The design, concrete security and efficiency of isogeny-based cryptography

Maria Corte-Real Santos
University College London
Department of Computer Science

Submitted to University College London (UCL) in partial fulfilment of the requirements for the award of the degree of Doctor of Philosophy.

Thesis submission date: October 19, 2024

Declaration

I, Maria Corte-Real Santos confirm that the work presented in my thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the thesis.

Maria Corte-Real Santos
London, United Kingdom
October 19, 2024

Abstract

The security of most public-key cryptosystems that are currently deployed rely on the hardness of the discrete logarithm problem or of the integer factorisation problem. In 1994, Shor gave a devastating polynomial-time quantum attack against these problems, showing that cryptosystems that rely on their hardness are not secure in the presence of a quantum adversary. Considering the increased investment in the development of large-scale quantum computers, in 2016, NIST began an effort to standardise post-quantum secure key encapsulation mechanisms and signature schemes.

We work with a specific type of post-quantum cryptography considered by NIST: isogeny-based cryptography, where security rests on the hardness of the isogeny problem. This problem asks, given two elliptic curves, to find a "nice" map, called an isogeny, between them. In the first part of this thesis, we explore the concrete security of this problem when working with supersingular elliptic curves. Viewing an elliptic curve as one-dimensional specialisation of a more abstract mathematical object, namely an abelian variety, we also study the generalisation of the isogeny problem to higher dimensions. In the next part, we focus on the recent shift towards using isogenies between two-dimensional abelian varieties to construct new protocols, and present efficient formulæ for computing such isogenies of odd degree.

We then focus on SQIsign, the only isogeny-based signature scheme that was submitted to NIST's alternate call for signatures. SQIsign boasts the smallest combined signature and public key sizes. However, finding SQIsign-friendly parameters has proved to be a difficult task. We provide a solution to this by presenting the first practical parameters for all security levels. Despite this, SQIsign is still considerably slower than other alternatives. Noting that SQIsign is most interesting in scenarios that require small signature sizes and fast verification, we also present work that accelerates SQIsign verification, without sacrificing the signature size.

Impact Statement

The results presented in this thesis contribute to the development of post-quantum secure cryptosystems, focusing particularly on isogeny-based cryptography. This thesis demonstrates how to gain a better understanding of the concrete security of the hardness assumption underlying isogeny-based cryptography. This is important for selecting parameters that reach a certain level of security, and improving the general confidence in isogeny-based cryptography. One of the main obstacles for isogeny-based cryptography to be used in practice is its efficiency. In this thesis, we furthermore introduce new techniques to accelerate isogeny-based cryptography. This includes techniques that are targeted specifically towards SQIsign, a promising isogeny-based signature scheme, as well as methods that improve the efficiency of isogenies between higher-dimensional generalisations of elliptic curves, which have shown to be indispensable in isogeny-based cryptography since the polynomial time attacks against prominent isogeny-based scheme SIDH/SIKE in 2022.

Since 2016, the National Institute of Standards and Technology (NIST) has looked towards standardising post-quantum cryptography in response to the substantial advancement of quantum computing. NIST cryptographic standards inform what cryptography is deployed by the United States Government and in industry. We note that they have impact outside the United States as many international bodies seek to also comply with NIST standards.

Most pertinent to this thesis is NIST's alternate call for post-quantum secure signature schemes in 2023, to which SQIsign was submitted and has progressed to Round 1. Our work on parameter finding for SQIsign directly impacted the methods used to find parameters for all security levels for the NIST submission.

SQIsign is currently the candidate that comes closest to the data sizes transmitted in pre-quantum elliptic curve signatures, however signing and verification are slow. The variant of SQIsign presented in this thesis, namely AprèsSQI, specifically targets applications where the amount of data transmitted is crucial and fast verification is desirable. A few common examples include long-term signatures, specifically public-key certificates, code updates for small devices, and smart cards. If SQIsign is standardised through the NIST process, it is plausible that this variant of SQIsign will become interesting for such applications in industry.

Finally, all the content in this thesis has been published to conference proceedings or journals, as well as open access repositories. The accompanying software has also been made publicly available on GitHub, and mostly written in open source languages (except for `MAGMA`). Therefore, it is easily available to other researchers who are looking to use it to work on isogeny-based cryptography, thus facilitating the future impact of this work.

Acknowledgements

My love for Mathematics began while I was a student at Murray Edwards College at the University of Cambridge, and I owe a special thanks to my Directors of Studies Holly Krieger and Ana Khukhro for encouraging me to start a PhD.

My PhD was funded by the UK EPSRC grant EP/S022503/1 that supports the Centre for Doctoral Training in Cybersecurity delivered by UCL's Departments of Computer Science, Security and Crime Science, and Science, Technology, Engineering, and Public Policy.

Thank you to my supervisors Philipp Jovanovic and Sarah Meiklejohn for their time, patience, and support throughout these four years, and for giving me the confidence to tackle the problems that I found interesting.

Another person who deserves my sincere thanks is Craig Costello. Thank you for taking me under your wing and introducing me to isogeny-based cryptography. Without your friendship, constant encouragement and mentoring, my PhD would not have been the same.

Thank you to Vladimir Dockchister and his research group for allowing me to take part in your reading groups and pretend to be a Mathematician every once in a while!

Thank you to Michael Naehrig and (again) to Craig Costello, for hosting me at Microsoft Research as a Summer intern in 2023. That summer in Seattle was very important to me, and it was a wonderful haven in the middle of my PhD that convinced me to stay in research. Much of that was due to you both.

A special thank you to Sam, for all his support throughout my PhD.

Thank you to the SICQ. Together we got a Best Rump Session award and a Best Early Career Paper award, but most importantly we shared many laughs.

Thank you to Benjamin Smith, for hosting me twice in Paris, and for his incredible insights and advice.

Thank you to all my (not yet mentioned) collaborators, Giacomo Bruno, Victor Flynn, Francisco Rodríguez-Henríquez, and Bruno Sterner, from whom I learnt many invaluable things.

Thank you to Simona Samardjiska for hosting me at Radboud University, and Damien Robert for hosting me at the Univeristy of Bordeaux, for a week each during my PhD.

I want to (again) thank Craig Costello, Jonathan Komada Eriksen, Sam Frengley, Philipp Jovanovic, Sarah Meiklejohn, and Michiel Van Beirendonck for giving their feedback on a preliminary draft of this thesis.

Thank you to my sister for being a big part of my support system in London. Last, but certainly not least, thank you to my parents. I am eternally grateful for their unwavering support and love, and for encouraging me to continuously chase my dreams.

Table of contents

Introduction

In 1976, Diffie and Hellman [136] made a major breakthrough in cryptography. Or, as they aptly put it:

"*We stand today on the brink of a revolution in cryptography.*"

Up until that point, communication was encrypted using a secret key that had been previously agreed to through other means (for example, in person). Such a scenario would be very impractical in today's digital age. Indeed, before online communication could take place with another person or server, one would have to meet to agree on a shared encryption key. Diffie and Hellman proposed to instead split the cryptographic primitive in two parts: a private operation done by each party using a *private key* leading to a piece of data that can be made public and used by other parties, called a *public key*. Two parties can now use their private key and the public keys to derive a *shared secret key* known only to the parties involved in this *key exchange*, which can henceforth be used to encrypt their communications. Though the idea is simple, it is initially unclear how one can build such a primitive. Diffie and Hellman's article was revolutionary in that they further provided an instantiation of this idea using the hardness of the discrete logarithm problem in finite fields $\mathbb{F}_p$, where p is a large prime: the Diffie–Hellman key exchange. The Diffie–Hellman protocol has had a deep impact on modern cryptography; it provides a practical and secure way for two parties to establish a shared secret over an insecure network, and thus forms the backbone of many secure communication protocols in use today.

From this work, the field of *public-key cryptography* was invented, thus ushering in the age of modern cryptography. Modern cryptography has proved essential to protect our online communications by allowing us to send messages in a confidential manner, whilst ensuring they are authentic and have not been altered. Cryptography is used in a variety of real world applications, such as instant messaging and electronic commerce. Other forms of public-key cryptography have since been constructed, most notably the RSA cryptosystem by Rivest, Shamir, and Adleman [270] based on the hardness of factoring large integers, and *elliptic curve cryptography* (ECC).

The use of *elliptic curves* in cryptography was independently proposed by Miller [236] and Koblitz [202] in the 1980s. Elliptic curves are *algebraic curves* that also naturally form a group under the *addition of points*. Therefore, they provide an alternative instantiation of the prime order groups used in Diffie–Hellman key-exchange, where the security rests on the hardness of the discrete logarithm problem in the elliptic curve group. Having keys of smaller size (in bytes) compared to other alternatives that give the same security, ECC has become a standard in public-key cryptography. It is used in, for example, the TLS protocol [135] that facilitates secure internet communications, and popular messaging applications such as WhatsApp [319] and Signal [291]. ECC has proved to be incredibly flexible, and more recently has been used in many blockchain platforms and cryptocurrencies, such as Ethereum, for generating *digital signatures*, a primitive that verifies the authenticity of digital information.

In his seminal 1995 paper, Shor exhibited a quantum polynomial time algorithm against both the discrete logarithm problem and the integer factorisation problem [288]. If large-scale quantum computers are built, all the primitives we have introduced thus far, which are essential for the security of our online communications, will become insecure. The attack by Shor highlighted the power of quantum computers. This sparked the race to develop new cryptography that can be implemented using classical computers but can withstand attacks from quantum ones. This branch of cryptography is called *post-quantum cryptography.*

In 2016, NIST initiated an effort to develop and standardize one or more *public-key encapsulation mechanisms* (which provide secure encryption) and digital signature schemes using post-quantum cryptography, with evaluation criteria being a scheme's cost and performance, algorithm and implementation characteristics, and security [310]. This security is categorised in five levels, depicted in Table 1, with NIST placing a focus on Levels I, III, IV.

Security	**Description**
Level I	At least as hard to break as `AES128` (Exhaustive Key Search)
Level II	At least as hard to break as `SHA256` (Collision Search)
Level III	At least as hard to break as `AES192` (Exhaustive Key Search)
Level IV	At least as hard to break as `SHA384` (Collision Search)
Level V	At least as hard to break as `AES256` (Exhaustive Key Search)

TABLE 1: Categories of NIST security levels

In 2020, NIST announced the schemes that had progressed to Round 3 of the standardisation effort, categorised into five main families of post-quantum cryptography:

- Lattice-based cryptography: Constructs cryptographic protocols using lattices, and their security is based on the hardness of computational problems involving these lattices. Lattice-based schemes are the most well-rounded in terms of security, performance, and communication cost.

- Code-based cryptography: Builds cryptographic schemes using error-correcting codes, such as Goppa codes, and is the oldest form of post-quantum cryptography. However, the key sizes are quite large.

- Multivariate-based cryptography: Based on the hardness of solving large systems of multivariate polynomial equations, these cryptosystems perform well in practice. A recent break of the promising multivariate-based signature scheme Rainbow has, however, decreased the confidence in their security [29].

- Hash-based cryptography: Constructs protocols using hash functions, and their security is based on the security of cryptographic hash functions. As such, there is strong confidence in their security. However, their key sizes are large, and they have only been used to construct signature schemes.

- Isogeny-based cryptography: Rather than using points on a single curve as in ECC, we build cryptographic protocols using maps between elliptic curves that have certain properties, called *isogenies*. The security of these schemes is based on the *general isogeny problem*: given two elliptic curves E_1, E_2 defined over a finite field, find an isogeny that maps E_1 to E_2.

In this thesis, we explore *isogeny-based cryptography*. The use of isogenies for cryptography was first proposed in 2006 independently by Couveignes [108] and Rostovtsev and Stolbunov [275]. Using *ordinary* elliptic curves, these proposals gave protocols that were largely impractical. Later, Charles, Lauter, and Goren [76] built a cryptographic hash function using isogenies between *supersingular* elliptic curves. Following this, De Feo, Jao, and Plût [123] proposed SIDH, a Diffie–Hellman-like key exchange built from isogenies between supersingular elliptic curves. After optimisations due to Costello, Longa, and Naehrig [104], SIDH became a more practical proposal, and its corresponding encryption scheme SIKE was submitted to the NIST standardisation effort [10]. This brought increased interest to isogeny-based cryptography.

