_i]P
10         if K ≠ ∞ then
11             Compute a degree ℓ_i isogeny φ : E_A → E_{A'} with ker(φ) = ⟨K⟩:
12             A ← A', P ← φ(P), k ← k/ℓ_i, e_i ← e_i − s.
```

more efficiently. We give implementation results according to our contributions in Section 3.5. Section 3.6 concludes this chapter and presents the current state-of-the-art from follow-up related work that was published subsequent to the publication of this chapter's material in [114].

3.2 Implementations of CSIDH

We recall some facts from Section 2.4 about the setup of CSIDH, and describe the algorithmic design of the class group action computation. We follow the choices and implementation by Castryck et al. [35] here. The algorithmic description and notation follow along the lines of Algorithm 1.

First, we define a prime $p = 4 \cdot \ell_1 \cdot \cdots \cdot \ell_n - 1$, where $\ell_1, \ldots, \ell_n$ are small distinct odd primes. Then we choose a supersingular curve E_0 over $\mathbb{F}_p$, usually we pick $E_0 : y^2 = x^3 + x$. Therefore we have $\#E_0(\mathbb{F}_p) = p + 1$, and $E_0(\mathbb{F}_p)$ contains points of order ℓ_i for $i = 1, \ldots, n$. Note that the factor 4 is required to ensure that we can use Montgomery curves.

The private key contains n integers sampled from an interval $[-B, B]$, i.e., has the form $(e_1, \ldots, e_n)$. Following Section 2.4, it determines a class group element $\mathfrak{a} \in \mathrm{cl}(\mathcal{O})$ through $\mathfrak{a} = \prod_i \mathfrak{l}_i^{e_i}$. Thus, for each i the absolute value $|e_i|$ determines how many isogenies of degree ℓ_i are to be computed, while the sign of e_i states if we have to use points defined over $\mathbb{F}_p$ or $\mathbb{F}_{p^2} \setminus \mathbb{F}_p$ on the current curve E to generate their kernels. This corresponds to points in $\ker(\pi - 1)$ resp. $\ker(\pi + 1)$, for which we also write $E[\pi - 1]$ resp. $E[\pi + 1]$.

For the computation of isogenies, we could use a naïve approach, for each required isogeny producing a point of suitable order on the current curve to be able to generate the respective kernel for Vélu's formulas. However, Algorithm 1 follows a more efficient approach by combining several isogeny computations from only one sampled point in the following way. At each step, we choose a random point P on the current curve by sampling a random $x \in \mathbb{F}_p$, and check in which of the cases above this leads us by checking the minimal field of definition of the corresponding y-coordinate by a square root check via a Legendre symbol computation. We then eliminate the possible unwanted factors in the order of P by multiplying it by $4 \cdot \prod_{j \notin S} \ell_j$, where the set S collects all the i for which e_i has the corresponding sign.

After this, we fix an ℓ_i with $i \in S$ and iterate over all ℓ_j for $j \in S$ to compute $k/\ell_i = \prod_{j \in S, j \neq i} \ell_j$, and compute the point $K = [k/\ell_i]P$. In other words, we remove all possibly remaining factors from the order of K except for ℓ_i, and check whether the resulting K has order ℓ_i and can be used as kernel generator for computing an ℓ_i-isogeny, i.e., if $K \neq \infty$. If so, we compute the corresponding isogeny φ and push P through via computing $\varphi(P)$. Then we move to the next prime from S and proceed in the same way. However, we don't have to consider the previously encountered ℓ_i in the multiplication that produces the potential kernel generator point, since the isogeny evaluation of P already eliminates the respective factor from its order, or in the other case, the order of P did not contain the previous ℓ_i as factor in the first place. Thus, we can update k after each step via $k \leftarrow k/\ell_i$, making the required multiplications gradually smaller.

After this loop is completed, we proceed in the same way, and sample new random points until all of the required isogenies are computed. The resulting curve then forms the public key, or the shared secret, respectively. Note that the computational effort in Algorithm 1 highly depends on the private key, which determines the number of required isogenies. Therefore, for the practical usage of CSIDH, it is important to transform this into a constant-time algorithm without adding much computational overhead.

Public keys can efficiently be validated by checking for supersingularity. We can simply sample a random point P on the curve corresponding to the received public key, and for each ℓ_i, compute $Q_i = [(p+1)/\ell_i]P$. For all i with $Q_i \neq \infty$, we compute $[\ell_i]Q_i$ and $d = \prod \ell_i$. If any of these $[\ell_i]Q_i \neq \infty$, the curve cannot be supersingular, since $\#E(\mathbb{F}_p) \nmid p+1$. If this is not the case, and $d > 4\sqrt{p}$, the curve must be supersingular, as can be seen from the Hasse interval and Lagrange's theorem (see [35]). Otherwise, the procedure can be repeated with a different point P. Following this approach, it is not possible to wrongly classify an ordinary curve as supersingular. Therefore, we can check if a public key has been honestly generated, and thus can prevent certain kinds of active attacks.

Choice of parameters. The following discussions and implementation results refer to the CSIDH-512 parameter set proposed in [35] for NIST's post-quantum security level 1. In particular, $p = 4 \cdot \ell_1 \cdot \cdots \cdot \ell_{74} - 1$, where $\ell_1, \ldots, \ell_{73}$ are the 73 smallest distinct odd primes

and $\ell_{74} = 587$. The integer elements of the private keys $(e_1, \ldots, e_{74})$ are chosen from the interval $[-5, 5]$, which means that the keyspace has size $11^{74} \approx 2^{256} \approx \sqrt{p}$. Assuming that there are not many equivalent key vectors, this heuristically covers most of the class group since $\mathrm{cl}(\mathcal{O}) \approx \sqrt{p}$. This parameter set leads to a prime size and public key lengths of 511 bit resp. 64 bytes. As mentioned before, the assessment of the exact quantum security and thus the appropriate choice of parameters and is still an open problem.

3.3 Elliptic curve point multiplications

Let $\alpha = \frac{p+1}{4} = \ell_1 \cdot \ell_2 \cdot \cdots \cdot \ell_n$. For the sake of simplicity, we consider a private key $(e_1, \ldots, e_n)$, where all $e_i > 0$, and return to the general case later. We quickly reiterate the behavior of Algorithm 1 in this case and describe a simple but effective speedup. First, we sample random points on the current curve E_0 until finding P with a y-coordinate defined over the corresponding field $\mathbb{F}_p$, and set $P_0 = [4]P$ in order to remove the possible factor 4 from its order. Then we compute $K_0 = [\frac{\alpha}{\ell_1}]P_0$. If $K_0 = \infty$, the order of P does not contain the factor ℓ_1, and we cannot use it to compute an isogeny of degree ℓ_1 and set $P_1 = P_0$ and $E_1 = E_0$. If however $K_0 \neq \infty$, then K_0 must have order ℓ_1 and can be used as generator of the kernel of an isogeny of degree ℓ_1, mapping to a curve E_1. In this case, we pull P_0 through the isogeny and obtain a point $P_1 \in E_1$. Note that this implies that the order of P_1 is the order of P_0 divided by ℓ_1. Therefore, for checking if we can use P_1 to compute an isogeny of degree ℓ_2, it suffices to compute $K_1 = [\frac{\alpha}{\ell_1 \cdot \ell_2}]P_1$ and proceed as before. Following this approach, the required factor for the scalar multiplication of P_j to obtain K_j reduces at each step, until only the factor ℓ_n remains for the computation of K_{n-1}, and no more multiplication is required for obtaining K_n.

The implementation of [35] goes through the primes in ascending order, starting with small degree isogenies. However, we found it advantageous to change the direction of the loop, i.e., go through the primes in descending order. By doing this, we can eliminate the larger factors of $p + 1$ first, and therefore end up with multiplications by significantly smaller factors as we proceed through the loop. Note that as soon as all isogenies of a certain degree have been computed, e.g. $|e_i|$ isogenies of degree ℓ_i, we include the factor ℓ_i in the first multiplication to compute P_0, making sure that the order of P_0 is not divisible by ℓ_i. We can then ignore the factor ℓ_i in the loop, which slightly reduces the advantage of our approach every time this occurs. However, we note that our approach is advantageous as long as at least two factors are left in the loop.

Figure 3.1 shows the effect of our approach, compared to the implementation of [35]. Note that per bit of the factor of an elliptic curve point multiplication one step in the Montgomery ladder is carried out, i.e., one combined doubling and differential addition. Therefore, in the first loop, at each multiplication the computational effort is reduced by δ_i times the cost of a ladder step, where δ_i is the difference between the two plots for a given

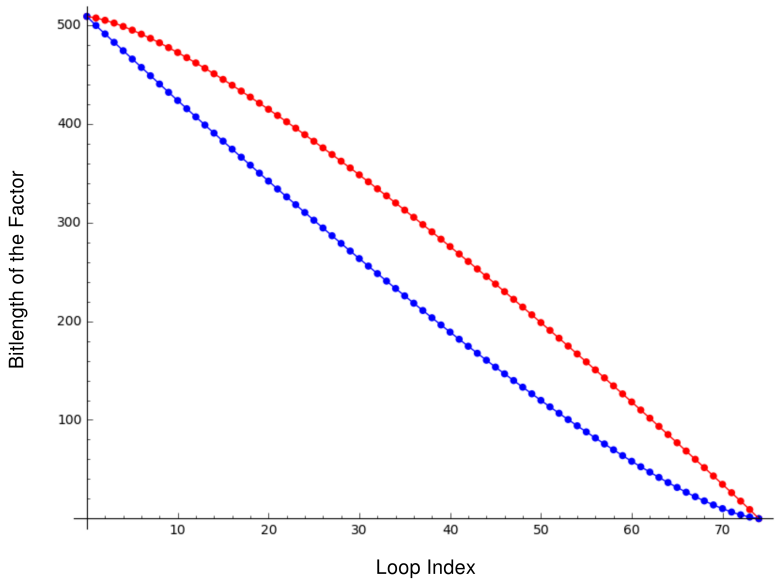


Figure 3.1: Bitlengths of factors during the first loop, when all e_i have the same sign. The red line follows the algorithm of [35], the blue line follows our described approach.

$i < n$, and hence $\delta_i \cdot (8\mathbf{M} + 4\mathbf{S} + 8\mathbf{a})$. As discussed before, the number of saved operations may reduce in the following loops.

In the general case, our assumption that all elements of the private key share the same sign obviously does not hold. However, the described effect will translate at a lower scale to both of the distinct computations for the sets $S_+ = \{\ell_i \mid e_i > 0\}$ and $S_- = \{\ell_i \mid e_i < 0\}$ corresponding to the private key $(e_1, \ldots, e_n)$. Indeed, when plotting the bitlengths of the factors in the respective first loops in such cases, this leads to two plots similar to Figure 3.1, only at a lower scale.

3.4 Isogeny computations

The algorithm of [35] uses isogeny formulas for Montgomery curves by Costello and Hisil [45] and Renes [131]. We will treat point evaluations and computations of coefficients of image curves separately. First, we present the isogeny formulas of [45], which can be used for the computation of isogenies in CSIDH.

Let K be a point of order $\ell = 2d + 1$ on a Montgomery curve $E : by^2 = x^3 + ax^2 + x$. Then we can compute the coordinate map of the unique (up to compositions by isomorphisms) ℓ-isogeny $\varphi : E \to E'$ with $\ker \varphi = \langle K \rangle$ by

$$\varphi : (x, y) \mapsto (f(x), y \cdot f'(x)),$$

where

$$f(x) = x \cdot \prod_{i=1}^{d} \left(\frac{x \cdot x_{[i]K} - 1}{x - x_{[i]K}} \right)^2,$$

$f'(x)$ is its derivative, and $x_{[i]K}$ denotes the x-coordinate of $[i]K$. The curve parameters a' and b' of E' can be computed by $a' = (6\tilde{\sigma} - 6\sigma + a) \cdot \pi^2$ and $b' = b \cdot \pi^2$, where we define

$$\sigma = \sum_{i=1}^{d} x_{[i]K}, \quad \tilde{\sigma} = \sum_{i=1}^{d} \frac{1}{x_{[i]K}}, \quad \text{and} \quad \pi = \prod_{i=1}^{d} x_{[i]K}.$$

Note that these formulas make use of the fact that $[i]K = -[\ell - i]K$ and thus $x_{[i]K} = x_{[\ell-i]K}$ for all $k \in \{1, \ldots, (\ell - 1)/2\}$.

3.4.1 Point evaluations

Since we work with XZ-only projective Montgomery coordinates, we have to represent $f(x)$ projectively. This is done in [45] by writing $(X_i : Z_i) = (x_{[i]K} : 1)$ for $i = 1, \ldots, d$, $(X : Z) = (x_P : 1)$ for the point P, at which the isogeny should be evaluated, and $\varphi(P) = (X' : Z')$ for the resulting point. Then

$$X' = X \cdot \left(\prod_{i=1}^{d} (X \cdot X_i - Z_i \cdot Z) \right)^2, \text{ and}$$
$$Z' = Z \cdot \left(\prod_{i=1}^{d} (X \cdot Z_i - X_i \cdot Z) \right)^2.$$

The implementation of [35] makes direct use of these formulas by going through the $(X_i : Z_i)$ for $i = 1, \ldots, d$ and computing the pairs $(X \cdot X_i - Z_i \cdot Z)$ and $(X \cdot Z_i - X_i \cdot Z)$ at a cost of $4\mathbf{M} + 2\mathbf{a}$ per step. However, we can also use the observation by Costello and Hisil in [45] to reduce the cost to $2\mathbf{M} + 4\mathbf{a}$ per step by

$$X' = X \cdot \left(\prod_{i=1}^{d} \Big[(X - Z)(X_i + Z_i) + (X + Z)(X_i - Z_i) \Big] \right)^2, \text{ and}$$
$$Z' = Z \cdot \left(\prod_{i=1}^{d} \Big[(X - Z)(X_i + Z_i) - (X + Z)(X_i - Z_i) \Big] \right)^2,$$

assuming that $X + Z$ and $X - Z$ are precomputed, and hence save $d \cdot (2\mathbf{M} - 2\mathbf{a})$ per isogeny evaluation.

3.4.2 Computing the codomain curve

An efficient computation of the codomain curve parameters is not as straightforward as for the point evaluations. This is due to the fact that the required parameters σ and $\tilde{\sigma}$ consist of sums of fractions. Therefore, Costello and Hisil give two different approaches to compute the isogenous curve [45].

The first approach uses the fact that the projective parameters $(a' : 1) = (A' : C')$ of the isogenous curve E' can be recovered from the knowledge of the three 2-torsion points of E'. Therefore, it is possible to recover the required curve parameters of E' by computing the 2-torsion points of E and pushing one of these points through the odd-degree isogeny, which preserves its order on the image curve. However, in contrast to SIDH, we only work over the field $\mathbb{F}_p$ instead of $\mathbb{F}_{p^2}$, while the required points of order 2 are not defined over $\mathbb{F}_p$ in the CSIDH setting.

Their second approach uses the fact that the curve parameters can be recovered from the knowledge of the x-coordinates of two points on the curve and their difference. While these points are typically available in SIDH during the key generation phase, this is not the case for CSIDH, where we only want to compute the isogenous curve and evaluate one point.

In [35], Castryck et al. compute the image curve by defining $c_j \in \mathbb{F}_p$ such that

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

as polynomials in w. Then they observe that

$$(A' : C') = (\hat{\pi}(a - 3\hat{\sigma}) : 1) = (ac_0c_{\ell-1} - 3(c_0c_{\ell-2} - c_1c_{\ell-1}) : c_{\ell-1}^2),$$

following the formulas and notation from Renes [131], where

$$\hat{\pi} = \prod_{i=0}^{\ell-1} x_{[i]K}, \quad \text{and} \quad \hat{\sigma} = \sum_{i=0}^{\ell-1} \left(x_{[i]K} - \frac{1}{x_{[i]K}} \right).$$

In their implementation, this is computed iteratively, going through the $(X_i : Z_i)$ for $i = 2, \ldots, d$, updating the required values at a cost of $6\mathbf{M} + 2\mathbf{a}$ per step. The final computations then take further $8\mathbf{M} + 3\mathbf{S} + 6\mathbf{a}$ to compute the curve parameters $(A' : C')$.

Using twisted Edwards curves for the codomain curve computation. Our approach to speed up this computation exploits the well-known correspondence between Montgomery and twisted Edwards curves. Given a Montgomery curve $E_{A,B} : Bv^2 = u^3 + Au^2 + u$, we can switch to a birationally equivalent twisted Edwards curve $E_{a,d} : ax^2 + y^2 = 1 + dx^2y^2$, where

$$A = \frac{2(a+d)}{a-d} \text{ and } B = \frac{4}{a-d},$$

via the coordinate map

$$(u, v) \mapsto (x, y) = \left(\frac{u}{v}, \frac{u-1}{u+1} \right),$$

and back via its inverse

$$(x, y) \mapsto (u, v) = \left(\frac{1+y}{1-y}, \frac{1+y}{(1-y)x} \right).$$

In [115] it is shown how to switch to and from twisted Edwards curves in the SIDH setting, which also applies to CSIDH, where Montgomery XZ-only coordinates and projective curve parameters $(A : C)$ are used, ignoring the Montgomery parameter b. Following this and [34], a Montgomery point $(X^M : Z^M)$ can be transformed to the corresponding Edwards YZ-coordinates $(Y^E : Z^E)$ by the map

$$(X^M : Z^M) \mapsto (Y^E : Z^E) = (X^M - Z^M : X^M + Z^M),$$

and the Montgomery parameters $(A : C)$ to the corresponding twisted Edwards parameters (a_E, d_E) by

$$a_E = A + 2C \text{ and } d_E = A - 2C.$$

Since this allows us to switch efficiently between equivalent Montgomery and twisted Edwards curves in CSIDH at a cost of 3**a** for the curve parameters and 2**a** for point coordinates, we may as well use isogeny formulas for twisted Edwards curves. Therefore, we state the twisted Edwards isogeny formulas given by Moody and Shumow in [118].

Let K be a point of order $\ell = 2d + 1$ on a twisted Edwards curve $E : a_E x^2 + y^2 = 1 + d_E x^2 y^2$. Then we can compute the coordinate map of the unique (up to compositions by isomorphisms) ℓ-isogeny $\varphi : E \to E'$ with $\ker(\varphi) = \langle K \rangle$ by

$$\varphi(P) = \left(\prod_{Q \in \langle K \rangle} \frac{x_{P+Q}}{y_Q}, \prod_{Q \in \langle K \rangle} \frac{y_{P+Q}}{y_Q} \right).$$

The curve E' is defined by the parameters

$$a'_E = a_E^\ell \quad \text{and} \quad d'_E = \pi_y^8 d_E^\ell, \quad \text{where} \quad \pi_y = \prod_{i=1}^{d} y_{[i]K}.$$

Since the coordinate map is not as simple to compute as for Montgomery curves, we are only interested in the computation of the image curve parameters. Writing $(Y_i^E : Z_i^E)$ for the projective coordinates of $[i]K$ for $i = 1, \ldots, d$, we can transform the formulas from above to the projective case by

$$a'_E = a_E^\ell \cdot \pi_Z^8 \quad \text{and} \quad d'_E = d_E^\ell \cdot \pi_Y^8,$$

where

$$\pi_Y = \prod_{i=1}^{d} Y_i^E \quad \text{and} \quad \pi_Z = \prod_{i=1}^{d} Z_i^E.$$

We can therefore use these formulas to compute the curve parameters of the image curves in CSIDH by switching to twisted Edwards coordinates and points, and switch back to the corresponding Montgomery curve after the isogeny computations by

$$(A' : C') = (2(a'_E + d'_E) : a'_E - d'_E),$$

again at a cost of 3**a**.

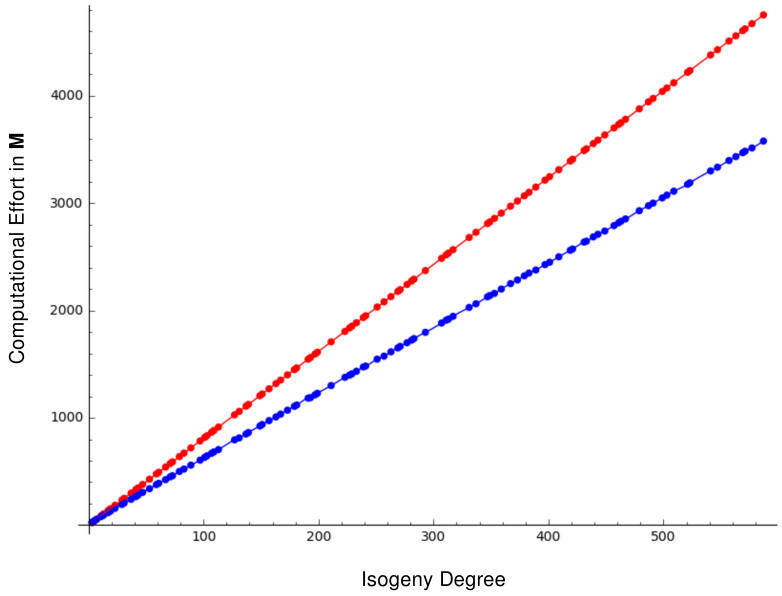


Figure 3.2: Costs of different prime-degree isogeny computations. The red line uses Montgomery point evaluations from [45] and codomain curve computations from [35], while the blue line uses the same Montgomery point evaluations and twisted Edwards codomain curve computations.

Note that the parameters a'_E and d'_E can be computed efficiently: While going through the $(X_i : Z_i)$ on the Montgomery curve for $i = 1, \ldots, d$, we can compute the corresponding Edwards coordinates $(Y_i^E : Z_i^E)$ at a cost of $2\mathbf{a}$. However, the required sums and differences naturally occur at the point evaluation part, and hence do not introduce any computational overhead. We can then compute π_Y and π_Z iteratively by $1\mathbf{M}$ each per step. Compared to the algorithm of [35], this saves $4\mathbf{M} + 2\mathbf{a}$ per step. Furthermore, we have to compute π_Y^8 and π_Z^8 by three squarings each, and a_E^ℓ and d_E^ℓ, which can be done efficiently, e.g. via a square-and-multiply approach. We further note that the latter computation does not require any values generated during the loop through the $(X_i : Z_i)$. This means that especially hardware architectures that allow for parallel computations would benefit from this, since the computation of a_E^ℓ and d_E^ℓ can be done in parallel to the loop through the $(X_i : Z_i)$.

Figure 3.2 compares the costs of a combined image curve computation and point evaluation for different prime degrees, where the red line arises from using the Montgomery isogenies from [35], including the optimizations from [45], and the blue line from using our approach utilizing twisted Edwards curves to compute the isogenous curve. The cost is measured in field multiplications, assuming that $1\mathbf{S} = 0.8\mathbf{M}$ and $20\mathbf{a} = 1\mathbf{M}$. In this case, we can derive a reduction of the costs by approximately 25% for the largest primes in the CSIDH-512 parameter set. We note that the speedup with different ratios of field

operations does not significantly differ from this, since the main difference between the approaches lies in the number of multiplications. To obtain the cost of the computations of a_E^ℓ and d_E^ℓ, we used a square-and-multiply approach. Since the exponents ℓ_i are small fixed numbers, it is also possible to precompute the optimal addition chains, and therefore save some computational effort compared to square-and-multiply. However, we found that even for the largest ℓ_i from the current parameter set, this saves at most four multiplications. Hence, the benefit of this is rather small compared to the increased length of the code and the more complicated implementation.

Note that for $\ell \leq 5$, our approach is slightly more expensive than the Montgomery approach. Therefore, in this case the Montgomery approach can be used. However, the benefit of this is very small compared to the total computational effort. Thus, for the sake of simplicity, it might be better to use our approach for all involved ℓ_i.

It is further noted in [35] that for a fixed prime ℓ one could reduce the computational effort by finding an appropriate representative of the isogeny modulo (a factor of) the ℓ-division polynomial ψ_ℓ, as done in [47] for 3- and 4-isogenies. However, every required isogeny degree would have to be implemented separately, resulting in major complications for implementations.

3.5 Implementation results

As proof-of-concept and for measuring the efficiency of our work, we took the mentioned implementation of Castryck et al. [35] as reference, and added our optimizations. The implementation is written in C and uses $\mathbb{F}_p$-arithmetic in assembly. The parameters in use are the CSIDH-512 parameters described in Section 3.2. The validation of public keys is not included in the following discussion and results.

The first optimization is the precomputation of the curve parameters $(A + 2C : 4C)$ each time before entering the Montgomery ladder, as also done in SIDH [47]. This only saves a few additions per ladder step, which however overall add up to a significant amount of field additions.

The other optimizations are as described above; one results from rearranging the factors in the class group action evaluation algorithm, and the other from more efficient isogeny computations by using the point evaluation from [45] and our twisted Edwards approach for the codomain curve computations.

Table 3.1 lists the influence of the different optimizations on the overall performance. In the respective implementations, only the mentioned optimization was used, while leaving other parts of the reference implementation from [35] unchanged. For the last entry, we combined all the described optimizations and achieved a reduction of the total computational effort by 25%, yielding a speed-up factor of 1.33. The latter implementation is available at `https://github.com/michael-meyer/phdthesis-code`.

Table 3.1: Performance comparison of the class group action evaluation in CSIDH with different optimizations applied. All timings are given in 10^6 clock cycles and were measured on an Intel Core i7-6500 Skylake processor, averaged over 10 000 runs.

	Clock cycles $\times 10^6$	Speedup factor
Castryck et al. [35]	138.6	-
Rearranging factors	126.5	1.096
Isogeny optimization	118.2	1.173
Combination of all optimizations	103.9	1.334

3.6 Conclusion and current state-of-the-art

In this chapter we presented ways to speed up the class group action evaluation used in CSIDH. However, for practical applications an efficient constant-time implementation is required in order to avoid leakage of information on the private key from the runtime of CSIDH. Such an implementation is presented in Chapter 4. Note that all the algorithmic improvements from this chapter also apply to this constant-time implementation.

Subsequent to the publication of the material presented in this chapter in [114], more work on efficient implementations of CSIDH's main operations has been published. We briefly review the current state-of-the art.

For the scalar multiplications in CSIDH, Cervantes-Vázquez, Chenu, Chi-Domínguez, De Feo, Rodríguez-Henríquez, and Smith observe that instead of using the Montgomery ladder, one can use optimal differential addition chains [37]. This is due to the fact that only scalar multiplications by the involved ℓ_i and 4 take place, which means that differential addition chains can be precomputed for each ℓ_i, resulting in an average speedup of 25%. Note that this improvement is compatible with the multiplication speedup obtained in this chapter.

In [37] it is further proposed to entirely work with twisted Edwards curves. Recall from Section 2.3 that we can define YZ-only twisted Edwards differential addition and doubling formulas that arise from transforming points to the Montgomery model, apply the efficient Montgomery operations, and map the results back to the twisted Edwards model. The same can be done with the Montgomery isogeny point evaluation formulas from [45], which are used in this chapter. Remarkably, this approach is slightly faster, saving $(4d-4)\mathbf{a}$ for an isogeny of degree $\ell = 2d+1$ when working with twisted Edwards coordinates. Other publications suggest that different approaches to construct CSIDH entirely with Edwards curves does not lead to significant speedups [119, 101].

A more significant speedup for large-degree isogeny computations has been achieved by Bernstein, De Feo, Leroux, and Smith [13]. For a prime degree ℓ and kernel gener-

ator K, the isogeny codomain curve and point evaluation formulas used in this chapter can be rewritten in terms of the kernel polynomial $h_S(x) = \prod_{s \in S}(x - x_{[s]K})$ with $S = \{1, 3, 5, \ldots, \ell - 2\}$. Thus, computing isogenies efficiently hinges on the ability of evaluating this polynomial efficiently. Inspired by integer factoring algorithms due to Strassen [148] and Pollard [127], Bernstein et al. make use of a baby-step giant-step approach that reduces from the linear complexity of a naïve computation to an asymptotic complexity of $\widetilde{O}(\sqrt{\ell})$. Further, in practice their approach outperforms the usual isogeny formulas already at isogeny degrees in the range of 100 [13, 2].

Isogenies of degree 2 are not available in the CSIDH setting, however Castryck and Decru [32] show that when using a prime with $p \equiv 7 \bmod 8$ and moving to the surface of the isogeny graph, horizontal 2-isogenies are available, and can be computed deterministically, i.e., without sampling points of order 2. Similarly, in [33], Castryck, Decru, and Vercauteren show how to compute isogenies of odd prime degree deterministically, obtaining a speedup for degrees $\ell \leq 13$.

Chapter 4

An efficient constant-time implementation of CSIDH

4.1 Introduction

As mentioned in Chapter 3, constant-time algorithms are required for most real-world applications of cryptosystems, since otherwise the runtime may leak information on the respectively utilized private key. In this chapter we present the first practical constant-time implementation of CSIDH, including many optimizations that reduce the computational effort. Our implementation induces a relatively small slowdown by a factor of 3.03 compared to the average runtime of the fastest variable-time implementation from [114]. On the other hand, the worst-case runtime of the variable-time implementation from [114] is slightly slower than our constant-time implementation. As in Chapter 3 we restrict to an implementation of CSIDH-512, but note that the same ideas can be applied to different parameter sets.

Note that there are two different notions of constant-time implementations, as explained in [17]. In our case, it suffices to work with the notion stating that the running time does not depend on the choice of the private key, but may vary due to randomness. The second notion specifies strict constant time, meaning that the running time must be the same for every execution of the algorithm, independent of private keys or randomness. Throughout this thesis, "constant time" refers to the first notion described above.

Related work. In [17], Bernstein, Lange, Martindale, and Panny describe constant-time implementations in the second notion from above, which is required for running the quantum oracle in a quantum attack. In this chapter, we follow the mentioned different approach for an efficient constant-time implementation, but reuse some of the techniques from [17].

Organization. The rest of this chapter is organized as follows. The following section briefly recalls the CSIDH class group action algorithm and presents leakage scenarios based on time, power analysis, and cache timing. In Section 4.3 we suggest different methods to avoid these leakages and propose a constant-time implementation. Section 4.4 contains a straightforward application of our suggested methods, and various optimizations. Thereafter, we provide implementation results in Section 4.5. Section 4.6 concludes this chapter and presents the current state-of-the-art from follow-up related work that was published subsequent to the publication of this chapter's material in [113]. Details on parameters and the full algorithms of our approach are given in Section 4.A and Section 4.B, respectively.

4.2 Leakage scenarios

Recall from Chapter 3 that we choose a prime of the form $p = 4 \cdot \ell_1 \cdot \cdots \cdot \ell_n - 1$, where the ℓ_i are small distinct odd primes, and a supersingular starting curve $E_0(\mathbb{F}_p)$. A private key consists of a tuple $(e_1, \ldots, e_n)$, where the integers e_i are sampled from an interval $[-B, B]$. The absolute value $|e_i|$ specifies how many ℓ_i-isogenies have to be computed, and the sign of e_i determines whether points on the current curve or on its twist have to be used as kernel generators. For the full CSIDH class group action evaluation, we refer to Algorithm 1.

It is obvious and already noted in [35] that the variable-time CSIDH implementations of [35, 114] are not side-channel resistant. In this chapter we focus on three scenarios that can leak information on the private key. Note that the second scenario features a stronger attacker than the first. Further, there will naturally be many more scenarios for side-channel attacks.

Timing leakage. Since the private key in CSIDH specifies how many isogenies of each degree have to be computed, it is obvious that this (up to additional effort for point multiplications due to the random sampling of points) determines the running time of the algorithm. As stated in [114], the worst-case running time occurs for the private key $(5, 5, \ldots, 5)$, and is more than 3 times slower than in the average case. The other extreme is the private key $(0, 0, \ldots, 0)$, which would require no computations at all. However, in an implementation protected against timing attacks, the running time should be independent of the private key.

Power analysis. Instead of focusing on the running time, we now assume that an attacker can measure the power consumption of an execution of the algorithm. We further assume that from the measurement, the attacker can determine blocks which represent the two main primitives in CSIDH, namely point multiplications and isogeny computations, and can distinguish these from each other. Now assume that the attacker can distinguish the loop iterations in line 1 of Algorithm 1. Then the attacker can determine which private

key elements share the same sign from the isogeny blocks that are performed in the same loop, since they have variable running time based on the isogeny degree. This significantly reduces the possible key space and therefore also the complexity of revealing the correct key.

Cache timing attacks. In general, data flow from the secret key to branch conditions and array indices must be avoided in order to achieve protection against cache timing attacks [14]. Our implementation follows these guidelines to avoid vulnerabilities against the respective possible attacks.

4.3 Mitigating leakage

In this section we present ideas on how to fix these possible leakages in an implementation of CSIDH. We outline the most important ideas here, and give details on how to implement them efficiently in CSIDH-512 in Section 4.4.

Dummy isogenies. It seems obvious that one should compute a constant number of isogenies of each degree ℓ, and discard the results of those not required by the private key, in order to obtain a running time independent of the private key. However, if we compute a dummy isogeny, i.e., compute an ℓ-isogeny and discard the results, an additional scalar multiplication $[\ell]P$ is required, since P only loses the factor ℓ from its order if it gets pushed through an isogeny. Thus, if utilized isogenies and dummy isogenies are computed in the same loop,[7] each dummy isogeny must be followed by such a scalar multiplication. In order to avoid leakage from this, we thus have to perform a scalar multiplication by ℓ after each ℓ-isogeny, which induces some computational overhead.

However, we can design a dummy isogeny algorithm that allows for avoiding such an additional multiplication. In particular, the ℓ-isogeny algorithm must update the curve parameters and evaluate the curve point P in case of a real isogeny, and compute $[\ell]P$ in the dummy case. For this, a unified algorithm that computes both cases with the same number and sequence of operations is required to achieve side-channel resistance. This can be done as follows.

Since the isogeny algorithm computes all scalar multiples of the kernel generator K up to $[\frac{\ell-1}{2}]K$, one can initially replace K by P, and perform two more differential additions to compute $[\ell]P$, while the curve parameters remain unchanged. In consequence, such a dummy isogeny implicitly simply performs a scalar multiplication by ℓ. Therefore, the order of the output point $[\ell]P$ is not divisible by ℓ then, which is important for using this point to compute correct kernel generators in following iterations. Note that since we require a unified isogeny algorithm, the two extra differential additions also have to be

[7] This is required, since otherwise, an attacker in the second leakage scenario can determine the private key easily.

performed for utilized isogenies. Thus, this introduces a minor computational overhead compared to the isogenies from [114], but in general this is cheaper than an additional scalar multiplication by ℓ.

The only ingredient we need to ensure the constant-time property is a side-channel resistant way to replace a point K by P. As usual, this is realized in practice via a constant-time swap function, which swaps two inputs in constant time if a decision bit is set, or leaves them unchanged if the decision bit is not set. Note that also the determination of the required decision bit, which indicates if a dummy or real isogeny has to be computed, is performed in constant time in our implementation. We refer to Chapter 5 for more details on this.

Balanced vs. unbalanced private keys. Using dummy isogenies to spend a fixed time on isogeny computations is not enough for a constant-time implementation, however. Another problem lies in the point multiplications in line 7 and 9 of Algorithm 1. We illustrate the potential timing leakage through an example. When considering the private keys $(5, 5, 5, \dots)$ and $(5, -5, 5, -5, \dots)$, we observe that for the former key the running time is 50% higher than for the latter key, even though the number of isogenies is equal in both computations. The reason for this is that in the first case, in order to compute one isogeny of each degree, the multiplication in line 7 is only a multiplication by 4, and the multiplication in line 9 has a factor of bitlength 509 in the first iteration, 500 in the second iteration, etc.

For the second key, following Algorithm 1, we have to perform one loop through the odd i and one through the even i in order to compute one isogeny of each degree ℓ_i. Therefore, the multiplications in line 7 have 254-bit resp. 259-bit factors, while the bitlengths of the factors in the multiplications in line 9 are $252, 244, \dots$, resp. $257, 248, \dots$ (see Figure 4.1). In total, adding up the bitlengths of all factors, we can measure the cost of all required point multiplications for the computation of one isogeny per degree, since one Montgomery ladder step is performed per bit. For simplicity, we assume that the condition in line 10 of Algorithm 1 never fails. For the former key, we end up with 16813 bits, while for the latter key we only have 9066 bits.

This can be generalized to any private key; the more unbalanced the key elements (or the products of the respective ℓ_i) are, i.e., many of them share the same sign, the more the computational effort grows, compared to the perfectly balanced case from above. This behavior depends on the private key and can therefore leak information. Hence, it is clear that we have to prevent this in order to achieve a constant-time implementation.

One way to achieve this is to use constant-time Montgomery ladders that always run to the maximum bitlength, no matter how large the respective factor is. However, this would lead to a massive increase in running time. Another possibility for handling this is to only choose key elements of a fixed sign. Then we have to adjust the interval from which we sample the integer key elements, e.g. from $[-5, 5]$ to $[0, 10]$ in CSIDH-512, in order to reach

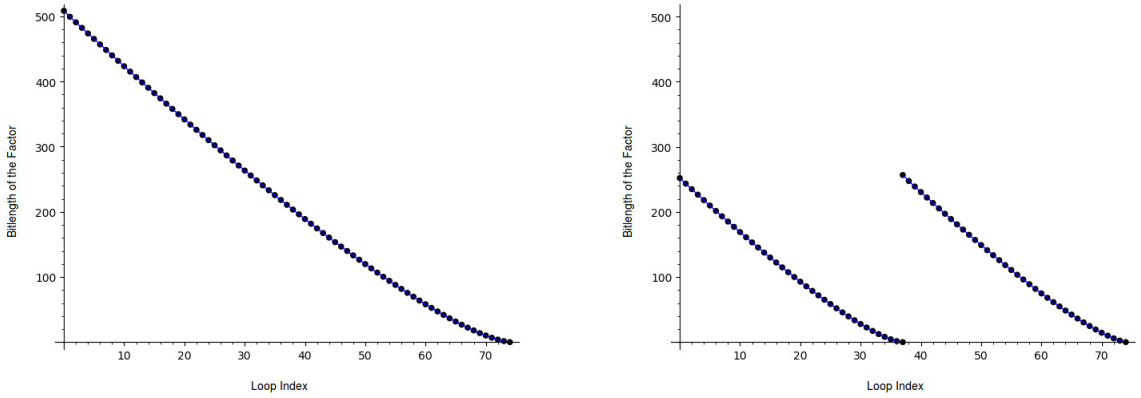


Figure 4.1: Bitlengths of the factors of the involved scalar multiplications for computing one isogeny per degree for the keys $(5, 5, \ldots, 5)$ (left) and $(5, -5, 5, -5, \ldots)$ (right).