As the youngest type of post-quantum cryptography, there was a large effort to study the classical and quantum security of SIKE [228, 259, 262], and more generally of isogeny-based cryptography [33, 107, 132]. The main hardness assumption underpinning the security of protocols built from supersingular elliptic curves and their isogenies is the general supersingular isogeny problem: given supersingular E_1, E_2 defined over $\overline{\mathbb{F}}_p$, find the isogeny $\varphi : E_1 \to E_2$. This leads us to the first contribution of this thesis: in Part II, we give an in-depth analysis of the concrete complexity of the best classical attacks against the general supersingular isogeny problem. Viewing an elliptic curve as a one-dimensional specialisation of a more abstract mathematical object, namely an *abelian variety*, we also study the generalisation of the isogeny problem to two dimensions. In both these settings, we go further and introduce new attacks with decreased concrete complexity, which allow us to obtain a firmer grasp on the hardness of the isogeny problem in dimension 1 and 2.

The security of SIKE relied on a variant of the isogeny problem where an attacker is given more information. This lead to a wave of polynomial time key recovery attacks which exploited this extra information using higher dimensional isogenies (i.e., isogenies between higher dimensional abelian varieties) to recover the secret key [67, 227, 271]. These attacks were devastating to SIKE's security, and as a result it was retracted from the NIST standardisation process. However, this was far from being the death of isogenies. In fact, remarkably, the attacks against SIKE have shown to be a very powerful constructive tool, breathing new life into isogeny-based cryptography. The attacks and subsequent works (e.g., [19]) demonstrated that higher-dimensional isogenies are crucial to both the understanding of the security of isogeny-based cryptography in dimension-1, and to the construction of efficient and advanced protocols using isogenies. In Part III, we advance the state-of-the-art on the computation of two-dimensional isogenies to facilitate their use in cryptography.

In 2022, NIST standardised the lattice-based encapsulation mechanism called CRYSTALS-KYBER [47], and digital signatures CRYSTALS-Dilithium [140], Falcon [161], and the hash-based signature SPHINCS+ [28]. Except for SPHINCS+, all of these schemes are based on the computational hardness of problems involving structured lattices. As such, NIST launched a call for additional digital signature schemes for the post-quantum cryptography standardisation process [309].

“*NIST is primarily interested in additional general-purpose signature schemes that are* ***not based on structured lattices****. For certain applications, such as certificate transparency, NIST may also be interested in signature schemes that have* ***short signatures*** *and* ***fast verification****.*” - NIST [309, pg. 2]

Parallel to the development of SIDH, work by Kohel, Lauter, Petit, and Tignol [203] introduced the *Deuring correspondence* to isogeny-based cryptography, connecting the world of *quaternion algebras* to supersingular elliptic curves and their *endomorphism rings*. These two ingredients led to the development of isogeny-based signature schemes, such as the GPS signature scheme [174] and SQIsign [125]. In Part IV, we focus on SQIsign, the sole isogeny-based candidate that was submitted to NIST's alternate call for signatures, which has since entered Round 1. It offers the smallest combined signature and public key sizes, and its security relies on isogeny-based hardness assumptions. It is therefore a promising alternative to lattice-based signatures in the NIST standardisation effort, and beyond. However, its signing and verification algorithms are very slow. With this in mind, the work in the final part of this thesis makes new strides on improving the efficiency of SQIsign.

Organisation of thesis

Part I: Preliminaries

- Chapter 1: We introduce cryptographic preliminaries that will be needed for the rest of the thesis. Most notably, we formally define the main cryptographic protocol of study: signature schemes constructed from identification protocols. We also present a dictionary of foundational mathematical concepts that may be unfamiliar to a more cryptographic audience.
- Chapter 2: We survey the main geometric objects underlying the contributions of this thesis. This chapter covers algebraic curves and their Picard groups, abelian varieties and (separable) isogenies between them. We focus in particular on principally polarised abelian varieties of dimension 1, namely elliptic curves, and of dimension 2.
- Chapter 3: Motivated by the Deuring correspondence, we switch gears to a more algebraic setting and introduce maximal orders and ideals in quaternion algebras.
- Chapter 4: Now that we have laid the mathematical groundwork, we look at these objects within a cryptographic context. In this chapter, we focus on the use of isogenies between abelian varieties in cryptography. After introducing superspecial abelian varieties and the superspecial isogeny graph, we define the general isogeny problem in dimension g and the best attacks against it. Due to its conjectured hardness, this problem underlies the security of all isogeny-based protocols.
- Chapter 5: In this chapter, we discuss how the Deuring correspondence can be used as a useful tool in isogeny-based cryptography. We first show that the quaternionic version of the general isogeny problem is solved in polynomial time by the KLPT algorithm. Following this, we introduce SQIsign, an isogeny-based signature scheme built from supersingular elliptic curves, the Deuring correspondence and the KLPT algorithm.

- Chapter 6: We present a comprehensive review of the literature surrounding the topics studied, and detail the main contributions of the thesis.

The preliminaries in this thesis span many topics in algebraic geometry, number theory and isogeny-based cryptography. To help a reader (who wants to focus on a particular chapter) navigate the necessary content, we provide a flowchart of dependencies in Figure 1.

Part II: On the concrete complexity of the isogeny problem

- Chapter 7: In this chapter, we study the concrete complexity of the best classical attack against the general isogeny problem in dimension 1. We extend this by introducing a variant of the attack with improved concrete complexity.
- Chapter 8: Using a similar framework, we analyse and improve upon the concrete complexity of the best classical algorithm that solves the general isogeny problem in dimension 2.

Part III: Two-dimensional isogenies

- Chapter 9: Since the SIDH/SIKE attacks, there has been a shift towards using (N, N)-isogenies, i.e., isogenies of degree N between two-dimensional (principally polarised) abelian varieties, to build isogeny-based protocols. Though $(2, 2)$-isogenies have thus far mostly been used due to their efficiency, it is likely that (N, N)-isogenies for larger N will be needed for future research. In this chapter, we give a general method to construct (N, N)-isogenies for any odd N, and exhibit the power of this method by presenting efficient and explicit formulæ for computing $(3, 3)$-isogenies.

Part IV: Accelerating SQIsign

- Chapter 10: We present a novel method to finding pairs of integers $(x, x \pm 1)$ such that $x(x \pm 1)$ is only divisible by small primes. Using this, we show how to construct cryptographic sized primes as $p = 2x^n - 1$ so that $p^2 - 1$ is sufficiently smooth. Such primes can then be used as parameters in SQIsign.
- Chapter 11: This chapter introduces a variant of SQIsign called ApresSQI targeted towards applications requiring small signatures and fast verification. To this end, we increase the efficiency of SQIsign verification by allowing signing to occur over an extension field of $\mathbb{F}_{p^2}$, and thus incurring a small degradation in signing time.

- Chapter 12: We conclude with summarising the contributions of this thesis, particularly in light of recent work. We also give some brief comments on avenues for future work.

Publications

This thesis is, for the most part, a concatenation of published papers. Publications are ordered chronologically and the list of authors of each paper is ordered alphabetically.[1]

[1]See https://www.ams.org/profession/leaders/CultureStatement04.pdf

Publications included in the thesis

We begin by presenting the published works that are included in this thesis, discussing my personal contributions to each work.

[90] Maria Corte-Real Santos, Craig Costello, and Jia Shi. "Accelerating the Delfs–Galbraith Algorithm with Fast Subfield Root Detection". In *Advances in Cryptology - CRYPTO 2022. 2nd Annual International Cryptology Conference, Santa Barbara, CA, USA, August 15-18, 2022, Proceedings, Part III*, pages 285-314, 2022.

This paper studies the concrete complexity of the Delfs–Galbraith algorithm, the best attack against the dimension-1 supersingular isogeny problem. It forms the basis of Chapter 7.

[60] Giacomo Bruno, Maria Corte-Real Santos, Craig Costello, Jonathan Komada Eriksen, Michael Meyer, Michael Naehrig, and Bruno Sterner. "Cryptographic Smooth Neighbors". In *Advances in Cryptology - ASIACRYPT 2023 - 29th International Conference on the Theory and Application of Cryptology and Information Security, Guangzhou, China, December 4-8, 2023, Proceedings, Part VII*, pages 190-221, 2023.

This paper forms the basis of Chapter 10, and presents a new method to finding SQIsign-friendly primes for all NIST security levels.

[88] Maria Corte-Real Santos, Craig Costello, and Sam Frengley. "An Algorithm for Efficient Detection of (N, N)-Splittings and Its Application to the Isogeny Problem in Dimension 2". In *Public-Key Cryptography - PKC 2024 - 27th IACR International Conference on Practice and Theory of Public-Key Cryptography, Sydney, NSW, Australia, April 15-17, 2024, Proceedings, Part III*, pages 157-189, 2024.

This paper analyses and improves the concrete security of Costello–Smith algorithm, the best attack against the dimension-2 superspecial isogeny problem. It is presented in Chapter 8. It was awarded the Best Paper Award at PKC 2024.

[92] Maria Corte-Real Santos, Jonathan Komada Eriksen, Michael Meyer, and Krijn Reijnders. "AprèsSQI: Extra Fast Verification for SQIsign Using Extension-Field Signing". In *Advances in Cryptology - EUROCRYPT 2024 - 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland, May 26-30, 2024, Proceedings, Part I*, pages 63-93, 2024.

We present this paper in Chapter 11. It introduces a new variant of the isogeny-based signature schemes SQIsign, called AprèsSQI, which prioritises fast verification. It was awarded the Best Early Career Paper Award at Eurocrypt 2024.

[91] Maria Corte-Real Santos, Craig Costello, and Benjamin Smith. "Efficient $(3, 3)$-isogenies on fast Kummer surfaces". To appear in Research in Number Theory.

Moving to dimension 2, this paper introduces efficient formulæ for the computation of $(3, 3)$-isogenies. It forms the basis of Chapter 9. We develop algorithms to compute chains of $(3, 3)$-isogenies, which we use to construct a cryptographic hash function. By benchmarking our algorithms against the state-of-the-art, we show that they outperform those in the literature by at least a factor of 8.

Publications not included in this thesis

We now present the author's published works that have not been included in this thesis.

[89] Maria Corte-Real Santos, Craig Costello, and Michael Naehrig. "On Cycles of Pairing-Friendly Abelian Varieties". In *Advances in Cryptology - CRYPTO 2024 - 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2024, Proceedings, Part IX*, pages 221-253, 2024.

An avenue for realising scalable proof systems relies on the existence of cycles of pairing-friendly elliptic curves. In this work, we generalise the notion of cycles of pairing-friendly elliptic curves to study cycles of pairing-friendly abelian varieties, and give several new constructions that can be instantiated at any security level.

[277] Maria Corte-Real Santos, Jonathan Komada Eriksen, Michael Meyer, and Francisco Rodríguez-Henríquez. "Finding Practical Parameters for Isogeny-based Cryptography". *IACR Communications in Cryptology*, vol. 1, no. 3, 2024.

In this paper, we present new methods that combine previous techniques for finding suitable primes for isogeny-based cryptography. We apply these methods to find primes for SQIsign, AprèsSQI and public-encryption scheme POKE. The parameters we propose provide the best performance among all parameters proposed in the literature. Furthermore, the SQIsign primes we present were used in the NIST submission SQIsign (NIST).

[94] Maria Corte-Real Santos, and E. Victor Flynn. "Isogenies on Kummer Surfaces". To appear in Mathematics of Computation.

In this paper, we give a cleaner and more direct approach to the derivation of the *fast* model of the Kummer surface, which arises from the theory of theta functions. We then extend the methods presented in previous joint work with Costello and Smith [91], to give algorithms to compute (N, N)-isogenies for any odd N, both on the general Kummer surface (introduced by Cassels and Flynn) and on the fast model. We also provide an accompanying implementation in MAGMA.

Preprints

This author also has a paper that has yet to be published that is not included in the thesis.

[95] Maria Corte-Real Santos and Krijn Reijnders. "Return of the Kummer: a Toolbox for Genus-2 Cryptography". In *Cryptology ePrint Archive, Paper 2024/948*, 2024. https://eprint.iacr.org/2024/948

This paper expands the machinery we have for isogeny-based cryptography in genus 2 by developing a toolbox of algorithms for Kummer surfaces. To achieve this, we give a detailed landscape of Kummer surfaces, develop a new technique for sampling torsion points using profiles of Tate pairings, and present a new understanding of the connection between Richelot isogenies and isogenies constructed in the theta-model. We apply these new tools to show that one-dimensional SQIsign verification can be viewed as a two-dimensional isogeny defined over $\mathbb{F}_p$ between products of elliptic curves.

Code accompanying this thesis

The code accompanying this thesis has been written in MAGMA [49], SageMath [311] and Python. It is publicly available at the following GitHub repository:

https://github.com/mariascrs/thesis-code.

This repository contains submodules pointing towards the software associated to each chapter, from Chapter 7 to Chapter 11. Throughout this work, we also directly reference the relevant GitHub repositories in each chapter.

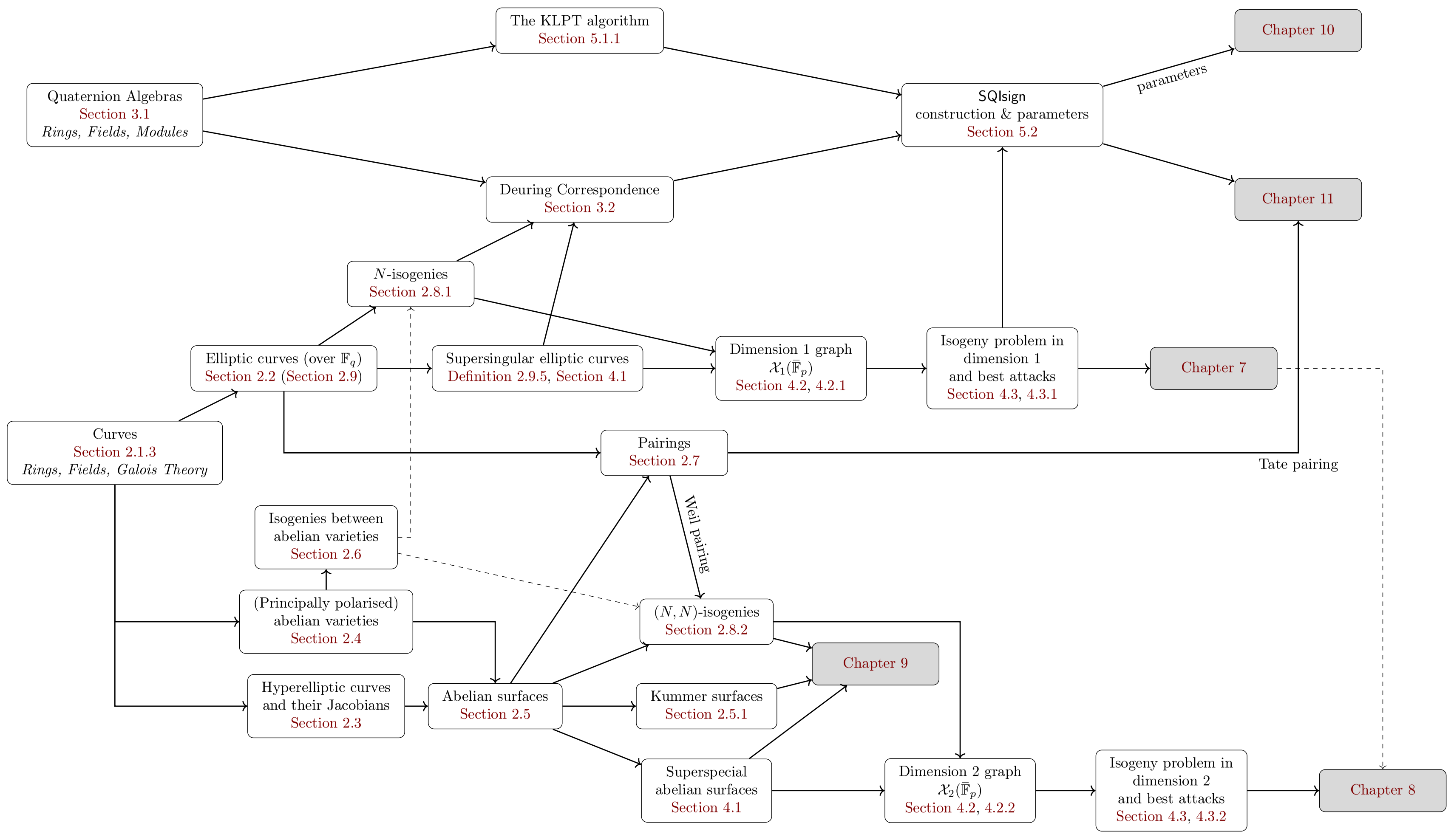


FIGURE 1: Visualising the dependencies between the preliminaries (boxes with white background) and the main chapters (boxes with grey background). In the left-most boxes, we detail in italics the mathematical preliminaries we assume from Section 1.2. The dotted arrows depict preliminaries which are desirable but not necessary.

List of Symbols

Symbols introduced in preliminaries

$\log(x)$	Base-2 logarithm of x
$\mathsf{A} \leftrightharpoons \mathsf{B}$	3-move interactive protocol between parties A and B
$\mathbb{Z}/N\mathbb{Z}$	Ring of integers modulo N
λ	The security parameter, or principal polarisation of an abelian variety (clear from context)
$\Bbbk$	Perfect field
$\operatorname{char} \Bbbk$	Characteristic of field $\Bbbk$ (usually > 5)
$\overline{\Bbbk}$	A (fixed) algebraic closure of a field $\Bbbk$
$\mathcal{O}_{\Bbbk}$	Ring of integers of the field $\Bbbk$
$\mathbb{F}_p$	Finite field of order p
$\mathbb{F}_{p^k}$	Degree k extension of $\mathbb{F}_p$
$\mathbb{F}_q$	Finite field of characteristic p (i.e., q is a power of p)
$\mathbb{Q}_p$	The p-adic numbers
$\mathbb{Q}(\sqrt{d})$	A quadratic field
$\operatorname{Gal}(\Bbbk'/\Bbbk)$	The Galois group of the field extension $\Bbbk'/\Bbbk$
$\mathbb{A}^n$	Affine n-space
$\mathbb{P}^n$	Projective n-space
$\mathbb{P}(w_0, \dots, w_n)$	Weighted projective n-space with weights $w_0, \dots, w_n$
C	A curve, usually hyperelliptic of genus $g \geq 1$
$\overline{\Bbbk}(C)$	Function field of a curve C
$\operatorname{Pic}^0(C)$	The Picard group of an irreducible non-singular projective curve C
$\operatorname{Jac}(C)$ or $\mathcal{J}_C$	The Jacobian of an irreducible non-singular projective curve C
π	The Frobenius endomorphism, or a quotient map (clear from context)
g	The genus of a curve C, or the dimension of an abelian variety
E	An elliptic curve
0_E	The identity of elliptic curve group, and often the point at infinity
j or $j(E)$	The j-invariant of an elliptic curve E

E^d	The quadratic twist of elliptic curve E by non-square $d \in \Bbbk^\times$
E^t	The unique quadratic twist of elliptic curve E defined over $\mathbb{F}_q$
$\mathrm{End}(E)$	The endomorphism ring of elliptic curve E
A	An abelian variety
0_A	The identity of the abelian variety
$[N]_A$	Multiplication-by-N endomorphism on the abelian variety A
$\oplus$ (or $+$)	The group law on an abelian variety A
$\ominus$ (or $-$)	The inverse of the group law on A
λ, μ, ν	Rosenhain invariants of a genus-2 curve
$\mathcal{K}$	A Kummer surface
I_2, I_4, I_6, I_{10}	The Igusa–Clebsch invariants of hyperelliptic curve C of genus 2
e_N	The N-Weil pairing
T_N	The Tate pairing of order N
t_N	The reduced Tate pairing of order N
G	The kernel of an $(N, \dots, N)$-isogeny, i.e., a maximal isotropic subgroup of $A[N]$
$\varphi, \widehat{\varphi}$	An $(N, \dots, N)$-isogeny and its dual
$\deg \varphi$	The degree of an isogeny φ
$D_{N,g}$	number of $(N, \dots, N)$-isogenies from a fixed domain p.p. abelian variety of dimension g
$\Phi_N(X, Y)$	Classical modular polynomial in $\mathbb{Z}[X, Y]$ of level N
$\Phi_{N,p}(X, Y)$	Classical modular polynomial in $\mathbb{F}_p[X, Y]$ of level N
$\mathcal{B}_{p,\infty}$	The quaternion algebra over $\mathbb{Q}$ ramified at p and ∞, and unramified elsewhere
$\bar{\alpha}$	Conjugate of an element $\alpha \in \mathcal{B}_{p,\infty}$
$\mathcal{O}$	Maximal order in $\mathcal{B}_{p,\infty}$
I, J, K	Ideals in $\mathcal{B}_{p,\infty}$
$\mathcal{O}_L(I)$	Left order of an ideal I
$\mathcal{O}_R(I)$	Right order of an ideal I
trd	The reduced trace of quaternion element α or ideal I
nrd	The reduced norm of quaternion element α or ideal I
$\bar{I}$	The conjugate of an ideal I in $\mathcal{B}_{p,\infty}$
E_0, $\mathcal{O}_0$	The elliptic curve $E_0\colon y^2 = x^3 + x$ with known endomorphism $\mathrm{End}(E_0) \cong \mathcal{O}_0$
I_φ	The ideal corresponding to an isogeny φ
$\mathcal{S}_g(\overline{\mathbb{F}}_p)$	Isomorphism classes of dimension-g superspecial p.p. abelian varieties

$\mathcal{X}_g(\bar{\mathbb{F}}_p)$	Graph with node set $\mathcal{S}_g(\bar{\mathbb{F}}_p)$ and edges being isogenies of p.p. abelian varieties
$\mathcal{X}_g(\bar{\mathbb{F}}_p, N)$	Graph with node set $\mathcal{S}_g(\bar{\mathbb{F}}_p)$ and edges being $(N, \ldots, N)$-isogenies
$\mathcal{S}_{p^2}$	The set of supersingular j-invariants in $\mathbb{F}_{p^2} \backslash \mathbb{F}_p$
$\mathcal{S}_p$	The set of supersingular j-invariants in $\mathbb{F}_p$
$\mathcal{J}_2(\bar{\mathbb{F}}_p)$	The set of superspecial p.p. abelian surfaces $\bar{\mathbb{F}}_p$-isomorphic to the Jacobian of a hyperelliptic genus-2 curve
$\mathcal{E}_2(\bar{\mathbb{F}}_p)$	The set of superspecial p.p. abelian surfaces $\bar{\mathbb{F}}_p$-isomorphic to $E_1 \times E_2$ for some supersingular elliptic curves E_1, E_2

SQIsign symbols

φ_{sk}	The secret isogeny with domain E_0 used as the signing key
D_{sk}	The degree of the secret isogeny
E_{pk}	The image of the secret isogeny φ_{sk} used as the public verification key
φ_{com}	The commitment isogeny with domain E_0
D_{com}	The degree of the commitment isogeny
E_{com}	The image of the commitment isogeny
φ_{chall}	The challenge isogeny with domain E_{com}
D_{chall}	The degree of the challenge isogeny
E_{chall}	The image of the challenge isogeny
φ_{resp}	The response isogeny, included in the SQIsign signature
2^e	The degree of the response isogeny
2^f	The degree of each block of the response isogeny

Part I

Preliminaries

Chapter 1

Cryptographic and Mathematical Preliminaries

Before we begin our journey towards introducing the central objects of our study – abelian varieties – and their use in isogeny-based cryptography, we present several important cryptographic and mathematical preliminaries. A reader with a strong cryptographic background may choose to skip Section 1.1 and simply consult this section for the definition of a term if necessary. Similarly, those with a good understanding of the mathematical definitions can skip Section 1.2.

1.1 Modern cryptography

We begin by defining the main cryptographic objects and concepts that we will be working with in this thesis. After presenting the basic building blocks we need throughout, we introduce digital signature schemes, a fundamental cryptographic primitive, which is used in a myriad of applications such as software distribution and financial transactions. Digital signatures will be central to the work in this thesis. We focus particularly on signatures schemes that are derived from identification schemes.

1.1.1 Basic definitions

We follow Katz and Lindell [201] to define basic concepts in modern cryptography that we will need throughout the thesis.

We begin by introducing some basic notation. If x is a binary string, we let $|x|$ denote its length. If S is a finite set, we denote by $\#S$ its size and $s \overset{\$}{\leftarrow} S$ denotes picking an element uniformly from S and assigning it to s.

1.1.1.1 Efficient computation

We define efficient computation as that which can be carried out in polynomial time. An algorithm is said to run in *polynomial time* if there exists a polynomial $f(\cdot)$ such that, for every input $x \in \{0,1\}^*$, the computation of the algorithm on x terminates in at most $f(|x|)$ operations. We use $y \leftarrow \mathsf{Alg}(x)$ to denote running an algorithm Alg on input x and assigning its output to y. We also often work with *probabilistic* polynomial time (PPT) algorithms, namely polynomial time algorithms with access to a source of randomness that yields unbiased random bits that are each independently equal to 1 or 0 with probability $\frac{1}{2}$. We often distinguish a probabilistic algorithm Alg using the notation $y \overset{\$}{\leftarrow} \mathsf{Alg}(x)$, for some input x and output y.

When determining the complexity of an algorithm or describing the size of certain inputs, we work asymptotically and use the following notation.