an equally large keyspace. This however doubles the computational effort for isogenies (combined normal and dummy isogenies). However, this will be the countermeasure of our choice, since also in a different leakage scenario, information on the sign distribution may leak.

Determining the sign distribution. In our second leakage scenario, featuring a power analysis attack, an attacker might determine the sign distribution of the key elements by identifying blocks of isogeny resp. dummy isogeny computations that are performed in the same loop of Algorithm 1. The sign distribution then trivially leaks, since the running time of an isogeny depends on its degree, and thus if some ℓ_i-isogenies are located in the same loop, the respective key elements e_i share the same sign. One way to mitigate this attack is to let each degree ℓ_i isogeny run as long as an ℓ_{max}-isogeny, where ℓ_{max} is the largest involved isogeny degree. As utilized in [17], this is possible because of the Matryoshka-doll structure of the isogeny algorithms. This would allow an attacker in the second leakage scenario to only determine the number of positive resp. negative elements, but not their exact distribution, at the cost of a large increase of computational effort. We can also again restrict to the case that we only choose non-negative (resp. only non-positive) key elements. Then there is no risk of leaking information about the sign distribution of the elements, since in this setting the attacker knows this beforehand, at the cost of twice as many isogeny computations.

Limitation to non-negative key elements. Since this choice eliminates both of the aforementioned possible leakages, we use the mentioned different interval to sample private key elements from. In CSIDH-512, this means using the interval $[0, 10]$ instead of $[-5, 5]$. As before, there are 11^{74} different key vectors to choose from. Castryck et al. argue in [35] that usually there are multiple vectors $(e_1, e_2, \ldots, e_n)$, which represent the same ideal

class, meaning that the respective keys are equivalent. However, they assume by heuristic arguments that the number of short representations per ideal class is small, i.e., the 11^{74} different keys $(e_1, e_2, \ldots, e_n)$, where all e_i are sampled from the interval $[-5, 5]$, represent not much less than 11^{74} ideal classes. If we now have two equivalent keys $e \neq f$ sampled from $[-5, 5]$, then we have a collision for our shifted interval as well, since shifting all elements of e and f by $+5$ results in equivalent keys $e' \neq f'$ with elements in $[0, 10]$, and vice versa. Therefore, the keyspace of our shifted version is equivalent to the one in CSIDH-512 as defined in [35].[8]

In the following sections we focus on optimized implementations, using the mentioned countermeasures against attacks, i.e., sampling key elements from the interval $[0, 10]$ and using dummy isogenies.

4.4 Efficient implementation

4.4.1 Straightforward implementation

Firstly, we describe the straightforward implementation of the evaluation of the class group action in CSIDH-512 with the choices from above, before applying various optimizations. We briefly go through the implementation aspects of the main primitives, i.e., point multiplications, isogenies and dummy isogenies, and explain why this algorithm runs in constant time, and does not leak information about the private key. Algorithm 2 presents the straightforward implementation of our constant-time approach.

Parameters. As described in [35], we have a prime number $p = 4 \cdot \ell_1 \cdot \ell_2 \cdot \cdots \cdot \ell_n - 1$, where the ℓ_i are small distinct odd primes. We further assume that we have $\ell_1 > \ell_2 > \cdots > \ell_n$. In CSIDH-512 we have $n = 74$, and we sample the elements of private keys $(e_1, e_2, \ldots, e_n)$ from $[0, 10]$.

Handling the private key. Similar to the original implementation of Castryck et al., we copy the elements of the private key in an array $e = (e_1, e_2, \ldots, e_n)$, where e_i determines how many isogenies of degree ℓ_i we have to compute. Furthermore, we set up another array $f = (10 - e_1, 10 - e_2, \ldots, 10 - e_n)$, to determine how many dummy isogenies of each degree we have to compute. As we go through the algorithm, we compute all the required isogenies and dummy isogenies, reducing e_i resp. f_i by 1 after each degree ℓ_i isogeny resp. dummy isogeny. We therefore end up with a total of 10 isogeny computations (counting isogenies and dummy isogenies) for each ℓ_i.

[8] One could also think of using the starting curve E', which is the result of applying the key $(5, 5, \ldots, 5)$ to the curve E_0. Then for a class group action evaluation, using key elements from $[-5, 5]$ and the starting curve E' is equivalent to using key elements from $[0, 10]$ and the starting curve E_0.

Algorithm 2: Constant-time evaluation of the class group action in CSIDH-512.

Input : $a \in \mathbb{F}_p$ such that $E_a : y^2 = x^3 + ax^2 + x$ is supersingular, and a list of integers $(e_1, \ldots, e_n)$ with $e_i \in \{0, 1, \ldots, 10\}$ for all $i \leq n$.

Output: $a' \in \mathbb{F}_p$, the curve parameter of the resulting curve $E_{a'}$.

1 Initialize $k = 4$, $e = (e_1, \ldots, e_n)$ and $f = (f_1, \ldots, f_n)$, where $f_i = 10 - e_i$.
2 **while** some $e_i \neq 0$ or $f_i \neq 0$ **do**
3 Sample random values $x \in \mathbb{F}_p$ until we have some x where $x^3 + ax^2 + x$ is a square in $\mathbb{F}_p$.
4 Set $P = (x : 1)$, $P \leftarrow [k]P$, $S = \{i \mid e_i \neq 0 \text{ or } f_i \neq 0\}$.
5 **foreach** $i \in S$ **do**
6 Let $m = \prod_{j \in S, j > i} \ell_j$.
7 Set $K \leftarrow [m]P$.
8 **if** $K \neq \infty$ **then**
9 **if** $e_i \neq 0$ **then**
10 Compute a degree ℓ_i isogeny $\varphi : E_a \to E_{a'}$ with $\ker(\varphi) = \langle K \rangle$:
11 $a \leftarrow a'$, $P \leftarrow \varphi(P)$, $e_i \leftarrow e_i - 1$.
12 **else**
13 Compute a degree ℓ_i dummy isogeny:
14 $a \leftarrow a$, $P \leftarrow [\ell_i]P$, $f_i \leftarrow f_i - 1$.
15 **if** $e_i = 0$ *and* $f_i = 0$ **then**
16 Set $k \leftarrow k \cdot \ell_i$.

Sampling random points. In line 3 of Algorithm 2, we have to find curve points on the current curve that are contained in $\ker(\pi - 1)$. As in [35] this can be done by sampling a random $x \in \mathbb{F}_p$, and computing y^2 by the curve equation $y^2 = x^3 + ax^2 + x$. We then check if y is defined over $\mathbb{F}_p$ by a Legendre symbol computation, i.e., by checking if $(y^2)^{(p-1)/2} \equiv 1 \pmod{p}$. If this is not the case, we simply repeat this procedure until we find a suitable point. Note that we require the curve parameter a to be in affine form. Since a will typically be in projective form after isogeny computations, we therefore have to compute the affine parameter each time before sampling a new point.

Elliptic curve point multiplications. Since we work with Montgomery curves, using only projective XZ-coordinates and projective curve parameters $a = A/C$, we can use the standard Montgomery ladder as introduced in [117], adapted to projective curve parameters as in [47]. This means that per bit of the factor, one combined doubling and differential addition is performed.

Isogenies. For the computation of isogenies, we use the formulas presented in [114]. They combine the Montgomery isogeny formulas by Costello and Hisil [45], and Renes [131] with the twisted Edwards formulas by Moody and Shumow [118], in order to obtain an efficient algorithm for the isogeny computations in CSIDH. For an ℓ_i-isogeny, this requires a point K of order ℓ_i as kernel generator, and the projective parameters A and C of the current curve. It outputs the image curve parameters A' and C', and the evaluation of the point P. As mentioned before, the algorithm computes all multiples of the point K up to the factor $\frac{\ell_i-1}{2}$. See e.g. [17] for more details.

Dummy isogenies. As described before, we want the degree ℓ_i dummy isogenies to output the scalar multiple $[\ell_i]P$ instead of an isogeny evaluation of P. Therefore, we interchange the points K and P in the original isogeny algorithm, such that it computes $[\frac{\ell_i-1}{2}]P$. We then perform two more differential additions, i.e., compute $[\frac{\ell_i+1}{2}]P$ from $[\frac{\ell_i-1}{2}]P$, P, and $[\frac{\ell_i-3}{2}]P$, and compute $[\ell_i]P$ from $[\frac{\ell_i+1}{2}]P$, $[\frac{\ell_i-1}{2}]P$, and P.

As mentioned before, we want isogenies and dummy isogenies of degree ℓ_i to share the same code in order to avoid conditional branches. Hence, the two extra differential additions are also performed in the isogeny algorithm, without using the results. In our implementation, a conditional point swap based on a decision bit ensures that the correct input point is chosen. This avoids conditional branches that depend on the private key in line 9 of Algorithm 2.

If one is concerned that a side-channel attacker can detect that the curve parameters A and C are not changed for some time (meaning that a series of dummy isogenies is performed), one could further rerandomize the projective representation of the curve parameter A/C by multiplying A and C by the same random number[9] $1 < \alpha < p$.

4.4.2 Constant-time property

We now explain why this algorithm runs in constant time. As already explained, we perform 10 isogeny computations (counting isogenies and dummy isogenies) for each degree ℓ_i. Furthermore, for a given degree ℓ_i, isogenies and dummy isogenies have the same running time, since they share the same code, and conditional branches are avoided. Therefore the total computational effort for isogenies is constant, independent of the respective private key. We also set the same condition (line 8 of Algorithm 2) for the kernel generator for the computation of a dummy isogeny, in order not to leak information.

Sampling random points and finding a suitable one does not run in constant time in Algorithm 2. However, the running time only depends on randomly chosen values, and does not leak any information on the private key.

Now for simplicity assume that we always find a point of full order, i.e., a point that can be used to compute one isogeny of each degree ℓ_i. Then it is easy to see that the total

[9] One could actually use an intermediate value $\alpha \in \mathbb{F}_p \setminus \{0, 1\}$ of the isogeny computation, since the factor is not required to be truly random.

computational effort for scalar multiplications in Algorithm 2 is constant, independent of the respective private key. If we now allow random points, we will typically not satisfy the condition in line 8 of Algorithm 2 for all i. Therefore, additional computations (sampling random points, and point multiplications) are required. However, this does not leak information about the private key, since this only depends on the random choice of curve points, but not on the private key.

Hence, we conclude that the implementation of Algorithm 2 as described here prevents the leakage scenarios considered in Section 4.2. It is however rather slow compared to the performance of variable-time CSIDH-512 in [114, 35]. In the following section we focus on ways to optimize and speed up the implementation.

4.4.3 Optimizations

Sampling points with Elligator. In [17], Bernstein, Lange, Martindale, and Panny pointed out that Elligator [15], specifically the Elligator 2 map, can be used in CSIDH in order to sample points over the required field of definition. Since we only need points defined over $\mathbb{F}_p$, this is especially advantageous in our situation. For $a \neq 0$ the Elligator 2 map works as follows (see [17]):

- Sample a random $u \in \{2, 3, \ldots, (p-1)/2\}$.
- Compute $v = a/(u^2 - 1)$.
- Compute e, the Legendre symbol of $v^3 + av^2 + v$.
- If $e = 1$, output v. Otherwise, output $-v - a$.

Therefore, for all $a \neq 0$ we can replace the search for a suitable point in line 3 of Algorithm 2 at the cost of an extra inversion. However, as explained by Bernstein et al., one can precompute $1/(u^2 - 1)$ for some values of u, e.g. for $u \in \{2, 3, 4, \ldots\}$. Then the cost is essentially the same as for the random choice of points, but we always find a suitable point in this way, compared to the probability of $1/2$ when sampling random points. However, Cervantes-Vázquez, Chenu, Chi-Domínguez, De Feo, Rodríguez-Henríquez, and Smith [37] pointed out that applying the method from [17] may leak information in our case if not applied correctly, i.e., trying for *all* precomputed values u to result in a point of desired order.[10] Thus, they describe a projective version of the Elligator 2 map, which is equally efficient, and mitigates this leakage, see [37] for details. Note that the probability for actually finding points of suitable order appears to be almost unchanged when using Elligator instead of random points, as discussed in [17].

For $a = 0$, Bernstein et al. [17] and Cervantes-Vázquez et al. [37] describe similar methods to exploit the Elligator 2 map in the respective scenarios.

[10] Note that the original version of [113], where this chapter's material was published, was vulnerable to this potential leakage, since it departed from [17] in only trying potential values u until the first succeeds in finding a point of desired order.

Table 4.1: Number of Montgomery ladder steps for computing one isogeny of each degree in CSIDH-512 for different numbers of batches m.

m	1	2	3	4	5	6	7
Ladder steps	16813	9066	6821	5959	5640	5602	5721

SIMBA (Splitting Isogenies into Multiple BAtches). In Section 4.3, we analyzed the running time of the variable-time CSIDH-512 algorithm for the keys $e_1 = (5, 5, \dots, 5)$ and $e_2 = (5, -5, 5, -5, \dots)$. For the latter, the running time is significantly faster, because of the smaller multiplications during the loop (line 9 of Algorithm 1), see Figure 4.1. We adapt and generalize this observation here, in order to speed up our constant-time implementation.

Consider for our setting the key $(10, 10, \dots, 10)$ and assume for simplicity that we can again always choose points of full order. In order to split the indices in two sets (exactly as Algorithm 1 does for the key e_2), we define the sets $S_1 = \{1, 3, 5, \dots, 73\}$ and $S_2 = \{2, 4, 6, \dots, 74\}$. Then the loops through the ℓ_i for $i \in S_1$ resp. $i \in S_2$ require significantly smaller multiplications, while only requiring to compute $[4k]P$ with $k = \prod_{i \in S_2} \ell_i$ resp. $k = \prod_{i \in S_1} \ell_i$ beforehand. We now simply perform 10 loops for each set, and hence this approach achieves exactly the same speedup over Algorithm 2, as Algorithm 1 does for the key e_2 compared to e_1, by using two batches of indices instead of only one.

A natural question is whether splitting the indices in two sets already results in the best possible speedup. We generalize the observation from above, now splitting the indices into m batches, where $S_1 = \{1, m+1, 2m+1, \dots\}$, $S_2 = \{2, m+2, 2m+2, \dots\}$, etc.[11] Before starting a loop through the indices $i \in S_j$ with $1 \leq j \leq m$, one now has to compute $[4k]P$ with $k = \prod_{h \notin S_j} \ell_h$. The number and size of these multiplications grows when m grows, so we can expect that the speedup turns into a slowdown when m is too large.

In order to find the best choice for m, we computed the total number of Montgomery ladder steps during the computation of one isogeny of each degree in CSIDH-512 for different m, with the assumptions from above. We did not take into account here that when m grows, we will have to sample more points (which costs at least one Legendre symbol computation each), since this depends on the cost ratio between Montgomery ladder steps and Legendre symbol computations in the respective implementation. Considering this potential overhead, Table 4.1 shows that the optimal choice is expected to be in the range $m \in \{4, 5, 6\}$.

If we now come back to the choice of points through Elligator, our assumption regarding kernel generator rejections does not hold anymore, and with very high probability, we will need more than 10 loops per index set. Typically, after 10 loops through each batch the

[11] Note that in [17] a similar idea is described. However, in their algorithm only two isogeny degrees are covered in each iteration. Our construction makes use of the fact that we restrict to intervals of non-negative numbers for sampling the private key elements.

large degree isogenies will be finished due to their small failure rate of $1/\ell$ for an ℓ-isogeny, while there are some small degree isogenies left to compute. In this case our optimization backfires, since in this construction the indices of the missing ℓ_i will be distributed among the m different batches. We therefore need large multiplications in order to only check a few small degrees per batch. Hence, it is beneficial to define a number $\mu \geq 10$, and merge the batches after μ steps, i.e., simply going back to Algorithm 2 for the computation of the remaining isogenies. We dub this construction SIMBA-m-μ.

Sampling private key elements from different intervals. Instead of sampling all private key elements from the interval $[0, 10]$, and in total computing 10 isogenies of each degree, one can sample the key elements from different intervals for each isogeny degree, as done in [61]. For a private key $e = (e_1, e_2, \ldots, e_n)$, we can choose an interval $[0, B_i]$ for each e_i, in order to e.g. reduce the number of expensive large degree isogenies at the cost of computing more small degree isogenies. We require $\prod_i (B_i + 1) \approx 11^{74}$ in order to obtain the same security level as before. For the security implication of this choice, similar arguments as in Section 4.3 apply.
Trying to find the optimal parameters B_i leads to a large integer optimization problem, which is not likely to be solvable exactly. Therefore, we heuristically searched for parameters that are likely to improve the performance of CSIDH-512. We present them in Section 4.5 and Section 4.A.

Note that if we choose $B = (B_1, \ldots, B_n)$ different from $B = (10, 10, \ldots, 10)$, the benefit of our optimizations above will change accordingly. Therefore, we adapted the parameters m and μ in our implementation according to the respective choice of B.

Skip point evaluations. As described before, the isogeny algorithms compute the image curve parameters, and push a point P through the isogeny. However, in the last isogeny per loop, the latter is unnecessary, since we choose a new point after the isogeny computation anyway. Therefore, skipping the point evaluation part in these cases saves some computational effort.

Application to variable-time CSIDH. Note that many of the optimizations from above are also applicable to variable-time CSIDH-512 implementations as in [114] or [35]. We could therefore also speed up the respective implementation results using the mentioned methods.

4.5 Implementation results

We implemented our optimized constant-time algorithm in C, using the implementation accompanying [114], which is based on the implementation from the original CSIDH paper by Castryck et al. [35]. For example, the implementation of the field arithmetic in

Table 4.2: Performance of one class group action evaluation in CSIDH-512 with the mentioned parameters. All timings were measured on an Intel Core i7-6500 Skylake processor, averaged over 1 000 runs.

Clock cycles $\times 10^6$	Wall clock time
314.5	121.3 ms

assembly is the same as in [35]. Our final algorithm, containing all the optimizations from above, can be found in Section 4.B.

Since we described different optimizations that can influence one another, it is not straightforward to decide which parameters B, m, and μ to use. Therefore, we tested various choices and combinations of parameters B, m, and μ, assuming $\ell_1 > \ell_2 > \cdots > \ell_n$. The parameters and implementation results can be found in Section 4.A. The best parameters we found are given by

$$\begin{aligned} B = [&5,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,8,8,8,8,8,8,11,11,11,11\\ &11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,13,13,13,13,13,13,13,13\\ &13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13] \end{aligned}$$

using SIMBA-5-11, where the key element e_i is sampled from $[0, B_i]$. We do not claim that these are the best parameters; there might be better choices that we did not consider in our experiments.

We further tried to rearrange the order of the primes ℓ_i in the different loops. As pointed out in [114], it is beneficial to go through the ℓ_i in descending order. However, if we suppress isogeny point evaluations in the last iteration per loop, this means that these savings refer to small ℓ_i, and therefore the impact of this is rather small. Hence, we put m large primes at the end of the list of primes ℓ_i, therefore requiring more computational effort for point multiplications, which is however in some situations outweighed by the larger savings from not evaluating points.

In this way, the best combination we found for CSIDH-512 is $\ell_1 = 349, \ell_2 = 347, \ell_3 = 337, \ldots, \ell_{69} = 3, \ell_{70} = 587, \ell_{71} = 373, \ell_{72} = 367, \ell_{73} = 359$, and $\ell_{74} = 353$, using SIMBA-5-11 and B from above, where the B_i are reordered accordingly to the ℓ_i.

In Table 4.2, we give the cycle count and running time for the implementation using the parameters from above.[12] The code is available at `https://github.com/michael-meyer/phdthesis-code`.

To give a comparison that mainly shows the impact of SIMBA and the different choice of B, we also ran the straightforward implementation according to Algorithm 2 with $B =$

[12]Note that this implementation uses the described version of Elligator that is potentially vulnerable to timing leakage. However, the performance of the Elligator usage from [37] roughly equals the performance of the version used in [113]. Thus, the impact on the total running times is negligible.

$[10, 10, \ldots, 10]$, also using Elligator. In this case, we measured 621.5 million clock cycles in the same setting as above.

Compared to the average performance of the variable-time implementation from [114], the results from Table 4.2 mean a slowdown by a factor of 3.03. However, as mentioned, also the variable-time implementation can benefit from the optimizations from this chapter.

We emphasize that although our results depend on the CSIDH-512 parameter set, it is clear that the described optimizations can be easily adapted to other parameter sets and security levels.

4.6 Conclusion and current state-of-the-art

This chapter presented the first practical constant-time implementation of CSIDH with a relatively small overhead by a factor of 3.03 compared to the average running time of the variable-time implementation from [114]. However, despite being efficient, this implementation is orders of magnitude slower than other quantum-resistant schemes. In order to further improve the performance of CSIDH, the following extensions of our ideas and other approaches for constant-time implementations have been published after the publication of this chapter's content in [113].

Onuki, Aikawa, Yamazaki, and Takagi [122] show that negative exponents in a CSIDH constant-time implementation can be allowed in our approach by always keeping an additional point, i.e., always having points $P_+ \in E[\pi - 1]$ and $P_- \in E[\pi + 1]$ available on the current curve E. This allows to choose the correct point to compute the order-ℓ_i kernel generator K via constant-time point swaps prior to computing an ℓ_i-isogeny, such that $K \in E[\pi - 1]$ if the corresponding $e_i > 0$ (resp. $f_i > 0$ for the dummy case), or $K \in E[\pi + 1]$ if $e_i < 0$ (resp. $f_i < 0$ for the dummy case). This means that we always have to push a second point through isogenies, which induces computational overhead compared to our approach. On the other hand, exponents can e.g. be chosen from the intervals $[-5, 5]$ instead of $[0, 10]$, and thus we only have to compute half as many isogenies in total. Overall, [122] reports a speedup of 29%, using SIMBA and other optimizations from this chapter.

Apart from several speedups as described in Section 3.6, Cervantes-Vázquez, Chenu, Chi-Domínguez, De Feo, Rodríguez-Henríquez, and Smith [37] propose a dummy-free variant of the constant-time implementation from [122] that e.g. only allows for even key elements $e_i \in [-10, 10]$. The required $|e_i|$ isogenies can then be computed as usual, followed by $(10 - |e_i|)/2$ isogenies in both positive and negative direction. Thus, instead of using dummy isogenies, we compute an equal number of isogenies in both directions, such that they effectively cancel each other. This allows for an increased side-channel protection, e.g. against fault attacks. However, it is easy to see that the increased number of total isogenies leads to a slowdown by a factor of 2 compared to the approach from [122].

Furthermore, [37] proposes a derandomized version of CSIDH. For this cause, the underlying prime p must be increased such that enough ℓ_i are available to cover a keyspace of size 2^{256} with exponents in $\{-1, 0, 1\}$, resp. $\{-1, 1\}$ for a dummy-free variant. Then it suffices to have full order points P_+ and P_- available in order to compute the class group action without randomly sampling points. Note that the generation of such points for a public curve can be combined with the key validation process, see [37].

A different approach on setting up a constant-time CSIDH algorithm has been given by Chi-Domínguez and Rodríguez-Henríquez [39]. They adapt the notion of optimal strategies from SIDH, and apply this idea to CSIDH. In particular, during the computation of the kernel generator, intermediate points can be stored and pushed through the following isogeny, therefore potentially requiring much smaller multiplications in order to compute the next kernel generators. Compared to the implementation from [122] using SIMBA, [39] reports a speedup of 3.3%. A slightly larger speedup has been obtained by Hutchinson, LeGrow, Koziel, and Azarderakhsh [92] by systematically investigating optimal SIMBA strategies.

All of the aforementioned implementations use the traditional isogeny formulas for Montgomery resp. twisted Edwards curves. However, the recent squareroot complexity formulas [13] can be used to speed up constant-time implementations (see [2]). The same is true for the deterministic isogenies of small degrees [32, 33], which have not yet been analyzed in this setting.

While all these implementations are secure against certain side-channel attacks, e.g. aiming at running time leakage, other approaches such as fault injections have received relatively little attention. In the following chapter we describe practical fault attacks against the implementation from [122] and present countermeasures.

4.A Detailed implementation results

We tested several parameters in a dynamical implementation, as explained in this chapter. The setting is the same as in Section 4.5. As parameters $B_0, \ldots, B_4$ we chose

$$
\begin{aligned}
B_0 &= [10, 10, 10, \ldots, 10], \\
B_1 &= [1, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 8, 8, 8, 8, 8, 8, 12, 12, 12, 12, \\
&\quad 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, \\
&\quad 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14], \\
B_2 &= [5, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 8, 8, 8, 8, 8, 8, 11, 11, 11, 11, \\
&\quad 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 13, 13, 13, 13, 13, 13, 13, 13, \\
&\quad 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13], \\
B_3 &= [2, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 10, 10, 10, 10, 10, 10, \\
&\quad 10, 10, 10, 10, 10, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, \\
&\quad 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16], \\
B_4 &= [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 12, 20, 20, 20, 20, 20, 20, \\
&\quad 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, \\
&\quad 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20].
\end{aligned}
$$

We measured many different combinations with different m and μ in a dynamic version of our implementation, running SIMBA-m-μ as described above, averaging the running time over 1000 runs per parameter set, given in 10^6 clock cycles. For each B_i, we present the three best combinations we found in Table 4.3.

Table 4.3 does not include speedups from rearranging the order of the primes ℓ_i for the different parameters in order to take advantage of the skipped point evaluations, as described in Section 4.5. However, all parameter sets seem to benefit equally from this, which means that the ranking of the parameters from Table 4.3 stays unchanged.

4.B Algorithms

In this section we describe our constant-time algorithm, containing the optimizations from above. We split the application of SIMBA in two parts: SIMBA-split (Algorithm 3) splits the isogeny computations in m batches, and SIMBA-merge (Algorithm 4) merges them after μ rounds.

Algorithm 5 shows the full class group action evaluation. We recommend to additionally analyze our implementation, provided in Section 4.5.

Table 4.3: Performance of one class group action evaluation in CSIDH-512 with different combinations of parameters. All timings are given in 10^6 clock cycles, and were measured on an Intel Core i7-6500 Skylake processor, averaged over 1000 runs.

B	1st		2nd		3rd	
0	μ=10 m=5	338.1	μ=10 m=6	343.5	μ=11 m=5	343.7
1	μ=12 m=4	329.3	μ=14 m=4	330.6	μ=13 m=4	330.8
2	μ=11 m=5	326.5	μ=12 m=5	327.0	μ=11 m=4	327.6
3	μ=16 m=4	333.8	μ=17 m=4	337.6	μ=16 m=3	339.3
4	μ=20 m=3	397.5	μ=20 m=4	399.0	μ=21 m=3	399.5

Algorithm 3: SIMBA-split.

Input : $e = (e_1, \ldots, e_n)$, $B = (B_1, \ldots, B_n)$, m.
Output: $e^i = (e_1^i, \ldots, e_n^i)$, $f^i = (f_1^i, \ldots, f_n^i)$, k_i for $i \in \{0, \ldots, m-1\}$.

Initialize $e^i = f^i = (0, 0, \ldots, 0)$ and $k_i = 4$ for $i \in \{0, \ldots, m-1\}$.
foreach $i \in \{1, \ldots, n\}$ **do**
    $e_i^{i \bmod m} \leftarrow e_i$
    $f_i^{i \bmod m} \leftarrow B_i - e_i$
    foreach $j \in \{1, \ldots, m\}$ **do**
        if $j \not\equiv i \bmod m$ **then**
            $k_i \leftarrow k_i \cdot \ell_i$

Algorithm 4: SIMBA-merge.

Input : $e^i = (e_1^i, \ldots, e_n^i)$ and $f^i = (f_1^i, \ldots, f_n^i)$ for $i \in \{0, \ldots, m-1\}$, m.
Output: $e = (e_1, \ldots, e_n)$, $f = (f_1, \ldots, f_n)$, and k.

Initialize $e = f = (0, 0, \ldots, 0)$, and $k = 4$.
foreach $i \in \{1, \ldots, n\}$ **do**
    $e_i \leftarrow e_i^{i \bmod m}$
    $f_i \leftarrow f_i^{i \bmod m}$
    if $e_i = 0$ *and* $f_i = 0$ **then**
        $k \leftarrow k \cdot \ell_i$

Algorithm 5: Constant-time evaluation of the class group action in CSIDH-512.

Input : $a \in \mathbb{F}_p$ such that $E_a : y^2 = x^3 + ax^2 + x$ is supersingular, a list of integers $(e_1, \ldots, e_n)$ with $0 \leq e_i \leq B_i$ for all $i \leq n$, $B = (B_1, \ldots, B_n)$, m, μ.

Output: $a' \in \mathbb{F}_p$, the curve parameter of the resulting curve $E_{a'}$.

Run SIMBA-split(e, B, m).
foreach $i \in \{1, \ldots, \mu\}$ **do**
  foreach $j \in \{1, \ldots, m\}$ **do**
    Run Elligator to find a point P, where $y_P \in \mathbb{F}_p$.
    $P \leftarrow [k_j]P$
    $S = \{\iota \mid e_\iota^j \neq 0 \text{ or } f_\iota^j \neq 0\}$
    foreach $\iota \in S$ **do**
      $\alpha = \prod_{\kappa \in S, \kappa > \iota} \ell_\kappa$
      $K \leftarrow [\alpha]P$
      if $K \neq \infty$ **then**
        if $e_\iota^j \neq 0$ **then**
          Compute a degree ℓ_ι isogeny $\varphi : E_a \to E_{a'}$ with $\ker(\varphi) = \langle K \rangle$:
          $a \leftarrow a'$, $P \leftarrow \varphi(P)$, $e_\iota^j \leftarrow e_\iota^j - 1$.
        else
          Compute a degree ℓ_ι dummy isogeny:
          $a \leftarrow a$, $P \leftarrow [\ell_\iota]P$, $f_\iota^j \leftarrow f_\iota^j - 1$.
        if $e_\iota^j = 0$ *and* $f_\iota^j = 0$ **then**
          Set $k_j \leftarrow k_j \cdot \ell_\iota$.
Run SIMBA-merge on inputs e^i and f^i for $i \in \{0, ..., m-1\}$, and m.
while some $e_i \neq 0$ or $f_i \neq 0$ **do**
  Run Elligator to find a point P, where $y_P \in \mathbb{F}_p$.
  Set $P = (x : 1)$, $P \leftarrow [k]P$, $S = \{i \mid e_i \neq 0 \text{ or } f_i \neq 0\}$.
  foreach $i \in S$ **do**
    Let $m = \prod_{j \in S, j < i} \ell_i$.
    Set $K \leftarrow [m]P$.
    if $K \neq \infty$ **then**
      if $e_i \neq 0$ **then**
        Compute a degree ℓ_i isogeny $\varphi : E_a \to E_{a'}$ with $\ker(\varphi) = \langle K \rangle$:
        $a \leftarrow a'$, $P \leftarrow \varphi(P)$, $e_i \leftarrow e_i - 1$.
      else
        Compute a degree ℓ_i dummy isogeny:
        $a \leftarrow a$, $P \leftarrow [\ell_i]P$, $f_i \leftarrow f_i - 1$.
      if $e_i = 0$ *and* $f_i = 0$ **then**
        Set $k \leftarrow k \cdot \ell_i$.

Chapter 5

Practical fault injection attacks on CSIDH

5.1 Introduction

As seen in the previous chapters, isogeny-based cryptography, and in particular SIDH and CSIDH, are promising candidates for quantum-resistant schemes. However, it seems that until now, isogeny-based PQC schemes have not received much attention in the implementation attack literature [41]. Only few fault-injection attacks on SIDH and more general investigations [83, 150, 80] have been discussed and published in the community so far. In [37] fault-injection attacks on a constant-time implementation of CSIDH have been discussed. However, all previous publications only consider attacks on a theoretical level and omit discussing a particular fault model, fault-attack method, and fault-injection technique. To the best of our knowledge in none of the publications the practical execution of fault-injection attacks has been investigated. Therefore, this is the first work on practical evaluations of the feasibility of fault attacks on an implementation of the CSIDH key exchange protocol.

In this chapter we focus on CSIDH, for which there are currently two proposals to design constant-time implementations. One approach uses dummy computations to achieve a running time independent of secret data [113, 122, 37], while the other is dummy-free [37]. The former approach is believed to be less secure against fault attacks, but is twice as fast as the latter. In this work we evaluate practical fault attacks on the former approach and present countermeasures, leading to a relatively small slowdown by a factor of 1.07, which yields a significantly better performance than the dummy-free alternative.

The contributions of this chapter are as follows: Firstly, we discuss practical attacker models for fault attacks and side-channel assisted fault attacks on constant-time CSIDH implementations with dummy isogenies. We then simulate all discussed attack models and perform practical experiments with low-budget attack equipment, namely a Chip-

Whisperer-Lite board, which includes a 32-bit ARM Cortex-M4 processor as target core. Lastly, we practically evaluate the performance of the proposed countermeasures.

We place the code used for this work in the public domain; it is available at https://github.com/michael-meyer/phdthesis-code. It includes the CSIDH implementation with and without countermeasures, the attack-simulation scripts, and attack scripts.

Remark 10. The majority of this work was done prior to the publication of asymptotically faster isogeny formulas by Bernstein, De Feo, Leroux, and Smith [13]. Some of our countermeasures rely on the structure of the isogeny computations in the implementations of [37, 113, 122]. Since this is significantly altered in the formulas from [13], it is unclear whether they can be protected by similar countermeasures. However, for small degrees the formulas used in this work are still faster, and it is yet unclear for which threshold the new formulas become faster in a constant-time implementation. Even if there are no similar countermeasures for [13], one could design a hybrid implementation, where the small degrees use protected dummy computations, while the larger degrees use the dummy-free approach.

Organization. This chapter is structured as follows. Section 5.2 provides details about isogenies and constant-time implementations of CSIDH. In Section 5.3 we introduce and discuss the different attack models before providing simulations and practical evaluation results in Section 5.4 and Section 5.5. Countermeasures against the discussed attacks are proposed in Section 5.6. Before we conclude the chapter in Section 5.8, we provide performance results of an implementation containing the proposed countermeasures in Section 5.7.

5.2 Preliminaries

We briefly review CSIDH and the computation of isogenies, expand upon dummy isogenies, and report on existing constant-time implementations of CSIDH.

5.2.1 CSIDH and isogenies

Recall from the previous chapters that in CSIDH, we define a prime of the form $p = 4 \cdot \ell_1 \cdot \cdots \cdot \ell_n - 1$, where $\ell_1, \ldots, \ell_n$ are small distinct odd primes, and work with supersingular elliptic curves in Montgomery form $E_a : y^2 = x^3 + ax^2 + x$ over $\mathbb{F}_p$. A private key is given by a vector of integers $(e_1, \ldots, e_n)$, where the entry e_i determines that $|e_i|$ isogenies of degree ℓ_i have to be computed, and the sign of e_i determines if an order-ℓ_i point on the current curve or its twist has to be taken as input. The entries are sampled from a small interval $[-B, B]$ to obtain an efficient computation. This class group action evaluation thus takes as input a curve E_a, computes the required chain of isogenies, and outputs a different curve $E_{a'}$.

Isogenies. As presented in Chapter 3, an ℓ-isogeny φ can be computed as follows. Let $K \in E$ be a point of order $\ell = 2d + 1$, and denote by $(X_i : Z_i)$ the projective coordinates of the point $[i]K$. Then [45] shows that

$$\begin{aligned}\varphi : (X : Z) \mapsto \Bigg(X \left(\prod_{i=1}^{d} (X - Z)(X_i + Z_i) + (X + Z)(X_i - Z_i) \right)^2 : \\ Z \left(\prod_{i=1}^{d} (X - Z)(X_i + Z_i) - (X + Z)(X_i - Z_i) \right)^2 \Bigg).\end{aligned} \tag{5.1}$$

The projective codomain curve parameter $a' = (A' : C')$ of $E_{a'}$ can be computed by exploiting the birational equivalence to a twisted Edwards curve [114]

$$(A' : C') = \left(2 \cdot \left((A+2)^\ell \pi_+^8 + (A-2)^\ell \pi_-^8 \right) : (A+2)^\ell \pi_+^8 - (A-2)^\ell \pi_-^8 \right), \tag{5.2}$$

where

$$\pi_+ = \prod_{i=1}^{d} (X_i + Z_i) \quad \text{and} \quad \pi_- = \prod_{i=1}^{d} (X_i - Z_i).$$

Dummy isogenies. As suggested in Chapter 4, constant-time algorithms of CSIDH often use dummy isogenies, since otherwise the running time is correlated to the secret key, which specifies the number of isogenies to be computed. These dummy computations perform the same instructions as real isogeny computations, but discard the results. Thus, they allow for a fixed number of isogeny computations, independent of the respective private key.

In order to speed up computations, dummy isogenies are designed to compute $[\ell]P$ for the input point P. This has to be done, since for a real isogeny of degree ℓ, the order of P loses the factor ℓ by being pushed through if the kernel generator K was computed from P. Therefore, a dummy isogeny would require a subsequent multiplication $[\ell]P$, which is prevented by performing this computation inside the dummy algorithm. To this end, a dummy isogeny swaps the input points K (kernel generator point) and P (point to be evaluated), to compute $[(\ell - 1)/2]P$ in the kernel computation part. Then two further differential additions suffice to compute $[\ell]P$. However, this method requires to perform these two further additions in a real isogeny as well, and discard their results, in order to achieve a constant-time behavior.

Figure 5.1 and Figure 5.2 show the different computation blocks that are contained in the degree ℓ isogeny algorithm. For real and dummy isogenies, the green blocks are necessary computations in order to produce a valid output, while the red blocks entirely consist of dummy computations, whose results are discarded. Note that these figures do not show conditional swaps, which are necessary to avoid conditional branches based on the private key. We refer to [113] and the accompanying implementation for more details.

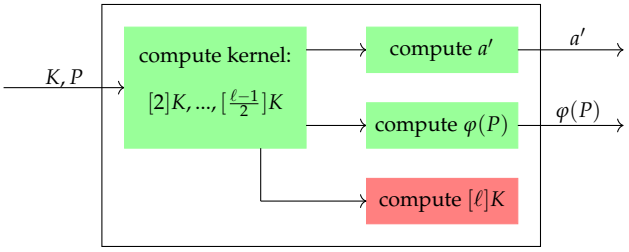


Figure 5.1: Real isogeny

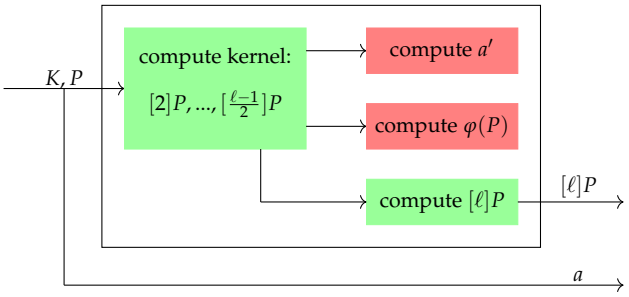


Figure 5.2: Dummy isogeny

5.2.2 Constant-time algorithms

As presented in Chapter 4, Meyer, Campos, and Reith (MCR) [113] pointed out that in addition to the variable number of isogenies, also the sign distribution of the key elements may leak information through the running time. Thus, they proposed a constant-time algorithm of CSIDH by using dummy isogenies, and by changing the secret key intervals from $[-B, B]^n$ to $[0, 2B]^n$. As a result, for any secret key the performance is the same as for the action of the integer vector $(2B, \ldots, 2B)$. This cost is about twice as much as that of the action of $(B, \ldots, B)$, which is the worst case in the variable-time algorithm. Further, they proposed several optimizations, such as the batching technique SIMBA or the usage of the point sampling method Elligator [15], which was first used in the context of CSIDH in [17], and obtain a speedup factor of roughly 2.