Definition 1.1.1. For two functions $f, g : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$, we say $f(n) = o(g(n))$ if for all $C > 0$, there exists $N \in \mathbb{N}$ such that for all $n > N$, we have $f(n) < Cg(n)$. We say $f(n) = O(g(n))$ if there exists a constant $C > 0$, such that there exists an $N \in \mathbb{N}$ with $f(n) < Cg(n)$ for all $n > N$. Furthermore, we define $\widetilde{O}(f(n)) := O(f(n)\log(n)^k)$ for some $k \in \mathbb{N}$.[1]

We say $f(n) = \omega(g(n))$ if for all $C > 0$, there exists $N \in \mathbb{N}$ such that for all $n > N$, we have $f(n) > Cg(n)$. We say $f(n) = \Omega(g(n))$ if there exists a constant $C > 0$, such that there exists $N \in \mathbb{N}$ with $f(n) > Cg(n)$ for all $n > N$.

Intuitively, the little-o notation says that f is dominated by g (for any constant factor C), whereas the big-O notation describes f that is bounded above by g (up to a constant factor C). The notation $\widetilde{O}$ ignores logarithmic factors in the big-O notation, and we often use this for clarity of presentation. Conversely, the ω notation says that f dominates g (for any constant factor C), and the Ω notation tells us that f is bounded below by g (up to a constant factor C).

Example 1.1.2. A polynomial time algorithm runs in $O(\ell^k)$ for some $k \in \mathbb{N}$, where ℓ is the length of the input. If an algorithm runs in $O(2^{f(\ell)})$ for some polynomial f, it is said to run in *exponential time.*

In Chapters 7 and 8 we will seek to understand the *concrete* complexity of certain algorithms, namely, we will explicitly determine the constant C. In the context of cryptography, the concrete complexity is often important to fine-tune parameters in order to achieve a certain level of security.

We also introduce notation used to analyse the average-case complexity of an algorithm.

Definition 1.1.3. For two functions $f, g : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$, we say $f(n) = \Theta(g(n))$ if there are constants $C_1, C_2 > 0$ and $N \in \mathbb{N}$ such that $C_1 g(n) < f(n) < C_2 g(n)$ for all $n > N$. We define $\widetilde{\Theta}(f(n)) := \Theta(f(n)\log(n)^k)$ for some $k \in \mathbb{N}$.

1.1.1.2 Game-based security in cryptography

We fix a parameter λ, called the *security parameter*, with unary representation 1^λ, which measures the security of a cryptographic protocol. We use code-based games in our security definitions [21]. A *game* $\mathsf{Game}_{\mathsf{A}}^{\mathsf{sec}}(\lambda)$ is played with respect to a security notion sec and adversary A, modelled as a PPT algorithm. It has a MAIN procedure whose output is the output of the game. We denote by $P(\mathsf{Game}_{\mathsf{A}}^{\mathsf{sec}}(\lambda))$ the probability that this output is equal to 1. In certain games, the adversary A will have access to a *random oracle*, which responds to every unique query with a random response chosen uniformly from its output domain. If a query is repeated, it responds in the same way.

In modern cryptography, a scheme is said to be secure according to some security notion sec if it can be broken only with very small probability. In particular, if an adversary A can win the corresponding game $\mathsf{Game}_{\mathsf{A}}^{\mathsf{sec}}$ with probability $1/f(n)$ where f is a polynomial taking positive values, then it is not secure. Conversely, we consider our scheme to be secure if the probability is asymptotically smaller than $1/f(n)$ for every polynomial f taking positive values. To capture this in our security definitions, we define *negligible functions.*

[1] $\log(x)$ will always denote the base-2 logarithm of x.

Definition 1.1.4. Let $\epsilon : \mathbb{N} \to \mathbb{R}_{>0}$ be a function. We say that ϵ is *negligible* if, for every polynomial $f(\cdot)$ there exists an $N \in \mathbb{N}$ such that for all $n > N$, we have $\epsilon(n) < \frac{1}{f(n)}$.

As we see in later sections, we say a scheme is secure (with respect to a security property sec) if the probability of an adversary A winning the corresponding game $\mathsf{Game}_{\mathsf{A}}^{\mathsf{sec}}$ is bounded above by a negligible function $\epsilon(\lambda)$.

Similarly, we will say that a problem is *hard* if there is no polynomial time algorithm in λ that solves the problem. In public-key cryptography, we build new cryptosystems whose security is tied to the hardness of such problems. For example, if there is an efficient attack against the discrete logarithm problem in (large) finite fields, the Diffie–Hellman key exchange [136] would no longer be secure.

1.1.2 Cryptographic hash functions

A fundamental primitive in modern cryptography is a cryptographic hash function. We follow Mittelbach and Fischlin [240] and Rogaway and Shrimpton [274].

A *(keyed) hash function* is a function $\mathsf{H}\colon \mathcal{K} \times \mathcal{M} \to \mathcal{Y}$, where $\mathcal{K}$ and $\mathcal{Y}$ are finite nonempty sets and $\mathcal{M}$ and $\mathcal{Y}$ are sets of strings. We insist that $\mathcal{Y} = \{0,1\}^\ell$ for some $\ell > 0$. The number ℓ is called the *hash length* of H. Furthermore, we insist that if $m \in \mathcal{M}$, then $\{0,1\}^{|m|} \subseteq \mathcal{M}$; as stated by Rogaway and Shrimpton [274, §2], this assumption is convenient, and a reasonable hash function should satisfy this. We usually write the first argument to H as a subscript so that $\mathsf{H}_k(m) = \mathsf{H}(k, m)$ for all $m \in \mathcal{M}$.
To obtain a cryptographic hash function, we give formal definitions of two notions of security. Firstly, in cryptography we will want our hash functions to yield few collisions, where a collision is a pair of inputs $m \neq m'$ such that $\mathsf{H}_k(m) = \mathsf{H}_k(m')$.

Definition 1.1.5. Let λ be the security parameter. A hash function $\mathsf{H}\colon \mathcal{K} \times \mathcal{M} \to \{0,1\}^\ell$ is *collision resistant* if for any PPT adversary A we have $P(\mathsf{Game}_{\mathsf{A}}^{\mathsf{coll}}) \leq \epsilon(\lambda)$, where $\epsilon : \mathbb{N} \to [0,1]$ is a negligible function and $\mathsf{Game}_{\mathsf{A}}^{\mathsf{coll}}$ is defined as follows:

```
MAIN $\mathsf{Game}_{\mathsf{A}}^{\mathsf{coll}}(\lambda)$
    $k \xleftarrow{\$} \mathcal{K}$
    $m, m' \xleftarrow{\$} \mathsf{A}(1^\lambda, k)$.
    return $(m \neq m') \wedge (\mathsf{H}_k(m) = \mathsf{H}_k(m'))$
```

The next security definition ensures that it is infeasible for an adversary to efficiently compute a preimage of the hash function.

Definition 1.1.6. Let λ be the security parameter and consider a hash function $\mathsf{H}\colon \mathcal{K} \times \mathcal{M} \to \{0,1\}^\ell$ and suppose $n > 0$ is such that $\{0,1\}^n \subseteq \mathcal{M}$. We say H is *preimage resistant* if for any PPT adversary A we have $P(\mathsf{Game}_{\mathsf{A}}^{\mathsf{pre}}) \leq \epsilon(\lambda)$, where $\epsilon : \mathbb{N} \to [0,1]$ is a negligible function and $\mathsf{Game}_{\mathsf{A}}^{\mathsf{pre}}$ is defined as follows:

```
MAIN $\mathsf{Game}_{\mathsf{A}}^{\mathsf{pre}}(\lambda)$
```

```
$k \xleftarrow{\$} \mathcal{K}$
$m \xleftarrow{\$} \{0,1\}^n$
$y \leftarrow \mathsf{H}_k(m)$
$m' \xleftarrow{\$} \mathsf{A}(1^\lambda, k, y)$.
**return** $\mathsf{H}_k(m') = y$
```

Given these two security definitions, we are ready to define cryptographic hash functions.

Definition 1.1.7. Let λ be the security parameter. A *cryptographic hash function* is a hash function $\mathsf{H}\colon \mathcal{K} \times \mathcal{M} \to \{0,1\}^{\ell(\lambda)}$ that is collision resistant and preimage resistant.

For some applications it suffices to rely on a security requirement that is weaker than collision resistance. We say that a hash function H is *second-preimage resistance* if, for $k \xleftarrow{\$} \mathcal{K}$ and $m \xleftarrow{\$} \{0,1\}^n \subseteq \mathcal{M}$ with $y := \mathsf{H}_k(m)$, it is infeasible for any PPT adversary A to find $m' \neq m$ such that $y = \mathsf{H}_k(m')$. If H is collision resistant, then it is also second-preimage resistant.

Remark 1.1.8. For simplicity, we usually consider *unkeyed* hash functions, which can be modeled as a keyed hash function with a single fixed key k that is known to everyone.

1.1.3 Identification schemes

In this section, we recall the standard cryptographic notions of Σ-protocols and identification schemes following the lecture notes by Damgård [113] and Venturi [315]. This will prove most important for Section 5.2 of Chapter 5, where we define the isogeny-based signature scheme SQIsign constructed from an identification protocol. Another good general reference for an interested reader is Chapter 8 by Katz [200].

1.1.3.1 Σ-Protocols

Let $R : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}$ be a relation defining a language $\mathcal{L}_R := \{x\colon \exists w \text{ s.t. } R(x,w) = 1\}$. We consider protocols between a prover P and a verifier V, both modelled as PPT algorithms with respect to the security parameter λ. The prover holds a witness w for a value $x \in \mathcal{L}$, and their goal is to convince the verifier of this through an interactive protocol. We can view x to be an instance of some computational problem, and w is a solution to that instance. We restrict to 3-move interactive protocols of the following form:

1. P sends a message a.
2. V sends a random t-bit string b.
3. P sends a reply c, and V outputs `accept` or `reject`.

This interaction will yield a transcript (a, b, c) where a is the *commitment*, b is the *challenge*, and c is the *response*. Intuitively, the role of the commitment step is to bind the prover to some information at the beginning of interaction to ensure that they behave correctly in the response phase. In the challenge phase, the verifier sends a random string that the prover should not be able to anticipate. We denote such 3-move interactive protocols between prover and verifier by $\mathsf{P}(x, w) \leftrightharpoons \mathsf{V}(x)$.

Definition 1.1.9. A protocol is said to be a Σ-*protocol* for a relation R if it is of the above 3-move form. We say that a Σ-protocol is:

- $(1-\delta)$-*Correct* if for all $x \in \mathcal{L}_R$ we have that $\pi = (\mathsf{P}(x, w) \leftrightharpoons \mathsf{V}(x))$ is a valid transcript with probability $1 - \delta$ (over the randomness of all involved algorithms) for $R(x, w) = 1$. If $\delta = 0$, then the Σ-protocol is said to be *perfectly correct.*
- *Special sound* if on input $x \in \mathcal{L}_R$, and any pair of accepting transcripts $\pi = (a, b, c)$ and $\pi' = (a, b', c')$ with $b \neq b'$, we can efficiently compute w such that $R(x, w) = 1$.
- *(Computationally) Honest verifier zero-knowledge* if there exists an efficient simulator S, which on input $x \in \mathcal{L}_R$ and $b \overset{\$}{\leftarrow} \{0, 1\}^t$, outputs an accepting transcript $\pi = (a, b, c)$ that are (computationally) indistinguishable from transcripts of the real protocol.

The special soundness property implies that a Σ-protocol for relation R is always an interactive proof system for the language $\mathcal{L}_R$ with soundness error 2^{-t}.

1.1.3.2 Identification schemes

An identification scheme is an interactive protocol between a prover P and a verifier V, where the prover aims to convince the verifier that it knows some secret without revealing anything about it.

Definition 1.1.10. An *identification (ID) scheme* is a tuple of PPT algorithms $(\mathsf{Gen},\mathsf{P},\mathsf{V})$ specified as follows:

- $(\mathsf{pk}, \mathsf{sk}) \overset{\$}{\leftarrow} \mathsf{Gen}(1^\lambda)$: Takes as input the security parameter λ and outputs a key pair $(\mathsf{pk}, \mathsf{sk})$.
- $\pi = (\mathsf{P}(\mathsf{pk}, \mathsf{sk}) \leftrightharpoons \mathsf{V}(\mathsf{pk}))$: Denotes an interactive protocol, where P holds the key pair $(\mathsf{pk}, \mathsf{sk})$ and V holds pk, at the end of which a transcript π is output.
- The verifier V outputs `accept` if π is valid, or `reject` otherwise.

We say that the ID scheme is *correct* if the verifier outputs `accept` with probability 1 over the choice of $(\mathsf{pk}, \mathsf{sk}) \overset{\$}{\leftarrow} \mathsf{Gen}(1^\lambda)$ and over the randomness in the involved algorithms.