Onuki, Aikawa, Yamazaki, and Takagi (OAYT) [122] proposed an idea for mitigating the increase of the computational cost due to the key interval $[0, 2B]$. By keeping two points $P_0 \in E[\pi + 1]$ and $P_1 \in E[\pi - 1]$ in each step in the algorithm, where π denotes the Frobenius endomorphism, one can compute isogenies for positive signs and negative signs of a secret key in the same loop. By always choosing the point P_s that suits the sign of e_i for computing the kernel generator of an ℓ_i-isogeny, the correlation between running time and sign distribution is eliminated. Thus, this method allows for the use of the secret key intervals $[-B, B]^n$, and therefore halves the number of total isogenies at the cost of an additional point evaluation per isogeny. We describe their approach in Algorithm 6.

Algorithm 6: Constant-time class group action (OAYT approach)

Input : $a \in \mathbb{F}_p$ s.t, E_a is supersingular, $m \in \mathbb{N}$,
$(e_1, \ldots, e_n)$ s.t. $-m \leq e_i \leq m$ for $i = 1, \ldots, n$.
Output: $a' \in \mathbb{F}_p$ s.t. $E_{a'} = (\mathfrak{l}_1^{e_1} \cdots \mathfrak{l}_n^{e_n}) * E_a$.

Set $e'_i = m - |e_i|$ for $i = 1, \ldots, n$.
while some $e_i \neq 0$ or $e'_i \neq 0$ **do**
  Set $S = \{i \mid e_i \neq 0 \text{ or } e'_i \neq 0\}$.
  Set $k = \prod_{i \in S} \ell_i$.
  Generate $P_0 \in E_a[\pi + 1]$ and $P_1 \in E_a[\pi - 1]$ by Elligator.
  Let $P_0 \leftarrow [(p+1)/k]P_0$ and $P_1 \leftarrow [(p+1)/k]P_1$.
  for $i \in S$ **do**
    Set s the sign bit of e_i.
    Set $K = [k/\ell_i]P_s$.
    Let $P_{1-s} \leftarrow [\ell_i]P_{1-s}$.
    if $K \neq \infty$ **then**
      if $e_i \neq 0$ **then**
        Compute $\varphi : E_a \to E_{a'}$ with $\ker \varphi = \langle K \rangle$.
        Let $a \leftarrow a'$, $P_0 \leftarrow \varphi(P_0)$, $P_1 \leftarrow \varphi(P_1)$, and $e_i \leftarrow e_i - 1 + 2s$.
      else
        Compute dummy isogeny:
        Let $a \leftarrow a$, $P_s \leftarrow [\ell_i]P_s$, and $e'_i \leftarrow e'_i - 1$.
    Let $k \leftarrow k/\ell_i$.
return A.

Note that, for the sake of simplicity, optimizations such as SIMBA are not described in Algorithm 6. We refer to [113, 122] for more details.

In [37], Cervantes-Vázquez, Chenu, Chi-Domínguez, De Feo, Rodríguez-Henríquez, and Smith (CCCDRS) obtained a speedup for the MCR and OAYT implementations by using twisted Edwards curves. Further, they proposed a dummy-less implementation in order to improve the resistance against fault attacks, at the cost of a slowdown by a factor of 2.

5.3 Attacker models

The attacker we are modeling in this work is deploying safe-error analysis to detect the dummy isogenies within CSIDH, i.e., they inject a single fault during the computation of the CSIDH group action and observe if an occurring fault impacts the shared secret. An adversary who can reliably skip or corrupt an isogeny computation of a chosen degree at

a chosen index can easily recover the full secret key with a relatively small number of fault injections. However, due to various sources of randomness during the execution, it is impossible to always corrupt the intended operation, and without side-channel information an adversary cannot know which isogeny was affected. Therefore, we propose three different attacker models with increasing capabilities to evaluate the impact of the resulting attacks.

In general, we assume that an adversary is able to repeatedly trigger an evaluation of the group action using the same secret key. The input curve may be the same for all evaluations, but may also be different. As CSIDH allows a static-static key exchange, this is likely how a key exchange is implemented. The attacker is able to inject faults that will set variables to random values or skip instructions. An attacker is limited to observe whether both parties obtained the same shared secret, e.g., by observing failure later in the protocol. Expressed in a more formal way, this model is the same as the second oracle from [80]. We propose the following three attackers with increasing capabilities. Attacker 1 and Attacker 2 are limited to fault injection, while Attacker 3 can also obtain additional side-channel information.

- **Attacker 1: Shotgun at the CSIDH.** Our weakest adversary model assumes that the attacker can reliably cause a fault during the computation of the CSIDH group action, but has no control over the location of the fault. They can then observe how often this leads to a wrong shared secret. This proportion of failures intuitively is depending on the ratio of real vs. dummy isogenies. While this is a rather weak adversary model, it nicely demonstrates the inherent problem of dummy operations in the context of fault injection attacks.
  The main limitation of Attacker 1 is that they have no control over the operation that is affected. Since the isogeny computations make up about 42% of cycles during the group action on the Cortex-M4, the attacker is likely to hit an isogeny computation relatively often. However, they have no knowledge of the order of the faulty isogeny computation which limits the information they can learn about the secret key.

- **Attacker 2: Aiming at isogenies at index i.** A slightly more powerful adversary can target isogeny computations at positions of their choice. This does not fully allow to target isogenies of a chosen degree, as the isogenies may be evaluated out of order due to point rejections. However, since the first evaluated isogenies have relatively large orders ℓ_i, and the point rejection probability is $1/\ell_i$, the sequence of the first isogenies is almost deterministic and the individual isogenies can be targeted easily. We evaluate how many isogenies the adversary can realistically attack in Section 5.4. For all entries of the secret key with $e_i = 0$, the injected fault will not change the result, and an adversary immediately knows this part of the secret key. For the remaining e_i the adversary has reduced the search space.

- **Attacker 3: Aiming at isogeny computations and tracing the order.** Our most powerful attacker model complements Attacker 2 by additionally allowing the adversary to trace the faulty isogeny computation to determine the degree of the isogeny that the fault was injected into. Since the isogeny order determines the running time of the isogeny computation, the order might be recovered from a power trace, e.g., using Simple Power Analysis [104].

One could imagine yet another adversary who is capable of setting certain e_i of the secret key to a chosen value. A possible attack would be as follows: For each e_i try all possible values and observe for which value the derived shared secret is correct. For the CSIDH-512 parameters proposed in [122], this would require at most 882 successful fault injections for fully recovering the secret key. This attack would also apply to dummy-free implementations like [37]. Note, however, that this adversary is overly powerful especially when assuming the low-cost fault injection equipment we are targeting in this work. Therefore, we focus on more realistic fault models for the remainder of this chapter which can be achieved using relatively cheap clock-glitching equipment.

5.4 Simulation

To gain a better understanding of how many fault injections an adversary would require to obtain a certain key space reduction or key recovery, we simulate the three previously defined adversary models and mount practical experiments on them. By using simulations, we can run much more experiments than when running the actual CSIDH implementation on a microcontroller target that would be suitable for fault injection attacks.

5.4.1 Attack 1

For the simulation of Attack 1, we implemented a Python script that simulates all operations that are performed within CSIDH-512 in the OAYT implementation. Our approach works as follows: We use our implemented cost-simulation to output a transcript containing each point multiplication, isogeny computation, etc., in addition to their cost measured in $\mathbb{F}_p$-multiplications. We then select a fault position using the strategy corresponding to the attack model (e.g., a uniformly random number between 0 and the total number of measured $\mathbb{F}_p$-multiplications for Attacker 1) and determine the impact of a fault occurring at that position. This script is parameterized by the relative cost of each field operation over $\mathbb{F}_p$, which we experimentally determine for our target implementation.

In general there can be two outcomes of a simulated fault injection:

- A fault was injected into an operation that was not a dummy operation which will lead to a wrong shared secret in most cases, which can be observed by the adversary.

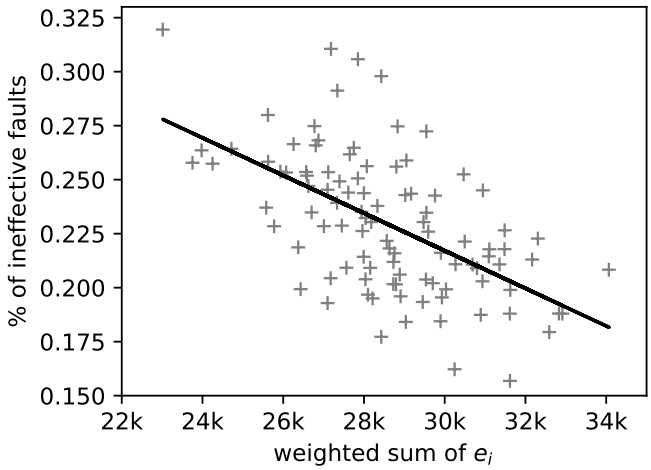


Figure 5.3: Simulation results for Attack 1 using 100 random CSIDH-512 secret keys. 500,000 faults are injected into random operations during the group action.

- A fault was injected into a dummy operation, i.e., there is no change in the shared secret. This can be considered an ineffective fault.

Note that there are some special cases, where a fault can be injected into a non-dummy operation, but the resulting shared secret is not influenced by this. Although these cases are rather rare, our simulation still considers them, in order to give more realistic results.

In Attack 1, the adversary simply observes the percentage of fault injections that yield a wrong shared secret. This proportion depends on the secret key as it correlates with the proportion of real versus dummy operations.

Results. We simulated the attack for 100 randomly selected CSIDH-512 keys and performed 500,000 runs with a single fault injection at a random location during the entire group action per key. Figure 5.3 shows the plot of the probability of an ineffective fault in relation to the weighted sum of the secret key. As the runtime of an individual isogeny is linear in its degree, the time spent in ℓ_i-isogenies is proportional to $|e_i| \cdot \ell_i$. Therefore, we compute the weighted sum as $\sum |e_i|\ell_i$ which corresponds to the approximate time spent in real isogenies. From the simulation, it is easy to see that the probability of seeing a faulty shared secret is correlated with the secret key. An adversary learning this probability, can also infer information about the secret key. The more faults are injected, the more evident this relationship becomes.

Impact. After obtaining the percentage of ineffective faults for a large enough number of rounds, the attacker now wants to gain information on the used secret key. However, we provide an example to show that this does not lead to a large reduction of the possible

key space. Suppose the attacker obtains a percentage that allows them to assume that the weighted sum of the secret key is less than 24,000. Then, by a Monte Carlo method, we can estimate that roughly 1% of all the possible CSIDH-512 keys satisfy this condition. This means that the search space reduces from 2^{256} to roughly 2^{249}. Since the measured correlation between the obtained percentage and weighted sum of the key is not even strong enough to allow for an assumption as in this example, we conclude that this attack is not able to significantly reduce the respective search space.

5.4.2 Attack 2

In the proposed constant-time implementations based on dummy isogenies (MCR and OAYT), the calculation for a certain e_i from the secret key vector $(e_1, \ldots, e_n)$ follows a certain pattern. In particular, first real and then dummy isogenies are calculated (see lines 12 - 17 in Algorithm 6). Thus, it is sufficient to determine within this calculation sequence where the first dummy isogeny occurs, in order to obtain the absolute value of each e_i.

In Attack 2 and Attack 3, we assume that the attacker knows spots in the isogeny computation for the respective degree which reveal whether it is a real or dummy isogeny with a single fault injection. Such critical spots (according to Figure 5.1 and Figure 5.2) in the code can be empirically determined in advance with manageable effort. In our experiments, we achieved an accuracy of over 95% with a single fault injection.

Results. For the second attack, it suffices to simply determine up to which isogeny computation the algorithm is likely to be deterministic, i.e., no points are going to be resampled. Since the probability of point rejection for a given degree ℓ_i is $1/\ell_i$, the sequence of the first isogeny computations is deterministic with high probability, due to the relatively large degrees. For example, the attacker knows with a probability of 71% that the first 23 isogeny computations run without point rejection in the OAYT implementation. This makes it easy to target these first 23 isogenies and find out whether they are real or dummy computations with relatively few fault attempts. Extending this number of 23 isogenies leads to a quickly increasing probability for point rejections, thus preventing unambiguous results for later isogenies.

Impact. Assuming that the first 23 isogenies are attacked, the space reduction achieved in this attack model is from 2^{256} to 2^{177} in the best case, where all the respective key elements are 0, and roughly to 2^{244} in the average case. For the average case, we assume that $1/11$ of the respective key elements, which lie in the range $[-5, 5]$ resp. $[0, 10]$, are 0. In the worst case, i.e., none of the respective key elements being 0, the keyspace is reduced to 2^{253}.

5.4.3 Attack 3

Since in this attack model the attacker is also able to trace the order of the isogeny calculation, a divide-and-conquer approach provides the most effective strategy.

Results. Both constant-time implementations (MCR and OAYT) use a bound vector $B = (B_1, B_2, \ldots, B_n)$ defining the intervals from which each secret exponent e_i must be sampled. The number of fault injections required to obtain the absolute value of a certain e_i depends on the corresponding B_i from the bound vector and on the number of attempts needed to distinguish a real from a dummy isogeny. For each individual degree, the attacker simply performs a binary search until the calculation of the first dummy isogeny is identified.

For a specific e_i, the adversary proceeds as follows. They first attack the $\lceil B_i/2 \rceil$-th isogeny calculation. If this is a dummy isogeny, they can simply perform a binary search through the lower half of the list by attacking the $\lceil B_i/4 \rceil$-th isogeny, etc., until the precise position of the first executed dummy isogeny is identified. Likewise, if the $\lceil B_i/2 \rceil$-th calculation is not a dummy isogeny, they can perform a binary search through the upper half. Thus, for a given secret key the worst-case number of required injections Γ can be calculated by

$$\Gamma = \sum_{i=1}^{n} \left(\lceil \log_2(B_i) \rceil \cdot \gamma_i \right),$$

where γ_i is the number of necessary injections to identify the type (dummy or real) of a single ℓ_i-isogeny, depending on the practical setup.

Impact. In the case of the MCR implementation with secret key vector entries $e_i \in [0, 10]$, this means that at most 296 injections are required in the worst case for a full key recovery, where we assume the optimal case of $\gamma_i = 1$ for all i. In the case of the OAYT implementation with secret key vector entries $e_i \in [-5, 5]$, our strategy requires at most 222 injections,[13] but only determines the private key entries up to their sign, and thus leads to a space reduction to 2^{74} in the worst case and to $2^{67.06}$ in the average case. The remaining search complexity can be further reduced to roughly 2^{38} in the worst case resp. $2^{34.5}$ in the average case by a meet-in-the-middle approach as follows (see [35]).

Let E be the initial curve and E' the public key curve. Assume that the attacker knows the degree $\ell_1^{|e_1|} \cdot \ldots \cdot \ell_{74}^{|e_{74}|}$ of the secret isogeny, i.e., knows the key up to the signs of the e_i. For simplicity, we assume all the exponents e_i are non-zero, so there are 2^{74} possible keys. Further, we initially consider all e_i to be positive. Then the attacker computes $[\ell_1^{s_1 e_1}, \ldots, \ell_{37}^{s_{37} e_{37}}] * E$ for all $(s_1, \ldots, s_{37}) \in \{-1, 1\}^{37}$ and stores all the resulting curves. This costs 2^{37} group action executions and uses storage of 2^{37} $\mathbb{F}_p$-elements. Next, they compute $[\ell_{38}^{-s_{38} e_{38}}, \ldots, \ell_{74}^{-s_{74} e_{74}}] * E'$ for $(s_{38}, \ldots, s_{74}) \in \{-1, 1\}^{37}$ and check whether the resulting

[13] For the deviating bound vectors used in the implementations of [113] and [122] the worst-case number of fault injections is 268 resp. 192.

curve equals a previously stored curve. If $[\ell_1^{s_1 e_1}, \ldots, \ell_{37}^{s_{37} e_{37}}] * E = [\ell_{38}^{-s_{38} e_{38}}, \ldots, \ell_{74}^{-s_{74} e_{74}}] * E'$ for some choice of the s_i, then the secret key is given by $(s_1 e_1, \ldots, s_{74} e_{74})$. In the worst case, this computation costs $2 \cdot 2^{37}$ group actions and 2^{37} equivalence checks of $\mathbb{F}_p$-elements, and uses storage of 2^{37} $\mathbb{F}_p$-elements.

5.5 Practical experiments

All our fault-injection experiments were performed on a ChipWhisperer-Lite (CW1173) 32-bit basic board, which includes a 32-bit STM32F303 ARM Cortex-M4 processor as the target core. The attacks were implemented in Python (version 3.6.9) using the ChipWhisperer open source toolchain[14] (version 5.1.3). An ARM plain C implementation of CSIDH, based on the implementation by Onuki, Aikawa, Yamazaki, and Takagi (OAYT) [122], was implemented for our project.

To reduce the time required for all experiments on the target board, we reduced the key space from 11^{74} to 3^2, i.e., our secret consists of two elements in $\{-1, 0, 1\}$. Furthermore, in Attack 1 we only compute isogenies with the smallest degrees (3 and 5). The practical experiments can thus be seen as a simplified demonstration of the feasibility of the described attacks.

In all implemented attacks, the isogenies are calculated without randomness, i.e., suitable points and private keys were precomputed. To require only one CSIDH action call per experiment, Bob's public key and the resulting shared secret for Alice's given public key were calculated in advance. Specifically, in all the implemented scenarios Alice's computation of the shared secret is attacked.

In our setup the fault is injected by suddenly increasing the clock frequency, hence, forcing the target core to skip an instruction.

Table 5.1 shows the results for the practical attacks. While the rate for Attack 1 increases slightly for keys containing more real isogenies, the increase for random-based (without knowledge of critical points) Attack 2 is much higher. The results for Attack 2 also apply to Attack 3.

5.6 Countermeasures

We describe countermeasures for the OAYT implementation, but note that this also applies to MCR, and, with slight modifications, to the CCCDRS implementation containing dummy computations.

It is evident that Attack 3 is the main threat that should be considered for countermeasures. Thus, we analyze the required countermeasures for the involved dummy computations during isogenies. However, the simulation of Attack 1 shows that there are other

[14] https://github.com/newaetech/chipwhisperer, commit 887e6c7

Table 5.1: Results for Attack 1 and Attack 2

type	key	# of trials	faulty shared secret
Attack 1	{0,0}	5000	19.8%
	{0,1}	5000	27.3%
	{-1,1}	5000	32.8%
Attack 2	{0,1}	5000	2.1%
	{-1,1}	5000	16.4%

parts of the CSIDH algorithm that could leak some information on the private key, if specifically attacked as in the Attack 3 model. Therefore, we describe the further required countermeasures, such that the resulting implementation is secure against leakage in all three attack models.

A rather simple countermeasure would be to randomize the order in which real and dummy isogenies for a specific degree are computed, instead of always computing the real ones first. However, Attacker 3 can still attack this with a slightly larger number of faults, using a probabilistic method to obtain the key elements. In contrast to this, our idea for countermeasures against the described fault injection attacks is to redesign the algorithm such that any fault injection will lead to the output of an error instead of the output curve. This means that an attacker does no longer see if the injected fault affected a real or a dummy operation, and is thus effective against all three attack models we described.

A key function that is frequently used is a check for equality. This is performed in constant time and therefore does not leak any information. The presented countermeasures are designed for our described specific attack model, i.e., the adversary is limited to injecting exactly one fault, which can either be a random fault or an instruction skip.

5.6.1 Isogenies

In order to reach security against Attack 3, we have to be able to detect faults during the dummy computations of isogenies. However, we stress that we require a unified isogeny algorithm, which computes a real isogeny or dummy isogeny of given degree in constant time, based on a decision bit $b \in \{0, 1\}$. This means that countermeasures for one of the two cases must be executed in *both* cases to maintain the constant-time property. However, it must be ensured that the verifications only lead to the output of an error in the relevant case. This is implemented via the function `cverify`(x, y, b), which always checks whether $x = y$ via the constant-time check for equality, but only outputs the result if $b = 1$.

Throughout this section, we assume that the decision bit is set as $b = 0$ if a dummy isogeny is to be computed, and $b = 1$ for the real isogeny case.

Real isogenies. As depicted in Section 5.2, the two additionally required differential additions (DADDs) are the only dummy computations in a real isogeny. Since their output is discarded, we have to validate that no fault has been injected during their execution. However, in this case the validation is straightforward. The DADDs are designed to compute $K' = [\ell]K$ for an ℓ-isogeny (see Section 5.2). Thus, in real isogenies, the result must be the point ∞, since K has order ℓ. This means that we can simply call $\texttt{cverify}(K', \infty, b)$, in order to perform this validation only in the case of real isogenies.

Dummy isogenies. In dummy isogenies, the dummy computations are given by the codomain curve computation and the point evaluation, as described in Section 5.2. However, in this case the involved dummy computations do not allow for an elegant verification of point orders or supersingularity as in all the other cases in this section. Instead, we will make use of a conditional addition function $\texttt{cadd}(x, y, b)$, which outputs $x + by$, i.e., x if $b = 0$ or $x + y$ if $b = 1$. This function is implemented by first calling a conditional set function, which takes as input y and b, is initialized by the output value 0, and overwrites this output by y if $b = 1$. Note that this function is implemented to run in constant time, in order to prevent leakage. Then, we call the usual addition function for $\mathbb{F}_p$-elements, and obtain the desired output in constant time.

While we have to maintain the structure of computations in the case of real isogenies (i.e., for $b = 1$), we have to make changes to them in order to obtain verifiable results in the dummy case (i.e., for $b = 0$). In order to achieve this in a constant-time manner, we make use of the conditional add function. Analogously, we define a conditional subtraction $\texttt{csub}(x, y, b)$, and can compute $\texttt{cadd2}(x, y, b)$ with result $bx + by$ through two calls to the conditional set function.

Dummy codomain curve computation. Instead of using multiples of the kernel generator K, dummy isogenies use multiples of the input point P_s. Thus, the output does not refer to a special type of curve or follow any other special property that can be validated.

Recall that the codomain curve parameters are computed by Equation (5.2). It is evident that different steps during the computations of A' and C' contain similar terms, and mostly differ in sign changes. Therefore, our strategy to evaluate these computations in the dummy case is to manipulate some of them with conditional additions, in order to obtain $A' = C'$. To reach this, our algorithm is designed in a way such that a fault injection in any line of code leads to $A' \neq C'$ in the dummy case. On the other hand, it obviously computes the correct output parameters in the case of real isogenies. Algorithm 7 details this method. Again we make use of the conditional verify function, to only possibly raise an error if $b = 0$.

Algorithm 7: Protecting the codomain curve computation

Input : Curve parameters $A, C \in \mathbb{F}_p$, degree ℓ, kernel points $(X_i : Z_i)$ for $1 \leq i \leq (\ell-1)/2$, decision bit $b \in \{0,1\}$.

Output: Curve parameters $A', C' \in \mathbb{F}_p$, error variable *error*.

1 Set $\pi_+ \leftarrow 1$, $\pi_- \leftarrow 1$.
2 **for** $i \in \{1, \ldots, (\ell-1)/2\}$ **do**
3 $\quad t_0 \leftarrow \texttt{cadd}(X_i, Z_i, b)$ // $t_0 = X_i \;|\; t_0 = X_i + Z_i$
4 $\quad t_1 \leftarrow \texttt{csub}(X_i, Z_i, b)$ // $t_1 = X_i \;|\; t_1 = X_i - Z_i$
5 $\quad \pi_+ \leftarrow \pi_+ \cdot t_0$ // $\pi_+ = \prod X_i \;|\; \pi_+ = \prod (X_i + Z_i)$
6 $\quad \pi_- \leftarrow \pi_- \cdot t_1$ // $\pi_- = \prod X_i \;|\; \pi_- = \prod (X_i - Z_i)$
7 $t_0 \leftarrow \texttt{cadd2}(C, C, b)$ // $t_0 = 0 \;|\; t_0 = 2C$
8 $t_1 \leftarrow (A - t_0)^\ell \cdot \pi_-^8$ // $t_1 = A^\ell \cdot \pi_-^8 \;|\; t_1 = (A - 2C)^\ell \cdot \pi_-^8$
9 $t_0 \leftarrow (A + t_0)^\ell \cdot \pi_+^8$ // $t_0 = A^\ell \cdot \pi_+^8 \;|\; t_0 = (A + 2C)^\ell \cdot \pi_+^8$
10 $A' \leftarrow \texttt{cadd}(t_1, t_0, b)$ // $A' = t_1 \;|\; A' = t_0 + t_1$
11 $A' \leftarrow \texttt{cadd}(A', A', b)$ // $A' = t_1 \;|\; A' = 2(t_0 + t_1)$
12 $C' \leftarrow \texttt{csub}(t_0, t_1, b)$ // $C' = t_0 \;|\; C' = t_0 - t_1$
13 $error \leftarrow \texttt{cverify}(A', C', \neg b)$ // `if` $b = 0$: `verify that` $A' = C'$
14 **return** $A', C', error$.

Dummy point evaluation. Analogously to the codomain curve computation, there is no possibility to check for the correct executions of this part through point order checks in the dummy case. Thus, we resort to the same strategy as for the codomain curve computation.

The output points are computed by Equation (5.1). We analogously manipulate the computations to output values satisfying $X' = Z'$ in the dummy case. As above, a fault to any line of code will result in output values with $X' \neq Z'$, and in the real isogeny case, the original algorithm stays unchanged. This method is detailed in Algorithm 8. Note that we are required to run this algorithm twice per isogeny, since both points P_0 and P_1 must be pushed through an isogeny at each step.

Non-dummy computations. In addition to the faults aiming at dummy computations, we need to be able to detect faults in non-dummy computations as well, in order to output an error instead of the output at the end of the algorithm. Otherwise, the attacker could still observe the difference between these cases.

To this end, we note that the output of an isogeny consists of the codomain curve parameters and the evaluated points. If a fault is injected during the computation of the codomain curve, then (with very high probability) the resulting parameters will not refer to a supersingular curve anymore. This can be deduced from the fact that the probability of a random parameter $a = A/C$ to define a supersingular curve is roughly $1/\sqrt{p}$, and therefore negligible [35]. Thus, the resulting curve at the end of the algorithm will most

Algorithm 8: Protecting the point evaluation

Input : Input point $(X : Z)$, degree ℓ, kernel points $(X_i : Z_i)$ for $1 \leq i \leq (\ell - 1)/2$, decision bit $b \in \{0, 1\}$.
Output: Output point $(X' : Z')$, error variable *error*.

1 $t_+ \leftarrow \texttt{cadd}(X, Z, b)$ // $t_+ = X$ | $t_+ = X + Z$
2 $t_- \leftarrow \texttt{csub}(X, Z, b)$ // $t_- = X$ | $t_- = X - Z$
3 Set $\pi_X \leftarrow 1$, $\pi_Z \leftarrow 1$.
4 **for** $i \in \{1, \ldots, (\ell - 1)/2\}$ **do**
5 $t_0 \leftarrow \texttt{cadd}(X_i, Z_i, b)$ // $t_0 = X_i$ | $t_0 = X_i + Z_i$
6 $t_1 \leftarrow \texttt{csub}(X_i, Z_i, b)$ // $t_1 = X_i$ | $t_1 = X_i - Z_i$
7 $t_0 \leftarrow t_- \cdot t_0$ // $t_0 = X \cdot X_i$ | $t_0 = (X - Z)(X_i + Z_i)$
8 $t_1 \leftarrow t_+ \cdot t_1$ // $t_1 = X \cdot X_i$ | $t_1 = (X + Z)(X_i - Z_i)$
9 $t_2 \leftarrow \texttt{cadd}(t_1, t_0, b)$ // $t_2 = t_1$ | $t_2 = t_0 + t_1$
10 $t_3 \leftarrow \texttt{csub}(t_0, t_1, b)$ // $t_3 = t_0$ | $t_3 = t_0 - t_1$
11 $\pi_X \leftarrow \pi_X \cdot t_2$
12 $\pi_Z \leftarrow \pi_Z \cdot t_3$ // if $b = 0$: $\pi_X = \pi_Z$
13 $X' \leftarrow \texttt{cadd}(\neg b, X, b)$ // $X' = 1$ | $X' = X$
14 $Z' \leftarrow \texttt{cadd}(\neg b, Z, b)$ // $Z' = 1$ | $Z' = Z$
15 $X' \leftarrow X' \cdot \pi_X^2$ // $X' = \pi_X^2$ | $X' = X \cdot \pi_X^2$
16 $Z' \leftarrow Z' \cdot \pi_Z^2$ // $Z' = \pi_Z^2$ | $Z' = Z \cdot \pi_Z^2$
17 $error \leftarrow \texttt{cverify}(X', Z', \neg b)$ // if $b = 0$: verify that $X' = Z'$
18 **return** $X', Z', error$.

likely not be supersingular. It therefore suffices to perform a single supersingularity check, e.g. as done in the public key validation in [35], at the end of the algorithm, and output an error in case of a non-supersingular curve. Instead of using the validation from [35], we use a different, slightly relaxed approach. We simply sample a random point Q on the curve, and check that $[p + 1]Q = \infty$. This method is much faster, but has a small chance to output false positives, so is not usable as public key validation. However, we heuristically checked the probability for false positives, and found that in 10^8 experiments with random curve parameters, our method and the rigorous verification always produced the same result. Thus, it seems to be infeasible for an attacker to exploit this relaxed supersingularity check.

The case of output points will be handled in detail in the following section.

5.6.2 Point orders and scalar multiplications

Scalar multiplications take place in line 6 and 9–10 in Algorithm 6, and are intended to produce points of the desired orders. If during such a multiplication a fault is injected

randomly, i.e., not aiming to produce a specific faulty output, then the probability for still generating a point of desired order is negligible.[15] The same is true for faults injected during the point evaluation of a real isogeny. For detecting such a fault, it therefore suffices to check if the output point P has the required order ℓ by verifying that $[\ell]P = \infty$.

However, it is not required to perform such a check after each scalar multiplication resp. isogeny. Indeed, it suffices to check point orders in the two following situations.

- At the end of each run through a batch of isogenies, if all computations are running correctly, then both points P_0 and P_1 must be the point at infinity ∞ at the end of the for-loop in line 7 of Algorithm 6. Thus, if we verify this at the end of each run through a batch, we are able to detect faults even if the respective faulty point P_s is not used to generate a kernel input point for an isogeny anymore after the fault is injected. This ensures the correctness of the scalar multiplications in lines 6 and 10 of Algorithm 6, and of the involved isogeny point evaluations.

- In order to validate the scalar multiplication in line 9 of Algorithm 6, we need to verify that K indeed has the correct order ℓ for each isogeny. This is done by calculating $[\ell]K$ and verifying that the result equals ∞ for each isogeny. Note that in the case of real isogenies, a faulty point K leads to wrong results that can be detected anyway; however, in the dummy case, the input point K is discarded, so this order check is indeed required. In order to keep our algorithm constant-time, this therefore has to be done in both cases.

All other scalar multiplications do not require separate order checks, since faults can be detected by the mentioned verifications.

Remark 11. Theoretically, the attacker could try to inject a fault such that a specific output point is produced, although this is not possible in our attacker model. In particular, the above verification does not detect a fault, if the order of the output point divides the desired order. If the attacker produces a point that lies on the same curve as the correct output point would (i.e., not on its twist), then this does not lead to a wrong computation, and therefore does not lead to possible leakage. Note that in CSIDH any point K of order ℓ on the same curve produces the same ℓ-isogeny codomain curve. It only makes a difference if $K \in E[\pi - 1]$ or $K \in E[\pi + 1]$.

This also explains a possible attack strategy: The adversary forces the output of a point on the twist with order dividing the expected order. Thus, the respective isogenies are computed with the wrong direction in the isogeny graph, which leads to leakage.

However, this is only a theoretical attack. Indeed, the chance for this to happen by accident is negligible, and to specifically map a point to a point on the twist of the same

[15] This follows since the required orders are always small, and for each prime factor $\ell_i | \#E[\pi - 1] = \#E[\pi + 1] = p + 1$, the probability for the order of a random point with x-coordinate in $\mathbb{F}_p$ to contain the factor ℓ_i is $1 - 1/\ell_i$.

order, with an unknown curve, seems infeasible. Even computing such a point with known curve would require to compute an irrational endomorphism, which, if possible in general, would completely break CSIDH on its own [36]. Computing small prime order points, thus possibly having an order dividing the expected order, could otherwise be done through division polynomials. However, as explained above, the attacker does not know the current curve (except for the starting curve) when injecting a fault, which means that division polynomials cannot be computed.

However, there is a simple, but rather costly, countermeasure to prevent attacks of this fashion. It suffices to check if the input kernel generator K lies on the correct curve via a Legendre symbol computation for each isogeny, and output an error otherwise. Although this seems not to be necessary for the reasons above, we report on the performance implications for this in Section 5.7.

5.6.3 Other functions

The CSIDH constant-time algorithms from [113, 122] feature some more functions outside of the scope of the analysis above. Compared to isogeny computations and scalar multiplications, their share of the total running time is small. Nevertheless, we review if countermeasures against fault injections for these functions are required.

The mentioned functions include the Elligator map [15], a method for efficient point sampling. For our discussion and in our implementation, we use the projective Elligator implementation from [37]. Further, the constant-time conditional point swap function `cswap` plays an important role in all current constant-time CSIDH implementations, and we review a method to prevent obvious loop-abort faults.

Apart from these functions, there are different functions such as integer multiplications. However, we disregard them here, since any fault to these functions is detectable through our described methods, e.g. through point order checks.

Conditional point swaps. The `cswap` function takes two $\mathbb{F}_p$-values and a decision bit $b \in \{0, 1\}$ as input, and swaps the input values if $b = 1$. However, this is performed in constant time, independent of the value of b. The swapping of two elliptic curve points therefore requires two separate executions of `cswap` for their two coordinates.

If one of these swaps is skipped or subject to a fault injection, this means that the respective X- and Z-coordinates of the two points no longer fit together as before if $b = 1$. An attacker could thus distinguish the cases $b = 0$ and $b = 1$, and therefore gain information. An easy mitigation is to replace each such pair of swaps by four swaps, such that each of them must be correctly executed in order to produce the valid resulting point. Thus, a fault injection would be detected by the methods from above, at a negligible cost.

Elligator. The Elligator map as used in [37] efficiently samples projective points $P_0 \in E_a[\pi + 1]$ and $P_1 \in E_a[\pi - 1]$ on the current curve E_a, where the cost is dominated by

one Legendre symbol calculation. However, if a fault is injected there, we can no longer guarantee that indeed $P_0 \in E_a[\pi + 1]$ and $P_1 \in E_a[\pi - 1]$. Deviating from this would mean that we compute isogenies with a wrong sign, and therefore obtain a wrong output curve, which can cause leakage.

We mitigate this by computing the Legendre symbol for both of these output points, and thereby making sure that $P_0 \in E_a[\pi + 1]$ and $P_1 \in E_a[\pi - 1]$ is satisfied.

Loop-abort faults. As mentioned in [37] and also applied to SIDH in [83], loop-abort faults can lead to a stopping of the algorithm, although not all required isogenies have been computed. In the CSIDH implementation featuring dummy isogenies, this can lead to leakage, since a correctly established shared secret in this case means that all the skipped isogenies would have been dummy computations. As usual, this can be prevented by using multiple counters, in order to make it much harder for the attacker to achieve an undetected loop-abort. In the CSIDH implementations from MCR and OAYT, there already are several counters, so it suffices to compare them before outputting the resulting curve, and thereby checking if one of them has been manipulated to abort the loop.

Decision bits. In many cases, decision bits must be set, such as b, which decides whether a real or dummy isogeny must be computed, or a decision bit that decides if P_0 or P_1 is used to compute the kernel generator for an isogeny. For our attack models, we could disregard these parts because of the low computational cost and the attacker's limited accuracy, but anyway we provide a simple countermeasure for leakage through an injected fault here. Since in our model the attacker only performs one fault injection, we can simply compute the respective bit twice, check if both computations obtained the same result, and output an error otherwise.

Remark 12. We note that also the dummy-free implementation of [37] offers attack surface; e.g. it is vulnerable to attacks aiming at the `cswap` function, Elligator, or some of the decision bit choices, which means that our discussion on these functions also applies to the dummy-free implementation.

5.7 Performance results

We implemented the countermeasures described in Section 5.6 into the implementation that was used in Section 5.5 to investigate the performance overhead of the proposed countermeasures. Like most previous work on CSIDH implementations, we entirely focus on the CSIDH-512 parameter set. The proposed attacks and countermeasures, however, apply to other parameter sets as well. The code was compiled with `arm-none-eabi-gcc`[16] Version 10.1.0. Table 5.2 contains the performance results without and with the countermeasures

[16] `https://developer.arm.com/`

Table 5.2: Performance results for one group action for the CSIDH-512 implementation on the ARM Cortex-M4 without and with countermeasures (CM). Averaged over 10 evaluations. Countermeasures for the theoretical twist attack are evaluated separately.

	STM32F407 (24 MHz) [clock cycles]		STM32F303 (7.4 MHz) [clock cycles]	
w/o CM	15 523M		15 721M	
w/ CM (w/o twist)	16 322M		16 751M	
overhead	804M	+5%	1 030M	+7%
w/ CM (w/ twist)	20 907M		21 486M	
overhead	5 384M	+35%	5 765M	+37%

implemented. We report cycle counts for both the STM32F303, which is the core on the 32-bit ChipWhisperer Lite, and the STM32F407 which is used in various post-quantum cryptography implementations in the literature and the benchmarking project PQM4 [97]. Our benchmarking code is primarily based on PQM4 and we follow the common practice of down-clocking the STM32F407 to 24 MHz to avoid flash wait states impacting the performance results. We report the average over 10 evaluations of the group action. The overhead of the presented countermeasures is 5% to 7%, and therefore relatively small compared to generic countermeasures like duplicating isogeny computations. The cost for the described twist attack countermeasures is slightly larger, namely 35% to 37% in total, including all other countermeasures. However, as described above, this attack is only of theoretical nature, which means that the former implementation suffices in practice. Note that the implementation of the arithmetic is a portable C implementation that was not heavily optimized for performance for this platform yet. It is therefore expected that all implementations can be further improved in terms of speed.

5.8 Conclusion

In this chapter we provided the first practical discussion on fault injection attacks on CSIDH. We introduced several attack models using low-budget equipment, simulated the impact of the attacks, and demonstrated the feasibility in a simplified experiment. We provided countermeasures against the proposed attacks and showed that the corresponding overhead falls well short of the twofold slowdown of dummy-free implementations.

However, there are several other topics to discuss in this context. Firstly, it is not clear whether similar countermeasures can be applied to the new asymptotically faster isogeny formulas from [13]. Furthermore, it remains an open task to investigate stronger attack models and different side-channel attacks.

Chapter 6

Threshold schemes from isogeny assumptions

6.1 Introduction

Threshold cryptography and secret sharing are large areas of interest in the cryptographic community since the late 1970s, when Shamir [136] and Blakley [24] published the first secret sharing schemes. In 1989, Desmedt and Frankel [66] constructed a practical threshold cryptosystem based on Shamir's secret sharing and ElGamal encryption [73].

The goal of a k-out-of-n, or (k, n)-threshold scheme is to split a secret key into multiple shares and distribute them among n parties, each party receiving one share. Then, for a certain threshold $k \leq n$, any k collaborating parties must be able to compute the cryptographic operation, e.g. decrypt or sign, without learning the secret key, while any set of less than k parties must be unable to do so.

After the publication of Desmedt and Frankel's scheme, several other threshold protocols were proposed; among others, a threshold variant of ElGamal signatures by Harn [90], a threshold DSA scheme by Gennaro, Jarecki, Krawczyk, and Rabin [85], and Desmedt and Frankel's and Shoup's threshold RSA signature schemes [67, 138]. More recently, applications of threshold schemes in the context of blockchains and cryptocurrencies led to a renewed interest in threshold ECDSA schemes [70, 84].

However, all of these schemes are either based on discrete logarithm or integer factorization problems, and are thus not quantum-resistant, since they fall prey to Shor's algorithm [137]. Only very recently, Cozzo and Smart [53] reviewed the post-quantum signature schemes that entered the second round of the NIST PQC standardization process [121] for threshold variants. Their main observation is that only the multivariate-based schemes LUOV [20] and Rainbow [69] allow for a natural threshold construction.

Another popular family of post-quantum schemes is provided by isogeny-based cryptography [94, 93]. While this family is not represented in the NIST PQC track for signatures, isogeny-based signatures have recently attracted much attention [59, 64, 19]. In this

chapter we introduce the first isogeny-based threshold encryption and signature schemes, based on Shamir's secret sharing.

Our schemes are simple adaptations of Desmedt and Frankel's and related schemes to the *Hard Homogeneous Spaces (HHS)* framework. This framework was introduced by Couveignes [51], to generalize both discrete logarithm and isogeny-based schemes. Encryption schemes for HHS were first proposed by Couveignes [51] and Rostovtsev and Stolbunov [133], then improved by De Feo, Kieffer and Smith [61], and eventually led to the development of CSIDH by Castryck, Lange, Martindale, Panny, and Renes [35].

The possibility of signature schemes based on HHS was first suggested by Couveignes [51] and Stolbunov [145, 146], although no instantiation was known until recently, when Beullens, Kleinjung, and Vercauteren introduced CSI-FiSh [19]. Before that, an alternative signature scheme based on a weaker notion of HHS, named SeaSign, was presented by De Feo and Galbraith [59].

Our Contributions. We introduce threshold variants of the Couveignes-Rostovtsev-Stolbunov encryption and signature schemes, based on Shamir's secret sharing. To make the results more easily accessible to non-experts, we first present our schemes in an abstract way, using the language of HHS, and only later we analyze their instantiation using CSIDH / CSI-FiSh.

The encryption scheme is a direct adaptation of [66]; the signature scheme is similar to threshold versions of Schnorr signatures [134]. Both schemes can only be proven secure in a *honest-but-curious* security model [28]; we skip the easy proof for the encryption scheme, and we focus on the more technical one for the signature scheme, which we prove secure in a *static corruptions* model, under a generalization of the Decision Diffie-Hellman Group Action (DDHA) assumption of Stolbunov.

We conclude with an analysis of the instantiations of the schemes based on isogeny graphs, in particular on the supersingular isogeny graphs used in CSIDH and CSI-FiSh.

We view this work as an initial step towards practical threshold schemes based on HHS and isogenies. Several technical improvements, such as better security properties and proofs, are necessary before these protocols can be considered truly practical. We discuss these issues at the end of this chapter.

Organization. Section 6.2 recalls basic facts on secret sharing, threshold cryptography, and HHS. Section 6.3 then introduces threshold encryption and signature schemes based on HHS, and reviews their security features. In Section 6.4 we give details about the instantiation of these threshold schemes using isogeny graphs. In Section 6.5 we conclude by summarizing open problems towards practical applications of our schemes, and review follow-up work that was published after the publication of this chapter's material in [63].

6.2 Preliminaries

We briefly recall two fundamental constructions in group-theoretic cryptography. The first, Shamir's secret sharing [136], lets a *dealer* split a secret s into n *shares*, so that any k shares are sufficient to reconstruct s; it is a basic primitive upon which several threshold protocols can be built.

The second, Couveignes' *Hard Homogeneous Spaces* (HHS) [51], is a general framework that abstracts some isogeny protocols, and that eventually inspired CSIDH [35]. Although most popular isogeny-based primitives are not, strictly speaking, instances of HHS, the protocols introduced in this work require an instance of an HHS in the strictest sense, and will thus be presented using that formalism.

6.2.1 Shamir's secret sharing & threshold cryptosystems

Shamir's scheme relies on polynomial interpolation to construct a k-out-of-n threshold secret sharing, for any pair of integers $k \leq n$.

Concretely, a prime $q > n$ is chosen, and the secret s is sampled from $\mathbb{Z}/q\mathbb{Z}$. To break the secret into shares, the dealer samples random coefficients $c_1, \ldots, c_{k-1} \in \mathbb{Z}/q\mathbb{Z}$ and forms the polynomial

$$f(x) = s + \sum_{i=1}^{k-1} c_i x^i;$$

then they form the shares $s_1 = f(1), \ldots, s_n = f(n)$ and distribute them to the n participants, denoted by $\mathcal{P}_1, \ldots, \mathcal{P}_n$. We shall call i the *identifier* of a participant $\mathcal{P}_i$, and s_i their *share*.

Any k participants, but no less, can reconstruct f using Lagrange's interpolation formula, and then recover s by evaluating f at 0. Explicitly, a set of participants $\mathcal{P}_i$, with indices taken from a set $S \subset \{1, \ldots, n\}$ of cardinality at least k, can recover the secret s in a single step through the formula

$$s = f(0) = \sum_{i \in S} f(i) \cdot \prod_{\substack{j \in S \\ j \neq i}} \frac{j}{j - i}.$$

Shamir's secret sharing enjoys *perfect* or *information theoretic* security, meaning that less than k shares provide no information on the secret. Indeed, assuming that $k - 1$ participants, w.l.o.g. $\mathcal{P}_1, \ldots, \mathcal{P}_{k-1}$, put their shares together, the map

$$(s, c_1, \ldots, c_{k-1}) \mapsto \big(f(0), f(1), \ldots, f(k-1)\big)$$

is, by Lagrange's formula, an isomorphism of $(\mathbb{Z}/q\mathbb{Z})$-vector spaces; hence, each tuple $\big(s = f(0), f(1), \ldots, f(k-1)\big)$ is equally likely to occur.

Threshold schemes. A major step towards practical threshold schemes based on Shamir's secret sharing was Desmedt and Frankel's threshold variant of ElGamal decryption [66]; a similar approach to design threshold signatures was proposed by Harn [90]. Many other threshold protocols follow a similar pattern, colloquially referred to as *secret sharing in the exponents*, that we are now going to briefly recall.

Let the secret $s \in \mathbb{Z}/q\mathbb{Z}$ and the shares s_i be distributed as above. Let G be a cyclic group of order q, and let g be a generator. Assuming that discrete logarithms are hard in G, the participants' goal is to compute the *shared key* g^s without letting anyone learn the secret s. We can again use Lagrange interpolation, but this time in the exponent:

$$g^s = g^{\sum s_i \prod \frac{j}{j-i}}.$$

To make this idea into a protocol, each party computes g^{s_i} from its share s_i, and sends it to all other parties. Given at least k shares s_i of the key with $i \in S$ and $\#S \geq k$, any party can then compute the shared key as

$$g^s = \prod_{i \in S} (g^{s_i})^{L^S_{0,i}},$$

where the exponents

$$L^S_{l,i} = \prod_{\substack{j \in S \\ j \neq i}} \frac{j-l}{j-i} \mod q \tag{6.1}$$

can be precomputed from public information.

If broadcasting the shares g^{s_i} to all participants is too expensive, an alternative is to send them to a central *combiner*, who is then in charge of computing g^s and finalizing the protocol. As we shall see later, this flexibility will be lost in our setting.

Secret sharing in rings. The proof of perfect security of Shamir's secret sharing scheme fundamentally relies on $\mathbb{Z}/q\mathbb{Z}$ being a field. For reasons that will become apparent later, we shall need to adapt the scheme to non-prime q, and thus to general rings of modular integers. This presents two problems: ensuring that no impossible inversions happen when computing the coefficients $L^S_{l,i}$ in Equation (6.1), and proving security in the more general setting. These obstacles are not difficult to overcome, as already highlighted in, e.g., RSA-based threshold schemes [138]; we briefly explain how this is done.

Impossible inversions arise during the reconstruction of the shared secret whenever one of the denominators $(j-i)$ in Lagrange's formula is not coprime to q. If q_1 is the smallest prime factor of q, then there can be at most q_1 distinct values modulo q_1; however, any identifier i congruent to 0 modulo q_1 must be prescribed, since otherwise $f(i) \bmod q_1$ would leak information on $s \bmod q_1$. Hence, at most $q_1 - 1$ participants can take part to Shamir's scheme in $\mathbb{Z}/q\mathbb{Z}$; for example, using $1, 2, \ldots, q_1 - 1$ as identifiers ensures that no difference of two of them shares a common factor with q.

Perfect security of the scheme is also achieved by restricting the identifiers to $1, 2, \ldots, q_1 - 1$, or any other set of integers distinct and non-zero modulo all divisors of q, thus restricting the number of participants to $n < q_1$. We formally prove this below.

Proposition 5. *Let q be an integer with prime factorization $q = \prod q_i^{e_i}$. Assume q_1 is the smallest of the prime factors, let $k \leq n < q_1$, and sample $s, c_1, \ldots, c_{k-1} \in \mathbb{Z}/q\mathbb{Z}$ uniformly at random. Let*

$$f(x) = s + \sum_{i=1}^{k-1} c_i x^i$$

and let $x_1, \ldots x_{k-1} \in \mathbb{Z}/q\mathbb{Z}$ be distinct and non-zero modulo all q_i. Associate a random variable S to s, and random variables Y_i to each $f(x_i)$.

The random variables $S, Y_1, \ldots Y_{k-1}$ are independent; in particular Shamir's (k, n)-secret sharing scheme over $\mathbb{Z}/q\mathbb{Z}$ is perfectly secure, in the sense that, given the shares $f(x_1), \ldots, f(x_{k-1})$, every secret s is equally likely to have originated them.

Proof. Consider the map

$$\rho \ : \ (s, c_1, \ldots, c_{k-1}) \mapsto \big(f(0), f(x_1), \ldots, f(x_{k-1})\big);$$

since all $x_i \bmod q_j$ are distinct and non-zero, its reduction modulo q_j is an isomorphism of $\mathbb{Z}/q_j\mathbb{Z}$-vector spaces; thus, by the Chinese Remainder Theorem, ρ is an isomorphism of $\mathbb{Z}/q\mathbb{Z}$-modules.

Introducing random variables Y_0 for $f(0)$ and C_i for the c_i's, we have that

$$\begin{aligned} P\{Y_0 = f(0), Y_1 = f(x_1), \ldots, Y_{k-1} = f(x_{k-1})\} \\ = P\{S = s, C_1 = c_1, \ldots, C_{k-1} = c_{k-1}\} = q^{-k}, \end{aligned}$$

from which we deduce that $P\{Y_i = f(x_i)\} = q^{-1}$. In particular, since $s = f(0)$,

$$\begin{aligned} P\{S = s, Y_1 = f(x_1), \ldots, Y_{k-1} = f(x_{k-1})\} \\ = P\{S = s\} \cdot P\{Y_1 = f(x_1)\} \cdots P\{Y_{k-1} = f(x_{k-1})\} \end{aligned}$$

for any $s, f(x_1), \ldots, f(x_{k-1})$, implying that S and the Y_i's are independent. □

6.2.2 Hard homogeneous spaces

Hard Homogeneous Spaces (HHS) were introduced by Couveignes in [51] as a generalization of Diffie-Hellman schemes. A *principal homogeneous space*, or *$\mathcal{G}$-torsor* is a set $\mathcal{E}$ endowed with a faithful and transitive group action by a group $\mathcal{G}$.[17] In other words, it is defined by a mapping

$$\begin{aligned} \mathcal{G} \times \mathcal{E} &\to \mathcal{E}, \\ \mathfrak{g} * E &= E', \end{aligned}$$

satisfying the following properties:

[17] The reader will excuse our extravagant font choices for set and group elements: our goal is to be consistent with the notation used in Section 6.4 for isogeny-based HHS.

- *Compatibility:* $\mathfrak{g}' * (\mathfrak{g} * E) = (\mathfrak{g}'\mathfrak{g}) * E$ for any $\mathfrak{g}, \mathfrak{g}' \in \mathcal{G}$ and $E \in \mathcal{E}$;
- *Identity:* $\mathfrak{e} * E = E$ if and only if $\mathfrak{e} \in \mathcal{G}$ is the identity element;
- *Transitivity:* for any $E, E' \in \mathcal{E}$ there exists a unique $\mathfrak{g} \in \mathcal{G}$ such that $\mathfrak{g} * E = E'$;

In particular, if $\mathcal{G}$ is finite, these axioms imply that $\#\mathcal{G} = \#\mathcal{E}$.

Couveignes defines a HHS as a finite principal homogeneous space with some additional algorithmic properties. He requires that the following problems can be solved efficiently (e.g., in polynomial time):

- *Group operations:* decide whether a string $\mathfrak{g}$ represents an element of $\mathcal{G}$, decide whether $\mathfrak{g} = \mathfrak{g}'$, compute $\mathfrak{g}^{-1}$ and $\mathfrak{g}\mathfrak{g}'$;
- *Sampling:* sample uniformly random elements from $\mathcal{G}$;
- *Membership:* decide whether a string E represents an element of $\mathcal{E}$, decide whether $E = E'$;
- *Action:* Given $\mathfrak{g}$ and E, compute $\mathfrak{g} * E$.

Furthermore, the following problems should be hard (e.g., not known to be solvable in polynomial time):

- *Vectorization:* Given $E, E' \in \mathcal{E}$, find $\mathfrak{g} \in \mathcal{G}$ such that $\mathfrak{g} * E = E'$;
- *Parallelization:* Given $E, E', F \in \mathcal{E}$, such that $E' = \mathfrak{g} * E$, find $F' = \mathfrak{g} * F$.

As a simple example, let $\mathcal{E}$ be a group of prime order q, then $\mathcal{G} = (\mathbb{Z}/q\mathbb{Z})^\times$ acts on $\mathcal{E} \setminus \{1\}$ by $a * g = g^a$. In this case, the Vectorization problem is the discrete logarithm problem in $\mathcal{E}$, and the Parallelization problem is the Computational Diffie–Hellman problem. Hence any discrete logarithm group is also a HHS.

Couveignes' original proposal used as HHS sets of ordinary elliptic curves over finite fields, with complex multiplication by a quadratic imaginary order $\mathcal{O}$; indeed, these are torsors for the class group $\mathrm{cl}(\mathcal{O})$, and the Vectorization and Parallelization problems are not known to be easily solvable. Based on this HHS, he defined key exchange as a straightforward generalization of the Diffie-Hellman protocol, and he also sketched an interactive identification scheme.

However, Couveignes' proposal presents several difficulties, as neither the group action nor random sampling are known to be easily computable. Independently from Couveignes, Rostovtsev and Stolbunov [133, 145] proposed a key exchange scheme based on the same group action, but with a different representation of elements of $\mathrm{cl}(\mathcal{O})$. This proposal had the benefit of making key exchange feasible, if not practical, and subsequent research [61] eventually led to the development of CSIDH [35], an efficient key exchange scheme based on the action of a quadratic class group on a set of supersingular curves.

Nevertheless, none of these constructions satisfies exactly the axioms of a HHS, since, for example, the cost of evaluating $\mathfrak{g} * E$ in CSIDH is in the worst case exponential in the size of $\mathfrak{g}$. While every group element has an equivalent representation that permits to efficiently evaluate the action, computing such a representation is difficult in general. This is not a problem for key-exchange schemes based on CSIDH, but, for example, it makes identification and signature schemes more involved and less efficient than what Couveignes had originally envisioned [59, 64].

The roadblock in all these constructions is the fact that the structure of the class group $\mathrm{cl}(\mathcal{O})$ is unknown, and it is thus impossible to have a unique representation for its elements. The best algorithm for computing the class group structure runs in sub-exponential time, and is thus neither practical nor scalable; nevertheless the application to isogeny-based signatures motivated Beullens, Kleinjung, and Vercauteren [19] to run an intensive computation for the CSIDH-512 parameter set, which allowed them to construct CSI-FiSh, an efficient isogeny-based signature scheme.

Currently, CSI-FiSh is the only known instance of HHS based on isogenies: group elements have unique representation, the group action can be evaluated efficiently, and the Vectorization and Parallelization problems are believed to be hard, both classically and quantumly. Unfortunately, parameter generation requires exponential time in the security parameter, thus CSI-FiSh is a HHS only in a practical sense for a specific security level, but not in the asymptotic sense.

In the next sections we are going to introduce threshold schemes based on HHS; then we will give more details on CSI-FiSh, and look at how the threshold schemes can be instantiated with it.

6.3 Threshold schemes from HHS

We now present threshold schemes based on Hard Homogeneous Spaces.

Let a group $\mathcal{G}$ and a set $\mathcal{E}$ be given, such that $\mathcal{G}$ acts faithfully and transitively on $\mathcal{E}$ and the HHS axioms are satisfied. We are going to require an additional property: that an element $\mathfrak{g} \in \mathcal{G}$ of order q is known, and we shall write q_1 for the smallest prime divisor of q. In particular, these hypotheses imply that there is an efficiently computable embedding $\mathbb{Z}/q\mathbb{Z} \hookrightarrow \mathcal{G}$ defined by $a \mapsto \mathfrak{g}^a$, which we are going to exploit to embed Shamir's secret sharing in the HHS.

Notation. From now on we will use capital letters $E, F, \ldots$ to denote elements of the HHS $\mathcal{E}$, and gothic letters $\mathfrak{a}, \mathfrak{b}, \mathfrak{g}, \ldots$ to denote elements of the group $\mathcal{G}$. Following [19], it will be convenient to see $\mathbb{Z}/q\mathbb{Z}$ as acting directly on $\mathcal{E}$: we will write $[a]$ for $\mathfrak{g}^a$, and $[a]E$ for $\mathfrak{g}^a * E$, where $\mathfrak{g}$ is the distinguished element of order q in $\mathcal{G}$.[18] Be wary that under this notation $[a][b]E = [a+b]E$.

[18] Note that this action is only transitive if $\mathfrak{g}$ generates $\mathcal{G}$.

Remark 13. The additional hypothesis excludes, in particular, HHS of unknown order, such as CSIDH (outside of the parameter set shared with CSI-FiSh).

Note that, assuming the factorization of q is known, given any element of $\mathcal{G}$ it is easy to test whether it is of order q. Nevertheless, in some instances it may be difficult to decide whether an element $\mathfrak{g}' \in \mathcal{G}$ belongs to $\langle \mathfrak{g} \rangle$; this may happen, for example, if $\mathcal{G} \simeq (\mathbb{Z}/q\mathbb{Z})^2$. This will not impact the protocols we define here, but is an important property to consider when designing threshold protocols in the general HHS setting. At any rate, for instantiations based on CSI-FiSh it is always easy to test membership of $\langle \mathfrak{g} \rangle$.

On the other hand, unless $\mathcal{G} = \langle \mathfrak{g} \rangle$, it is a well known hard problem (exponential in $\log q$) to decide whether given $E, E' \in \mathcal{E}$ there exists $a \in \mathbb{Z}/q\mathbb{Z}$ such that $E' = [a]E$. Indeed, a generic solution to this problem would imply an efficient generic algorithm for solving many instances of discrete logarithms [35].

We now describe a distributed algorithm to compute the group action of $\langle \mathfrak{g} \rangle$ on $\mathcal{E}$ in a threshold manner, and explain how it impacts the communication structure of threshold protocols. Then we present two simple threshold protocols, a KEM and a signature, directly adapted from their non-threshold counterparts.

6.3.1 Threshold group action

Like in Section 6.2, we assume that the participants $\mathcal{P}_1, \mathcal{P}_2, \dots$ possess shares $s_i = f(i)$ of a secret $s \in \mathbb{Z}/q\mathbb{Z}$; their goal is to evaluate the group action $[s]E_0$ for any given $E_0 \in \mathcal{E}$, without communicating their shares s_i.

Let $S \subset \{1, \dots, n\}$ be a set of cardinality at least k, and recall the definition of the Lagrange coefficients in Equation (6.1):

$$L_{l,i}^S = \prod_{\substack{j \in S \\ j \neq i}} \frac{j-l}{j-i} \mod q.$$

Then the participants $\mathcal{P}_i$ for $i \in S$ determine the shared secret by $s = \sum_{i \in S} s_i \cdot L_{0,i}^S$. For the sake of simplicity, we will assume that $S = \{1, \dots, k\}$.

The participants coordinate as follows. First, E_0 is sent to $\mathcal{P}_1$, who starts by computing

$$E_1 = \left[s_1 \cdot L_{0,1}^S \right] E_0.$$

The resulting E_1 is passed on to $\mathcal{P}_2$, who continues by computing

$$E_2 = \left[s_2 \cdot L_{0,2}^S \right] E_1 = \left[s_2 \cdot L_{0,2}^S + s_1 \cdot L_{0,1}^S \right] E_0.$$

This procedure repeats analogously for the parties $\mathcal{P}_3, \dots, \mathcal{P}_{k-1}$, and at last $\mathcal{P}_k$ can compute

$$E_k = \left[s_k \cdot L_{0,k}^S \right] E_{k-1} = \left[\sum_{i \in S} s_i \cdot L_{0,i}^S \right] E_0 = [s]E_0.$$

Algorithm 9: Threshold variant of the group action computation.

Input : $E_0 \in \mathcal{E}$, set of participants S.
Output: $[s]E_0$.

Set $E \leftarrow E_0$.
foreach $i \in S$ **do**
    If $E \notin \mathcal{E}$, participant $\mathcal{P}_i$ outputs $\perp$ and the algorithm stops.
    Participant $\mathcal{P}_i$ outputs $E \leftarrow \left[s_i \cdot L_{0,i}^S\right] E$.
return E.

Communication structure. Comparing the algorithm to classical threshold Diffie-Hellman protocols as in Section 6.2.1, it is obvious that there are differences in their structures. There, each party $\mathcal{P}_i$ computes $g_i = g^{s_i}$ from its secret share s_i and a common generator g. Anyone can then compute $g_i^{L_{0,i}^S}$ for each $i \in S$, and multiply the results to obtain g^s.

In our HHS setting, the situation is different. First, $\left[s_i \cdot L_{0,i}^S\right] E$ cannot be computed from the knowledge of $[s_i]E$ and $L_{0,i}^S$, thus only $\mathcal{P}_i$ can compute it. Consequently, each participant has to know in advance the set S of parties taking part to the computation, in order to apply $L_{0,i}^S$.

Further, it is not possible to introduce a combiner, who could proceed as in the classical case by receiving the different $\left[s_i \cdot L_{0,i}^S\right] E_0$ and combining them to obtain $[s]E_0$, since in general the set $\mathcal{E}$ is not equipped with a compatible group operation $\mathcal{E} \times \mathcal{E} \to \mathcal{E}$. Therefore, it is necessary to adopt a sequential round-robin communication structure:

$$\xrightarrow{E_0,S} \mathcal{P}_1 \xrightarrow{E_1,S} \mathcal{P}_2 \xrightarrow{E_2,S} \dots \xrightarrow{E_{k-1},S} \mathcal{P}_k \xrightarrow{[s]E_0} .$$

Note that the order of the $\mathcal{P}_i$ can be changed without affecting the final result.

However, this means that $\mathcal{P}_k$ is the only party who ends up knowing the result of the group action. If a cryptographic protocol needs to handle this element secretly, our algorithm is only suitable for situations where only one participant is required to know the secret result. Algorithm 9 summarizes the described approach in the general case.

In a different setting where all participants are required to secretly know the final result, several modifications are possible. For example, when encrypted channels between the participants exist, the last participant can simply distribute through them the resulting $[s]E_0$.

Alternatively, k parallel executions of Algorithm 9, each arranging the participants in a different order, let all participants know the final result. The cost of this modification is rather high: $O(k^2)$ elements of $\mathcal{E}$ need to be transmitted, and $O(k^2)$ group actions evaluated. This can be improved to $O(k \log k)$ transmitted elements of $\mathcal{E}$ (but still $O(k^2)$ group actions) using a binary splitting strategy.

Remark 14. Algorithm 9 does nothing to prevent corrupted participants from leading to an incorrect output. While threshold schemes based on discrete logarithms can often detect and correct malicious behavior (using, e.g., error correcting codes [85]), this is more difficult for HHS. Indeed, there seems to be no way for a participant to verify the previous participant's output in Algorithm 9, outside of generic zero-knowledge techniques.

6.3.2 Threshold HHS ElGamal decryption

The first application we present for our threshold group action is *threshold decryption*, a direct adaptation of [66].

Inspired by the classical ElGamal encryption scheme [73], a PKE protocol in the HHS settings was first introduced by Stolbunov [133, 145, 146]. We briefly recall it here, using the terminology of KEMs.

Public parameters: A HHS $(\mathcal{E}, \mathcal{G})$, a *starting element* $E_0 \in \mathcal{E}$, and a hash function H from $\mathcal{E}$ to $\{0,1\}^\lambda$.

Keygen: Sample a secret key $\mathfrak{a} \in \mathcal{G}$, output $\mathfrak{a}$ and the public key $E_a = \mathfrak{a} * E_0$.

Encaps: Sample $\mathfrak{b} \in \mathcal{G}$, output $K = H(\mathfrak{b} * E_a)$ and $E_b = \mathfrak{b} * E_0$.

Decaps: Given E_b, if $E_b \in \mathcal{E}$ output $K = H(\mathfrak{a} * E_b)$, otherwise output $\perp$.

The **Decaps** routine is easily adapted into a threshold algorithm requiring k participants to collaborate in order to recover the decryption key K. This also requires modifying **Keygen**, which must now be executed by a trusted dealer and integrate Shamir's secret sharing.

Public parameters: A HHS $(\mathcal{E}, \mathcal{G})$ with a distinguished element $\mathfrak{g} \in \mathcal{G}$ of order q, a *starting element* $E_0 \in \mathcal{E}$, and a hash function H from $\mathcal{E}$ to $\{0,1\}^\lambda$.

Keygen:

- Sample a secret $s \in \mathbb{Z}/q\mathbb{Z}$ and generate shares $s_i \in \mathbb{Z}/q\mathbb{Z}$ using Shamir's secret sharing;
- Distribute privately s_i to participant $\mathcal{P}_i$;
- Output public key $E_a = [s]E_0$.

Encaps: Sample $\mathfrak{b} \in \mathcal{G}$, output $K = H(\mathfrak{b} * E_a)$ and $E_b = \mathfrak{b} * E_0$.

Decaps: Given E_b and a set S of participants, $\#S \geq k$, run Algorithm 9 to compute $E = [s]E_b$; output $\perp$ if the algorithm returns $\perp$, otherwise output $K = H(E)$.

The asymmetry of the scheme will not be lost on the reader: while the shared secret for the threshold group is restricted to be in $\langle \mathfrak{g} \rangle$, there are no restrictions for **Encaps**. Although it would be completely possible (maybe even desirable for practical reasons) to restrict secrets to $\langle \mathfrak{g} \rangle$ also in the encapsulation, we do not do so because there is no known way for decapsulation to test whether E_b has been generated this way.

It is clear that this scheme achieves the stated goal of threshold decryption: upon receiving a ciphertext, at least k participants must agree to decrypt in order to recover the key K; only the last participant in the chain learns K. If less than k participants agree to decrypt, the key K cannot be recovered; however this security property is only guaranteed when all participants behave honestly.

When allowing for corruptions, the scheme immediately becomes broken. Indeed in Algorithm 9, when a participant beyond the first receives an input, they are unable to link it to the ciphertext E_b. This makes it possible to trick an unwilling participant $\mathcal{P}$ into helping decrypt a message: let c be such a message, a group of $k-1$ participants only has to wait for a message c' that $\mathcal{P}$ is willing to decrypt; when $\mathcal{P}$ agrees, they submit to it an intermediate value of a computation for c, which $\mathcal{P}$ is unable to distinguish from one for c'. Contrast this to the original ElGamal threshold decryption of Desmedt and Frankel [66], where each participant performs its computation directly on the input.

Because of this, the security of the protocol can only be proven in a *honest-but-curious* model. We skip the easy security proof, and leave the search for more refined threshold decryption protocols for future work.

6.3.3 Threshold signatures

An identification scheme in the HHS framework was first sketched by Couveignes [51]; in his PhD thesis [146] Stolbunov also suggested applying the Fiat-Shamir transform [76] to it to obtain a signature scheme. Nevertheless these schemes stood out of reach until recently, when the class group computation for CSIDH-512 was completed [19]; CSI-FiSh is effectively Stolbunov's scheme, combined with optimizations introduced in SeaSign [59].

CSI-FiSh and its ancestors can be easily adapted into threshold protocols. We start by recalling the basic interactive zero-knowledge identification scheme: a prover Peggy wants to convince a verifier Vic that she knows a secret element $\mathfrak{a} \in \mathcal{G}$ such that $E_a = \mathfrak{a} * E_0$. They proceed as follows:

- Peggy samples a random $\mathfrak{b} \in \mathcal{G}$ and commits to $E_b = \mathfrak{b} * E_0$.
- Vic challenges with a random bit $c \in \{0, 1\}$.
- If $c = 0$, Peggy replies with $\mathfrak{z} = \mathfrak{b}$; otherwise she replies with $\mathfrak{z} = \mathfrak{b}\mathfrak{a}^{-1}$.
- If $c = 0$, Vic verifies that $\mathfrak{z} * E_0 = E_b$; otherwise, he verifies that $\mathfrak{z} * E_a = E_b$.

It is immediately seen that the scheme is correct, thanks to the properties of homogeneous spaces, and that it has soundness $1/2$. For the zero-knowledge property, it is crucial that elements in $\mathcal{G}$ can be sampled uniformly, and that they have unique representation. See [146, 59, 19] for detailed proofs.

We now adapt this scheme into a threshold signature by applying the Fiat-Shamir transform and Shamir's secret sharing as before.

We let again $(\mathcal{E}, \mathcal{G})$ be a HHS with a distinguished element $\mathfrak{g}$ of order q, we fix a starting element $E_0 \in \mathcal{E}$, and a hash function $H : \{0,1\}^* \to \{0,1\}^\lambda$. We assume that a trusted dealer has sampled a random secret $s \in \mathbb{Z}/q\mathbb{Z}$, securely distributed shares s_i to the participants $\mathcal{P}_i$, and published the public key $E_s = [s]E_0$.

Here is a sketch of how participants $\mathcal{P}_1, \ldots, \mathcal{P}_k$ can cooperate to sign a message m:

- In the commitment phase, the participants collaborate to produce a random element $[b]E_0$ in a way similar to Algorithm 9, by producing each a random value $b_i \in \mathbb{Z}/q\mathbb{Z}$ and evaluating $E_i = [b_i]E_{i-1}$.
- Once $E_k = [b]E_0$ is computed, the challenge bit c is obtained from the hash $H(E_k, m)$.
- If $c = 0$, each $\mathcal{P}_i$ outputs $z_i = b_i$, else each $\mathcal{P}_i$ outputs $z_i = b_i - s_i \cdot L^S_{0,i}$.
- The signature is $(c, z = \sum z_i)$.

To verify the signature it suffices to check that $H([z]E_0, m) = 0 \ldots$, if $c = 0$, or that $H([z]E_s, m) = 1 \ldots$, if $c = 1$. Of course, this sketch must be repeated λ times, in order to ensure the appropriate level of security.

The complete signing algorithm is summarized in Algorithm 10. As presented there, it is rather inefficient in terms of signature size and signing/verification time. However, all the key/signature size compromises presented in CSI-FiSh [19] are compatible with our threshold adaptation, and would produce a more efficient signature scheme. The details are left to the reader.

Security analysis

We conclude with a study of the security of the threshold signature scheme. Like the other schemes presented here, it is only secure against (static) honest-but-curious adversaries; however the security proof is more technical, and we give it in more detail. Since our threshold signature has the same public key and produces the same signatures as the Stolbunov/CSI-FiSh non-threshold scheme, we are able to use Gennaro et al.'s security model [85], with the appropriate modifications to handle a trusted dealer. In a nutshell, security in this model is proven by showing that the transcript of the threshold protocol can be simulated given only the signature, even in presence of up to $k-1$ corrupted participants; then, security follows from the unforgeability of the non-threshold signature scheme. We start with a brief description of the model.

Algorithm 10: Threshold HHS signature.

Input : Message m, participant set S.
Output: A signature on m.

Set $(E_1^0, \ldots, E_\lambda^0) \leftarrow (E_0, \ldots, E_0)$.
Let $k \leftarrow 0$.
foreach $i \in S$ **do**
    Let $k \leftarrow k + 1$.
    foreach $1 \leq j \leq \lambda$ **do**
        If $E_j \notin \mathcal{E}$, participant $\mathcal{P}_i$ outputs $\perp$ and aborts the protocol.
        $\mathcal{P}_i$ samples $b_{i,j} \in \mathbb{Z}/q\mathbb{Z}$ uniformly at random.
        $\mathcal{P}_i$ outputs $E_j^k \leftarrow [b_{i,j}]E_j^{k-1}$.
Let $c_1 \cdots c_\lambda \leftarrow H(E_1^k, \ldots, E_\lambda^k, m)$.
foreach $i \in S$ **do**
    foreach $1 \leq j \leq \lambda$ **do**
        if $c_j = 0$ **then**
            $\mathcal{P}_i$ outputs $z_{i,j} = b_{i,j}$.
        else
            $\mathcal{P}_i$ outputs $z_{i,j} = b_{i,j} - s_i \cdot L_{0,i}^S$.
foreach $1 \leq j \leq \lambda$ **do**
    Let $z_j = \sum_{i \in S} z_{i,j}$.
return the signature $(c_1 \cdots c_\lambda, z_1, \ldots, z_\lambda)$.

Communication model. We assume the n parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$ have access to a broadcast channel they use to exchange messages when executing the signature protocol. On top of that, each participant has access to a private channel with the trusted dealer $\mathcal{T}$, that they use to receive the secret shares.

The adversary. We consider a *static honest-but-curious* adversary, i.e., one that chooses up to $k - 1$ players to corrupt at the beginning of the unforgeability game, and then observes all their communications, including the secret shares received from the dealer; other than that, all parties strictly follow the protocol. In the literature, this type of adversary is often also called *semi-honest* or *passive*.

The *view* of an adversary is the probability distribution on the transcript of all the information seen by it during the protocol execution: this includes secret shares, the message m to sign, the messages received from other parties, and the resulting signature.

Unforgeability. A threshold signature scheme is *unforgeable* if no polynomial-time adversary $\mathcal{A}$ can produce a signature for a previously unsigned message m, given the view of $\mathcal{A}$ for adaptively chosen messages $m_1, ..., m_Q$. This definition is analogous to the usual notion of UF-CMA. In other words, this means that $\mathcal{A}$ does not learn enough information from transcripts of protocol executions to forge a valid signature.

Simulatability. Gennaro et al. proved that a threshold signature scheme is unforgeable if the underlying signature scheme is, and the threshold scheme is *simulatable*. This is defined as there being a polynomial-time *simulator* $\mathcal{S}$ that takes as input a message m, the public key E_s, a valid signature on m, and the shares of the corrupted participants, and outputs transcripts that are computationally indistinguishable from the view of the adversary. Intuitively, this means that the adversary gains no more information from seeing the transcript, than from the signature alone.

The trusted dealer. Unlike the threshold scheme of Gennaro et al., our signature scheme does not feature a distributed key generation. We thus adopt a hybrid model, where the generation of the trusted shares is modeled by an ideal functionality $\mathcal{F}_{\mathcal{T}}$, that executes Shamir's secret sharing, publishes the public key, and distributes the secret shares to each participant through the private channel.

In particular, the adversary is not able to tamper with $\mathcal{F}_{\mathcal{T}}$, and the distinguisher has no knowledge of the master secret generated by it.

We will prove simulatability under a new assumption that we call *Power-DDHA*. This decision version of the *Scalar-HHS* problem of Felderhoff [75] is a generalization of the Decision Diffie–Hellman Group Action (DDHA) introduced by Stolbunov [145], and is related to the *P*-DDH assumption introduced by Kiltz for discrete logarithm groups [100].

Problem 1 (Power-DDHA problem)**.** *Let $(\mathcal{E}, \mathcal{G})$ be a HHS. Let $E \in \mathcal{E}$ and $1 < a < \#\mathcal{G}$ an integer; let $\mathfrak{s}$ be a uniformly random element in $\mathcal{G}$. The* a-Power-DDHA problem *is: given $(a, E, \mathfrak{s} * E, F)$, where $F \in \mathcal{E}$ is an element, either sampled from the uniform distribution on $\mathcal{E}$, or $F = \mathfrak{s}^a * E$, decide from which distribution F is drawn.*

Remark 15. The special case of (-1)-Power-DDHA where the HHS is instantiated with a graph of $\mathbb{F}_p$-isomorphism classes of supersingular curves, and E is the special curve $E : y^2 = x^3 + x$, is known to be solvable efficiently. Other "special" curves in the graph also enjoy this property, see [36].