The most basic form of security for ID schemes is *passive* security. It aims to capture the notion that it should be hard for a dishonest prover to convince the verifier that they know the secret key corresponding to a particular public key. Passive security will be sufficient for our purposes of constructing a signature scheme.

Definition 1.1.11. We say $(\mathsf{Gen}, \mathsf{P}, \mathsf{V})$ is a *passively secure ID scheme* if, for all PPT adversaries A there exists a negligible function $\epsilon : \mathbb{N} \to [0, 1]$ such that $P(\mathsf{Game}^{\mathsf{id}}_{\mathsf{A}}) \leq \epsilon(\lambda)$ where $\mathsf{Game}^{\mathsf{id}}_{\mathsf{A}}$ is defined as follows:

```
MAIN $\mathsf{Game}^{\mathsf{id}}_{\mathsf{A}}(\lambda)$
    $Q \leftarrow \emptyset$
```

```
        $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{Gen}(1^\lambda)$
        $s \xleftarrow{\$} \mathsf{A}^{\textsc{Transcript}}(1^\lambda, \mathsf{pk})$
        $\pi \leftarrow (\mathsf{A}(s) \leftrightharpoons \mathsf{V}(\mathsf{pk}))$
        **return** $(\pi \text{ valid}) \wedge (\#Q = O(\mathsf{poly}(\lambda))) \wedge (\pi \notin Q)$

    $\underline{\textsc{Transcript}()}$

        $\pi_i \leftarrow \mathsf{P}(\mathsf{pk}, \mathsf{sk}) \leftrightharpoons \mathsf{V}(\mathsf{pk})$
        $Q \leftarrow Q \cup \{\pi_i\}$
        **return** $\pi_i$
```

An ID scheme can be constructed from a Σ-protocol for a hard relation R.

Definition 1.1.12. A relation R is said to be *hard* if there exists a PPT algorithm Gen that, on input of the security parameter $\lambda \in \mathbb{N}$, will output a pair (x, w) such that $R(x, w) = 1$, and for all PPT adversaries A we have $P(\mathsf{Game}_{\mathsf{A}}^{\mathsf{hard}}) \leq \epsilon(\lambda)$, for a negligible function $\epsilon \colon \mathbb{N} \to [0, 1]$. Here, $\mathsf{Game}_{\mathsf{A}}^{\mathsf{hard}}$ is defined as follows:

```
    $\underline{\textsc{Main}\ \mathsf{Game}_{\mathsf{A}}^{\mathsf{id}}(\lambda)}$

        $(x, w) \xleftarrow{\$} \mathsf{Gen}(1^\lambda)$
        $w' \xleftarrow{\$} \mathsf{A}(1^\lambda, x)$
        **return** $R(x, w') = 1$
```

The idea to obtain an ID scheme from a Σ-protocol of such a relation R is to generate a pair $(x, w) \leftarrow \mathsf{Gen}(1^\lambda)$ and to define $\mathsf{pk} := x$ and $\mathsf{sk} := w$. The execution of the scheme then proceeds as the underlying Σ-protocol. More formally, we have the following theorem, whose proof can be found in, for example, the lecture notes by Venturi [315, Theorem 5].

Theorem 1.1.13. *Let* (P, V) *be a* Σ*-protocol for a hard relation* R *with challenge space of size* $\omega(\log \lambda)$*. Then* (P, V) *is a passively secure ID scheme.*

1.1.4 Signature schemes from the Fiat–Shamir transform

We now exhibit an important application of identification schemes, namely the construction of provably secure signature schemes in the *random oracle model* using the Fiat–Shamir transform. In the random oracle model, we assume there is a publicly-accessible oracle that implements a completely random function, usually instantiated with a cryptographic hash function. A deep understanding of the random oracle model will not be needed for this work; indeed, we will only use it in this section to understand the construction of SQIsign, an isogeny-based signature scheme introduced in Section 5.2. We refer to Bellare and Rogaway [20] for more details.

First, we define digital signatures and the security properties we require them to have.

Definition 1.1.14. A *signature scheme* $\Pi = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Verify})$ is composed of the triple of polynomial time algorithms defined as follows:

- $(\mathsf{vk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KGen}(1^\lambda)$: The key generation algorithm takes as input the security parameter $\lambda \in \mathbb{N}$, and outputs a key pair $(\mathsf{vk}, \mathsf{sk})$ composed of the public verification key vk and private signing key sk.
- $\sigma \xleftarrow{\$} \mathsf{Sign}(\mathsf{sk}, m)$: The signing algorithm takes as input the signing key sk and a message $m \in \{0,1\}^*$ (and random coins), and returns a signature σ.
- $\mathtt{accept}/\mathtt{reject} \leftarrow \mathsf{Verify}(\mathsf{vk}, \sigma, m)$: The verification algorithm takes as input the verification key vk, the message m and the signature σ, and outputs `accept` or `reject`. We often depict `accept` by 1 and `reject` by 0.

We say that the signature scheme Π is correct if for all $m \in \{0,1\}^*$ and all $(\mathsf{vk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KGen}(1^\lambda)$, we have $\mathsf{Verify}(\mathsf{vk}, m, \mathsf{Sign}(\mathsf{sk}, m)) = 1$ with probability 1 over the choice of $(\mathsf{vk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{Gen}(1^\lambda)$ and over the randomness of all involved algorithms.

For security, we require that, given polynomially many valid signatures and (chosen) message pairs, it should be hard to forge a signature on a new message.

Definition 1.1.15. We say that a signature scheme $\Pi = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Verify})$ is *unforgeable against chosen-message attacks* (UF-CMA) if for all PPT adversaries A there exists a negligible function $\epsilon : \mathbb{N} \to [0,1]$ such that $P(\mathsf{Game}_{\mathsf{A}}^{\mathsf{UF\text{-}CMA}}) \leq \epsilon(\lambda)$ in the following game:

```
$\underline{\textsc{Main}\ \mathsf{Game}_{\mathsf{A},\Pi}^{\mathsf{UF\text{-}CMA}}(\lambda)}$
    $Q \leftarrow \emptyset$
    $(\mathsf{vk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KGen}(1^\lambda)$
    $(m, \sigma) \xleftarrow{\$} \mathsf{A}^{\textsc{Sign}}(1^\lambda, \mathsf{pk})$
    **return** $(\mathsf{Verify}(\mathsf{vk}, \sigma, m)) \wedge (m \notin Q)$

$\underline{\textsc{Sign}(m)}$
    $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m)$
    $Q \leftarrow Q \cup \{m\}$
    **return** $\sigma$
```

1.1.4.1 The Fiat–Shamir transform

The Fiat–Shamir transform [151] allows us to construct a signature scheme from a Σ-protocol for any hard relation R, or equivalently from a passively secure ID scheme. The intuitive idea of the transformation is to replace the challenge sampled by the verifier by the output of a cryptographic hash function, which takes as input the commitment a and message m. This is sufficient to ensure that the prover does not have access to the challenge before generating the commitment, and binds the commitment to the message that will be signed in that session. More formally, we construct the signature scheme Π as follows:

- $(\mathsf{vk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KGen}(1^\lambda)$: The key generation algorithm fixes a hash function $\mathsf{H}\colon \{0,1\}^* \to \{0,1\}^*$, runs the generation protocol $(x, w) \leftarrow \mathsf{Gen}(1^\lambda)$ of the underlying ID scheme where $(x, w) \in \mathcal{L}_R$, and outputs verification key $\mathsf{vk} := (\mathsf{H}, x)$ and signing key $\mathsf{sk} := (w, \mathsf{H}, x)$.

- $\sigma \xleftarrow{\$} \mathsf{Sign}(\mathsf{sk}, m)$: The signing algorithm performs a run of the underlying Σ-protocol by parsing sk as (w, H, x), computing the challenge $b = \mathsf{H}(x, m, a)$, and returns $\sigma := (a, c)$ as a signature on m, where a is the commitment and c is the response (see Section 1.1.3.1).

- `accept/reject` $\leftarrow \mathsf{Verify}(\mathsf{vk}, \sigma, m)$: The verification algorithm parses vk as (H, x) and σ as (a, c), computes $b' = \mathsf{H}(x, m, a)$, and returns `accept` (or 1) if (a, b', c) is a valid transcript, and `reject` (or 0) otherwise.

Fiat and Shamir [151] showed that this construction gives a secure signature scheme (in the sense of Definition 1.1.15), leading to the following theorem. See, for example Venturi [315, Theorem 7], for a proof.

Theorem 1.1.16. *Let* (P, V) *be a* 3*-round passively secure ID scheme, such that the first message* a *has conditional min-entropy*[2] $\omega(\log \lambda)$ *given the public key* pk*. Then, the signature scheme derived by applying the Fiat–Shamir transform is UF-CMA secure in the random oracle model.*

Remark 1.1.17. With a view to the work presented in this thesis, all signature schemes are constructed from identification protocols. As a result, we will make heavy use of Theorem 1.1.16. As we are also interested in security against quantum adversaries, it is natural to seek an extension of the Fiat–Shamir transform that is secure in the *Quantum Random Oracle Model* (QROM), a quantum version of the random oracle model introduced by Boneh, Dagdelen, Fischlin, Lehmann, Schaffner, and Zhandry [39]. Unruh [313] proposed an adaptation of the Fiat–Shamir transform that is secure in the QROM. However, Unruh's transform is expensive.

1.2 Mathematical preliminaries

The purpose of this section is to point to references for mathematical concepts that may not be known to a more general audience, but will be assumed throughout the thesis.

Rings. We refer to Artin [6, Chapter 11], or Cohn [85] for a complete description of ring theory. Most pertinent to this thesis is the definition of the characteristic of a ring [85, §6.3], integral domains (namely non-zero commutative rings such that product of any two non-zero elements is non-zero) [6, §11.7], and the following two examples of rings.

Example 1.2.1. We briefly fix notation for two types of rings that we will need for Chapter 3: division rings and matrix rings.

- A *division ring* is a non-trivial ring in which division by non-zero elements is defined. In particular, for every non-zero element a, there exists a multiplicative inverse a^{-1} such that $a \cdot a^{-1} = a^{-1} \cdot a = 1$. In Chapter 3, we introduce an example of a division ring: quaternion algebras over $\mathbb{Q}$.

- A *matrix ring* is a set of matrices with entries in a ring R that form a ring under matrix addition and matrix multiplication. For example, 2×2 matrices with entries in the real numbers $\mathbb{R}$, denoted $M_2(\mathbb{R})$, is a matrix ring.

[2]If $P = (p_1, \ldots, p_n)$ is a finite probability distribution, its min-entropy is defined as $\log(1/p_{\max})$ where $p_{\max} = \max_i(p_i)$. It is the most conservative way of measuring the unpredictability of a set of outcomes.

A reader should also be familiar with polynomial rings as defined in [6, §11.2], and ideals in such rings [6, Definition 11.3.13].

Fields. A field $\Bbbk$ is a commutative division ring. Field theory is a vast topic and good references for this topic are Cameron [63] or Milne [238]. For the work in this thesis, a reader should be familiar with the field of fractions (or quotient field) corresponding to an integral domain [63, §2.14] and field extensions [297, §4.1]. An important field extension is the algebraic closure of $\Bbbk$, denoted $\overline{\Bbbk}$, as defined in Milne [238, Chapter 6]. We will often use properties of field extensions, including their degree [238, pg. 14], their transcendental degree [238, Chapter 9] and whether they are separable or (purely) inseparable [238, Chapter 3]. Furthermore, in Chapters 7 and 10 we work with the trace and norm map of a field extension, as defined by, for example, Milne [238, pg. 82]. We briefly detail three examples of fields that are used in this work.

Example 1.2.2 (Finite Fields)**.** In cryptography, the main example of a field that we work with is a *finite field*. A field is said to *finite* if it has finite cardinality. If $\Bbbk$ is a finite field, then it has prime characteristic p and has order p^k for some $k \in \mathbb{N}$. In fact, all finite fields with characteristic p and order p^k are isomorphic. We can therefore identify all finite fields with the same order p^k, and denote them by $\mathbb{F}_{p^k}$. A finite field is an example of a *perfect field*. We define $\Bbbk$ to be perfect if every finite extension of $\Bbbk$ is separable [238, pg. 33]. Another example of a perfect field is the rational numbers $\mathbb{Q}$.

Example 1.2.3 (The p-adic numbers)**.** For our discussion on quaternion algebras defined over $\mathbb{Q}$ in Section 3.1, it will be important to study the behaviour at particular primes p. This means that we study what our quaternion algebra looks like when it is defined over the p-adic numbers $\mathbb{Q}_p$. A p-adic number $a \in \mathbb{Q}_p$ is given by the series

$$a = \sum_{i=k}^{\infty} a_i p^i = a_k p^k + a_{k+1} p^{k+1} + \cdots ,$$

where k is a (possibly negative) integer and $0 \leq a_i < p$. For more details, see [45, Chapter 1, §3].