This obstacle is easy, but tedious, to circumvent in the proof of the next theorem. We leave the details to the reader.

Felderhoff proved that the search version of Power-DDHA (Scalar-HHS) is equivalent to Parallelization whenever the order of $\mathcal{G}$ is known and odd [75]. We also recall the formal definition of the Vectorization problem, also known as *Group Action Inverse Problem* [145].

Problem 2 (GAIP). *Let $(\mathcal{E}, \mathcal{G})$ be a HHS, let E, F be uniformly random elements of $\mathcal{E}$. The* Group Action Inverse Problem *asks to compute $\mathfrak{a} \in \mathcal{G}$ such that $E = \mathfrak{a} * F$.*

It is clear that GAIP is harder than Power-DDHA: given a GAIP solver one can simply apply it to $(E, \mathfrak{s} * E)$, and then use the answer to solve Power-DDHA.

Theorem 5. *Under the Power-DDHA assumption, the signature scheme of Algorithm 10 is simulatable.*

Stolbunov's signature scheme is proven secure in the ROM under GAIP (see [146, 59, 19]); since GAIP is harder than Power-DDHA, we immediately get the following theorem.

Corollary 1. *Under the Power-DDHA assumption, the signature scheme of Algorithm 10 is unforgeable, when the hash function H is modeled as a random oracle.*

Proof of Theorem 5. Observe that the public key $E_s = [s]E_0$ uniquely determines s; but that, together with the $k-1$ corrupted shares, uniquely determines the polynomial f in Shamir's secret sharing, and thus all other shares. We shall denote by $s_1, \ldots, s_n$ these uniquely determined shares, note however that the simulator only knows the corrupted ones.

Let $(c_1 \cdots c_\lambda, z_1, \ldots, z_\lambda)$ be a signature, and let S be the set of k signers (who signs a given message is decided by the adversary). To simulate a transcript, the simulator draws integers $z_{i_1,j}, \ldots, z_{i_{k-1},j} \in \mathbb{Z}/q\mathbb{Z}$ at random, for any $1 \leq j \leq \lambda$, and sets $z_{i_k,j} = z_j - z_{i_1,j} - \cdots - z_{i_{k-1},j}$. Since z_j is uniformly distributed, it is clear all $z_{i,j}$ also are. These values make the second part of the transcript (lines 12–15 in Algorithm 10).

To complete the transcript, the simulator now needs to output commitments $E_j^{k_i}$ (line 8), where for each $i \in S$ we denote by $1 \leq k_i < k$ the position of i in S. We start with the case where S contains only one uncorrupted participant, which can be simulated perfectly.

If $c_j = 0$ the simulator simply sets

$$E_j^{k_i} = \left[b_{k_1,j} + b_{k_2,j} + \cdots + b_{k_i,j}\right] E_0 = \left[z_{k_1,j} + z_{k_2,j} + \cdots + z_{k_i,j}\right] E_0,$$

as in Algorithm 10. If $c_j = 1$, define the sequence

$$\begin{aligned} E_s^0 &= E_0, \\ E_s^{k_i} &= \left[s_i \cdot L_{0,i}^S\right] E_s^{k_i - 1}, \end{aligned}$$

so that $E_s = E_s^k$. The simulator can compute all curves $E_s^{k_i}$ as follows: assume the uncorrupted participant $\mathcal{P}_i$ is in position k_i in S, for any $k' < k_i$ it computes $E_s^{k'}$ *directly*:

$$E_s^{k'} = \left[\sum_{i \in S, k_i \leq k'} s_i \cdot L_{0,i}^S\right] E_0,$$

whereas for all $k' \geq k_i$ it computes it *backwards* from E_s:

$$E_s^{k'} = \left[\sum_{i \in S, k_i > k'} -s_i \cdot L_{0,i}^S\right] E_s.$$

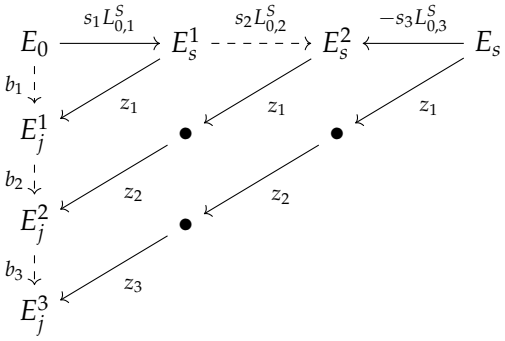


Figure 6.1: Recomputation of $E_j^{k_i}$ given $z_{i,j}$.

Then, the commitments are computed as

$$E_j^{k_i} = \left[z_{k_1,j} + z_{k_2,j} + \cdots + z_{k_i,j}\right] E_s^{k_i},$$

which is immediately seen as being the same as in Algorithm 10, thanks to $b_{i,j} = z_{i,j} + s_i \cdot L^S_{0,i}$. An example of this computation where participants $\mathcal{P}_1$ and $\mathcal{P}_3$ are corrupted and participant $\mathcal{P}_2$ is not is pictured in Figure 6.1.

Because all the choices are uniquely determined once the values $z_{i,j}$ have been chosen, it is clear that this transcript is perfectly indistinguishable from a real one, even for a computationally unbounded distinguisher.

We are left with the case where the set S contains more than one uncorrupted participant; in this case, we will resort to random sampling. For simplicity, we will assume that sets S are always sorted in increasing order, so that the relative order of the participants' actions does not change from one signature to another.

Like above, we start one *direct* chain from E_0, and one *backwards* from E_s; both chains stop when they encounter an uncorrupted participant $\mathcal{P}_i$. Now, let $E_s^{k_i - 1}$ be the last curve in the *direct* chain, we set the next curve $E_s^{k_i} = [r_i]E_0$, where r_i is sampled uniformly from $\mathbb{Z}/q\mathbb{Z}$. We also store r_i in association with S, and keep it for reuse the next time the adversary queries for the set S.

We continue the *direct* chain from $E_s^{k_i}$, either using the knowledge of $s_i \cdot L^S_{0,i}$ for corrupted participants, or sampling a random r_i for uncorrupted ones; we stop when we meet the *backwards* chain. An example of this process is pictured below:

$$E_0 \xrightarrow{\mathbf{r_1}} \mathbf{E_s^1} \xrightarrow{s_2 L^S_{0,2}} E_s^2 \xrightarrow{\mathbf{r_3}} \mathbf{E_s^3} \xrightarrow{r_4} E_s^4 \xrightarrow{s_4 L^S_{0,4}} E_s$$

we write in bold data that is obtained through random sampling; the value r_4 is implicitly determined by the other four values. After we have determined this data, we compute the $E_j^{k_i}$'s and complete the transcript as before.

Now, this transcript is no longer indistinguishable from the real view of the adversary, however we argue that it still is computationally indistinguishable assuming Power-DDHA. Indeed, when $c_j = 1$, the distinguisher is able to recover $E_s^{k_i}$ from $E_j^{k_i}$ as $E_s^{k_i} = \left[-z_{k_1,j} - z_{k_2,j} - \cdots - z_{k_i,j}\right] E_j^{k_i}$. This means that the distinguisher will collect many pairs of the form $\left(E, \left[s_i \cdot L_{0,i}^{S}\right] E\right)$ (in queries where $\mathcal{P}_i$ is the only uncorrupted participant in S), and many others of the form $(E', [r_i]E')$ (where the expected relation would be $\left(E', \left[s_i \cdot L_{0,i}^{S'}\right] E'\right)$ instead). In general, it will be the case that $E' = [b]E$ for some $b \in \mathbb{Z}/q\mathbb{Z}$ not necessarily known to the distinguisher; however, by subtracting known factors coming from corrupted players, the distinguisher can reduce to a distinguishing problem between tuples $([\sum s_i']E_0, [\sum s_i' a_i]E_0)$ and $([\sum s_i']E_0, [r]E_0)$, where the s_i' are unknowns related to uncorrupted shares s_i, the a_i are known (and possibly 0), and r is random. This is an instance of a problem more general than Power-DDHA, and is thus at least as hard as Power-DDHA.

Hence, assuming Power-DDHA is hard, no polynomial time algorithm can distinguish between the simulated transcript and the real interaction, thus proving that the threshold scheme is simulatable. □

Remark 16. It is evident from the proof that the security of the (n, n)-threshold signature scheme can be proven without assuming Power-DDHA. The appearance of this surprising assumption seems an artifact related to the limitations of the HHS framework; indeed, the analogous scheme based on discrete logarithms can be proven as hard as standard Schnorr signatures without additional assumptions [144]. We hope that further research will improve the state of security proofs for HHS threshold schemes.

Remark 17. Although our scheme is unforgeable in a (static) honest-but-curious model, it is obviously *non-robust*: any participant can lead to an invalid signature without being detected. Robustness can be added using generic zero-knowledge techniques, however it would be interesting to achieve it in a more efficient bespoke fashion.

Another desirable improvement would be to prove security in a stronger *adaptive* corruptions model, where the adversary can query the signing oracle before choosing which participants to corrupt.

6.4 Instantiations based on isogeny graphs

We now describe an instantiation of the previous schemes based on a principal homogeneous space of supersingular elliptic curves defined over a finite field $\mathbb{F}_p$. To this end, we recall the description of CSIDH from Section 2.4.3, put in the context of HHS.

It was first observed by Delfs and Galbraith [65] that the set of all supersingular curves defined over a prime field $\mathbb{F}_p$ partitions into one or two *levels*, each level being a principal homogeneous space for the class group of an order of the quadratic imaginary field $\mathbb{Q}(\sqrt{-p})$, in a way analogous to the well known theory of *complex multiplication*.

These principal homogeneous spaces were first used for a cryptographic purpose in CSIDH [35], however only the precomputation performed recently by Beullens et al. for the signature scheme CSI-FiSh [19] permits to turn one of these into a true HHS.

We now briefly recall some key facts on CSIDH and CSI-FiSh, before turning to the instantiation of our threshold schemes.

6.4.1 Supersingular complex multiplication

From now on we let p be a prime, $\mathbb{F}_p$ the field with p elements, and $\overline{\mathbb{F}_p}$ an algebraic closure. An elliptic curve E defined over $\mathbb{F}_p$ is said to be *supersingular* if and only if $\#E(\mathbb{F}_p) = p+1$. It is well known that there are approximately $p/12$ isomorphism classes of supersingular curves, all defined over $\mathbb{F}_{p^2}$; of these, $O(\sqrt{p})$ are defined over $\mathbb{F}_p$.

Let E be a supersingular curve defined over $\mathbb{F}_p$, an *endomorphism* is an isogeny from E to itself, and it is said to be *defined over* $\mathbb{F}_p$ (or $\mathbb{F}_p$-rational) if it commutes with the *Frobenius endomorphism* π. The $\mathbb{F}_p$-rational endomorphisms of E form a ring, denoted by $\mathrm{End}_{\mathbb{F}_p}(E)$, isomorphic to an order[19] of $\mathbb{Q}(\sqrt{-p})$; more precisely, it is isomorphic to either $\mathbb{Z}[\sqrt{-p}]$ or $\mathbb{Z}[(\sqrt{-p}+1)/2]$. Let $\mathcal{O}$ be such an order, the *class group* $\mathrm{cl}(\mathcal{O})$ is the quotient of the group of invertible ideals of $\mathcal{O}$ by the group of its principal ideals; it is a finite abelian group.

The set of all supersingular curves with $\mathrm{End}_{\mathbb{F}_p}(E)$ isomorphic to a given order $\mathcal{O} \subset \mathbb{Q}(\sqrt{-p})$ is called the *horizontal isogeny class* associated to $\mathcal{O}$. A straightforward extension to the theory of complex multiplication states that the horizontal isogeny class of $\mathcal{O}$, up to $\mathbb{F}_p$-isomorphism, is a principal homogeneous space for $\mathrm{cl}(\mathcal{O})$. To make this into a HHS, an efficient (e.g., polynomial in $\log(p)$) algorithm to evaluate the action of $\mathrm{cl}(\mathcal{O})$ is needed. This is where isogenies play an important role. Fix an isomorphism $\mathrm{End}_{\mathbb{F}_p}(E) \simeq \mathcal{O}$, for any invertible ideal $\mathfrak{a}$, the action $\mathfrak{a} * E$ can be computed as follows: first define the $\mathfrak{a}$-*torsion* subgroup of E as

$$E[\mathfrak{a}] = \{P \in E(\overline{\mathbb{F}_p}) \mid \alpha(P) = 0 \text{ for all } \alpha \in \mathfrak{a}\},$$

this is a finite subgroup of E, and it is stabilized by the Frobenius endomorphism π; then the unique isogeny $\varphi : E \to E/\langle E[\mathfrak{a}]\rangle$ with kernel $E[\mathfrak{a}]$ is such that $\mathfrak{a} * E = E/\langle E[\mathfrak{a}]\rangle$. It follows that, if $\mathfrak{a}$ and $\mathfrak{b}$ are two ideals in the same class, i.e., such that $\mathfrak{a} = (\alpha)\cdot\mathfrak{b}$ for some element $\alpha \in \mathcal{O}$, then $E/\langle E[\mathfrak{a}]\rangle \simeq E/\langle E[\mathfrak{b}]\rangle$.

The curve $E/\langle E[\mathfrak{a}]\rangle$ can be efficiently computed using an isogeny evaluation algorithm [152, 74], however the complexity of this operation is polynomial in the degree of the isogeny, or, equivalently, in the norm $N(\mathfrak{a}) = \#(\mathcal{O}/\mathfrak{a})$. This implies that the action of an element $\mathfrak{a} \in \mathrm{cl}(\mathcal{O})$ can only be efficiently computed when a representative of small norm of $\mathfrak{a}$ is known, or, more generally, when a decomposition

$$\mathfrak{a} = \prod_i \mathfrak{l}_i$$

with all $\mathfrak{l}_i$ of small norm is known.

[19] In this context, an order is a $\mathbb{Z}$-module isomorphic to $\mathbb{Z} \oplus \omega\mathbb{Z} \simeq \mathbb{Z}[\omega]$ for some $\omega \notin \mathbb{Q}$.

Now, for any prime ℓ, the ideal $(\ell) \subset \mathcal{O}$ is either prime, or it splits into a product of two (possibly equal) conjugate prime ideals $\mathfrak{l}\bar{\mathfrak{l}} = (\ell)$ of norm ℓ. In the former case, there are no invertible ideals of norm ℓ in $\mathcal{O}$; in the latter, $\mathfrak{l}$ and $\bar{\mathfrak{l}}$ are the only ideals of norm ℓ, and they are the inverse of one another in $\mathrm{cl}(\mathcal{O})$. Asymptotically, about 50% of the primes ℓ split, thus we may hope to form a basis of generators of $\mathrm{cl}(\mathcal{O})$ of norms bounded by $\mathrm{polylog}(p)$, such that any element of $\mathrm{cl}(\mathcal{O})$ can be represented as a product of $\mathrm{polylog}(p)$ elements of the basis.[20]

This representation for the elements of $\mathrm{cl}(\mathcal{O})$ using a *smooth basis* is at the heart of the Couveignes–Rostovtsev–Stolbunov key exchange scheme, and of CSIDH. However, having a smooth basis may not be enough: to have a HHS, one still needs to be able to rewrite any element of $\mathrm{cl}(\mathcal{O})$ as a compact product of smooth elements. This is the key difference between CSIDH and CSI-FiSh, as we shall see next.

6.4.2 CSIDH and CSI-FiSh

CSIDH was designed to make evaluating the group action of $\mathrm{cl}(\mathcal{O})$ as efficient as possible. To this end, a prime p of the form

$$p + 1 = 4 \prod_{i=1}^{n} \ell_i$$

is selected, where $\ell_1, \ldots, \ell_{n-1}$ are the first $n-1$ odd primes, and ℓ_n is chosen so to make p prime. This choice guarantees several desirable properties:

- The curve $E : y^2 = x^3 + x$ has $\mathbb{F}_p$-rational endomorphism ring isomorphic to $\mathbb{Z}[\pi]$, where $\pi = \sqrt{-p}$ is the image of the Frobenius endomorphism of E;
- All curves in the horizontal isogeny class of $\mathbb{Z}[\pi]$ can be written in the form $y^2 = x^3 + Ax^2 + x$, and the coefficient A uniquely characterizes the $\mathbb{F}_p$-isomorphism class;
- All ℓ_i split in $\mathbb{Z}[\pi]$ as $(\ell_i) = \mathfrak{l}_i\bar{\mathfrak{l}}_i = \langle \ell_i, \pi - 1 \rangle \cdot \langle \ell_i, \pi + 1 \rangle$;
- For any curve E, the $\mathfrak{l}_i$-torsion subgroup is easily found as $E[\mathfrak{l}_i] = E[\ell_i] \cap E(\mathbb{F}_p)$.

The first two properties ensure that supersingular isomorphism classes are easy to construct and represent uniquely, the third guarantees[21] that a number exponential in n of ideal classes of $\mathbb{Z}[\pi]$ can be efficiently represented and its action evaluated, the fourth enables some important optimizations for computing isogenies of degree ℓ_i.

In CSIDH and optimized variants [114, 113, 122, 37], all ideal classes are implicitly represented as products

$$\mathfrak{a} = \prod_{i=1}^{n} \mathfrak{l}_i^{e_i},$$

[20] Jao, Miller and Venkatesan [95] showed that it is indeed possible to bound the norms by $O(\log^2(p))$, assuming the Generalized Riemann Hypothesis.

[21] This guarantee is only heuristic: it is possible, although unlikely, that all $\mathfrak{l}_i$ have small order in $\mathrm{cl}(\mathbb{Z}[\pi])$, and thus generate a small subgroup.

with the exponents e_i in some box $[-B_i, B_i]$ (negative exponents are interpreted as powers of $\bar{\mathfrak{l}}_i$). Explicitly, the representation of an ideal class $\mathfrak{a}$ is simply the vector of exponents $(e_1, \ldots, e_n)$. The action of such ideals can be evaluated in time $\mathrm{poly}(B_i, e_i, n)$ using isogeny formulas.

In practice, a single parameter set has been fully specified for CSIDH, corresponding to the NIST post-quantum level 1.[22] The set has $n = 74$, $\ell_{73} = 373$, and $\ell_{74} = 587$, yielding a prime p of approximately 512 bits; we shall refer to it as CSIDH-512. Protocols based on CSIDH-512 usually sample exponents in a box $[-5, 5]$, which heuristically covers almost all the class group, and which permits to evaluate one class group action in under 30ms [114].

However, based on this data only, CSIDH is *not* a HHS. Indeed, all axioms of an HHS are satisfied but two: it is not possible to efficiently evaluate the action of *any* element of $\mathrm{cl}(\mathbb{Z}[\pi])$, and it is not always possible to test equality of two elements of $\mathrm{cl}(\mathbb{Z}[\pi])$. Take for example the exponent vector $(2^{128}, 0, \ldots, 0)$, corresponding to the ideal $\mathfrak{a} = \langle 3, \pi - 1\rangle^{2^{128}}$; this is a valid element of $\mathrm{cl}(\mathbb{Z}[\pi])$, however without further knowledge its action can only be evaluated through 2^{128} isogeny evaluations. Hopefully, $\mathfrak{a}$ has an equivalent representation on the basis $\mathfrak{l}_1, \ldots, \mathfrak{l}_n$ with much smaller exponents, however we have no way to compute it and, even if we were given it, we could not test their equality.

These problems go away once we have computed the group structure of $\mathrm{cl}(\mathbb{Z}[\pi])$. More precisely, we need to know the *relation lattice* of $\mathfrak{l}_1, \ldots, \mathfrak{l}_n$, i.e., the lattice

$$\Lambda = \left\{ (e_1, \ldots, e_n) \;\middle|\; \prod_{i=0}^{n} \mathfrak{l}_i^{e_i} = 1 \right\},$$

which yields a representation of the class group as $\mathrm{cl}(\mathbb{Z}[\pi]) \simeq \mathbb{Z}^n/\Lambda$. Now, equality of two exponent vectors e, f can be tested by checking that $e - f \in \Lambda$, and any exponent vector e can be evaluated efficiently by finding an (approximate) closest vector $f \in \Lambda$ and evaluating $e - f$ instead.

Neither computing the relation lattice, nor computing a good reduced basis for it are easy tasks: the former requires subexponential time in $\log(p)$, and the latter exponential time in n.[23] Nevertheless, the computation for the CSIDH-512 parameter set happens to be just within reach of contemporary computers, as proven by Beullens et al. [19]: they managed to compute the structure of the class group, which happens to be cyclic of order

$$\begin{aligned} \#\mathrm{cl}(\mathbb{Z}[\pi]) = 3 \cdot 37 \cdot 1407181 \cdot 51593604295295867744293584889 \cdot \\ 315994145046819958530082787455878322049 09 \approx 2^{257.136}, \end{aligned} \tag{6.2}$$

and a BKZ-reduced basis for the relation lattice. In particular, they found out that the ideal $\mathfrak{l}_1 = \langle 3, \pi - 1\rangle$ generates $\mathrm{cl}(\mathbb{Z}[\pi])$.

[22] NIST defines the security of level 1 as being equivalent to AES-128.

[23] Using a quantum computer, the relation lattice can be computed in polynomial time, however lattice reduction still requires exponential time.

Thanks to CSI-FiSh, we thus dispose of a HHS with quantum security estimated at the NIST-1 security level, although scaling to higher security levels currently looks problematic.

6.4.3 Instantiation of the threshold schemes

Given the CSI-FiSh data, we can now instantiate our threshold schemes. However, it is evident by Equation (6.2) that the full group $\langle \mathfrak{l}_1 \rangle = \mathrm{cl}(\mathbb{Z}[\pi])$ is not suitable for them, because the smallest prime factor of its order is 3, thus limiting the schemes to just 2 participants. We may instead choose as generator $\mathfrak{l}_1^3$, which limits the schemes to 36 participants, or $\mathfrak{l}_1^{111}$, allowing more than a million participants.[24]

Efficiency. The performance of our schemes can be readily inferred from that of the CSI-FiSh signature scheme.

To evaluate the action of an ideal in $\mathrm{cl}(\mathbb{Z}[\pi])$, CSI-FiSh first solves an approximate closest vector problem using Babai's nearest plane algorithm [5], and an algorithm by Doulgerakis, Laarhoven, and de Weger [71]; then uses the isogeny evaluation algorithm of CSIDH. The average cost for one evaluation is reported to be $135.3 \cdot 10^6$ cycles (40–50ms on a commercial CPU), which is only 15% slower than the original CSIDH evaluation.[25]

In the encryption scheme, each participant computes exactly one class group action. Since the participants must do their computations sequentially, the total time for decryption is multiplied by the number of participants; the time for encryption, on the other hand, is unaffected by the number of participants, indeed the threshold nature of the protocol is transparent to the user.

In the signature scheme, using the optimization described in [19], depending on the choice of parameters each participant computes between 6 and 56 group actions. Since the group action largely dominates the cost of the whole signing algorithm, we can expect to complete a (k, n)-threshold signature in approximately $k \cdot t \cdot 135.3 \cdot 10^6$ cycles, where $6 \leq t \leq 56$. However, the t group actions by each participant are independent and can be computed in parallel; since the round-robin evaluation in the threshold scheme leaves plenty of idle cycles for participants while they wait for other participants' results, by carefully staggering the threshold group evaluations the k participants can evaluate the t group actions with the same efficiency as the non-threshold scheme, as long as $k \leq t$. According to [19, Tables 3,4], this would provide, for example, quantum-resistant threshold signatures for up to 16 participants in under 1 second, with public keys of 4 KB and signa-

[24] An alternative way to allow up to 36 participants is to use the action of $\mathrm{cl}(\mathbb{Z}[(\pi+1)/2])$ on the horizontal isogeny class of $y^2 = x^3 - x$: the class group is 3 times smaller than $\mathrm{cl}(\mathbb{Z}[\pi])$, and still generated by $\langle 3, \pi - 1 \rangle$. Because the two class group actions are compatible, the CSI-FiSh data can easily be repurposed for this variant without additional computations. This approach is detailed in [32].

[25] Benchmarks in [19] are based on the original CSIDH implementation [35]. A speed-up of roughly 30% is to be expected using the techniques in [114].

ture size of only 560 B. Another example are 1880 B signatures with public key size of 128 B and k up to 56 in under 3 seconds; other interesting compromises are possible. These numbers compare favorably to other post-quantum threshold signatures that are expected to run in seconds [53], and may be especially interesting for side-channel protected implementations of CSI-FiSh.

Attacks. The security of the threshold schemes is essentially the same as that of the original single-participant signature and encryption schemes.

The fact that secrets are sampled in a subgroup of $\mathrm{cl}(\mathbb{Z}[\pi])$ of index 3 or 111 has a minor impact on security, as cryptanalyses can exploit this information to speed up their searches.

In the classical setting, the best algorithm for both Vectorization and Parallelization is a random-walk approach [65] that finds a path between two supersingular curves in $O(\sqrt{\#\mathrm{cl}(\mathbb{Z}[\pi])}) = O(\sqrt[4]{p})$. If, like in our case, we restrict to a vertex set that is x times smaller, the random walk algorithm will find a collision approximately $\sqrt{x}$ times faster. Hence, we expect a loss in classical security of less than 4 bits.[26]

Note that this gain is optimal: if an algorithm could solve the Vectorization problem in a subgroup of size N/x more than $O(\sqrt{x})$ times faster, then by a divide and conquer approach the Vectorization problem in the full group of size N could be solved in less than $O(\sqrt{N})$ operations.

A similar gain can also be obtained in the best quantum algorithm for solving the Vectorization problem [107, 106, 130]. However, since its complexity is sub-exponential, the final gain is even less than 4 bits. The exact quantum security of CSIDH and CSI-FiSh is currently debated [35, 17, 21, 25, 124], nevertheless whatever the final consensus turns out to be, the quantum security of our threshold schemes will be extremely close to it.

6.5 Conclusion and current state-of-the-art

We introduced threshold variants of encryption and signature schemes based on Hard Homogeneous Spaces, and efficient quantum-safe instantiations thereof based on isogeny graphs of supersingular curves (CSIDH).

Our schemes are similar to well known Diffie–Hellman-style threshold schemes, however they are sharply different in the communication structure: whereas classical schemes have participants output messages in parallel with little coordination, our schemes impose a strictly sequential round-robin message passing style. Apparently, this limitation trickles down, negatively affecting many aspects: security properties, security proofs, efficiency.

In our ElGamal-style decryption algorithm, only one participant learns the cleartext, and we are only able to prove security in a *honest-but-curious* setting. While the commu-

[26] In reality, it is well known that the size of the search space can also be reduced by 3 in the original CSIDH, by *walking to the surface*. Thus, the only reduction in security comes from the factor of 37.

nication structure is slightly less problematic for the signature scheme, its security too can only be proven in a *honest-but-curious* setting with *static corruptions*. Interesting questions for future research are efficient protocols where all participants learn the cleartext, or with stronger security properties, such as the ability to detect malicious participants.

Finally, the instantiation of our schemes is limited by the feasibility of parameter generation: to the present date the only available parameter set is the CSIDH-512 HHS, as computed by Beullens et al. Higher security levels would require extremely intensive computations that are currently out of reach.

Follow-up work. In [54] Cozzo and Smart propose Sashimi, an alternative threshold signature scheme based on HHS. Instead of using Shamir secret sharing, they utilize replicated secret sharing, and achieve active security via zero-knowledge proofs. However, this leads to a much slower performance; in particular, [54] reports running times of roughly 5 minutes per party in the case of a threshold scheme instantiation using CSI-FiSh and CSIDH-512. Cozzo and Smart also define an actively secure distributed key generation (DKG) protocol which is not robust, however.

This is mitigated by Beullens, Disson, Pedersen, and Vercauteren [18]. They propose CSI-RAShi, a robust and actively secure DKG protocol in the context of HHS using Shamir secret sharing. In particular, they propose replacements for components of the discrete logarithm-based DKG from Gennaro et al. [86] that do not generalize to the HHS setting; for example, Beullens et al. introduce the *piecewise verifiable proofs* primitive, which is used to replace the Pedersen commitments [123] from the DKG of [86]. The resulting DKG runs in roughly $4.5 + 18n$ seconds when instantiated with CSIDH-512, where n is the number of participants.

Chapter 7

Searching parameters for B-SIDH and SQISign

7.1 Introduction

In this chapter we study the problem of finding *twin smooth* integers, i.e., finding two consecutive integers of a given size, m and $m+1$, whose product is as smooth as possible. Though the literature on the role of smooth numbers in computational number theory and cryptography is vast (see for example the surveys by Pomerance [128] and Granville [88]), the problem of finding consecutive smooth integers of cryptographic size has only been motivated very recently: optimal instantiations of the key exchange scheme B-SIDH [44] and the digital signature scheme SQISign [62] require a large prime that lies between two smooth integers, and this is a special case of the twin smooth problem in which $2m+1$ is prime.

This chapter presents a sieving algorithm for finding twin smooth integers that improves on the methods used in [44] and [62]. The high-level idea is to use two monic polynomials of degree n that split in $\mathbb{Z}[x]$ and that differ by a constant, i.e.,

$$a(x) = \prod_{i=1}^{n}(x - a_i) \text{ and } b(x) = \prod_{i=1}^{n}(x - b_i), \tag{7.1}$$

where $a(x) - b(x) = C$ for $C \in \mathbb{Z}$. Whenever $\ell \in \mathbb{Z}$ such that $a(\ell) \equiv b(\ell) \equiv 0 \bmod C$, it follows that the integers $a(\ell)/C$ and $b(\ell)/C$ differ by 1.

Assume that $|\ell| \gg |a_i|$ and $|\ell| \gg |b_i|$ for $1 \leq i \leq n$, and fix a smoothness bound B. Rather than directly searching for two consecutive B-smooth integers m and $m+1$, roughly of size N, the search instead becomes one of finding a value of ℓ such that the $2n$ (not necessarily distinct) integers

$$\ell - a_1, \ldots, \ell - a_n, \ell - b_1, \ldots, \ell - b_n, \tag{7.2}$$

each of size roughly $N^{1/n}$, are B-smooth. For $n > 1$, and under rather mild heuristics, the probability of finding twin smooth integers in this fashion is significantly greater than

the searches used in [44] and [62]. Put another way, the same computational resources are likely to succeed in finding twin smooth integers subject to an appreciably smaller smoothness bound.

To search for $\ell \approx N^{1/n}$ such that the $2n$ integers in (7.2) are B-smooth, we adopt the simple sieve of Eratosthenes as described by Crandall and Pomerance [55, §3.2.5]; this identifies all of the B-smooth numbers in an arbitrary interval. If w is the largest difference among the $2n$ integers in $\{a_i\} \cup \{b_i\}$, then a sliding window of size $|w|$ can be used to scan the given interval for simultaneous smoothness among the integers in (7.2). This approach has a number of benefits. Firstly, smooth numbers in a given interval can be recognized once-and-for-all, meaning we can combine arbitrarily many solutions to (7.1) into one scan of the interval. Secondly, different processors can scan disjoint intervals in parallel, and each of the interval sizes can be tailored to the available memory of the processor. Finally, the simple sieve we use to identify the smooth numbers in an interval (which is the bottleneck of the overall procedure) is open to a range of modifications and improvements – see Section 7.7.

The approach in this paper hinges on being able to find solutions to (7.1). Such solutions are related to a classic problem in Diophantine Analysis.

7.1.1 The Prouhet-Tarry-Escott problem

The Prouhet-Tarry-Escott (PTE) problem of size n and degree k asks to find two distinct multisets of integers $\{a_1, \ldots, a_n\}$ and $\{b_1, \ldots, b_n\}$ for which

$$\begin{aligned} a_1 + \cdots + a_n &= b_1 + \cdots + b_n, \\ a_1^2 + \cdots + a_n^2 &= b_1^2 + \cdots + b_n^2, \\ &\vdots \qquad\quad \vdots \qquad\quad \vdots \\ a_1^k + \cdots + a_n^k &= b_1^k + \cdots + b_n^k. \end{aligned}$$

The most interesting case is $k = n - 1$, which is maximal (see Section 7.3), and such *ideal* solutions immediately satisfy (7.1). For example, when $n = 4$, the sets $\{0, 4, 7, 11\}$ and $\{1, 2, 9, 10\}$ are such that

$$\begin{aligned} 0 + 4 + 7 + 11 &= 1 + 2 + 9 + 10 &&= 22, \\ 0^2 + 4^2 + 7^2 + 11^2 &= 1^2 + 2^2 + 9^2 + 10^2 &&= 186, \\ 0^3 + 4^3 + 7^3 + 11^3 &= 1^3 + 2^3 + 9^3 + 10^3 &&= 1738, \end{aligned}$$

from which it follows (see Proposition 6) that

$$a(x) = x(x-4)(x-7)(x-11) \text{ and } b(x) = (x-1)(x-2)(x-9)(x-10)$$

differ by a constant $C \in \mathbb{Z}$. Indeed, $a(x) - b(x) = -180$.

Origins of the PTE problem are found in the 18th century works of Euler and Goldbach, and it remains an active area of investigation [27, 26, 30]. In 1935, Wright [154] conjectured that ideal solutions to the PTE problem should exist for all n, but at present this conjecture is open: for $n = 11$ and for $n \geq 13$, no ideal solutions to the PTE problem have been found, see [26, p. 94] and [30, p. 73]. However, Borwein states that "heuristic arguments suggest that Wright's conjecture should be false. [...] It is intriguing, however, that ideal solutions exist for as many n as they do" [26, p. 87].

The PTE solutions that *are* known for $n \in \{2, 3, 4, 5, 6, 7, 8, 9, 10, 12\}$ are a nice fit for our purposes. If we were to fix a smoothness bound, B, and then search for the largest pair of consecutive B-smooth integers we could find, having PTE solutions for n as large as possible would be helpful. But for our cryptographic applications (see Section 7.1.3), we will instead fix a target range for our twin smooth integers to match a given security level, and then aim to find the smoothest twins within that range. In this case, the degree n of $a(x)$ and $b(x)$ cannot be too large, since a larger n means fewer $\ell \in \mathbb{Z}$ to search over. Ideally, n needs to be large enough such that the splitting of $a(x)$ and $b(x)$ into n linear factors helps with the smoothness probability, but small enough so that we still have ample $\ell \in \mathbb{Z}$ to find $a(\ell)$ and $b(\ell)$ such that

(i) $a(\ell) \equiv b(\ell) \equiv 0 \bmod C$,

(ii) $(m, m+1) = (b(\ell)/C, a(\ell)/C)$ are B-smooth, and (optionally)

(iii) $2m + 1$ is prime.

It turns out that those $n \leq 12$ for which PTE solutions are known are the *sweet spot* for our target applications, where $2^{240} \leq m \leq 2^{512}$.

7.1.2 Prior methods of finding twin smooth integers

After defining *twin smooth integers* for concreteness, we recall previous methods used to find large twin smooth integers.

Definition 11 (Twin smooth integers)**.** For a given $B > 1$, we call $(m, m+1)$ with $m \in \mathbb{Z}$ a pair of *twin B-smooth integers* or *B-smooth twins* if $m \cdot (m+1)$ contains no prime factor larger than B.

As Lehmer notes in [109], consecutive pairs of smooth integers have occurred in 18th century works and have been mentioned by Gauss in the context of computing logarithms of integers.

Hildebrand [91, Corollary 2] has shown that there are infinitely many pairs of consecutive smooth integers $(m, m+1)$, however this result notably holds for a smoothness bound that depends on m. More precisely, there are infinitely many such pairs of m^{ε}-smooth integers for any fixed $\varepsilon > 0$. An analogous result holds for tuples of k consecutive smooth integers (for any k), as shown by Balog and Wooley [6].

For a fixed, constant smoothness bound B, the picture is different. A theorem by Størmer [147] states that there is only a finite number of such pairs. We begin with some historical results which show that deterministically computing the largest pair of consecutive B-smooth integers requires a number of operations that is exponential in the number of primes up to B.

Solving Pell equations. Fix B, let $\{2, 3, \ldots, q\}$ be the set of primes up to B with cardinality $\pi(B)$, and suppose that m and $m+1$ are both B-smooth. Let $x = 2m+1$, so that $x-1$ and $x+1$ are also B-smooth, and let D be the squarefree part of the product $(x-1)(x+1)$, so that $x^2 - 1 = Dy^2$ for some $y \in \mathbb{Z}$. Since the product $(x-1)(x+1)$ is B-smooth, it follows that Dy^2 is B-smooth, which (since D is squarefree) means that

$$D = 2^{\alpha_2} \cdot 3^{\alpha_3} \cdot \cdots \cdot q^{\alpha_q}$$

with $\alpha_i \in \{0, 1\}$ for $i = 2, 3, \ldots, q$. For each of the $2^{\pi(B)}$ squarefree possibilities for D, an effective theorem of Størmer [147] (and further work by Lehmer [109]) reverses the above argument and proposes to solve the $2^{\pi(B)}$ Pell equations

$$x^2 - Dy^2 = 1,$$

finding *all* of the solutions for which y is B-smooth, and in doing so finding the complete set of B-smooth consecutive integers m and $m+1$.

Ideally, this process could be used to deterministically find optimally smooth consecutive integers at any size, by increasing B until the largest pair of twin smooths is large enough. For example, the largest pair of twin smooth integers with $B = 3$ is $(8, 9)$, the largest pair of twin smooth integers with $B = 5$ is $(80, 81)$, and the largest pair of twin smooth integers with $B = 7$ is $(4374, 4375)$. Unfortunately, solving $2^{\pi(B)}$ Pell equations becomes infeasible before the size of m grows large enough to meet our requirements. For $B = 113$, [44] reports that the largest twins $(m, m+1)$ found upon solving all 2^{30} Pell equations have $m = 19316158377073923834000 \approx 2^{74}$, and the largest twins found among the set when adding the requirement that $2m+1$ is prime have $m = 75954150056060186624 \approx 2^{66}$.

The extended Euclidean algorithm. One naïve way of searching for twin smooth integers is to compute B-smooth numbers m until either $m-1$ or $m+1$ also turns out to be B-smooth. A much better method, which was used in [44, 13, 62], is to instead choose two coprime B-smooth numbers α and β that are both of size roughly the square root of the targets m and $m+1$. Since α and β are coprime, Euclid's extended GCD algorithm outputs two integers (s, t) such that $\alpha s + \beta t = 1$ with $|s| < |\beta/2|$ and $|t| < |\alpha/2|$. We can then take $\{m, m+1\} = \{|\alpha s|, |\beta t|\}$, and the probability of m and $m+1$ being B-smooth is now the probability that $s \cdot t$ is B-smooth. The key observation here is that the product $s \cdot t$ with $s \approx t$ is much more likely to be B-smooth than a random integer of similar size. In

Section 7.2 we will develop methods and heuristics that allow us to closely approximate these probabilities.

Searching with $m = x^n - 1$. The method from [44] that proved most effective in finding twin smooth integers with $2^{240} \leq m \leq 2^{256}$ is by searching with $(m, m+1) = (x^n - 1, x^n)$ for various n, where the best instances were found with $n = 4$ and $n = 6$. Our approach can be seen as an extension of this method, where the crucial difference is that for $n > 2$ the polynomial $x^n - 1$ does not split in $\mathbb{Z}[x]$, and the presence of higher degree terms significantly hampers the probability that values of $\ell^n - 1 \in \mathbb{Z}$ are smooth. For example, with $n = 6$ we have $m = (x^2 - x + 1)(x^2 + x + 1)(x - 1)(x + 1)$ and, assuming $B \ll \ell$, the probability that integer values of this product are B-smooth is far less than if it was instead a product of six monic, linear terms. On the other hand, the probability that $m + 1$ is B-smooth for a given ℓ is the probability that ℓ itself is B-smooth, which works in favor of the *non-split* method. However, as we shall see in the sections that follow, this is not enough to counteract the presence of the higher degree terms. Furthermore, several of the PTE solutions we will be using also benefit from repeated factors.

7.1.3 Cryptographic applications of twin smooth integers

As described in Section 2.4, two supersingular isogeny-based schemes have recently been proposed that require a new type of instantiation. Rather than defining primes p for which either $p - 1$ *or* $p + 1$ is smooth (as in SIDH/SIKE), the key exchange scheme B-SIDH [44] and the digital signature scheme SQISign [62] instead require primes for which (large factors of) both $p - 1$ *and* $p + 1$ are smooth. As both of those papers discuss, finding primes that lie between two smooth integers is not an easy task, but the practical incentive to do this is again related to the compactness of these schemes: B-SIDH's public keys are even smaller than the analogous SIDH/SIKE compressed public keys, and the sum of the SQISign public key and signature sizes is significantly smaller than those of all of the remaining NIST signature candidates.

In both B-SIDH and SQISign, the overall efficiency of the protocol is closely tied to the smoothness of $p - 1$ and $p + 1$. Roughly speaking, any prime ℓ appearing in the factorizations of these two integers implies that an ℓ-isogeny needs to be computed somewhere in the protocol. Such ℓ-isogenies have traditionally been computed in $O(\ell)$ field operations using Vélu's formulas [152], but recent work by Bernstein, De Feo, Leroux, and Smith [13] improved the asymptotic complexity to $\widetilde{O}(\sqrt{\ell})$ by clever use of a baby-step giant-step algorithm. Nevertheless, the large ℓ-isogenies that are required in these protocols dominate the runtime, and the best instantiations of both schemes will use large primes p lying between two integers that are as smooth as possible.

In this chapter we will view the search for such primes as one that imposes an additional stipulation on the more general problem of finding twin smooth integers: crypto-

graphically useful instances of the twin smooth integers $(m, m+1)$ are those where the sum $2m+1$ is a prime, p.

Security analyses of B-SIDH and SQISign suggest that it is possible to relax the requirements and to tolerate *cofactors* that divide either or both of $p-1$ and $p+1$ and have prime factors somewhat larger than the target smoothness bound, such that (the size of) any primes dividing these cofactors have no impact on the efficiency. For simplicity and concreteness, we will focus our analysis on the pure problem of finding twin smooth integers that disallows any primes larger than our smoothness bound, but we will oftentimes point out the modifications and relaxations that account for cofactors; this is discussed in Section 7.7.

The heuristic analysis summarized in Table 7.3 predicts that sieving with PTE solutions finds twin smooth integers $(m, m+1)$ that are smoother than one expects to find using the same computational resources and the prior methods described in Section 7.1.2. Indeed, in Section 7.6 we present a number of examples we found with our sieve whose largest prime divisors are several bits smaller than the largest prime divisors in instantiations found in the literature. In reference to Table 7.3, we briefly sketch some intuition on how these smoother examples translate into practical speedups. For example, the best prior instantiation of a prime p with $2^{240} \leq p < 2^{256}$ found that $(p-1)$ and $(p+1)$ are simultaneously 2^{19}-smooth, whereas our sieve found a similarly sized p subject to a smoothness bound of 2^{15}. Given the current (square root) complexity of state-of-the-art ℓ-isogeny computations, this suggests that the most expensive isogeny computed in our example will be roughly 4 times faster than that of the prior example.

Organization. First time readers may benefit from first jumping straight to Section 7.5, where all the theory developed in Sections 7.2–7.4 is put into action by way of a full worked example. Section 7.2 gathers some results that allow us to approximate the smoothness probabilities of both integers and integer-valued polynomials. 7.3 starts by making the connection between our method of finding twin smooth integers and the PTE problem, before going into the theory of the PTE problem and showing how to generate infinitely many solutions for certain degrees. Section 7.4 describes our sieving algorithm. Section 7.6 presents some of the best examples found with our sieve and compares them with the previous examples in the literature. Section 7.7 discusses a number of possible modifications and improvements to the sieve.

7.2 Smoothness probabilities

In this section we recall some well-known results concerning smoothness probabilities that will be used to analyze various approaches throughout the paper: Section 7.2.1 shows how to approximate the probability that $m \gg B$ is B-smooth using the Dickman–de Bruijn

function; Section 7.2.2 shows how to approximate the probability that integer values of a polynomial $f(x) \in \mathbb{Z}[x]$ are B-smooth.

7.2.1 Smoothness probabilities for large N

Recall that an integer is said to be B-smooth if it does not have any prime factor exceeding B. Let

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

be the number of positive B-smooth integers. For each real number $u > 0$, Dickman's theorem [55, Theorem 1.4.9] states that there is a real number $\rho(u) > 0$ such that

$$\frac{\Psi(N, N^{1/u})}{N} \sim \rho(u) \text{ as } N \to \infty. \tag{7.3}$$

Dickman described $\rho(u)$ as the unique continuous function on $[0, \infty)$ that satisfies $\rho(u) = 1$ for $0 \leq u \leq 1$, and $\rho'(u) = -\frac{\rho(u-1)}{u}$ for $u > 1$. For $1 \leq u \leq 2$, $\rho(u) = 1 - \ln(u)$, but for $u > 2$ there is no known closed form for $\rho(u)$. Nevertheless, it is easy to evaluate $\rho(u)$ (up to any specified precision) for a given value of u, and popular computer algebra packages (like Magma and Sage) have this function built in.

In this chapter we will be using (7.3) to approximate the probability that certain large numbers are smooth. For example, with $N = 2^{128}$ and $u = 8$, the value $\rho(8) \approx 2^{-25}$ approximates the probability that a 128-bit number is 2^{16}-smooth. With u fixed, this approximation becomes better as N tends towards infinity. Using $\rho(u)$ as the *smoothness probability* assumes the heuristic that $N^{1/u}$-smooth numbers are uniformly distributed in $[1, N]$.

While there are methods to more precisely estimate $\Psi(N, B)$, see e.g. [143] and [7], we are content with the simple approximation given by ρ. Using a basic sieve to identify smooth integers, we have counted all B-smooth integers up to $N = 2^{43}$ for B up to 2^{16} and compared their numbers with those predicted by the Dickman–de Bruijn function. Except for the lower end of the studied interval and for very small smoothness bounds, we have found the approximation by ρ to be sufficiently close to the actual values.

7.2.2 Smoothness heuristics for polynomials

For a polynomial $f(x) \in \mathbb{Z}[x]$, define

$$\Psi_f(N, B) = \#\{1 \leq m \leq N \mid f(m) \text{ is } B\text{-smooth}\}.$$

Throughout the paper we will use the following conjecture (see [110, Eq. 1.4] and [88, Eq. 1.20]) as a heuristic to estimate the probability that $f(N)$ is $N^{1/u}$-smooth.

Heuristic 1. *Suppose that the polynomial $f(x) \in \mathbb{Z}[x]$ has distinct irreducible factors over $\mathbb{Z}[x]$ of degrees $d_1, d_2, \ldots, d_k \geq 1$, respectively, and fix $u > 0$. Then*

$$\frac{\Psi_f(N, N^{1/u})}{N} \sim \rho(d_1 u) \cdot \cdots \cdot \rho(d_k u) \tag{7.4}$$

as $N \to \infty$.

With $B = N^{1/u}$, Heuristic 1 says that for $m \leq N$, the probability of $f(m)$ being B-smooth is the product of the probabilities of each of its factors being B-smooth (these are computed via (7.3)). Martin proved this conjecture for a certain range of u [110, Theorem 1.1] that does not apply in our case. Heuristic 1 inherently assumes that the smoothness probabilities of each of the factors are independent of one another; here, the roots of our split polynomials all lie in relatively short intervals, and thus are not uniformly distributed in, say, $[1, N]$. For example, with $f(m) = \prod_{1 \leq i \leq d}(m - f_i) \in \mathbb{Z}$, any prime q that divides $m - f_1$ only divides $m - f_i$ for some $1 < i \leq d$ if $q \mid f_i - f_1$, which in particular means that any prime which is larger than the interval size can divide at most one of the (unique) $m - f_i$. Nevertheless, our experiments have shown Heuristic 1 to be a very useful approximation for our purposes; we simply use it as a means to approximate how many values of $m \in \mathbb{Z}$ need to be searched before we can expect to start finding twin smooth integers, and to draw comparisons between approaches for various target sizes.

7.3 Split polynomials that differ by a constant

Henceforth we will use $a(x)$ and $b(x)$ to denote two polynomials of degree $n > 1$ in $\mathbb{Z}[x]$ that differ by an integer constant $C \in \mathbb{Z}$, i.e., $a(x) - b(x) = C$. Moreover, unless otherwise stated, both a and b are assumed to split into linear factors over $\mathbb{Z}$, i.e.,

$$a(x) = \prod_{1 \leq i \leq n} (x - a_i) \quad \text{and} \quad b(x) = \prod_{1 \leq i \leq n} (x - b_i),$$

where the a_i and b_i (which are not necessarily distinct) are all in $\mathbb{Z}$.

The core idea of this paper is to search for twin smooth integers by searching over $\ell \in \mathbb{Z}$ such that

$$a(\ell) \equiv b(\ell) \equiv 0 \bmod C.$$

Then, the two polynomials $a_C(x) := a(x)/C$ and $b_C(x) := b(x)/C \in \mathbb{Q}[x]$ evaluate to integer values $a_C(\ell)$ and $b_C(\ell)$ at ℓ, and moreover

$$a_C(\ell) = b_C(\ell) + 1.$$

Since a and b split into n linear factors over $\mathbb{Z}$, $a_C(\ell)$ and $b_C(\ell)$ necessarily contain n integer factors of approximately the same size. In Section 7.4.4 we approximate the probability that $a_C(\ell)$ and $b_C(\ell)$ are B-smooth, and show that these probabilities are favorable (in the ranges of practical interest) compared to the previously known methods of searching for large twin smooths.

7.3.1 The Prouhet-Tarry-Escott problem

For degrees $n \leq 3$, infinite families of split polynomials $a(x)$ and $b(x)$ with $a(x) - b(x) = C \in \mathbb{Z}$ can be constructed by solving the system that arises from equating all but the constant coefficients. Although there are n equations in $2n$ unknowns, for $n > 3$ this process becomes unwieldy; the equations are nonlinear and we are seeking solutions that assume values in $\mathbb{Z}$. Moreover, relaxing the monic requirement (which permits $4n$ unknowns) and allowing for solutions in $\mathbb{Q}$ does not seem to help beyond $n > 3$. Fortunately, finding these pairs of polynomials is closely connected to the computational hardness of solving the PTE problem of size n.

Definition 12 (The Prouhet-Tarry-Escott problem)**.** The *Prouhet-Tarry-Escott (PTE) problem* of size n and degree k asks to find distinct multisets of integers $\mathcal{A} = \{a_1, \ldots, a_n\}$ and $\mathcal{B} = \{b_1, \ldots, b_n\}$, such that

$$\sum_{i=1}^{n} a_i^j = \sum_{i=1}^{n} b_i^j$$

for $j = 1, \ldots, k$. We abbreviate solutions to this problem by $[a_1, \ldots, a_n] =_k [b_1, \ldots, b_n]$ or $\mathcal{A} =_k \mathcal{B}$.

A classic result that links PTE solutions to polynomials is the following [27, Proposition 1].

Proposition 6. *The following are equivalent:*

$$\sum_{i=1}^{n} a_i^j = \sum_{i=1}^{n} b_i^j \quad \text{for} \quad j = 1, \ldots, k. \tag{7.5}$$

$$\deg\left(\prod_{i=1}^{n}(x - a_i) - \prod_{i=1}^{n}(x - b_i)\right) \leq n - (k+1). \tag{7.6}$$

Proposition 6 implies that for any PTE solution of size n and degree $k = n - 1$, the polynomials $a(x) = \prod_{i=1}^{n}(x - a_i)$ and $b(x) = \prod_{i=1}^{n}(x - b_i)$ differ by a constant. For a given n, this choice for k is the maximal possible choice [27, Proposition 2], hence the respective solutions are called *ideal solutions*. Ideal solutions are known for $n \leq 10$ and $n = 12$, but it remains unclear if there are ideal solutions for other sizes [30]. Unless stated otherwise, henceforth we will only speak of PTE solutions that are ideal solutions.

As we will see later, the most useful PTE solutions for our purposes are those for which the constant C is as small as possible. We now recall some useful results from the literature concerning the constants that can arise from PTE solutions.

Definition 13 (Fundamental constant C_n)**.** Let n be a positive integer, and write $C_{n,\mathcal{A},\mathcal{B}}$ for the associated constant of an ideal PTE solution $\mathcal{A} =_{n-1} \mathcal{B}$ of size n. Then we define

$$C_n = \gcd\{C_{n,\mathcal{A},\mathcal{B}} \mid \mathcal{A} =_{n-1} \mathcal{B}\}$$

as the *fundamental constant* associated to ideal PTE solutions of size n.

Table 7.1: Divisibility results for the PTE problem

n	Lower bound for C_n	Upper bound for C_n
2	1	1
3	2^2	2^2
4	$2^2 \cdot 3^2$	$2^2 \cdot 3^2$
5	$2^4 \cdot 3^2 \cdot 5 \cdot 7$	$2^4 \cdot 3^2 \cdot 5 \cdot 7$
6	$2^5 \cdot 3^2 \cdot 5^2$	$2^5 \cdot 3^2 \cdot 5^2$
7	$2^6 \cdot 3^3 \cdot 5^2 \cdot 7 \cdot 11$	$2^6 \cdot 3^3 \cdot 5^2 \cdot 7 \cdot 11$
8	$2^4 \cdot 3^3 \cdot 5^2 \cdot 7^2 \cdot 11 \cdot 13$	$2^8 \cdot 3^3 \cdot 5^2 \cdot 7^2 \cdot 11 \cdot 13$
9	$2^7 \cdot 3^3 \cdot 5^2 \cdot 7^2 \cdot 11 \cdot 13$	$2^9 \cdot 3^4 \cdot 5^2 \cdot 7^2 \cdot 11 \cdot 13 \cdot 17 \cdot 23 \cdot 29$
10	$2^7 \cdot 3^4 \cdot 5^2 \cdot 7^2 \cdot 13 \cdot 17$	$2^{11} \cdot 3^6 \cdot 5^2 \cdot 7^2 \cdot 11 \cdot 13 \cdot 17 \cdot 23 \cdot 37$
11	$2^8 \cdot 3^4 \cdot 5^3 \cdot 7^2 \cdot 11 \cdot 13 \cdot 17 \cdot 19$	none known
12	$2^8 \cdot 3^4 \cdot 5^3 \cdot 7^2 \cdot 11^2 \cdot 17 \cdot 19$	$2^{12} \cdot 3^8 \cdot 5^3 \cdot 7^2 \cdot 11^2 \cdot 13^2 \cdot 17 \cdot 19 \cdot 23 \cdot 29 \cdot 31$

A result by Kleiman [102] gives a lower bound on the fundamental constant.

Proposition 7. *Let n be a positive integer. Then $(n-1)! \mid C_n$.*

For concrete choices of n, more divisibility results are presented in the work of Rees and Smyth [129], and Caley [30]. These results form sharper bounds for C_n, and thus for constants arising from any given PTE solution. Upper bounds for C_n can be directly computed by taking the GCD of all known solutions of size n. This is detailed in [30], where for example it is stated that for $n = 9$ we have

$$2^7 \cdot 3^3 \cdot 5^2 \cdot 7^2 \cdot 11 \cdot 13 \mid C_9 \text{ and } C_9 \mid 2^9 \cdot 3^4 \cdot 5^2 \cdot 7^2 \cdot 11 \cdot 13 \cdot 17 \cdot 23 \cdot 29.$$

Table 7.1 is an updated version of [30, Table 3.2], and gives an overview of the bounds for the fundamental constants C_n. These results give estimates for the optimal choices of solutions for our searches. In particular, choosing solutions with associated constants close to the upper bound for C_n yields the best preconditions for finding twin smooth integers.

For our application of finding twin smooth integers, it may seem unnecessarily restrictive to only make use of PTE solutions, yielding monic polynomials a and b with integer roots. However, it can be proven that *all* polynomials that are split over $\mathbb{Q}$ and that differ by a constant arise from PTE solutions. In order to prove this, we make use of the following result ([27, Lemma 1], [30, Proposition 2.1.2]).

Proposition 8. *Let $[a_1, \ldots, a_n] =_k [b_1, \ldots, b_n]$ with associated constant C and M, K arbitrary integers with $M \neq 0$. Define a linear transform $h(x) = Mx + K$ and let $a'_i = h(a_i)$ and $b'_i = h(b_i)$ for $i = 1, \ldots, n$. Then $[a'_1, \ldots, a'_n] =_k [b'_1, \ldots, b'_n]$, and the associated constant is $C' = C \cdot M^n$.*

Two such solutions that are connected through a linear transform are called *equivalent*. Note that Proposition 8 also holds for the PTE problem over rational numbers instead of integers.

Corollary 2. *Let $a(x)$ and $b(x)$ be non-monic polynomials of degree n with rational roots $\mathcal{A} = \{a_1, \ldots, a_n\}$ and $\mathcal{B} = \{b_1, \ldots, b_n\}$, such that $a(x) - b(x) = C \in \mathbb{Q}$. Then $\mathcal{A} =_{n-1} \mathcal{B}$ for the PTE problem over $\mathbb{Q}$, and there is an equivalent solution $\mathcal{A}' =_{n-1} \mathcal{B}'$ to the PTE problem over $\mathbb{Z}$.*

Proof. Since $\deg(a(x) - b(x)) = 0$, Proposition 6 implies that $\mathcal{A} =_{n-1} \mathcal{B}$. Let $M \in \mathbb{Z}$ be a common denominator of $a_1, \ldots, a_n, b_1, \ldots, b_n$ and define the linear transform $h(x) = Mx$. Let $a'_i = h(a_i)$ and $b'_i = h(b_i)$ for $i = 1, \ldots, n$. Then $\mathcal{A}' = \{a'_1, \ldots, a'_n\}$ and $\mathcal{B}' = \{b'_1, \ldots, b'_n\}$ consist of integers, and by Proposition 8, $\mathcal{A}' =_{n-1} \mathcal{B}'$ is a solution for the PTE problem over $\mathbb{Z}$. □

Corollary 2 allows us to focus entirely on integer PTE solutions without imposing any further restrictions. For our search for smooth values of the polynomials, Proposition 8 further implies that we only have to search with one polynomial per equivalence class.

Corollary 3. *Let $\mathcal{A} = \{a_1, \ldots, a_n\}$, $\mathcal{B} = \{b_1, \ldots, b_n\}$ with $\mathcal{A} =_{n-1} \mathcal{B}$, and $\mathcal{A}' = \{a'_1, \ldots, a'_n\}$, $\mathcal{B}' = \{b'_1, \ldots, b'_n\}$ with $\mathcal{A}' =_{n-1} \mathcal{B}'$ be equivalent ideal PTE solutions. Let $a(x)$, $b(x)$, and $a'(x)$, $b'(x)$ be the respective polynomials such that $a(x) - b(x) = C \in \mathbb{Z}$ resp. $a'(x) - b'(x) = C' \in \mathbb{Z}$, and $h(x)$ be the associated linear transform. Then for given $x_{\min}$ and $x_{\max}$, $a_C(x)$ and $b_C(x)$ take on the same integer values for $x \in I = [x_{\min}, x_{\max}]$ as $a'_{C'}(x)$ and $b'_{C'}(x)$ for $x \in h(I)$.*

In order to efficiently identify equivalent solutions, we make use of Proposition 8 to define a representation of equivalence classes, which we call the *normalized form* of a class of solutions.

Definition 14 (Normalized form of PTE solutions)**.** A *normalized form* of a given PTE solution is a solution such that $a_1 \leq a_2 \leq \cdots \leq a_n$, $b_1 \leq b_2 \leq \cdots \leq b_n$, $0 = a_1 < b_1$, and $\gcd(a_1, \ldots, a_n, b_1, \ldots, b_n) = 1$.

Another classification of solutions, which is of importance for our searches, is the distinction between *symmetric* and *non-symmetric* solutions [26].

Definition 15 (Symmetric PTE solutions)**.** For n even, an *even ideal symmetric solution* to the PTE problem is of the form

$$[\pm a_1, \pm a_2, \ldots, \pm a_{n/2}] =_{n-1} [\pm b_1, \pm b_2, \ldots, \pm b_{n/2}].$$

For n odd, an *odd ideal symmetric solution* to the PTE problem is of the form

$$[a_1, a_2, \ldots, a_n] =_{n-1} [-a_1, -a_2, \ldots, -a_n].$$

It can immediately be seen that the normalized form of a symmetric solution is unique, but no longer has the form satisfying Definition 15. However, as is usual, we will still be calling these solutions symmetric, since they are symmetric with respect to the integer K (instead of symmetric with respect to 0, as in the classic formulation of Definition 15), where $h(x) = Mx + K$ is the linear transform connecting these solutions. Thus, we define solutions as non-symmetric if and only if their equivalence class does not contain a symmetric solution according to Definition 15.

Note that in the special case of non-symmetric solutions, the normalized form is not unique. In particular, if $[a_1, \ldots, a_n] =_{n-1} [b_1, \ldots, b_n]$ is a non-symmetric normalized solution, then so is the solution arising from the linear transform $h(x) = Mx + K$, where $M = -1$ and $K = \max\{a_n, b_n\}$. In this case, we take the solution with minimal b_1 to represent the normalized solution, and refer to the second normalized solution as the *flipped solution*.

Finally, in Section 7.4.4 we will see that PTE solutions with repeated factors have higher probabilities (than those without repeated factors) of finding twin smooth integers. The following result [30, Theorem 2.1.3] shows that repeated factors can only occur with multiplicity at most 2.

Proposition 9 (Interlacing). *Let $\mathcal{A} = \{a_1, \ldots, a_n\}$ and $\mathcal{B} = \{b_1, \ldots, b_n\}$ be an ideal PTE solution, where $a_1 \leq a_2 \leq \cdots \leq a_n$ and $b_1 \leq b_2 \leq \cdots \leq b_n$, and w.l.o.g., we assume that $a_1 < b_1$. Then, $a_1 \neq b_j$ for all j. If n is odd, we have*

$$a_1 < b_1 \leq b_2 < a_2 \leq a_3 < \cdots < a_{n-1} \leq a_n < b_n,$$

and if n is even, then

$$a_1 < b_1 \leq b_2 < a_2 \leq a_3 < \cdots < a_{n-2} \leq a_{n-1} < b_{n-1} \leq b_n < a_n.$$

7.3.2 PTE solutions

An important prerequisite for searching for twin smooth integers is a large number of normalized ideal PTE solutions with relatively small associated constants. To this end, we briefly review solutions from the literature as well as methods to construct ideal solutions. Henceforth, we will refer to normalized ideal PTE solutions only as PTE solutions.

A database of Shuwen collects several PTE solutions, both symmetric and non-symmetric [139]. In particular, special solutions, such as the smallest solutions with respect to the associated constants, and the first solutions found for each size, are presented there.

Apart from this, several methods for generating PTE solutions have been found. Parametric solutions are known for $n \in \{2, 3, 4, 5, 6, 7, 8, 10, 12\}$, and these can be used to generate infinitely many symmetric solutions [30]. However, the number of solutions with small associated constants is limited. For $n = 9$, only two non-equivalent solutions are known.

For $n \in \{5, 6, 7, 8\}$, we implemented the methods from [26] to generate as many symmetric solutions with small associated constants as possible. For $n = 10$ and $n = 12$, there are parametric symmetric solutions due to Smyth [142] and Choudhry and Wróblewski [155], resp., both following an earlier method from Letac [87]. In both methods, the two parameters that form solutions come from a quadratic equation in two variables. This equation can be transformed into an elliptic curve equation, and thus finding suitable parameters is equivalent to finding rational points on this elliptic curve. In [30, Section 6], Caley implements these methods by adding multiples of a non-torsion point, P, to the eight known torsion points.[27] However, it is evident from the underlying transforms that PTE solutions with small constants can only arise from rational elliptic curve points with small denominators in their coordinates. Caley's approach thus proves to be non-optimal for our aims, as the denominators in the coordinates of $[i]P$ become too large already for very small i, resulting in PTE solutions with huge constants. We implemented these methods with the curves and transforms from [30], but deviated from Caley's approach by first searching for non-torsion points with integer coordinates, resp. coordinates with very small denominators. We then followed Caley's algorithm and computed small multiples of these points and their sums with torsion points. Despite finding many PTE solutions, none of them proved to have an associated constant close to the upper bound for C_{10} resp. C_{12}. Further, taking the GCD of all found solutions, we did not succeed in reducing the known upper bounds for C_{10} resp. C_{12}.

For each size n, we identified an upper bound for constants that permit acceptable success probabilities for our searches, and collected as many solutions as possible up to this value. Table 7.2 reports on the numbers of solutions we found, including solutions from [139].

7.4 Sieving with PTE solutions

Our sieving algorithm consists of two phases. The first phase identifies the B-smooth numbers in a given interval (Section 7.4.1). The second phase then scans the interval using either a single PTE solution (Section 7.4.2) or the combination of many PTE solutions (Section 7.4.3).

7.4.1 Identifying smooth numbers in an interval

We follow the exposition of Crandall and Pomerance [55, §3.2.5] and adopt the simple sieve of Eratosthenes to identify the B-smooth integers in an interval $[L, R)$. We set up an array of $R - L$ integers corresponding to the integers $L, L + 1, \ldots, R - 1$, and initialize each entry with 1. For all primes with $p < B$, we identify the smallest non-negative $i \in \mathbb{Z}$,

[27] The elliptic curves that arise for $n = 10$ and $n = 12$ have Mordell-Weil-groups $\mathbb{Z}/4\mathbb{Z} \times \mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}$ resp. $\mathbb{Z}/4\mathbb{Z} \times \mathbb{Z}/2\mathbb{Z} \times \mathbb{Z} \times \mathbb{Z}$. Thus there are eight torsion points in each case, and the non-torsion groups are generated by one resp. two non-torsion points.

Table 7.2: Number of PTE solutions up to an upper bound for the constants. $C_{\min,n}$ denotes the smallest constant known for each degree.

n	$\lceil \log_2(C_{\min,n}) \rceil$	Bitlength of upper bound	# of solutions
5	13	50	49
6	14	50	2438
7	33	60	8
8	31	60	51
9	52	60	2
10	73	100	1
12	76	100	1

for which $L + i \equiv 0 \bmod p$, and multiply the array elements at positions $i + jp$ by p for all $j \in \mathbb{Z}$ such that $L \leq i + jp < R$. Additionally, for all primes with $p < \sqrt{R}$, we have to identify the maximal exponent e such that $p^e < R$, and analogously perform sieving steps with the relevant prime powers, where further multiplications by p take place. After this process is finished, the B-smooth integers in the interval are precisely those for which the number at position i is $L + i$. Subsequently, we transform this array of integers into a bitstring, where a '1' indicates a B-smooth number, while a '0' represents a non-smooth number.

This simple approach allows for several optimizations and modifications, some of which are discussed further in Section 7.7.

7.4.2 Searching with a single PTE solution

Assume that we are searching with a normalized ideal PTE solution of size n, writing $a(x) = \prod_{i=1}^{n}(x - a_i)$ and $b(x) = \prod_{i=1}^{n}(x - b_i)$, together with $C \in \mathbb{Z}$ such that $a(x) - b(x) = C$. We will assume $C > 0$, since $a(x)$ and $b(x)$ can otherwise swap roles accordingly, and as usual we write $a_C(x) = a(x)/C$ and $b_C(x) = b(x)/C$ as the two polynomials in $\mathbb{Q}[x]$.

We are searching for ℓ such that $m + 1 = a_C(\ell)$ and $m = b_C(\ell)$ are both B-smooth and of a given size, and thus the size of the constant C affects the size of the ℓ we should search over. Moreover, we only wish to search over the values of ℓ for which $a_C(\ell)$ and $b_C(\ell)$ are integers, and we determine this set of residues (modulo C) as follows. If $C = \prod p_i^{e_i}$ is the prime factorization of the constant, then for each prime-power factor we determine all residues $r_i \bmod p_i^{e_i}$ for which $a(r_i) \equiv b(r_i) \equiv 0 \bmod p_i^{e_i}$ (note that it is sufficient to check that one of $a(r_i)$ or $b(r_i)$ is a multiple of $p_i^{e_i}$). We then use the Chinese Remainder Theorem (CRT) to reconstruct the full set of residues $\{r \bmod C\}$ for which $a(r) \equiv b(r) \equiv 0 \bmod C$. Depending on the size of the constant, the full list of suitable residues may be rather large;

if not, they can be stored in a lookup table, but if so, only the smaller sets (i.e., the $\{r_i\}$ corresponding to $p_i^{e_i}$) need to be stored. We can then either loop over the suitable residues by constructing them on the fly using the CRT, or we can check whether a candidate ℓ is a suitable residue by reducing it modulo each of the $p_i^{e_i}$.

It is worth pointing out that when searching for cryptographic parameters with a single PTE solution, the condition that $2m + 1$ is prime can be used to discard the residues $\{\tilde{r} \bmod C\}$ for which $2b_C(r) + 1$ can never be prime if $r \equiv \tilde{r} \bmod C$. In a very rare number of cases, the polynomial $2b_C(x) + 1 = 2/C \cdot (b(x) + C/2)$ in $\mathbb{Q}[x]$ is such that $(b(x) + C/2)$ is reducible in $\mathbb{Z}[x]$, in which case the PTE solution can be completely discarded. For example, this happens for *both* of the PTE solutions with $n = 9$.

Recall from Section 7.3 that the constants of the PTE solutions are (for our purposes) always B-smooth. When processing an interval $[L, R)$, the problem therefore reduces to finding $\ell \in [L, R)$ such that all of the factors of $a(\ell)$ and $b(\ell)$ are marked as B-smooth. For the PTE solution in use, these factors are given by $\ell_i = \ell - i$, where $i \in \{a_1, \ldots, a_n, b_1, \ldots, b_n\}$. Note that since $a_1 = 0$ for our normalized representation, we have $\ell = \ell_0$. Starting with ℓ at the left end of the interval requires some care since for a given ℓ, we need to be able to check for the smoothness of all ℓ_i. Hence, to be able to cover the full space when processing consecutive intervals, we have to run the first phase of the sieve for a slightly larger interval, namely $[L - w, R)$ (overlapping to the left with the previous interval), where $w = \max\{a_n, b_n\}$. This allows us to process $\ell \in [L, R)$ such that ℓ_w will cover $[L - w, R - w)$.

In the second phase of the sieve we advance ℓ through all of the elements in the bitstring marked '1', each time checking the bits corresponding to the remaining ℓ_i, i.e., $i \in \{a_2, \ldots, a_n, b_1, \ldots, b_n\}$. If, at any time, we see that any of the ℓ_i corresponds to a '0', we advance ℓ such that it is aligned with the next '1' and repeat the process until all of the ℓ_i correspond to a '1'. At this point, we can then check whether ℓ is a suitable residue modulo C as above; if not, ℓ is again advanced to the next set bit, but if so, we have found twin smooth integers, and it is here that we can optionally check whether their sum is prime.

We note that when using a single PTE solution, the algorithm could be modified to sieve in arithmetic progressions given by the suitable residues modulo C. We leave the exploration of whether this can be more efficient than the above approach for future work.

In the case of a large interval $[L, R)$, the memory requirements can be significantly reduced by dividing $[L, R)$ into several subintervals, which can be processed separately. The only downside is that a naïve implementation of the first phase processes certain intervals twice due to the overlap of length w. This can be easily mitigated by copying the last w entries of the previous interval at each step. However, due to both the large (sub)intervals used in our implementation and the relatively small w's that arise in PTE solutions, the impact of this overlap is negligible in practice, so the naïve approach can be taken without a noticeable performance penalty.

Parallelization. The straightforward way to parallelize the sieve is to assign processors distinct subintervals of $[L, R)$, e.g. according to their own memory/performance capabilities. If many processors have rapid access to the same memory, then it may be faster for some resources being devoted to identifing smooth numbers in the next interval while the remaining resources sieve the current interval.

Negative input values. Until now we have only considered positive input values $\ell \in [L, R)$, but our approach also permits negative inputs to the polynomials $a(x)$ and $b(x)$. For example, for even n, this gives another pair of integers that could potentially be smooth. At first glance, this seems to imply that each time ℓ is advanced, we must also check the values $\ell'_i = \ell + i$ with $i \in \{a_1, \ldots, a_n, b_1, \ldots, b_n\}$ for smoothness. Moreover, it seems that the overlap of size w for each search interval must also be added to both sides. We note, however, that if the PTE solution in use is symmetric (see Definition 15), then the values ℓ'_i are the same as the values $(\ell + w)_i$, and thus are naturally checked by our previous algorithm at position $\ell + w$. This is not the case for general non-symmetric solutions, but for those non-symmetric solutions that are normalized (see Definition 14), we can instead search with positive inputs to the *flipped* solution arising from the linear transform $h(x) = -x + w$, which is especially beneficial when searching with many solutions simultaneously.

7.4.3 Searching with many PTE solutions

One of the main benefits of our sieve is that it can combine many PTE solutions into the same search and rapidly process them together. Many PTE solutions tend to share at least one non-zero element in common, and if checking this element returns a '0', all such solutions can be discarded at once. In what follows we describe a method to arrange the set of PTE solutions in a *tree*, such that (on average) a minimal number of checks is used to check the full set of solutions. Note that computing this tree is a one-time precomputation that is performed at initialization.

Suppose we have t solutions, written as $[a_{i,1}, \ldots, a_{i,n}] =_{n-1} [b_{i,1}, \ldots, b_{i,n}]$ for $1 \leq i \leq t$. Noting that $a_{i,1} = 0$ for all i, write $S_i = \{a_{i,2}, \ldots, a_{i,n}, b_{i,1}, \ldots, b_{i,n}\}$, i.e., S_i is the set of distinct non-zero integers in the i-th PTE solution. Now, as in the single solution sieve above, suppose we have advanced ℓ to a set bit at some stage of our sieving algorithm. Rather than checking each of the PTE solutions individually, we would like to share any checks that are common to multiple PTE solutions. The key observation is that we are highly unlikely[28] to have a PTE solution whose elements all correspond to '1', so in combining many PTE solutions we would ultimately like to minimize the number of checks required before we can rule all of them out and move ℓ to the next set bit.

In looking for the minimum number of checks whose failures rule out all PTE solutions, we are looking for a set H of minimal cardinality such that $H \cap S_i \neq \{\emptyset\}$ for $1 \leq i \leq t$,

[28] We assume that the smoothness bound is aggressive enough to make the smooth integers sparse.

i.e., the smallest-sized set that shares at least one element with each of the PTE solutions. Finding this set is an instance of the *hitting set problem*; this problem is NP-complete in general, but for the sizes of the problem in this paper, a good approximation is given by the greedy algorithm [98]. We start by looking for the element that occurs most among all of the S_i, call this g_1; we then look for the element that occurs most among the S_i that do not contain g_1, call this g_2; we then look for the element that occurs most among those S_i that do not contain g_1 or g_2, and continue in this way until we have $H = \{g_1, g_2, \ldots, g_h\}$ such that every S_i contains at least one of the g_j, for $1 \leq i \leq t$ and $1 \leq j \leq h$. This process naturally partitions the PTE solutions to fall under h different *branches*. For each PTE solution in a given branch, the corresponding element of the hitting set is removed and the process is repeated recursively until there is no common element between the remaining solutions, at which point they become *leaves*. In Section 7.5.2 we give a toy example with 20 PTE solutions that produces the tree in Figure 7.2. In this example the first hitting set is $\{1, 2\}$; if a search was to use these 20 solutions, then most of the time only two checks will be required before ℓ can be advanced to the next set bit.

At a high level, our multi-solution sieve then runs the same way as the single solution sieve in Section 7.4.2, except that we must traverse our tree each time ℓ is advanced. We do this by checking all of the elements of a the hitting set, and we only enter the branch corresponding to a given element if the associated check finds a '1' (an example sequence of checks is included in Section 7.5.2). This is repeated recursively until we either encounter a leaf, where we simply check the remaining elements sequentially, or until all of the elements in the hitting set at the current level of the tree return a '0', at which point we can move up to the branch above and continue. As mentioned above, in practice the most common scenario is that all of the elements in the highest hitting set correspond to a '0', and the number of checks performed in order to rule out the full set of PTE solutions is minimal. Note that checking the divisibility of $a(\ell)$ and $b(\ell)$ by the constant C is, in practice, best left until the point where a match is found. Since solutions have different constants and different sets of suitable relations, it is not useful to incorporate modular relations into the sieving step of the multi-solution algorithm.

The efficiency of checking all PTE solutions simultaneously is therefore heavily dependent on the size of the first hitting set. In cases where we have many PTE solutions (see Section 7.3.2), the first hitting set can be used to decide which PTE solutions to search with. If a pre-existing set of PTE solutions has a hitting set H, then including any additional solutions that share at least one element with H incurs nearly no performance cost.