Example 1.2.4 (Quadratic Fields)**.** Quadratic fields have found many applications in isogeny-based cryptography, particularly when working with ordinary elliptic curves defined over $\overline{\mathbb{F}}_p$ or supersingular elliptic curves defined over $\mathbb{F}_p$, both of which will be introduced in Section 2.9. In this manuscript, however, we focus on supersingular curves defined over $\mathbb{F}_{p^2}$, the reasons for which will be explained in Section 4.1. As such, we do not explicitly require in-depth knowledge on quadratic fields. We define them here briefly for completeness; for more details we refer the reader to Buell [61].

A quadratic field is given by $\mathbb{Q}(\sqrt{d})$ for some (uniquely defined) square-free integer $d \neq 0$. If $d > 0$, $\mathbb{Q}(\sqrt{d})$ is a *real* quadratic field, and when $d > 0$ it is an *imaginary* quadratic field. In cryptography, we mostly work with the latter case.

The *ring of integers* $\mathcal{O}_\Bbbk$ of a quadratic field $\Bbbk$ is the ring of all algebraic integers in $\Bbbk$ (i.e., field elements that are roots of monic polynomials with integer coefficients). Explicitly, for $\Bbbk = \mathbb{Q}(\sqrt{d})$, we have $\mathcal{O}_\Bbbk$ is $\mathbb{Z}\left[\frac{1+\sqrt{d}}{2}\right]$ for $d \equiv 1 \bmod 4$, and $\mathbb{Z}[\sqrt{d}]$ otherwise. For example, when $\Bbbk = \mathbb{Q}(\sqrt{-5})$ we have $\mathcal{O}_\Bbbk = \mathbb{Z}[\sqrt{-5}]$.

The class group of a quadratic field is a widely studied object both in mathematics and cryptography. To define it, we first introduce fractional ideals. An *ideal* I in the ring $\mathcal{O}_{\Bbbk}$ is a subset of $\mathcal{O}_{\Bbbk}$ such that: (a) I is subgroup of the additive group of $\mathcal{O}_{\Bbbk}$; and (b) for all $x \in I$ and $r \in \mathcal{O}_{\Bbbk}$, we have $rx \in I$. An ideal I is a *fractional ideal* if there exists a fixed algebraic integer $\alpha \in \mathcal{O}_{\Bbbk}$ such that for every $x \in I$, we have $\alpha x \in \mathcal{O}_{\Bbbk}$. An ideal I is *principal* if there exists an algebraic integer α such that $I = \{r\alpha \colon r \in \mathcal{O}_{\Bbbk}\}$. We can define the product ideal IJ of I and J to be the ideal containing all finite sums of products $\sum x_i y_i$ for $x_i \in I, y_i \in J$. In fact, the set of fractional ideals forms a group $J_{\Bbbk}$ under ideal multiplication, and the set of principal ideals $P_{\Bbbk} \subseteq J_{\Bbbk}$ is a subgroup.

The *class group* of $\Bbbk$ is the quotient group $J_{\Bbbk}/P_{\Bbbk}$. The order of the group, which is finite, is called the *class number* of $\Bbbk$.

Galois Theory. In Chapter 2, we define our objects over the algebraic closure $\overline{\Bbbk}$ of a field $\Bbbk$. However, it is important to understand when they are defined over the base field $\Bbbk$. To define this formally, we must work with Galois groups of the form $\mathrm{Gal}(\overline{\Bbbk}/\Bbbk)$. For a reader unfamiliar with Galois theory, the book by Stewart [297] gives a very readable introduction to the topic. The case where $\Bbbk = \mathbb{F}_q$ is particularly simple. However, for the purposes of the work in this thesis, it suffices to understand the explicit examples that we give in Sections 2.2 and 2.3, for which the question of whether it is defined over $\Bbbk$ becomes clearer.

Modules. A *module* is an analogue of a vector space, where the scalar field is replaced by a ring. A reader should be familiar with the definition of a (sub)module, given for example by Artin [6, §14.1].

Chapter 2

Abelian Varieties of Low Dimension

This chapter is dedicated to introducing the geometric objects that we study in this work: principally polarised abelian varieties of dimension 1 and 2. We begin by defining *affine* and *projective curves*, and present our two examples of interest: elliptic curves and hyperelliptic curves of genus 2. Further to this, we show how to associate a group to such curves using the construction of the *Picard group*. This will form our first examples of *principally polarised abelian varieties*, the main object of study in this thesis. As we will see in this chapter, an abelian variety is a *projective algebraic variety* that is also an *algebraic group*. The group structure of these objects is what enables their use in cryptography. For instance, by looking at specific examples of abelian varieties over finite fields, we can instantiate a group where the discrete logarithm problem is hard. We focus in particular on principally polarised abelian varieties of dimension 1 and 2, fully categorising all such varieties and detailing specific properties they hold. With our goal of isogeny-based cryptography in mind, we define *isogenies* between abelian varieties, namely a morphism with special properties which preserve the group structure. We finish by specialising our discussions to abelian varieties defined over a finite field. As we span many topics throughout this chapter, we indicate the relevant literature at the beginning of each section.

For a reader unfamiliar with the mathematical background, we refer to Section 1.2, which gives references for undefined terms in this chapter. For simplicity, we let $\Bbbk$ be a perfect field and fix an algebraic closure $\overline{\Bbbk}$ of $\Bbbk$.

2.1 Curves

In this section, we give a brief introduction to curves with their cryptographic applications in mind. We begin by giving a general description of affine and projective varieties, before specialising to those that define algebraic curves. We then introduce the Picard group associated to such a curve, which allows us to construct a group corresponding to any (smooth, irreducible) curve. We emphasize that the most important takeaways of this section are the two concrete examples of algebraic curves that we give in Section 2.2 and Section 2.3, namely (hyper)elliptic curves.

2.1.1 Affine and projective varieties

We first describe affine and projective varieties via the theory of algebraic sets. We show how to define a geometric object in affine or projective space as the vanishing set of multivariate polynomials. We refer to [164, 264, 292] for general references for this section.

2.1.1.1 Affine varieties

For a positive integer n and field $\mathbb{k}$ with algebraic closure $\overline{\mathbb{k}}$, we define *affine n-space* $\mathbb{A}^n(\overline{\mathbb{k}})$ to be the set of n-tuples
$$\mathbb{A}^n = \mathbb{A}^n(\overline{\mathbb{k}}) := \{P = (x_1, \dots, x_n) \colon x_i \in \overline{\mathbb{k}}\}.$$
In particular, $\mathbb{A}^1(\overline{\mathbb{k}}) = \overline{\mathbb{k}}$ is the *affine line*, and $\mathbb{A}^2(\overline{\mathbb{k}})$ is the *affine plane*. The *set of $\mathbb{k}$-rational points of $\mathbb{A}^n$* is the set
$$\mathbb{A}^n(\mathbb{k}) := \{P = (x_1, \dots, x_n) \in \mathbb{A}^n \colon x_i \in \mathbb{k}\}.$$

Remark 2.1.1. Let $\mathrm{Gal}(\overline{\mathbb{k}}/\mathbb{k})$ be the (absolute) Galois group of the extension $\overline{\mathbb{k}}/\mathbb{k}$. Then $\mathrm{Gal}(\overline{\mathbb{k}}/\mathbb{k})$ acts on $\mathbb{A}^n$ as follows. For $\sigma \in \mathrm{Gal}(\overline{\mathbb{k}}/\mathbb{k})$ and $P = (x_1, \dots, x_n) \in \mathbb{A}^n$, we define $\sigma(P) := (\sigma(x_1), \dots, \sigma(x_n))$. The set $\mathbb{A}^n(\mathbb{k})$ can instead be characterised by
$$\mathbb{A}^n(\mathbb{k}) = \{P \in \mathbb{A}^n \colon \sigma(P) = P \text{ for all } \sigma \in \mathrm{Gal}(\overline{\mathbb{k}}/\mathbb{k})\}.$$

For a polynomial $f \in \mathbb{k}[X_1, \dots, X_n]$, a point $P = (x_1, \dots, x_n) \in \mathbb{A}^n(\mathbb{k})$ is called a *zero* of f if $f(P) = f(x_1, \dots, x_n) = 0$. We are interested in the algebraic sets defined by the vanishing of polynomials $f \in \mathbb{k}[X_1, \dots, X_n]$.

Definition 2.1.2. Let $\overline{\mathbb{k}}[X_1, \dots, X_n]$ be a polynomial ring in n variables, and let $I \subset \overline{\mathbb{k}}[X_1, \dots, X_n]$ be an ideal. An *affine algebraic set* is any set of the form
$$V(I) = \{P \in \mathbb{A}^n \colon f(P) = 0 \text{ for all } f \in I\}.$$
Similarly, if V is an affine algebraic set, the *ideal of V*, denoted $I(V)$ is generated by
$$\{f \in \overline{\mathbb{k}}[X_1, \dots, X_n] \colon f(P) = 0 \text{ for all } P \in V\}.$$

An affine algebraic set V is defined over $\mathbb{k}$ if it can be generated by polynomials in $\mathbb{k}[X_1, \dots, X_n]$. In this case, the set of $\mathbb{k}$-rational points of V is given by the set $V(\mathbb{k}) = V \cap \mathbb{A}^n(\mathbb{k})$.

Definition 2.1.3. An *(irreducible) affine variety* is an affine algebraic set V for which $I(V)$ is a prime ideal in $\overline{\mathbb{k}}[X_1, \dots, X_n]$.

Remark 2.1.4. For an affine variety V, when $I(V) = (f)$ for $f \in \mathbb{k}[X_1, \dots, X_n]$, then $I(V)$ is a prime (and therefore V is irreducible) if f is irreducible over $\mathbb{k}$ (see, for example, [164, Example 4.15(ii)]).

Example 2.1.5. Consider $f = y^2 - x^3 - 17 \in \overline{\mathbb{F}}_5[x, y]$ defining an ideal $I = (f)$. Viewing f as a polynomial over $\overline{\mathbb{F}}_5[x]$, f is irreducible and so I is a prime ideal. The algebraic set V corresponding to I is an affine variety given by the single equation $y^2 = x^3 + 17$. The $\mathbb{F}_5$-rational points of V are
$$V(\mathbb{F}_5) = \{(2,0), (3,2), (3,3), (4,1), (4,4)\}.$$

To define the dimension of an affine variety V, we need to construct the function field of V. Consider the ring
$$\overline{\mathbb{k}}[V] = \overline{\mathbb{k}}[X_1, \dots, X_n]/I(V).$$

As $I(V)$ is prime, $\overline{\mathbb{k}}[V]$ is an integral domain. The *function field of* V is the quotient field (or field of fractions) $\overline{\mathbb{k}}(V)$ of the integral domain $\overline{\mathbb{k}}[V]$.

Definition 2.1.6. Let V be a variety. The *dimension of* V, denoted $\dim(V)$, is the transcendence degree of $\overline{\mathbb{k}}(V)$ over $\overline{\mathbb{k}}$.

For a variety V defined over $\mathbb{k}$, we can similarly define $\mathbb{k}[V]$ and $\mathbb{k}(V)$. They are the subsets of $\overline{\mathbb{k}}[V]$ and $\overline{\mathbb{k}}(V)$, respectively, that are fixed by $\mathrm{Gal}(\overline{\mathbb{k}}/\mathbb{k})$.

Example 2.1.5 (continuing from p. 35)**.** The dimension of $\mathbb{A}^n$ is n as $\overline{\mathbb{k}}(\mathbb{A}^n) = \overline{\mathbb{k}}(X_1, \dots, X_n)$. The affine variety V defined by $y^2 = x^3 + 17$ has function field

$$\overline{\mathbb{F}}_5(V) = \overline{\mathbb{F}}_5[x, y]/(y^2 - x^3 - 17),$$

and $\dim(V) = 1$. More generally, if $V \subset \mathbb{A}^n$ is given by a single non-constant polynomial equation $f(X_1, \dots, X_n) = 0$, then $\dim(V) = n - 1$ [292, Example I.1.4].

2.1.1.2 Projective varieties

Let $P = (x_0, \dots, x_n) \in \mathbb{A}^{n+1}(\mathbb{k})$ be a point in affine $(n+1)$-space. If $P \neq (0, \dots, 0)$, then it defines a unique line passing through P and $(0, \dots, 0)$. Rather than considering all non-zero points on this line distinct, we can identify them. Using this identification, we can define (weighted) projective n-space.