7.4.4 Success probabilities

In Table 7.3 we use Heuristic 1 to draw comparisons between our method of finding twin smooth integers and the prior methods discussed in Section 7.1.2. The entries in the table are the approximate smoothness bounds that should be used to give success probabilities of 2^{-20}, 2^{-30}, 2^{-40}, and 2^{-50}. The term *success probability* is used to estimate how large

a search space needs to be covered before we can expect to find twin smooth integers; these probabilities are computed directly via Equation (7.4). For example (refer to the bold element in the last row of the table), using *one* PTE solution with $n = 8$ and a smoothness bound of $B \approx 2^{26.9}$, we can expect to find a pair of twin smooth numbers in $[1, N] = [1, 2^{384}]$ after searching roughly 2^{20} inputs $\ell \in [1, N^{1/n}] = [1, 2^{48}]$, for which $a_C(\ell)$ and $b_C(\ell)$ are integers.[29] To find similarly sized twin smooth integers using the XGCD approach, we would have to search roughly 2^{20} elements with a smoothness bound of $B \approx 2^{41.5}$, or 2^{30} elements with a smoothness bound of $B \approx 2^{32.8}$; on the other hand, if we were using XGCD with the same $B \approx 2^{26.9}$ as the PTE solution, we should expect to have to search a space larger han 2^{40} before finding twin smooths.

We stress that Table 7.3 is merely intended as a rough guide to the size of the smoothness bounds we should use in a given search, and similarly to provide an approximate comparison between the methods. As mentioned in Section 7.2, Heuristic 1 makes the rather strong assumption that the elements in our PTE solutions are uniform in $[1, N^{1/n}]$, and using the Dickman–de Bruijn function is a rather crude blanket treatment of the concrete combinations of B, N and n of interest to us. Moreover, the best version of our sieve (like the one used in Section 7.6) combines hundreds of PTE solutions into one search, and extending a theoretical analysis to cover such a collection of solutions is unnecessary. We point out that the application of Heuristic 1 to our scenario further assumes that the denominator C gets absorbed by the different factors uniformly. In other words, we assume that after canceling the denominator, all factors of $a_C(\ell)$ and $b_C(\ell)$ roughly have the same size. Although this is not true in general, our experiments and the smoothness of C (see Section 7.3.1) suggest this to be a good approximation for the average case.

The elements of the table that are faded out correspond to instances of the respective search method where the size of the possible search space is not large enough to expect to find solutions with the given probability. Moreover, Table 7.3 does not incorporate the additional probabilities associated with the twin smooth integers having a prime sum. Searches for cryptographic parameters typically need to find several twin smooth integers before finding a pair with a prime sum, so our search spaces tend to be a little larger than Table 7.3 suggests.[30] We chose 2^{-20} as the largest success probability in the table under the assumption that any search for twin smooth integers will cover a space of size at least 2^{20}.

A number of rows in the lower section of the table are marked (*) to indicate that these are PTE solutions with repeated factors. Viewing Heuristic 1, we see that these solutions find twin smooth integers with a higher probability than those PTE solutions without re-

[29] The total number of inputs required for this (including the ones which lead to non-integer polynomial values) depends on the PTE solution and associated constant in use, and can easily be computed via the CRT approach described before.

[30] It is beyond the scope of this work to make any statements about the probability of a prime sum, except to say that in practice we observe that twin smooth sums have a much higher probability of being prime than a random number of the same size.

Table 7.3: Table of smoothness bounds and success probabilities for prior methods and our method. All numbers are given as base-2 logarithms. Further explanation in text.

method	N														
	256					384					512				
	n	probability				n	probability				n	probability			
		-50	-40	-30	-20		-50	-40	-30	-20		-50	-40	-30	-20
naïve	–	20.2	23.4	28.4	36.7	–	30.2	35.2	42.6	55.1	–	40.3	46.9	56.7	73.4
XGCD	–	15.9	18.4	21.9	27.7	–	23.9	27.5	32.8	41.5	–	31.9	36.7	43.7	55.3
$2x^n - 1$	4	15.6	17.8	20.8	25.8	6	19.9	22.6	26.4	32.3	6	26.6	30.1	35.2	43.1
	6	13.3	15.1	17.6	21.6	8	20.4	23.2	27.2	33.8	12	22.0	24.9	28.9	35.2
	8	13.6	15.5	18.2	22.5	10	20.3	23.1	27.2	33.8	16	25.8	29.3	34.6	43.5
	9	15.4	17.7	21.0	26.4	12	16.5	18.7	21.7	26.4	18	23.3	26.3	30.9	38.4
	10	13.5	15.4	18.2	22.5	16	19.3	22.0	25.9	32.7	20	23.2	26.3	31.0	38.5
	12	11.0	12.4	14.5	17.6	18	17.4	19.8	23.1	28.8	24	20.2	22.9	26.7	32.8
PTE	3	20.4	23.0	26.6	32.2	3	30.6	34.5	39.9	48.4	4*	30.6	34.5	39.9	48.4
	3*	16.2	18.4	21.6	26.6	3*	24.3	27.7	32.4	39.9	5	31.9	25.6	40.6	48.2
	4	17.8	20.0	22.9	27.5	4	26.7	29.9	34.4	41.2	6	29.1	32.2	36.6	43.0
	4*	15.3	17.2	20.0	24.2	4*	22.9	25.8	29.9	36.3	6*	25.2	28.2	32.2	38.5
	5	16.0	17.8	20.3	24.1	5	24.0	26.7	30.4	36.1	7	26.8	29.6	33.5	39.0
	6	14.5	16.1	18.3	21.5	6	21.8	24.2	27.5	32.3	8	24.9	27.5	30.9	35.8
	6*	12.6	14.1	16.1	19.3	6*	18.9	21.1	24.2	28.9	9	23.3	25.7	28.7	33.2
	7	13.4	14.8	16.7	19.5	7	20.1	22.2	25.1	29.3	10	22.0	24.1	26.8	31.1
	8	12.5	13.7	15.4	17.9	8	18.7	20.6	23.2	**26.9**	12	19.8	21.5	23.9	27.5

peated factors, which is why they show a lower smoothness bound (for a fixed probability). PTE solutions with repeated factors are only known for $n \in \{3, 4, 6\}$.

7.5 A worked example

We now give concrete examples found with the sieve described in Section 7.4, referring back to the theory developed in Section 7.3 where applicable. We first illustrate a simple search that uses a single PTE solution, and then move to combining many PTE solutions into the same sieve.

7.5.1 Searching with a single PTE solution

Suppose we are searching for twin smooth integers $(m, m+1)$ with $2^{240} \leq m < 2^{256}$. Table 7.3 suggests that the best chances of success are with $n \in \{6, 7, 8\}$, and in particular with the $n = 6$ solutions that have repeated factors. Since the search spaces using polynomials of degree $n = 7$ and $n = 8$ are rather confined when targeting $m < 2^{256}$ (see Table 7.3), for this example we use a PTE solution of size $n = 6$ containing repeated factors, namely

$$[1, 1, 8, 8, 15, 15] =_5 [0, 3, 5, 11, 13, 16], \tag{7.7}$$

which corresponds to the polynomials

$$a(x) = (x-1)^2(x-8)^2(x-15)^2, \quad b(x) = x(x-3)(x-5)(x-11)(x-13)(x-16).$$

Proposition 6 induces that $a(x)$ and $b(x)$ differ by an integer constant, which in this case is

$$C = a(x) - b(x) = 14400 = 2^6 3^2 5^2.$$

Observe that Proposition 7 guaranteed that C was a multiple of $(n-1)! = 5!$.

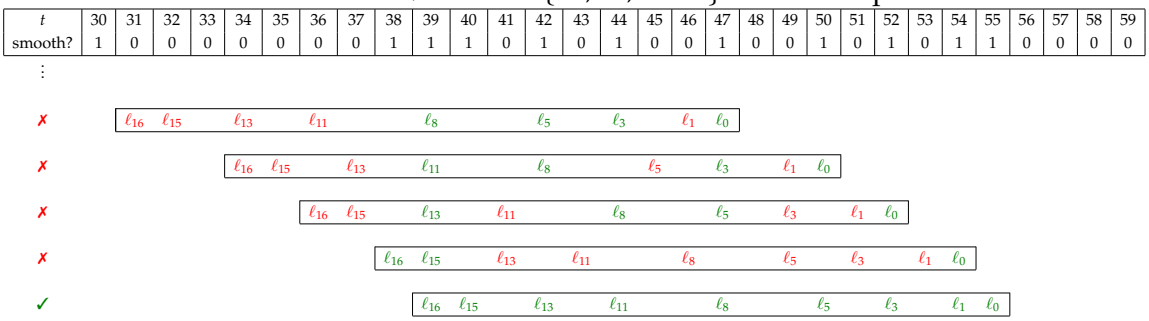
Given that $2^{13} < C < 2^{14}$, searching for m with $2^{240} \leq m < 2^{256}$ means searching for values ℓ such that $a(\ell)$ and $b(\ell)$ lie between 2^{254} and 2^{269}, so that $a_C(\ell)$ and $b_C(\ell)$ are then of the right size. Since $a(x)$ and $b(x)$ have degree 6, this means searching with $2^{42} \leq \ell < 2^{45}$.

Recall from Section 7.4 that our sieving algorithm alternates between two main phases. The first is independent of the PTE solution(s) we are searching with, and simply involves identifying all smooth numbers in a given interval (see Section 7.4.1). In this example, we chose interval sizes of $2^{20} = 1048576$, so at the conclusion of this first phase, we have a bitstring of length 1048576 to search over: a '1' in this string means the number associated with its index is B-smooth, while a '0' indicates that it is not.

With $B \approx 2^{16.1}$, Table 7.3 suggests that searching with the PTE solution in (7.7) will find twin smooth integers for roughly 1 in every 2^{30} values of ℓ that are tried. Thus, we set $B = 2^{16}$ and started the search at $\ell = 2^{42}$. With this ℓ and B, the Dickman–de Bruijn function tells us that we can expect the proportion of B-smooth numbers to be close to $\rho(42/16) \approx 0.103$.

At the top of Figure 7.1, we give 30 bits of an interval (found after sieving for some time) that correspond to $\ell = 5170314186700 + t$, for $t \in \{30, 31, \ldots, 59\}$. Here 11 of the 30 bits are 1, so the proportion of B-smooth numbers in this small interval is exceptionally high; indeed, these are the kinds of substrings we are sieving for, in hope that our PTE solution aligns favorably to find 1's in all of the required places. Viewing (7.7), we write $\ell_i = \ell - i$ for $i \in \{0, 1, 3, 5, 8, 11, 13, 15, 16\}$. As depicted in Figure 7.1, each step in the second phase starts by finding the next smooth number (i.e., the next '1' in the string), advancing $\ell = \ell_0$ to align there before sequentially checking from ℓ_1 through to ℓ_{16}. If, at any stage, one of the ℓ_i is aligned with a '0', we advance ℓ to the next '1' in the string

Figure 7.1: Sieving with the PTE solution $[1,1,8,8,15,15] =_5 [0,3,5,11,13,16]$ across the subinterval $\ell = 5170314186700 + t$ for $t \in \{30,31,\ldots 59\}$. Further explanation in text.



and repeat the procedure. Once we have finished processing a full interval (of size 2^{20} in this case), we advance to the next interval by first computing the string that identifies all B-smooth numbers, then processing the interval by aligning ℓ_0 with the next set bit, and checking the remaining ℓ_i.

In Figure 7.1 we see that when $\ell_0 = 5170314186747$, the next bit checked reveals that ℓ_1 corresponds to a '0', so this position is immediately discarded and we advance to the next set bit taking $\ell_0 = 5170314186750$. Again, ℓ_1 discovers a '0', so ℓ_0 advances to 5170314186752, and then to 5170314186754 (both of these also have ℓ_1 aligned with '0'). Advancing to $\ell_0 = 5170314186755$, we see that the remaining ℓ_i correspond to set bits and are thus all smooth, namely

$$\begin{aligned}
\ell_0 &= 5 \cdot 29 \cdot 31 \cdot 211 \cdot 557 \cdot 9787, & \ell_1 &= 2 \cdot 71 \cdot 919 \cdot 1237 \cdot 32029,\\
\ell_3 &= 2^{12} \cdot 11^2 \cdot 13 \cdot 277 \cdot 2897, & \ell_5 &= 2 \cdot 3 \cdot 5^3 \cdot 181 \cdot 4783 \cdot 7963,\\
\ell_8 &= 3^2 \cdot 23 \cdot 41 \cdot 83 \cdot 1117 \cdot 6571, & \ell_{11} &= 2^3 \cdot 3 \cdot 7^2 \cdot 17 \cdot 43 \cdot 191 \cdot 31489,\\
\ell_{13} &= 2 \cdot 103 \cdot 1093 \cdot 2663 \cdot 8623, & \ell_{15} &= 2^2 \cdot 5 \cdot 1163 \cdot 11927 \cdot 18637,\\
\ell_{16} &= 13 \cdot 53 \cdot 113 \cdot 3347 \cdot 19841.
\end{aligned}$$

The PTE solution (7.7) translates into the twin-smooth numbers

$$(m, m+1) = \left(\frac{\ell_0 \ell_3 \ell_5 \ell_{11} \ell_{13} \ell_{16}}{C}, \frac{(\ell_1 \ell_8 \ell_{15})^2}{C} \right).$$

In this case their sum is a prime p, which lies between the B-smooth numbers $2m$ and $2(m+1)$, namely

$$\begin{aligned}
p &= 2m + 1\\
&= 26531946489131985387630288088472672221025647530300250331041227602234368 01.
\end{aligned}$$

Remark 18. When searching with a single solution, in practice we only want to search over the $\ell \in \mathbb{Z}$ for which $a(\ell) \equiv b(\ell) = 0 \bmod C$. As described in Section 7.3, we use the

CRT to find these ℓ by first working modulo each of the prime power factors of C. In this case we find

- 40 residues $r_1 \in [0, 2^6)$ such that $a(\ell) \equiv b(\ell) \equiv 0 \bmod 2^6$ iff $\ell \equiv r_1 \bmod 2^6$;
- 9 residues $r_2 \in [0, 3^2)$ such that $a(\ell) \equiv b(\ell) \equiv 0 \bmod 3^2$ iff $\ell \equiv r_2 \bmod 3^2$;
- 15 residues $r_3 \in [0, 5^2)$ such that $a(\ell) \equiv b(\ell) \equiv 0 \bmod 5^2$ iff $\ell \equiv r_3 \bmod 5^2$.

Here we see that $a(\ell) \equiv b(\ell) \equiv 0 \bmod 3^2$ for all $\ell \in \mathbb{Z}$ (this can be seen immediately by looking at the expression for $a(x)$ above), so we can ignore the factor of 3^2 and work with the effective denominator $C' = 2^6 5^2 = 1600$. Of the 1600 possible residues in $[0, 2^6 3^5)$, we only search over the $40 \cdot 15 = 600$ values of ℓ that will produce $a(\ell) \equiv b(\ell) \equiv 0 \bmod C'$. In this case the list of residues is small enough that we can simply store them once and for all and avoid recomputing them on the fly with the CRT at runtime. However, many of the PTE solutions we use have much larger denominators and a much smaller proportion of residues to be searched over, and in these cases storing residues modulo each prime power and then using the CRT on the fly is much faster than looking up the full set of residues (modulo C) in one huge table.

For ease of exposition, we ignored this in the above example. Returning to Figure 7.1, we point out that none of the four values that were checked prior to finding the solution (i.e., $\ell = 5170314186700 + t$ with $t \in \{47, 50, 52, 54\}$) are such that $a(\ell) \equiv b(\ell) \equiv 0 \bmod C$. In fact, none of the other smooth ℓ's depicted in Figure 7.1 have this property; the previous smooth ℓ that does is $\ell = 5170314186728$, so in practice we would have advanced straight from this ℓ to the successful one.

Remark 19. Since the degree of a and b is even, negative values for ℓ will lead to valid positive twin smooth integers and possibly a corresponding prime sum. Negative values can be taken into account by considering the flipped solution (as defined at the end of Section 7.3.1). Because the solution considered here is symmetric, any pattern corresponding to a negative value also occurs for a positive value.

7.5.2 Sieving with many PTE solutions

We now turn to illustrating the full sieving algorithm that combines many PTE solutions into one search. The degree 6 sieves we used in practice combined hundreds of PTE solutions into one search (see Table 7.2), but for ease of exposition we will illustrate using the first 20 solutions (ordered by the size of the constant). These range from the solution S_1, which has $C = 14400 = 2^6 \cdot 3^2 \cdot 5^2$, to S_{20}, which has $C = 13305600 = 2^8 \cdot 3^3 \cdot 5^2 \cdot 7 \cdot 11$. These solutions are listed below.

$S_1 : [0,3,5,11,13,16] =_5 [1,1,8,8,15,15]$; $S_2 : [0,5,6,16,17,22] =_5 [1,2,10,12,20,21]$,
$S_3 : [0,4,9,17,22,26] =_5 [1,2,12,14,24,25]$, $S_4 : [0,7,7,21,21,28] =_5 [1,3,12,16,25,27]$,
$S_5 : [0,7,8,22,23,30] =_5 [2,2,15,15,28,28]$, $S_6 : [0,5,13,23,31,36] =_5 [1,3,16,20,33,35]$,
$S_7 : [0,8,9,25,26,34] =_5 [1,4,14,20,30,33]$, $S_8 : [0,7,11,25,29,36] =_5 [1,4,15,21,32,35]$,
$S_9 : [0,9,11,29,31,40] =_5 [1,5,16,24,35,39]$, $S_{10} : [0,8,11,27,30,38] =_5 [2,3,18,20,35,36]$,
$S_{11} : [0,5,16,26,37,42] =_5 [2,2,21,21,40,40]$, $S_{12} : [0,6,17,29,40,46] =_5 [1,4,20,26,42,45]$,
$S_{13} : [0,7,14,28,35,42] =_5 [2,3,20,22,39,40]$, $S_{14} : [0,10,13,33,36,46] =_5 [1,6,18,28,40,45]$,
$S_{15} : [0,9,17,34,36,46] =_5 [1,6,24,25,42,44]$, $S_{16} : [0,9,14,32,37,46] =_5 [2,4,21,25,42,44]$,
$S_{17} : [0,9,16,34,41,50] =_5 [1,6,20,30,44,49]$, $S_{18} : [0,11,15,37,41,52] =_5 [1,7,20,32,45,51]$,
$S_{19} : [0,7,21,35,49,56] =_5 [1,5,24,32,51,55]$, $S_{20} : [0,12,13,37,38,50] =_5 [2,5,22,28,45,48]$.

In regards to Remark 18, recall from Section 7.4 that each PTE solution has a different constant C and thus a different set of residues. In general these residues are incompatible with one another, so we choose to ignore them until the sieve identifies candidate pairs (ℓ, S_i), at which point we only mark the pair as a solution if the corresponding polynomials have $a(\ell) \equiv b(\ell) \equiv 0 \bmod C$.

Now, recall from Section 7.4 that our sieving tree is built by recursively identifying *hitting sets* among the set of solutions, and then removing the corresponding element in the hitting set from each solution. The first hitting set is (always) $\{0\}$, which is the root of our tree. After removing 0 from all of the solutions, we see that the next hitting set is $\{1,2\}$; some PTE solutions contain both 1 and 2, but 1 appears in more solutions than 2 does, so the solutions S_2 and S_3 occur in the branches that fall beneath 1 in the tree. Repeating this process produces the tree in Figure 7.2. Note that this is a precomputation that is done once-and-for-all before the sieve begins.

Again we target $2^{240} \leq m < 2^{256}$ by searching with $2^{42} \leq \ell < 2^{45}$, set our smoothness bound as $B = 2^{16}$, and alternate between identifying the B-smooth numbers in intervals of size $2^{20} = 1048576$, processing each interval by advancing through all of the set bits (smooth numbers) within it. Write $\ell_i = \ell - i$ as before. Here the hitting set has only two elements, so given that the probability of smoothness is roughly $\rho(42/16) \approx 0.103$, most of the time we will only need to check two neighboring bits (ℓ_1 and ℓ_2) before discarding each candidate ℓ.

Viewing Figure 7.2, we traverse the tree by moving down the levels and processing each subsequent hitting set from left to right. If, at any stage, we find a smooth number, we immediately move down a level and process the numbers branching beneath it. We are only permitted to move up a level and continue to the right once the entire hitting set at a given level has been checked. Finally, if at any stage we arrive at a leaf and find that all of the remaining numbers are smooth, we then identify this solution as a candidate. At this stage we check whether $a(\ell) \equiv b(\ell) \equiv 0 \bmod C$, in which case we have found twin

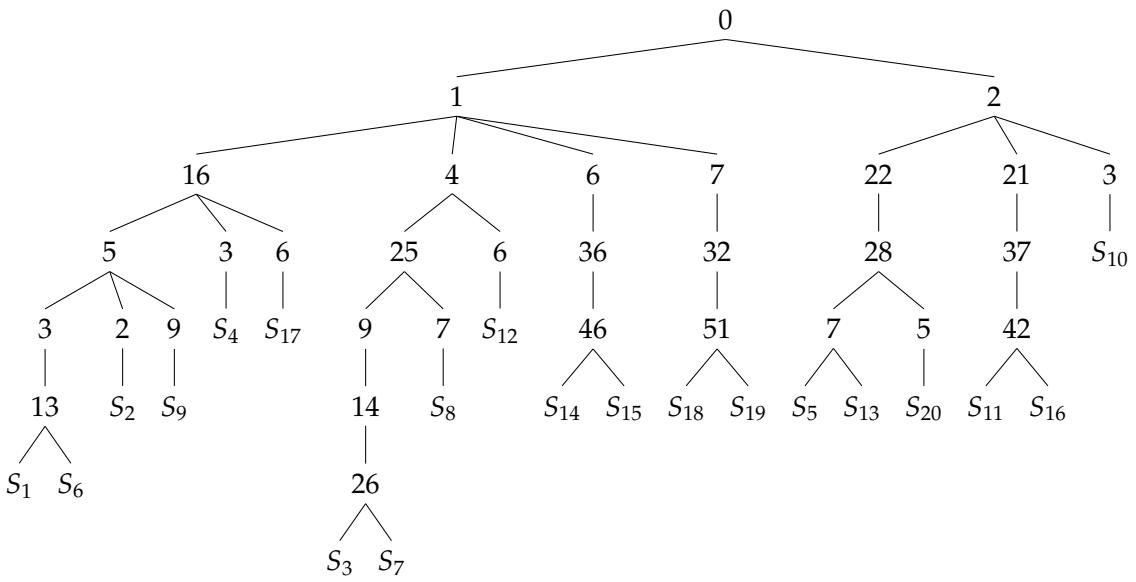


Figure 7.2: A sieving tree for 20 example PTE solutions. Further explanation in text.

smooth integers, and then optionally check whether their sum is a prime, in which case we have found cryptographically suitable parameters.

After some time, our sieve advances to the B-smooth number

$$\ell_0 = 5435932476400 = 2^4 \cdot 5^2 \cdot 199 \cdot 4817 \cdot 14177.$$

In this case the subsequent set of ordered checks made in traversing the tree in Figure 7.2 are given below (we use ✓ to indicate that ℓ_i is B-smooth, ✗ otherwise). Checking the entire leaf marked S_{17} is combined into Check 5 for brevity; the remaining values here are ℓ_i with $i \in \{9, 20, 30, 34, 41, 44, 49, 50\}$.

Check 1.	ℓ_1 ✓	*Check 2.*	ℓ_{16} ✓	*Check 3.*	ℓ_5 ✗	*Check 4.*	ℓ_3 ✗
Check 5.	S_{17} ✓	*Check 6.*	ℓ_4 ✗	*Check 7.*	ℓ_6 ✓	*Check 8.*	ℓ_{36} ✗
Check 9.	ℓ_7 ✗	*Check 10.*	ℓ_2 ✗				

At the conclusion of Check 5, we now know that all of the elements in

$$S_{17} \colon [0, 9, 16, 34, 41, 50] =_5 [1, 6, 20, 30, 44, 49]$$

are smooth, and thus we have found a candidate solution. Checks 6–10 are included to show how the sieve continues. It remains to check whether $\ell = 5435932476400$ gives $a(\ell) \equiv b(\ell) \equiv 0 \bmod C$, when

$$a(x) = x(x-9)(x-16)(x-34)(x-41)(x-50)$$

and

$$b(x) = (x-1)(x-6)(x-20)(x-30)(x-44)(x-49).$$

are such that $C = 7761600$. In this case we do find that $a(\ell) \equiv 0 \bmod C$ (which is sufficient), so we know that

$$m = \ell_0\ell_9\ell_{16}\ell_{34}\ell_{41}\ell_{50}/C \qquad \text{and} \qquad m+1 = \ell_1\ell_6\ell_{20}\ell_{30}\ell_{44}\ell_{49}/C$$

are both B-smooth integers. Indeed, factoring reveals that

$$\begin{aligned} m &= 2^5 \cdot 3^4 \cdot 5^2 \cdot 109 \cdot 173 \cdot 199 \cdot 233 \cdot 571 \cdot 677 \cdot 743 \cdot 1303 \cdot 2351 \cdot 2729 \\ &\quad \cdot 3191 \cdot 4817 \cdot 12071 \cdot 12119 \cdot 14177 \cdot 16979 \cdot 30389 \cdot 37159 \cdot 39979, \text{ and} \\ m+1 &= 13 \cdot 17 \cdot 23 \cdot 31 \cdot 61 \cdot 103 \cdot 263 \cdot 643 \cdot 1153 \cdot 1429 \cdot 1889 \cdot 2213 \cdot 3359 \\ &\quad \cdot 5869 \cdot 7951 \cdot 9281 \cdot 18307 \cdot 28163 \cdot 34807 \cdot 41077 \cdot 41851 \cdot 64231. \end{aligned}$$

In this case $2m+1$ is the product of two large primes, so a sieve for cryptographic parameters would continue by advancing to the next smooth ℓ_0 in the interval.

7.6 Cryptographic examples of twin smooth integers

We implemented the sieve including the tree structure for searching with multiple PTE solutions in Python 3.8 and used it to run our experiments. The code takes as input the left and right bounds of a desired interval to be searched, a size for the sub-intervals that are processed by the sieve at a time, as well as a smoothness bound and a list of PTE solutions. It then computes the PTE solution search tree and starts the sieve as described in Sections 7.4 and 7.5. Another input is a desired number of threads, between which the interval is divided and then run on the available processors in a multi-processing fashion.

After examining the PTE solution counts in Table 7.2 and the smoothness probabilities in Table 7.3, we chose to launch a sieve with 520 PTE solutions of size $n = 6$ that searched $\ell \in [2^{40}, 2^{45}]$ with a smoothness bound of $B = 2^{16}$ and intervals of size 2^{20}. The 520 solutions are all the ones we found that have a constant of at most 38 bits. The first hitting set of the PTE solutions had cardinality 13, and the Dickman–de Bruijn function estimates that the proportion of B-smooth numbers in our interval is $\rho(45/16) \approx 0.0715$. The search ran on 128 logical processors (Intel Xeon CPU E5-2450L @1.8GHz) for just over a week before the entire interval was scanned.

Table 7.4 reports one of the cryptographic primes that was found with our sieve for each bitlength between 240 and 255 (excluding 251, 253 and 254, for which no primes were found), and compares it to the primes found with prior methods in the literature. For the primes found using PTE solutions, we give the search parameter ℓ together with

the corresponding PTE solution, which is one of

$$\begin{aligned}
S_1&: [0,3,5,11,13,16] =_5 [1,1,8,8,15,15],\\
S_2&: [0,7,8,22,23,30] =_5 [2,2,15,15,28,28],\\
S_3&: [0,7,33,47,73,80] =_5 [3,3,40,40,77,77],\\
S_4&: [0,11,24,46,59,70] =_5 [4,4,35,35,66,66],\\
S_5&: [0,5,16,26,37,42] =_5 [2,2,21,21,40,40].
\end{aligned}$$

For each prime we report the smoothness bound B, which is the largest prime divisor of $(p-1)(p+1)$, together with its bitlength. In the case of the 241- and 250-bit primes, we see that $B < 2^{15}$. The smallest prior B corresponding to primes of around this size was the 19-bit $B = 486839$ from [44]. Referring back to Table 7.3, we see that a search through an interval of this size should find a few twin smooth integers with $B < 2^{15}$, but finding enough twin smooths with $B < 2^{14}$ to hope for a prime sum among them may have been out of the question.

To check whether $n = 6$ produces the smoothest twins of this size (as Table 7.3 predicts), we ran similar sieves using the 8 PTE solutions with $n = 7$ and the 51 PTE solutions with $n = 8$ with $B = 2^{18}$, and in both cases we covered the full range of possible inputs that would produce a $p < 2^{256}$. Despite finding a handful of twin smooth integers with $B < 2^{17}$, the search spaces were not large enough to find any primes among them.

As future work, we will be launching sieves that target $N = 2^{384}$ and $N = 2^{512}$. A small search with degree 6 found some example primes whose neighbors are 2^{24}-smooth, e.g. the 384-bit prime obtained with S_1 above for $\ell = 73786976891204925015$, but Table 7.3 suggests that the degree 8 PTE solutions will have a better chance of success at this level.

7.7 Relaxations and modifications

There are numerous ways to modify our sieving approach for performance reasons, or to relax the search conditions in order to precisely match the security requirements imposed by B-SIDH or SQISign.

Approximate sieves. There are several sieving optimizations discussed in [55, §3.2.5–3.3] that can be applied to the sieving phase of our algorithm. For large scale searches, it could be preferred to sacrifice the exactness of the sieve we implemented for more performant approximate sieves. For example, the smallest primes are the most expensive to sieve with due to the large number of multiplications. Thus, an approximate sieve can choose to skip these small primes (but still include the larger prime powers) and choose to tag numbers as being B-smooth as soon as the result is close enough to the expected number. This requires to choose an error bound, which also determines if and how many false positive and false negative results are going to occur.

Table 7.4: A comparison between some of the best instances found with our sieve and the best instances from the literature. Further explanation in text.

method	where		p (bits)	B	$\lceil \log_2 B \rceil$
XGCD	[13, App. A]		256	6548911	**23**
$p = 2x^n - 1$	[44, Ex. 5]		247	652357	**20**
	[44, Ex. 6]		237	709153	**20**
	[44, Ex. 7]		247	745309897	**30**
	[44, Ex. 8]		250	486839	**19**
PTE sieve	19798693013832	S_3	240	54503	**16**
	5170314186755	S_1	241	32039	**15**
	24924102654286	S_4	242	62501	**16**
	6781477697051	S_1	243	59557	**16**
	32519458118257	S_3	244	64591	**16**
	18675743644600	S_5	245	57457	**16**
	9244655038011	S_1	246	63803	**16**
	20173246926702	S_2	247	40289	**16**
	22687888853658	S_2	248	59497	**16**
	26042586248980	S_2	249	53959	**16**
	36144284257450	S_5	250	32191	**15**
	19052682871080	S_1	252	48733	**16**
	27071078665441	S_1	255	52069	**16**

A standard approach for sieving algorithms is discussed by Crandall and Pomerance [55, §3.2.5]. This approach replaces multiplications by additions in Eratosthenes-like sieves by choosing to represent numbers as their base-2 logarithms. Moreover, sieves can use approximate logarithms, i.e., round these logarithms to nearby integers and tolerate errors in the logarithms; for example, if we choose to tolerate errors up to $\log B$, then we are guaranteed that factors that are unaccounted for in the approximation are also less than the smoothness bound [55, p. 124]. Rather than accumulating products, we are then accumulating sums of relatively small integers. We have experimented with using approximate logarithms, but were not able to notice an improvement in our implementation when using this technique. This is probably due to our use of Python and the fact that all numbers we considered in our experiments were small enough to fit in a 64-bit machine word. A

more aggressively tuned implementation (e.g. a fine-grained C implementation) will most likely gain a substantial speedup in this case.

Recall from Section 7.4.2 that when a single PTE solution is used we are only interested to sieve the subset of integers for which $a(\ell) \equiv b(\ell) \equiv 0 \bmod C$. In this case it may be preferable to employ Bernstein's batch smoothness algorithm [8]; this can be used to gain a better overall complexity (per element) when sieving through an arbitrary set.

Lastly, we point out that the set of primes used in the factor base can be tailored to our needs. For example, if future research reveals that certain types of prime isogeny degrees are favored over others (i.e., when invoking the $\widetilde{O}(\sqrt{\ell})$ algorithm from [13]), then it may be preferable to increase the bound B and only include those primes in our sieve.

Non-smooth cofactors vs. fully smooth numbers. The security analyses of B-SIDH or SQISign suggest that both systems can tolerate a non-smooth cofactor in either or both of $p-1$ and $p+1$. In these cases, relaxing conditions in the second part of our sieve to allow non-smooth cofactors is straightforward. When searching with PTE solutions of size n, we could e.g. only require $n-1$ of the factors on each side to be B-smooth. The naïve way to do this when traversing the tree would be to incorporate a counter that only allows branches to be discarded when two non-smooth numbers have been discovered, but this approach makes things unnecessarily complicated and significantly slower, e.g. it no longer suffices to start the sieving procedure at each '1' in the interval, since ℓ_0 is now allowed to be non-smooth.

A much better approach can be taken by simply creating many relaxed PTE solutions from the original solution $\mathcal{A} =_{n-1} \mathcal{B}$, and including them in the solution tree. For example, if the security analysis corresponding to a search with $n = 6$ suggests we only need 5 smooth factors from each side of the PTE solution, then the solution $[0, 7, 11, 25, 29, 36] =_5 [1, 4, 15, 21, 32, 35]$ can be modified into 36 relaxed solutions, each of which corresponds from wiping out one number from $\mathcal{A}$ and one number from $\mathcal{B}$; these new solutions only include 10 distinct elements. By building a tree from these solutions and running the same algorithm as in Section 7.4, we are effectively allowing for one of the factors of the original solution to be non-smooth. The only minor modification required appears when 0 is wiped out from a solution, in which case we have to shift all elements such that the new solution contains 0, by the means of Proposition 8. We reiterate that all of these modifications are a one-time precomputation before the sieve begins. In the case of the PTE solutions with repeated factors, e.g. $[0, 3, 5, 11, 13, 16] =_5 [1, 1, 8, 8, 15, 15]$, we may not be able to tolerate a non-smooth cofactor that would arise from removing any of 1, 8 or 15 from the PTE solution. On the other hand, if the security analysis does permit such a cofactor (which appears to be the case for SQISign), then our relaxed solutions would either remove one of the repeated numbers from $\mathcal{B}$, or two of the numbers from $\mathcal{A}$; the latter would have a better success probability, but (assuming the hitting set remains unchanged) our tree approach would not pay any noticable overhead by including all such relaxations.

7.8 Conclusion

In this chapter we have introduced a new method for searching for twin smooth integers, which can potentially be used for the instantiation of the isogeny-based primitives B-SIDH [44] and SQISign [62]. Our method uses solutions to the PTE problem, which give rise to fully split polynomials that differ by constants. Although our results outperform prior attempts to find such numbers, we mainly focused on the lowest security level, i.e., numbers of roughly 256 bit. It thus remains an open task to conduct similar searches for higher security levels.

Another open task is to adapt our searches to the exact requirements of B-SIDH or SQISign as described in Section 7.7, and run the corresponding searches. This might lead to parameters that outperform the ones found in this chapter when used in practice.

Conclusion

We briefly recall the contributions of this thesis and follow-up work, and discuss open problems and research directions in the context of the topics of this thesis.

The first part of this thesis aimed at providing efficient and side-channel resistant implementations of CSIDH. In particular, we improved the efficiency of the involved computations (Chapter 3), provided an efficient constant-time implementation (Chapter 4), and reviewed practical fault injection attacks, their impact, and countermeasures (Chapter 5).

As presented in Section 3.6, other approaches to compute isogenies have recently been proposed; Bernstein et al. described a faster method to compute large degree ℓ-isogenies with an asymptotic complexity of $\widetilde{O}(\sqrt{\ell})$, which outperforms earlier approaches already for degrees in the range of 100 [13]. On the other hand, small degree isogenies can be computed in a deterministic way [33], i.e., without the need for sampling suitable points for each isogeny, and 2-isogenies can be used in CSIDH when moving to the surface of the isogeny graph [32]. Although constant-time implementations of CSIDH using the isogeny formulas from [13] have been studied in [2], it remains an open task to implement CSIDH including all these speedups, and determine where the optimal crossover points for using the respective isogeny approaches lie. Furthermore, it remains an open question which algorithmic approach for constant-time implementations achieves the best performance when combined with optimized isogeny computations; i.e., if the optimal strategy approach from [39] still outperforms SIMBA-based approaches as described in Chapter 4 and used in [113, 122, 37] when small degree isogenies are computed deterministically, or if even better constant-time approaches can be found.

Moreover, it is an important task to study further practical side-channel attacks on different CSIDH implementations, as e.g. initiated in Chapter 5, in order to take a step towards real-world applications. On the other hand, also more analysis on the exact quantum security of CSIDH is required. A major obstacle for this currently is the ambiguous definition of quantum security levels, which allow several interpretations of the security achieved by a certain CSIDH parameter set.

In Chapter 6 we defined threshold encryption and signature protocols based on CSIDH [35] and CSI-FiSh [19]. These schemes achieve a relatively fast performance, but are only secure in the rather weak honest-but-curious adversary model. As explained in Section 6.5, this is mitigated in [54] by using zero-knowledge proofs, which however leads to a far

worse performance. Furthermore, [18] introduced a robust and actively secure method to generate keys in a distributed way.

An open problem in this area certainly is to find threshold protocols that achieve both better security features than the protocols from Chapter 6, and a faster performance than the follow-up protocols from [54, 18]. Further, these protocols require the respective class group structure to be known, which currently limits the practical application to the CSIDH-512 parameter set. Thus, it is a natural question if progress in this area can make the computation of larger class groups feasible.

Apart from threshold protocols, the unique features of the group action underlying CSIDH allow for many more advanced protocols. In [3] a framework based on group actions is given, which allows for the simple usage of various hardness assumptions from isogeny-based cryptography in advanced protocols. As an example this framework is used to define isogeny-based smooth projective hashing, dual-mode PKE, two-message statistically sender-private OT, and Naor-Reingold style PRF.

Chapter 7 discussed the question of how to set up the recent isogeny-based key exchange B-SIDH [44] and the signature scheme SQISign [62]. In particular, both schemes require a large prime p, for which (large factors of) $p+1$ and $p-1$ are as smooth as possible. We made use of the PTE problem to obtain pairs of fully split polynomials with constant differences, and explained how they can be used to search for such primes.

Although our results significantly improve earlier results from [44, 13, 62], there is no reason to conclude that our search method is optimal. Moreover, we mainly restricted to NIST level 1 parameters, which means that searches for larger parameters still have to be conducted. When targeting primes for a specific protocol, i.e., either B-SIDH or SQISign, one can adapt our PTE method to meet the exact requirements of this protocol, which might lead to better results than the general purpose searches conducted in this thesis.

In a nutshell, isogeny-based cryptography provides promising tools for the design of quantum-resistant primitives. While SIDH/SIKE resp B-SIDH allow for efficient and very compact key exchange and key encapsulation, isogeny group actions allow for defining many more advanced protocols. Thus, compared to other post-quantum approaches, isogeny-based cryptography has the advantage of small keys and signatures, and its applicability to many different protocols, while being at least an order of magnitude slower than other post-quantum schemes, and having relatively new hardness assumptions. Important tasks for future work are therefore given by efficiency improvements, both for improving existing primitives and finding new, more efficient protocols, as well as security analyses that strengthen the confidence in the hardness of the underlying isogeny problems.

Bibliography

[1] 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 C. Cid and M. J. Jacobson Jr., editors, *Selected Areas in Cryptography – SAC 2018*, pages 322–343. Springer, 2018.

[2] 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. Cryptology ePrint Archive, Report 2020/1109, 2020. https://eprint.iacr.org/2020/1109.

[3] Navid Alamati, Luca De Feo, Hart Montgomery, and Sikhar Patranabis. Cryptographic group actions and applications. In S. Moriai and H. Wang, editors, *Advances in Cryptology – ASIACRYPT 2020*, pages 411–439. Springer, 2020.

[4] Reza Azarderakhsh, David Jao, Kassem Kalach, Brian Koziel, and Christopher Leonardi. Key compression for isogeny-based cryptosystems. In *Proceedings of the 3rd ACM International Workshop on ASIA Public-Key Cryptography*, pages 1–10, 2016.

[5] László Babai. On Lovász' lattice reduction and the nearest lattice point problem. *Combinatorica*, 6(1):1–13, 1986.

[6] Antal Balog and Trevor D. Wooley. On strings of consecutive integers with no large prime factors. *J. Austral. Math. Soc. (Series A)*, 64:266–276, 1998.

[7] Daniel J. Bernstein. Arbitrarily Tight Bounds On The Distribution Of Smooth Integers. In *Proceedings of the Millennial Conference on Number Theory*, pages 49–66, 2002.

[8] Daniel J. Bernstein. How to find smooth parts of integers, 2004. http://cr.yp.to/papers.html#smoothparts.

[9] Daniel J. Bernstein. Curve25519: new Diffie-Hellman speed records. In M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, editors, *Public-Key Cryptography – PKC 2006*, pages 207–228. Springer, 2006.

[10] Daniel J. Bernstein, Peter Birkner, Mark Joye, Tanja Lange, and Christiane Peters. Twisted Edwards Curves. In S. Vaudenay, editor, *Progress in Cryptology - AFRICACRYPT 2008*, pages 389–405. Springer, 2008.

[11] Daniel J. Bernstein, Peter Birkner, Tanja Lange, and Christiane Peters. ECM Using Edwards Curves. *Mathematics of Computation*, 82(282):1139–1179, 2013.

[12] Daniel J. Bernstein, Johannes Buchmann, and Erik Dahmen. *Post-Quantum Cryptography*. Springer, 2009.

[13] Daniel J. Bernstein, Luca De Feo, Antonin Leroux, and Benjamin Smith. Faster computation of isogenies of large prime degree. To appear in ANTS-XIV, 2020. https://eprint.iacr.org/2020/341.

[14] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. *Journal of Cryptographic Engineering*, 2(2):77–89, 2012.

[15] 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*, pages 967–980. ACM, 2013.

[16] Daniel J. Bernstein and Tanja Lange. Explicit-Formulas Database. https://www.hyperelliptic.org/EFD/.

[17] Daniel J. Bernstein, Tanja Lange, Chloe Martindale, and Lorenz Panny. Quantum circuits for the CSIDH: optimizing quantum evaluation of isogenies. In Y. Ishai and V. Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019*, pages 409–441, 2019.

[18] Ward Beullens, Lucas Disson, Robi Pedersen, and Frederik Vercauteren. CSI-RAShi: Distributed key generation for CSIDH. Cryptology ePrint Archive, Report 2020/1323, 2020. https://eprint.iacr.org/2020/1323.

[19] Ward Beullens, Thorsten Kleinjung, and Frederik Vercauteren. CSI-FiSh: Efficient Isogeny Based Signatures Through Class Group Computations. In S. D. Galbraith and S. Moriai, editors, *Advances in Cryptology – ASIACRYPT 2019*, pages 227–247. Springer, 2019.

[20] Ward Beullens, Bart Preneel, Alan Szepieniec, and Frederik Vercauteren. LUOV. Round 2 submission, NIST Post-Quantum Cryptography Standardization, 2019. https://www.esat.kuleuven.be/cosic/pqcrypto/luov/.

[21] Jean-François Biasse, Annamaria Iezzi, and Michael J. Jacobson Jr. A note on the security of CSIDH. In D. Chakraborty and T. Iwata, editors, *Progress in Cryptology - INDOCRYPT 2018*, pages 153–168. Springer, 2018.

[22] Jean-François Biasse, David Jao, and Anirudh Sankar. A quantum algorithm for computing isogenies between supersingular elliptic curves. In W. Meier and D. Mukhopadhyay, editors, *Progress in Cryptology – INDOCRYPT 2014*, pages 428–442. Springer, 2014.

[23] Ian Blake, Gadiel Seroussi, and Nigel Smart. *Elliptic curves in cryptography*, volume 265 of *London Mathematical Society Lecture Note Series*. Cambridge University Press, 1999.

[24] George R. Blakley. Safeguarding cryptographic keys. In *1979 International Workshop on Managing Requirements Knowledge (MARK)*, pages 313–318. IEEE, 1979.

[25] Xavier Bonnetain and André Schrottenloher. Quantum security analysis of CSIDH. In A. Canteaut and Y. Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020*, pages 493–522. Springer, 2020.

[26] Peter Borwein. The Prouhet-Tarry-Escott Problem. In *Computational Excursions in Analysis and Number Theory*, pages 85–95. Springer, 2002.

[27] Peter Borwein and Colin Ingalls. The Prouhet-Tarry-Escott Problem Revisited. http://www.cecm.sfu.ca/personal/pborwein/PAPERS/P98.pdf.

[28] Luís T. A. N. Brandão, Nicky Mouha, and Apostol Vassilev. Threshold Schemes for Cryptographic Primitives: Challenges and Opportunities in Standardization and Validation of Threshold Cryptography. NISTIR 8214, 2018. https://nvlpubs.nist.gov/nistpubs/ir/2019/NIST.IR.8214.pdf.

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

[30] Timothy Caley. *The Prouhet-Tarry-Escott problem*. PhD thesis, University of Waterloo, 2012.

[31] Fabio Campos, Matthias J. Kannwischer, Michael Meyer, Hiroshi Onuki, and Marc Stöttinger. Trouble at the CSIDH: Protecting CSIDH with Dummy-Operations against Fault Injection Attacks. In *Workshop on Fault Detection and Tolerance in Cryptography – FDTC 2020*, pages 57–65. IEEE, 2020.

[32] Wouter Castryck and Thomas Decru. CSIDH on the surface. In J. Ding and J.-P. Tillich, editors, *Post-Quantum Cryptography – 11th International Conference, PQCrypto 2020*, pages 111–129. Springer, 2020.

[33] Wouter Castryck, Thomas Decru, and Frederik Vercauteren. Radical isogenies. In S. Moriai and H. Wang, editors, *Advances in Cryptology – ASIACRYPT 2020*, pages 493–519. Springer, 2020.

[34] Wouter Castryck, Steven Galbraith, and Reza Rezaeian Farashahi. Efficient arithmetic on elliptic curves using a mixed Edwards-Montgomery representation. Cryptology ePrint Archive, Report 2008/218, 2008. http://eprint.iacr.org/2008/218.

[35] Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. CSIDH: An efficient post-quantum commutative group action. In T. Peyrin and S. D. Galbraith, editors, *Advances in Cryptology – ASIACRYPT 2018*, pages 395–427. Springer, 2018.

[36] Wouter Castryck, Lorenz Panny, and Frederik Vercauteren. Rational isogenies from irrational endomorphisms. In A. Canteaut and Y. Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020*, pages 523–548. Springer, 2020.

[37] 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 P. Schwabe and N. Thériault, editors, *Progress in Cryptology – LATINCRYPT 2019*, pages 173–193. Springer, 2019.

[38] Denis X. Charles, Kristin E. Lauter, and Eyal Z. Goren. Cryptographic hash functions from expander graphs. *Journal of Cryptology*, 22(1):93–113, 2009.

[39] Jesús-Javier Chi-Domínguez and Francisco Rodríguez-Henríquez. Optimal strategies for CSIDH. Cryptology ePrint Archive, Report 2020/417, 2020. https://eprint.iacr.org/2020/417.

[40] Andrew Childs, David Jao, and Vladimir Soukharev. Constructing elliptic curve isogenies in quantum subexponential time. *Journal of Mathematical Cryptology*, 8(1):1–29, 2014.

[41] Sreeja Chowdhury, Ana Covic, Rabin Yu Acharya, Spencer Dupee, Fatemeh Ganji, and Domenic Forte. Physical Security in the Post-quantum Era: A Survey on Side-channel Analysis, Random Number Generators, and Physically Unclonable Functions. arXiv preprint arXiv:2005.04344, 2020. https://arxiv.org/abs/2005.04344.

[42] Craig Costello. Pairings for beginners, 2012. http://craigcostello.com.au/pairings/PairingsForBeginners.pdf.

[43] Craig Costello. Supersingular Isogeny Key Exchange for Beginners. In K. G. Paterson and D. Stebila, editors, *SAC 2019 – International Conference on Selected Areas in Cryptography*, pages 21–50. Springer, 2019.

[44] Craig Costello. B-SIDH: supersingular isogeny Diffie-Hellman using twisted torsion. In S. Moriai and H. Wang, editors, *Advances in Cryptology – ASIACRYPT 2020*, pages 440–463. Springer, 2020.

[45] Craig Costello and Huseyin Hisil. A simple and compact algorithm for SIDH with arbitrary degree isogenies. In T. Takagi and T. Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017*, pages 303–329. Springer, 2017.

[46] Craig Costello, David Jao, Patrick Longa, Michael Naehrig, Joost Renes, and David Urbanik. Efficient compression of SIDH public keys. In J. Coron and J. B. Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017*, pages 679–706. Springer, 2017.

[47] Craig Costello, Patrick Longa, and Michael Naehrig. Efficient algorithms for supersingular isogeny Diffie-Hellman. In M. Robshaw and J. Katz, editors, *Advances in Cryptology – CRYPTO 2016*, pages 572–601. Springer, 2016.

[48] Craig Costello, Patrick Longa, Michael Naehrig, Joost Renes, and Fernando Virdia. Improved Classical Cryptanalysis of SIKE in Practice. In A. Kiayias, M. Kohlweiss, P. Wallden, and V. Zikas, editors, *Public-Key Cryptography – PKC 2020*, pages 505–534. Springer, 2020.

[49] Craig Costello, Michael Meyer, and Michael Naehrig. Sieving for twin smooth integers with solutions to the Prouhet-Tarry-Escott problem. To appear in EUROCRYPT 2021. https://eprint.iacr.org/2020/1283.

[50] Craig Costello and Benjamin Smith. Montgomery curves and their arithmetic. *Journal of Cryptographic Engineering*, 8(3):227–240, 2018.

[51] Jean-Marc Couveignes. Hard Homogeneous Spaces. Cryptology ePrint Archive, Report 2006/291, 2006. https://eprint.iacr.org/2006/291.

[52] David A. Cox. *Primes of the form $x^2 + ny^2$: Fermat, class field theory, and complex multiplication*. Wiley, 2nd edition, 2013.

[53] Daniele Cozzo and Nigel P. Smart. Sharing the LUOV: threshold post-quantum signatures. In *Cryptography and Coding – 17th IMA International Conference, IMACC 2019*, pages 128–153. Springer, 2019.

[54] Daniele Cozzo and Nigel P. Smart. Sashimi: Cutting up CSI-FiSh secret keys to produce an actively secure distributed signing protocol. In J. Ding and J.-P. Tillich, editors, *Post-Quantum Cryptography – 11th International Conference, PQCrypto 2020*, pages 169–186. Springer, 2020.

[55] Richard Crandall and Carl B. Pomerance. *Prime numbers: a computational perspective*, volume 182. Springer Science & Business Media, 2006.

[56] Joan Daemen and Vincent Rijmen. *The design of Rijndael*, volume 2. Springer, 2002.

[57] Luca De Feo. Mathematics of Isogeny Based Cryptography, 2017. http://arxiv.org/abs/1711.04062.

[58] Luca De Feo. Exploring isogeny graphs. Habilitation thesis, 2018. https://defeo.lu/hdr/.

[59] Luca De Feo and Steven D. Galbraith. SeaSign: Compact isogeny signatures from class group actions. In Y. Ishai and V. Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019*, pages 759–789, 2019.

[60] Luca De Feo, David Jao, and Jérôme Plût. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. *Journal of Mathematical Cryptology*, 8(3):209–247, 2014.

[61] Luca De Feo, Jean Kieffer, and Benjamin Smith. Towards practical key exchange from ordinary isogeny graphs. In T. Peyrin and S. Galbraith, editors, *Advances in Cryptology – ASIACRYPT 2018*, pages 365–394. Springer, 2018.

[62] Luca De Feo, David Kohel, Antonin Leroux, Christophe Petit, and Benjamin Wesolowski. SQISign: compact post-quantum signatures from quaternions and isogenies. In S. Moriai and H. Wang, editors, *Advances in Cryptology – ASIACRYPT 2020*, pages 64–93. Springer, 2020.

[63] Luca De Feo and Michael Meyer. Threshold schemes from isogeny assumptions. In A. Kiayias, M. Kohlweiss, P. Wallden, and V. Zikas, editors, *Public-Key Cryptography – PKC 2020*, pages 187–212. Springer, 2020.

[64] Thomas Decru, Lorenz Panny, and Frederik Vercauteren. Faster SeaSign signatures through improved rejection sampling. In J. Ding and R. Steinwandt, editors, *Post-Quantum Cryptography – 10th International Conference, PQCrypto 2019*, pages 271–285. Springer, 2019.

[65] Christina Delfs and Steven D. Galbraith. Computing isogenies between supersingular elliptic curves over $\mathbb{F}_p$. *Designs, Codes and Cryptography*, 78(2):425–440, 2016.

[66] Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In G. Brassard, editor, *Advances in Cryptology – CRYPTO '89*, pages 307–315. Springer, 1990.

[67] Yvo Desmedt and Yair Frankel. Shared generation of authenticators and signatures. In J. Feigenbaum, editor, *Advances in Cryptology – CRYPTO '91*, pages 457–469. Springer, 1991.

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

[69] Jintai Ding, Ming-Shing Chen, Albrecht Petzoldt, Dieter Schmidt, and Bo-Yin Yang. Rainbow. Round 2 submission, NIST Post-Quantum Cryptography Standardization, 2019. https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions.

[70] Jack Doerner, Yashvanth Kondi, Eysa Lee, and Abhi Shelat. Secure two-party threshold ECDSA from ECDSA assumptions. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 980–997. IEEE, 2018.

[71] Emmanouil Doulgerakis, Thijs Laarhoven, and Benne de Weger. Finding closest lattice vectors using approximate Voronoi cells. In J. Ding and R. Steinwandt, editors, *Post-Quantum Cryptography – 10th International Conference, PQCrypto 2019*, pages 3–22. Springer, 2019.

[72] Harold M. Edwards. A normal form for elliptic curves. *Bulletin of the American Mathematical Society*, 44:393–422, 2007.

[73] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory*, 31(4):469–472, 1985.

[74] Noam D. Elkies. Elliptic and modular curves over finite fields and related computational issues. In *Computational perspectives on number theory (Chicago, IL, 1995)*, volume 7 of *Studies in Advanced Mathematics*, pages 21–76. AMS International Press, 1998.

[75] Joël Felderhoff. Hard Homogenous Spaces and Commutative Supersingular Isogeny based Diffie–Hellman. Internship report, Inria, France, 2019. https://hal.archives-ouvertes.fr/hal-02373179.

[76] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In A. M. Odlyzko, editor, *Advances in Cryptology – CRYPTO '86*, pages 186–194. Springer, 1987.

[77] Mireille Fouquet and François Morain. Isogeny volcanoes and the SEA algorithm. In *International Algorithmic Number Theory Symposium*, pages 276–291. Springer, 2002.

[78] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In M. Wiener, editor, *Advances in Cryptology – CRYPTO '99*, pages 537–554. Springer, 1999.

[79] Steven D. Galbraith. *Mathematics of public key cryptography*. Cambridge University Press, 2012.

[80] Steven D. Galbraith, Christophe Petit, Barak Shani, and Yan Bo Ti. On the security of supersingular isogeny cryptosystems. In J. H. Cheon and T. Takagi, editors, *Advances in Cryptology – ASIACRYPT 2016*, pages 63–91. Springer, 2016.

[81] Steven D. Galbraith, Christophe Petit, and Javier Silva. Identification Protocols and Signature Schemes Based on Supersingular Isogeny Problems. In T. Takagi and T. Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017*, pages 3–33. Springer, 2017.

[82] Steven D. Galbraith and Frederik Vercauteren. Computational problems in supersingular elliptic curve isogenies. *Quantum Information Processing*, 17(10):265, 2018.

[83] Alexandre Gélin and Benjamin Wesolowski. Loop-abort faults on supersingular isogeny cryptosystems. In T. Lange and T. Takagi, editors, *Post-Quantum Cryptography – 8th International Conference, PQCrypto 2017*, pages 93–106. Springer, 2017.

[84] Rosario Gennaro and Steven Goldfeder. Fast Multiparty Threshold ECDSA with Fast Trustless Setup. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1179–1194. ACM, 2018.

[85] Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. Robust threshold DSS signatures. In U. Maurer, editor, *Advances in Cryptology – EUROCRYPT '96*, pages 354–371. Springer, 1996.

[86] Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. Secure Distributed Key Generation for Discrete-Log Cryptosystems. In J. Stern, editor, *Advances in Cryptology – EUROCRYPT '99*, pages 295–310. Springer, 1999.

[87] Albert Gloden. *Mehrgradige Gleichungen*. Noordhoff, 1944.

[88] Andrew Granville. Smooth numbers: computational number theory and beyond. *Algorithmic number theory: lattices, number fields, curves and cryptography*, 44:267–323, 2008.

[89] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219, 1996.

[90] Lein Harn. Group-oriented (t, n) threshold digital signature scheme and digital multisignature. *IEEE Proceedings-Computers and Digital Techniques*, 141(5):307–313, 1994.

[91] Adolf Hildebrand. On a conjecture of Balog. *Proceedings of the American Mathematical Society*, 95(4):517–523, 1985.

[92] Aaron Hutchinson, Jason LeGrow, Brian Koziel, and Reza Azarderakhsh. Further optimizations of CSIDH: a systematic approach to efficient strategies, permutations, and bound vectors. In M. Conti, J. Zhou, E. Casalicchio, and A. Spognardi, editors, *International Conference on Applied Cryptography and Network Security – 18th International Conference, ACNS 2020*, pages 481–501. Springer, 2020.

[93] 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, David Urbanik, and Geovandro Pereira.

SIKE. Round 3 submission, NIST Post-Quantum Cryptography Standardization, 2020. https://sike.org/.

[94] David Jao and Luca De Feo. Towards Quantum-Resistant Cryptosystems from Supersingular Elliptic Curve Isogenies. In B. Yang, editor, *Post-Quantum Cryptography – PQCrypto 2011*, pages 19–34. Springer, 2011.

[95] David Jao, Stephen D. Miller, and Ramarathnam Venkatesan. Expander graphs based on GRH with an application to elliptic curve cryptography. *Journal of Number Theory*, 129(6):1491–1504, 2009.

[96] Samuel Jaques and John M. Schanck. Quantum cryptanalysis in the RAM model: Claw-finding attacks on SIKE. In A. Boldyreva and D. Micciancio, editors, *Advances in Cryptology – CRYPTO 2019*, pages 32–61. Springer, 2019.

[97] Matthias J. Kannwischer, Joost Rijneveld, Peter Schwabe, and Ko Stoffelen. PQM4: Post-quantum crypto library for the ARM Cortex-M4. https://github.com/mupq/pqm4.

[98] Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

[99] Christian Karpfinger and Kurt Meyberg. *Algebra: Gruppen-Ringe-Körper*. Springer, 4th edition, 2017.

[100] Eike Kiltz. A Tool Box of Cryptographic Functions Related to the Diffie-Hellman Function. In C. P. Rangan and C. Ding, editors, *Progress in Cryptology – INDOCRYPT 2001*, pages 339–349. Springer, 2001.

[101] Suhri Kim, Kisoon Yoon, Young-Ho Park, and Seokhie Hong. Optimized method for computing odd-degree isogenies on Edwards curves. In S. D. Galbraith and S. Moriai, editors, *Advances in Cryptology – ASIACRYPT 2019*, pages 273–292. Springer, 2019.

[102] H. Kleiman. A note on the Tarry-Escott problem. *J. Reine Angew. Math.*, 278/279:48–51, 1975.

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

[104] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. In M. Wiener, editor, *Advances in Cryptology – CRYPTO '99*, pages 388–397. Springer, 1999.

[105] David R. Kohel. *Endomorphism rings of elliptic curves over finite fields*. PhD thesis, University of California, Berkeley, 1996.

[106] Greg Kuperberg. Another subexponential-time quantum algorithm for the dihedral hidden subgroup problem. *TQC,*. volume 22 of *LIPIcs*, pages 22-34. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2013.

[107] Greg Kuperberg. A subexponential-time quantum algorithm for the dihedral hidden subgroup problem. *SIAM Journal on Computing*, 35(1):170–188, 2005.

[108] Péter Kutas, Chloe Martindale, Lorenz Panny, Christophe Petit, and Katherine E. Stange. Weak instances of SIDH variants under improved torsion-point attacks. Cryptology ePrint Archive, Report 2020/633, 2020. https://eprint.iacr.org/2020/633.

[109] Derrick H. Lehmer. On a problem of Störmer. *Illinois Journal of Mathematics*, 8(1):57–79, 1964.

[110] Greg Martin. An asymptotic formula for the number of smooth values of a polynomial. *Journal of Number Theory*, 93:108–182, 2002.

[111] Kazuto Matsuo. SIDH over quadratic twists. In *Proc. of SCIS2019 – 2019 Symposium on Cryptography and Information Security*, 2019.

[112] Alfred J. Menezes, Tatsuaki Okamoto, and Scott A. Vanstone. Reducing elliptic curve logarithms to logarithms in a finite field. *IEEE Transactions on information Theory*, 39(5):1639–1646, 1993.

[113] Michael Meyer, Fabio Campos, and Steffen Reith. On Lions and Elligators: An efficient constant-time implementation of CSIDH. In J. Ding and R. Steinwandt, editors, *Post-Quantum Cryptography – 10th International Conference, PQCrypto 2019*, pages 307–325. Springer, 2019.

[114] Michael Meyer and Steffen Reith. A faster way to the CSIDH. In D. Chakraborty and T. Iwata, editors, *Progress in Cryptology – INDOCRYPT 2018*, pages 137–152. Springer, 2018.

[115] Michael Meyer, Steffen Reith, and Fabio Campos. On hybrid SIDH schemes using Edwards and Montgomery curve arithmetic. Cryptology ePrint Archive, Report 2017/1213, 2017. https://eprint.iacr.org/2017/1213.

[116] Victor S. Miller. Use of elliptic curves in cryptography. In H. C. Williams, editor, *Advances in Cryptology – CRYPTO '85*, pages 417–426. Springer, 1985.

[117] Peter L. Montgomery. Speeding the Pollard and Elliptic Curve Methods of Factorization. *Mathematics of Computation*, 48(177):243–264, 1987.

[118] Dustin Moody and Daniel Shumow. Analogues of Vélu's formulas for isogenies on alternate models of elliptic curves. *Mathematics of Computation*, 85(300):1929–1951, 2016.

[119] Tomoki Moriya, Hiroshi Onuki, and Tsuyoshi Takagi. How to construct CSIDH on Edwards curves. In S. Jarecki, editor, *Topics in Cryptology – CT-RSA 2020*, pages 512–537. Springer, 2020.

[120] Michael Naehrig and Joost Renes. Dual isogenies and their application to public-key compression for isogeny-based cryptography. In S. D. Galbraith and S. Moriai, editors, *Advances in Cryptology – ASIACRYPT 2019*, pages 243–272. Springer, 2019.

[121] National Institute of Standards and Technology (NIST). Post-Quantum Cryptography Standardization, 2016. https://csrc.nist.gov/Projects/post-quantum-cryptography/Post-Quantum-Cryptography-Standardization.

[122] Hiroshi Onuki, Yusuke Aikawa, Tsutomu Yamazaki, and Tsuyoshi Takagi. (Short Paper) A Faster Constant-Time Algorithm of CSIDH Keeping Two Points. In N. Attrapadung and T. Yagi, editors, *Advances in Information and Computer Security – 14th International Workshop on Security, IWSEC 2019*, pages 23–33. Springer, 2019.

[123] Torben P. Pedersen. A Threshold Cryptosystem without a Trusted Party. In D. W. Davies, editor, *Advances in Cryptology – EUROCRYPT '91*, pages 522–526. Springer, 1991.

[124] Chris Peikert. He gives C-sieves on the CSIDH. In A. Canteaut and Y. Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020*, pages 463–492. Springer, 2020.

[125] Christophe Petit. Faster algorithms for isogeny problems using torsion point images. In T. Takagi and T. Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017*, pages 330–353. Springer, 2017.

[126] Arnold K. Pizer. Ramanujan graphs and Hecke operators. *Bulletin of the American Mathematical Society*, 23(1):127–137, 1990.

[127] John M. Pollard. Theorems on factorization and primality testing. *Mathematical Proceedings of the Cambridge Philosophical Society*, 76(3):521–528, 1974.

[128] Carl B. Pomerance. The role of smooth numbers in number theoretic algorithms. In *Proceedings of the International Congress of Mathematicians*, pages 411–422. Springer, 1995.

[129] Elmer Rees and Christopher J. Smyth. On the constant in the Tarry-Escott problem. In *Cinquante Ans de Polynômes*, pages 196–208. Springer, 1990.

[130] Oded Regev. A subexponential time algorithm for the dihedral hidden subgroup problem with polynomial space. arXiv preprint quant-ph/0406151, 2004. https://arxiv.org/abs/quant-ph/0406151.

[131] Joost Renes. Computing isogenies between Montgomery curves using the action of $(0,0)$. In T. Lange and R. Steinwandt, editors, *Post-Quantum Cryptography – 9th International Conference, PQCrypto 2018*, pages 229–247. Springer, 2018.

[132] Ronald L. Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[133] Alexander Rostovtsev and Anton Stolbunov. Public-key cryptosystem based on isogenies. Cryptology ePrint Archive, Report 2006/145, 2006. http://eprint.iacr.org/2006/145.

[134] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In G. Brassard, editor, *Advances in Cryptology – CRYPTO '89*, pages 239–252. Springer, 1989.

[135] René Schoof. Elliptic curves over finite fields and the computation of square roots mod p. *Mathematics of computation*, 44(170):483–494, 1985.

[136] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[137] Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 124–134. IEEE, 1994.

[138] Victor Shoup. Practical Threshold Signatures. In B. Preneel, editor, *Advances in Cryptology – EUROCRYPT 2000*, pages 207–220. Springer, 2000.

[139] Chen Shuwen. The Prouhet-Tarry-Escott Problem. http://eslpower.org/TarryPrb.htm.

[140] Joseph H. Silverman. *The Arithmetic of Elliptic Curves*. Graduate Texts in Mathematics. Springer, 2nd edition, 2009.

[141] Simon Singh. *The Code Book: The Science of Secrecy from Ancient Egypt to Quantum cryptography*. Fourth Estate, 1999.

[142] Christopher J. Smyth. Ideal 9th-order multigrades and Letac's elliptic curve. *Mathematics of Computation*, 57(196):817–823, 1991.

[143] Jonathan P. Sorenson. A Fast Algorithm for Appoximately Counting Smooth Numbers. In W. Bosma, editor, *Algorithmic Number Theory, 4th International Symposium, ANTS-IV, 2000*, volume 1838 of *Lecture Notes in Computer Science*, pages 539–550. Springer, 2000.

[144] Douglas R. Stinson and Reto Strobl. Provably secure distributed Schnorr signatures and a (t, n) threshold scheme for implicit certificates. In V. Varadharajan and Y. Mu, editors, *Australasian Conference on Information Security and Privacy – ACISP 2001*, pages 417–434. Springer, 2001.

[145] Anton Stolbunov. Constructing public-key cryptographic schemes based on class group action on a set of isogenous elliptic curves. *Advances in Mathematics of Communications*, 4(2):215–235, 2010.

[146] Anton Stolbunov. Cryptographic schemes based on isogenies. Doctoral thesis, NTNU, 2012.

[147] Carl Størmer. Quelques théorèmes sur l'équation de Pell $x^2 - dy^2 = \pm 1$ et leurs applications. *Christiania Videnskabens Selskabs Skrifter, Math. Nat. Kl*, (2):48, 1897.

[148] Volker Strassen. Einige Resultate über Berechnungskomplexität. *Jahresbericht der Deutschen Mathematiker-Vereinigung*, 78:1–8, 1976.

[149] Seiichiro Tani. Claw finding algorithms using quantum walk. *Theoretical Computer Science*, 410(50):5285–5297, 2009.

[150] Yan Bo Ti. Fault Attack on Supersingular Isogeny Cryptosystems. In T. Lange and T. Takagi, editors, *Post-Quantum Cryptography – 8th International Conference, PQCrypto 2017*, pages 107–122. Springer, 2017.

[151] Paul C. Van Oorschot and Michael J. Wiener. Parallel collision search with cryptanalytic applications. *Journal of cryptology*, 12(1):1–28, 1999.

[152] Jacques Vélu. Isogénies entre courbes elliptiques. *C.R. Acad. Sc. Paris, Série A.*, 271:238–241, 1971.

[153] Lawrence C. Washington. *Elliptic curves: number theory and cryptography*. CRC press, 2nd edition, 2008.

[154] Edward M. Wright. On Tarry's problem (I). *The Quarterly Journal of Mathematics*, (1):261–267, 1935.

[155] Jaroslaw Wróblewski and Ajai Choudhry. Ideal solutions of the Tarry-Escott problem of degree eleven with applications to sums of thirteenth powers. *Hardy-Ramanujan Journal*, 31, 2008.

[156] Youngho Yoo, Reza Azarderakhsh, Amir Jalali, David Jao, and Vladimir Soukharev. A post-quantum digital signature scheme based on supersingular isogenies. In A. Kiayias, editor, *International Conference on Financial Cryptography and Data Security – FC 2017*, pages 163–181. Springer, 2017.

[157] Gustavo H. M. Zanon, Marcos A. Simplicio, Geovandro C. C. F. Pereira, Javad Doliskani, and Paulo S. L. M. Barreto. Faster key compression for isogeny-based cryptosystems. *IEEE Transactions on Computers*, 68(5):688–701, 2018.

Notation

The following lists contain symbols resp. acronyms (except for names of cryptographic schemes) used throughout this thesis in an alphabetical order. Note that the notation from different chapters overlaps in some cases; these will be explicitly assigned to the corresponding chapters.

Symbols

$[m]$	the multiplication-by-m map for points on elliptic curves.
∞	the point at infinity on an elliptic curve.
$\mathcal{A}$	a polynomial-time adversary (in Chapter 6).
$\mathcal{A}, \mathcal{B}$	multisets of integers (in Chapter 7).
$\mathbb{A}^n$	affine space of dimension n.
$a(x), b(x)$	polynomials that split in $\mathbb{Z}[x]$ and differ by a constant.
$\mathbf{a}, \mathbf{M}, \mathbf{S}, \mathbf{I}$	costs for a field addition, multiplication, squaring, inversion.
$\mathfrak{a}, \mathfrak{b}, \mathfrak{g}, \ldots$	ideals resp. elements of a group $\mathcal{G}$.
B	a smoothness bound (in Chapter 7).
$\mathrm{char}(\mathbb{K})$	the characteristic of a field $\mathbb{K}$.
$\mathrm{cl}(\mathcal{O})$	the ideal class group of $\mathcal{O}$.
$\deg \psi$	the degree of a map ψ.
$\mathcal{E}$	a hard homogeneous space.
E	an elliptic curve.
$E(\mathbb{K})$	an elliptic curve over $\mathbb{K}$.
$E[m]$	the m-torsion of an elliptic curve E.
$\mathcal{E}\ell\ell_p(\mathcal{O}, \pi)$	the set of supersingular elliptic curves over $\mathbb{F}_p$ with $\mathrm{End}_{\mathbb{F}_p}(E) \cong \mathcal{O}$.
$\mathrm{End}(E)$	the endomorphism ring of an elliptic curve E.
$\mathrm{End}_{\mathbb{F}_q}(E)$	the $\mathbb{F}_q$-rational endomorphism ring of an elliptic curve E.
$\mathbb{F}_q$	a finite field of q elements.
$\mathcal{F}_{\mathcal{T}}$	an ideal functionality.
$\mathcal{G}$	a group acting on a hard homogeneous space.
$G_{E,\mathbb{K},\ell}$	the ℓ-isogeny graph of the isogeny class of the elliptic curve E over $\mathbb{K}$.
H	a hash function (in Chapter 6).

H	a hitting set (in Chapter 7).
$j(E)$	the j-invariant of an elliptic curve E.
$\mathbb{K}$	a field.
$\overline{\mathbb{K}}$	the algebraic closure of a field $\mathbb{K}$.
$\ker \psi$	the kernel of a map ψ.
$L_{l,i}^{S}$	a Lagrange interpolation polynomial.
$\mathfrak{l}, \bar{\mathfrak{l}}$	prime ideals.
$O, \widetilde{O}$	big O complexity notation.
$\mathcal{O}$	an order of an imaginary quadratic field.
P, Q, R	points on an elliptic curve.
$\mathbb{P}^n$	projective space of dimension n.
$\mathcal{P}_i$	a participant of a threshold scheme.
π	the Frobenius endomorphism (in Chapters 1-6).
$\pi(x)$	the number of primes $p \leq x$ (in Chapter 7).
φ	an isogeny.
$\Psi(N, B)$	the number of positive B-smooth integers $x \leq N$.
$\mathbb{Q}$	the field of rational numbers.
$\mathbb{R}$	the field of real numbers.
ρ	the Dickman–de Bruijn function.
S	the set of participants of a threshold scheme.
$\mathcal{S}$	a simulator.
$\mathcal{T}$	the trusted dealer in a threshold scheme.
$\mathbb{Z}$	the ring of integers.
$\mathbb{Z}/n\mathbb{Z}$	the ring of integers modulo n.

Acronyms

CRT	Chinese Remainder Theorem.
DADD	differential addition.
DKG	distributed key generation.
ECC	elliptic curve cryptography.
GCD	greatest common divisor.
HHS	hard homogeneous space.
KEM	key encapsulation mechanism.
NIST	National Institute of Standards and Technology.
OT	oblivious transfer.
PKE	public-key encryption.
PQC	post-quantum cryptography.
PTE	Prouhet-Tarry-Escott problem.

PRF	pseudorandom function.
SIMBA	splitting isogenies into multiple batches.
UF-CMA	unforgeability under chosen-message attacks.
XGCD	extended Euclidean algorithm.

Abstract

This thesis aims at providing efficient and side-channel protected implementations of isogeny-based primitives, and at their application in threshold protocols. To this end, Chapter 1 introduces the topic and Chapter 2 provides the necessary background on elliptic curves, isogenies, and cryptographic protocols. The remainder of this thesis is based on a sequence of academic papers.

- Chapter 3 reviews the variable-time proof-of-concept implementation of CSIDH as published in [35] and introduces several optimizations. We restructure the algorithm in order to significantly reduce the cost for the required scalar multiplications. Furthermore, we show that the isogeny computations based on Montgomery curves from [35] can be significantly improved by using both Montgomery and twisted Edwards curves. In total, these improvements yield a speedup of 25% compared to the variable-time implementation from [35]. The content of this chapter was published in [114].

- Chapter 4 presents the first practical constant-time implementation of CSIDH. We describe how the variable-time implementations from [35, 114] leak information on the private key; in particular, this concerns the total number of isogenies and the sign distribution of the private key entries. We describe how this can be mitigated by using dummy isogenies and sampling only non-negative key elements, which leads to a significant slowdown in a straightforward implementation. However, we present several techniques to speed up the implementation, such as SIMBA, which processes the isogeny computations in separate batches. In total, our constant-time implementation achieves a rather small slowdown by a factor of 3.03 compared to the variable-time implementation from [114]. The content of this chapter was published in [113].

- Chapter 5 reviews practical fault injection attacks on CSIDH and presents countermeasures. While [37] presents a dummy-free constant-time implementation of CSIDH at the cost of a twofold slowdown, implementations that utilize dummy-isogenies [113, 122] are naturally vulnerable to fault injection attacks. We evaluate different attack models theoretically and practically, using low-budget equipment. Moreover, we present countermeasures that mitigate the proposed fault injection

attacks, only leading to a small performance overhead of 7%. The content of this chapter was published in [31].

- Chapter 6 initiates the study of threshold schemes based on the Hard Homogeneous Spaces (HHS) framework of Couveignes [51]. Although CSIDH as defined in [35] is not a HHS in the strictest sense, the record class group precomputation performed for the signature scheme CSI-FiSh [19] provides a strict HHS for the CSIDH-512 parameter set. Using the HHS equivalent of the technique of Shamir's secret sharing in the exponents, we thus adapt isogeny based schemes to the threshold setting. In particular, we present threshold versions of the CSIDH public key encryption, and the CSI-FiSh signature schemes. The main highlight is a threshold version of CSI-FiSh which runs almost as fast as the original scheme, for message sizes as low as 1880 B, public key sizes as low as 128 B, and thresholds up to 56; other speed-size-threshold compromises are possible. The content of this chapter was published in [63].

- Chapter 7 gives a sieving algorithm for finding pairs of consecutive smooth numbers that utilizes solutions to the Prouhet-Tarry-Escott (PTE) problem. Any such solution induces two degree n polynomials, $a(x)$ and $b(x)$, that differ by a constant integer C and completely split into linear factors in $\mathbb{Z}[x]$. We describe how such polynomials can be used to search for twin smooth integers. The motivation for finding large twin smooth integers lies in their application to compact isogeny-based post-quantum protocols, namely B-SIDH [44] and SQISign [62], which both require large primes that lie between two smooth integers. Finding such a prime can be seen as a special case of finding twin smooth integers under the additional stipulation that their sum is a prime p. When searching for cryptographic parameters with $2^{240} \leq p < 2^{256}$, an implementation of our sieve found primes p where $p+1$ and $p-1$ are 2^{15}-smooth; the smoothest prior parameters had a similar sized prime for which $p-1$ and $p+1$ were 2^{19}-smooth. The content of this chapter was published in [49].