Definition 2.1.7. We define *projective n-space* to be the set

$$\mathbb{P}^n = \mathbb{P}^n(\overline{\mathbb{k}}) := \left\{ (x_0 \colon x_1 \colon \cdots x_n) \colon x_i \in \overline{\mathbb{k}} \text{ with } \prod x_i \neq 0 \right\} \Big/ \sim,$$

where $\sim$ is the equivalence relation:

$$(x_0 \colon x_1 \colon \cdots \colon x_n) \sim (y_0 \colon y_1 \colon \cdots \colon y_n) \iff (y_0 \colon y_1 \colon \cdots \colon y_n) = (\lambda x_0 \colon \lambda x_1 \colon \cdots \colon \lambda x_n),$$

for some non-zero $\lambda \in \overline{\mathbb{k}}^{\times}$. The set of $\mathbb{k}$-*rational points in* $\mathbb{P}^n(\overline{\mathbb{k}})$ is defined as

$$\mathbb{P}(\mathbb{k}) := \{ (x_0 \colon x_1 \colon \cdots \colon x_n) \in \mathbb{P}^n \colon x_i \in \mathbb{k} \text{ for all } i \} \subseteq \mathbb{P}^n(\overline{\mathbb{k}}).$$

Example 2.1.8. When $n = 0$, the zero-dimensional projective space $\mathbb{P}^0(\overline{\mathbb{k}})$ is a single point. When $n = 1$, we have

$$\mathbb{P}^1(\overline{\mathbb{k}}) := \{ (x_0 \colon x_1) \colon x_0 \text{ or } x_1 \neq 0 \} / \sim \;\; = \;\; \{ (1 \colon x) \colon x \in \overline{\mathbb{k}} \} \cup \{ (0 \colon 1) \},$$

which is the *projective line*: an affine line with one extra point $(0 \colon 1)$, often called the *point at infinity*.

Remark 2.1.9. The Galois group $\mathrm{Gal}(\overline{\mathbb{k}}/\mathbb{k})$ acts on $\mathbb{P}^n$ by acting on the projective coordinates. For $P = (x_0 : \cdots : x_n) \in \mathbb{P}^n$ we have $\sigma\left((x_0 : \cdots : x_n)\right) := (\sigma(x_0) : \cdots : \sigma(x_n))$. This action is well-defined:

$$\sigma\left((\lambda x_0 : \cdots : \lambda x_n)\right) = (\sigma(\lambda)\sigma(x_0) : \cdots : \sigma(\lambda)\sigma(x_n)) = (\sigma(x_0) : \cdots : \sigma(x_n)).$$

Therefore, we can characterise the set of $\Bbbk$-rational points in $\mathbb{P}^n$ as $\mathbb{P}^n(\Bbbk) = \{P \in \mathbb{P}^n : \sigma(P) = P \text{ for all } \sigma \in \mathrm{Gal}(\overline{\Bbbk}/\Bbbk)\}$.

Definition 2.1.10. We define *n-dimensional weighted projective space* with pairwise coprime weights $\omega_0, \dots, \omega_n$ to be the set

$$\mathbb{P}(\omega_0, \dots, \omega_n)(\overline{\Bbbk}) := \{(x_0 \colon x_1 \colon \cdots \colon x_n) \mid x_i \in \overline{\Bbbk}, \text{ not all } x_i \neq 0\}/\sim,$$

where $\sim$ is the following equivalence relation:

$$(x_0 \colon x_1 \colon \cdots \colon x_n) \sim (y_0 \colon y_1 \colon \cdots \colon y_n) \iff (y_0 \colon y_1 \colon \cdots \colon y_n) = (\lambda^{\omega_0} x_0 \colon \lambda^{\omega_1} x_1 \colon \cdots \colon \lambda^{\omega_n} x_n),$$

for some non-zero $\lambda \in \overline{\Bbbk}$. We can define the set of $\Bbbk$-*rational points in* $\mathbb{P}(\omega_0, \dots, \omega_n)(\overline{\Bbbk})$ analogously to in Definition 2.1.7.

It will be important to note that affine n-space $\mathbb{A}^n$ can be embedded into projective n-space $\mathbb{P}^n$ via

$$\mathbb{A}^n \hookrightarrow \mathbb{P}^n, \quad (x_1, \dots, x_n) \mapsto (x_1 \colon \dots \colon x_n \colon 1).$$

By considering subsets of $\mathbb{P}^n$, we can define maps in the converse direction. Namely, let $U_i := \{(x_0 \colon \cdots \colon x_n) \in \mathbb{P}^n \mid x_i \neq 0\}$. The map $\phi_i : U_i \to \mathbb{A}^n$, defined as

$$(x_0 \colon x_1 \colon \cdots \colon x_n) \mapsto \left(\frac{x_0}{x_i}, \dots, \frac{x_{i-1}}{x_i}, \frac{x_{i+1}}{x_i}, \dots, \frac{x_n}{x_i}\right),$$

is a bijection. Fixing the index i, we often identify $\mathbb{A}^n$ as the subset $U_i \subset \mathbb{P}^n$ under this bijection.

To define projective algebraic sets, we instead consider *homogeneous polynomials*, i.e., polynomials $F \in \overline{\Bbbk}[X_0, \dots, X_n]$ such that $F(\lambda X_0, \dots, \lambda X_n) = \lambda^d F(X_0, \dots, X_n)$ for all $\lambda \in \overline{\Bbbk}$, where $d = \deg(F)$. This ensures that, if $F(P) = 0$ for some $P \in \mathbb{P}^n(\overline{\Bbbk})$, then every projective representative of that point is a zero of F as well.

Definition 2.1.11. Let I be a homogeneous ideal of $\overline{\Bbbk}[X_0, \dots, X_n]$. A *projective algebraic set* is a set

$$V(I) := \{P \in \mathbb{P}^n \colon F(P) = 0 \text{ for all homogeneous } F \in I\}.$$

For a projective algebraic set V, the *homogeneous ideal of* V is the ideal $I(V)$ of $\overline{\Bbbk}[X_0, \dots, X_n]$ generated by

$$\{F \in \overline{\Bbbk}[X_0, \dots, X_n] \colon F \text{ homogeneous and } F(P) = 0 \text{ for all } P \in V\}.$$

Remark 2.1.12 (Zariski Topology)**.** Affine and projective spaces are equipped with the structure of a topological space via the *Zariski topology.* The *closed* subsets of affine (resp. projective) n-space are affine (resp. projective) algebraic sets. Therefore, the Zariski topology on $\mathbb{A}^n$ or $\mathbb{P}^n$ induces a topology on any affine or projective algebraic set, respectively. For more details, we refer to [264, §3].

Definition 2.1.13. An *(irreducible) projective variety* is a projective algebraic set V such that the homogeneous ideal $I(V)$ is a prime ideal in $\overline{\mathbb{k}}[X_1, \ldots, X_n]$. Similarly to the affine case, we say that a projective algebraic variety V is defined over $\mathbb{k}$ if it can be generated by homogeneous polynomials $F \in \mathbb{k}[X_0, \ldots, X_n]$. Then, the set of $\mathbb{k}$-rational points of V is the set $V(\mathbb{k}) = V \cap \mathbb{P}^n(\mathbb{k})$.

Remark 2.1.14. As for the affine case (see Remark 2.1.4), for a projective variety V, when $I(V)$ is generated by a single homogeneous polynomial $F \in \mathbb{k}[X_1, \ldots, X_n]$, then $I(V)$ is a prime ideal if F is irreducible over $\mathbb{k}$.

Throughout this thesis, we want to work in both affine and projective space. To move from projective space to affine space, we have to make a *choice of affine patch*, which means we have to choose a bijection $U_i \longleftrightarrow \mathbb{A}^n$. We follow the usual convention and take U_n (i.e., $x_n \neq 0$) always. This is formalised in the following definition.

Definition 2.1.15. Let $F \in \mathbb{k}[X_0, \ldots, X_n]$ be a homogeneous polynomial. We define the *dehomogenization* F_H of F to be

$$F_H(X_1, \ldots, X_n) := F(X_1, \ldots, X_n, 1).$$

Conversely, given $f \in \mathbb{k}[X_1, \ldots, X_n]$ of degree d we define the *homogenization* f_H of f to be

$$f_H(X_0, \ldots, X_n) := X_n^d f(X_0/X_n, \ldots, X_{n-1}/X_n).$$

As a result, if V is a projective algebraic set with homogeneous ideal $I(V) \subset \overline{\mathbb{k}}[X_0, \ldots, X_n]$, we identify $V \cap \mathbb{A}^n$ with $\phi_n(V \cap U_n)$: it is an affine algebraic set with

$$I(V \cap \mathbb{A}^n) = \{F_H \colon F \in I(V)\}.$$

Namely, $I(V \cap \mathbb{A}^n)$ is the ideal generated by the dehomogenization of homogeneous polynomials generating $I(V)$. Similarly, the *projective closure* $\overline{V}$ of an affine algebraic set $V \subset \mathbb{A}^n$ is the projective algebraic set with

$$I(\overline{V}) = \{f_H \colon f \in I(V)\}.$$

We can now define many properties of projective varieties V in terms of its affine part $V \cap \mathbb{A}^n$. Firstly, we define the *dimension* of V as the dimension of $V \cap \mathbb{A}^n$. The *function field of* V, denoted $K(V)$, is the function field of $V \cap \mathbb{A}^n$.

Example 2.1.16. Consider the affine variety V over $\overline{\mathbb{k}}$ defined by the equation $y^2 = x^3 - 5x^2 + x$, with affine coordinates $(x, y) \in \mathbb{A}^2$. We have $\dim(V) = 1$. The variety V has a projective closure $\overline{V}$ given by the image embedding $V \hookrightarrow \mathbb{P}^2$ defined by $(x, y) \mapsto (x : y : 1)$. The projective variety $\overline{V}$ is generated by the equation

$$Y^2 Z = X^3 - 5X^2 Z + XZ^2,$$

and $\dim(\overline{V}) = 1$. The additional point $(0 : 1 : 0)$ on the projective curve is a distinguished point called the *point at infinity*. Such an affine variety V is called an affine curve, and $\overline{V}$ is a projective curve. These will be the focus of Section 2.1.3.

2.1.2 Maps between varieties

In this section, we briefly look at algebraic maps between varieties. Let us first define morphisms of affine varieties.

Definition 2.1.17. Let V_1 and $V_2 \subset \mathbb{A}^n$ be affine varieties. We say that a map $\phi\colon V_1 \to V_2$ is a *morphism of affine varieties* if $\phi = (\phi_1, \ldots, \phi_n)$, where $\phi_i \in \overline{\mathbb{k}}[X_1, \ldots, X_n]$ are polynomials, such that $\phi(P) := (\phi_1(P), \ldots, \phi_n(P)) \in V_2$ for a point $P \in V_1$. If $\phi_1, \ldots, \phi_n \in \mathbb{k}[X_1, \ldots, X_n]$, we say that ϕ is *defined over* $\mathbb{k}$.

Defining morphisms of projective varieties requires more care, and we first introduce rational maps.

Definition 2.1.18. Let V_1 and $V_2 \subset \mathbb{P}^n$ be projective varieties. A *rational map* from V_1 to V_2 is a map ϕ, usually denoted by $\phi\colon V_1 \dashrightarrow V_2$, of the form

$$\phi = (\phi_0\colon \ldots : \phi_n),$$

where $\phi_0, \ldots, \phi_n \in \overline{\mathbb{k}}(V_1)$ have the property that for every point $P \in V_1$ at which $\phi_0, \ldots, \phi_n$ are all defined, we have

$$\phi(P) = (\phi_0(P)\colon \ldots : \phi_n(P)) \in V_2.$$

We say that ϕ is defined over $\mathbb{k}$ if there exists $\lambda \in \overline{\mathbb{k}}^\times$ such that $\lambda\phi_0, \ldots, \lambda\phi_n \in \mathbb{k}(V_1)$. A *birational map* from V_1 to V_2 is a rational map $\phi\colon V_1 \dashrightarrow V_2$ such that there is a rational map $V_2 \dashrightarrow V_1$ inverse to ϕ.

A rational map $\phi = (\phi_0\colon \cdots \phi_n)\colon V_1 \dashrightarrow V_2$ is called *regular* at $P \in V_1$ if there exists a function $g \in \overline{\mathbb{k}}(V_1)$ such that $g\phi_0, \ldots, g\phi_n$ are all defined at P and at least one of $g\phi_0(P), \ldots, g\phi_n(P)$ are non-zero. Note that, for each P, we may need a different function g. Morphisms of projective varieties are rational maps that are regular at every point.

Definition 2.1.19. Let V_1 and $V_2 \subset \mathbb{P}^n$ be projective varieties. A *morphism* $\phi : V_1 \to V_2$ between V_1 and V_2 is a rational map that is regular at every point $P \in V_1$. The map ϕ is called an *isomorphism* if there exists a morphism $\psi\colon V_2 \to V_1$ such that $\phi \circ \psi$ and $\psi \circ \phi$ are identity maps on V_1 and V_2, respectively. We say that two varieties V_1, V_2 are *isomorphic over* $\mathbb{k}$ or $\mathbb{k}$-*isomorphic* if there exists an isomorphism defined over $\mathbb{k}$.

Example 2.1.20. For this example, we follow Silverman [292, Examples 2.3, 3.5]. Let $V \subset \mathbb{P}^2$ be the projective variety given by the equation

$$X^2 + Y^2 = Z^2,$$

defined over $\mathbb{k}$ with characteristic $\operatorname{char} \mathbb{k} \neq 2$. The morphism

$$\begin{aligned} \mathbb{P}^1 &\longrightarrow V, \\ (s : t) &\longmapsto (s^2 - t^2 : 2st : s^2 + t^2), \end{aligned}$$

has inverse

$$\begin{aligned} V &\longrightarrow \mathbb{P}^1, \\ (X : Y : Z) &\longmapsto (X + Z : Y). \end{aligned}$$

In this way, V and $\mathbb{P}^1$ are isomorphic as projective varieties.

2.1.3 Curves

We turn our attention to affine and projective curves. These will be the geometric object of most interest to us, and is in some sense the simplest type of variety. Indeed, an affine, resp. projective, curve C is an affine, resp. projective, variety of dimension 1.

As C is an algebraic variety, it has a corresponding ideal $I(C)$ which is generated by polynomials $f_1, \ldots, f_m \in \overline{\mathbb{k}}[X_1, \ldots, X_n]$ if affine, or homogeneous polynomials $F_0, \ldots, F_m \in \overline{\mathbb{k}}[X_0, \ldots, X_n]$ if projective. We say that C is generated by $f_1, \ldots, f_m$ or $F_0, \ldots, F_m$, respectively.

Recall that a point $P \in \mathbb{A}^n$ lies on the affine curve $C \subset \mathbb{A}^n$ if $f_i(P) = 0$ for all $i = 1, \ldots, m$. We denote the $\mathbb{k}$-rational points of C, as defined in Section 2.1.1.1, by $C(\mathbb{k})$. We similarly define the $\mathbb{k}$-rational points of a projective curve C following Section 2.1.1.2.

Definition 2.1.21. Let C be an affine curve generated by $f_1, \ldots, f_m \in \overline{\mathbb{k}}[X_1, \ldots, X_n]$. A point $P \in C$ is *non-singular* or *smooth* if the $m \times n$ matrix

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

has rank $n - 1$. If C is non-singular at every point P, we say that C is *non-singular*.

Remark 2.1.22. When C is generated by $f \in \overline{\mathbb{k}}[X_1, \ldots, X_n]$, Definition 2.1.21 reduces to C being non-singular if

$$\left(\frac{\partial f}{\partial X_1}(P), \ldots, \frac{\partial f}{\partial X_n}(P)\right) \neq (0, \ldots, 0).$$

When C is a projective curve we can identify C with the affine part $C \cap \mathbb{A}^n$. We then define C to be *non-singular* (or *smooth*) at P if $C \cap \mathbb{A}^n$ is non-singular (or smooth) at P.

Example 2.1.16 (continuing from p. 38)**.** The affine variety defined by $f = -y^2 + x^3 - 5x^2 + x \in \overline{\mathbb{k}}[x, y]$ is an irreducible affine curve C. Let us now explore properties of the curve C. It is smooth at a point $P = (x_P, y_P)$ with $f(x_P, y_P) = 0$ if and only if the partial derivatives $\left(\frac{\partial f}{\partial y}(P), \frac{\partial f}{\partial x}(P)\right) \neq (0, 0)$. As $\frac{\partial f}{\partial y} = -2y$, it is non-zero at P if and only if $y_P \neq 0$ and $\operatorname{char} \mathbb{k} \neq 2$. If $y = 0$, then consider the curve $x(x^2 - 5x + 1)$. The quadratic $x^2 - 5x + 1$ has non-zero discriminant $\Delta = 21 \in \mathbb{k}$ for $\operatorname{char} \mathbb{k} \neq 3, 7$, so in this case it does not have repeated roots in $\mathbb{k}$ and $\frac{\partial f}{\partial x}(P) \neq 0$. In conclusion, C is a non-singular irreducible affine curve for $\operatorname{char} \mathbb{k}$ coprime to 2, 3, and 7. In this case, the projective closure of C defined by $Y^2 Z = X^3 - 5X^2 Z + XZ^2 \subset \mathbb{P}^2$, is an *elliptic curve*, a special type of affine curve that we will define in Section 2.2. It has *distinguished point* $(0 : 1 : 0)$.

2.1.4 The Picard group

Let C be an irreducible non-singular projective curve defined over $\overline{\Bbbk}$. We now introduce the Picard group $\mathrm{Pic}^0(C)$ corresponding to C following Silverman [292, Chapter II], which allows us to associate a group to any projective curve C. This is important within the context of cryptography as this group structure is essential to instantiate cryptographic protocols. As an example, consider an *elliptic curve* $E/\Bbbk$ with elliptic curve group $E(\Bbbk)$ used to construct *elliptic-curve cryptography*. The Picard group allows us to generalise this construction to a larger family of curves C. We revisit this example in Section 2.2.

Definition 2.1.23. We define a *divisor* D to be a formal sum of points $P \in C$, i.e.,

$$D = \sum_{P \in C} n_P(P),$$

where $n_P \in \mathbb{Z}$ and there are only finitely many non-zero constants n_P. The *support* of D is defined to be the set of points $P \in C$ such that $n_P \neq 0$. We define the *degree* of D as $\deg(D) = \sum_{P \in C} n_P \in \mathbb{Z}$.

The set of all divisors $\mathrm{Div}(C)$ of a curve C, forms a commutative group with identity $\sum_{P \in C} 0(P)$, where the addition is defined pointwise as

$$\sum_{P \in C} n_P(P) + \sum_{P \in C} m_P(P) = \sum_{P \in C} (n_P + m_P)(P).$$

The set $\mathrm{Div}^0(C) := \{D \in \mathrm{Div}(C) \colon \deg(D) = 0\}$, of all divisors of degree 0 forms a subgroup of $\mathrm{Div}(C)$.

Another important type of divisor are those coming from rational functions f on the curve C. For a rational function $f \in \overline{\Bbbk}(C)^\times$, we can look at the formal sum

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

where $\mathrm{ord}_P(f)$ denotes the order of f at P. We have that $\mathrm{ord}_P(f) = a \in \mathbb{Z}$ where

$$a \begin{cases} < 0, & \text{if } f \text{ has a pole of order } -a \text{ at } P, \\ = 0, & \text{if } f \text{ is non-zero at } P, \\ > 0, & \text{if } f \text{ has a zero of order } a \text{ at } P. \end{cases}$$

As f only has a finite number of zeros and poles, only finitely many of the $\mathrm{ord}_P(f)$ are non-zero, and $\mathrm{div}(f)$ is a divisor. Such divisors are called *principal divisors*. The set of all principal divisors is noted $\mathrm{Prin}(C)$, which forms a subgroup of $\mathrm{Div}^0(C)$ as principal divisors have degree 0, see [292, Proposition II.3.1(b)].

Definition 2.1.24. The *Picard group* of a curve C is defined as the quotient group

$$\mathrm{Pic}^0(C) := \mathrm{Div}^0(C)/\,\mathrm{Prin}(C),$$

namely, $\mathrm{Pic}^0(C)$ is the group of divisors in $\mathrm{Div}^0(C)$ up to equivalence, where D_1 and D_2 are said to be equivalent if they differ by a principal divisor.

If C is defined over $\Bbbk$, then $\mathrm{Gal}(\overline{\Bbbk}/\Bbbk)$ acts on $\mathrm{Div}(C)$ and $\mathrm{Div}^0(C)$ by acting on the divisors as

$$\sigma(D) = \sum_{P \in C} n_P(\sigma(P)),$$

for $\sigma \in \mathrm{Gal}(\overline{\Bbbk}/\Bbbk)$. Then, we say that D is defined over $\Bbbk$ if $\sigma(D) = D$ for all $\sigma \in \mathrm{Gal}(\overline{\Bbbk}/\Bbbk)$. We let $\mathrm{Pic}^0_{\Bbbk}(C)$ be the subgroup of $\mathrm{Pic}^0(C)$ fixed by the Galois group $\mathrm{Gal}(\overline{\Bbbk}/\Bbbk)$.

Remark 2.1.25. If $D = n_1(P_1) + \cdots n_r(P_r)$ is defined over $\Bbbk$, it does not mean that each point $P_1, \ldots, P_r$ are defined over $\Bbbk$. We demonstrate this through an example. Consider the curve $C : y^2 = x^6 + 1$ of genus 2, defined over $\mathbb{F}_{11^2} = \mathbb{F}_{11}(i)$, where $i^2 + 1 = 0$. Consider the divisor $D = ((i, 0)) + ((-i, 0)) - 2(0_C) \in \mathrm{Div}^0(C)$. As each point in the support of D is defined over $\mathbb{F}_{11^2}$, it suffices to consider the Galois group $\mathrm{Gal}(\mathbb{F}_{11^2}/\mathbb{F}_{11})$, rather than $\mathrm{Gal}(\overline{\mathbb{F}}_{11}/\mathbb{F}_{11})$. We note that $\mathrm{Gal}(\mathbb{F}_{11^2}/\mathbb{F}_{11})$ is generated by the *Frobenius endomorphism* $\pi : \mathbb{F}_{11^2} \to \mathbb{F}_{11^2}$ mapping $x \mapsto x^{11}$. As

$$\pi(D) = (\pi(i, 0)) + (\pi(-i, 0)) - 2(\pi(0_C)) = ((-i, 0)) + ((i, 0)) - 2(0_C) = D,$$

we see that D is defined over $\mathbb{F}_{11}$. However, the points $(\pm i, 0) \in C(\mathbb{F}_{11^2})$ are not defined over $\mathbb{F}_{11}$.

Example 2.1.16 (continuing from p. 38)**.** Consider the line $L\colon y = ax + b$ and the affine curve C defined by $y^2 = x^3 - 5x^2 + x$ as before. To define divisors, we need to work with the corresponding projective curve $\overline{C}$ defined by $Y^2Z = X^3 - 5X^2Z + XZ^2$ with identity 0_C. By Bézout's theorem (see [168, §5.3]), a line L intersects C at exactly three points (counting multiplicities). The divisor $\mathrm{div}(L)$ tells us what these points of intersection are. Suppose first that L intersects C at 3 distinct points $P, Q, R \in C(\Bbbk)$, each with multiplicity 1. Then L also has a pole of order 3 at 0_C. Therefore, we have $\mathrm{div}(L) = (P) + (Q) + (R) - 3(0_C)$. If instead L intersects C at P with multiplicity 2, and at a point R with multiplicity 1, we have $\mathrm{div}(L) = 2(P) + (R) - 3(0_C)$. These lines are precisely those used in the group law $\oplus$ in $\mathrm{Pic}^0(\overline{C})$, which is given by point addition on an elliptic curve via the chord-and-tangent rule, as depicted in Figure 2.2 in Section 2.2. Indeed, in the first case we have $R = -(P \oplus Q)$, and in the second case $R = -[2]P$.

2.1.4.1 The genus of curves

The *genus* of a projective curve is an important invariant and arises as a consequence of the Riemann–Roch theorem. As before, let C be an irreducible non-singular projective curve defined over $\Bbbk$.

Let $D = \sum_{P \in C} n_P(P) \in \mathrm{Div}(C)$ be a divisor of C. We say that a divisor is *effective* if $n_P \geq 0$ for all $P \in C$. We denote this by $D \geq 0$. Then, if $D_1, D_2 \in \mathrm{Div}(C)$, we say that $D_1 \geq D_2$ if $D_1 - D_2 \geq 0$.

Using this notation, we can now describe zeros and poles of a function. For instance, if $\mathrm{div}(f) \geq 5(P)$ for $f \in \overline{\Bbbk}(C)^\times$, then we know that f has a zero of at least order 5 at P. Going further, for a divisor $D \in \mathrm{Div}(C)$ we define:

$$\mathcal{L}(D) := \{f \in \overline{\Bbbk}(C)^\times \mid \mathrm{div}(f) \geq -D\} \cup \{0\}.$$

$\mathcal{L}(D)$ is a finite dimensional $\overline{\Bbbk}$-vector space [298, Lemma 1.4.6, Proposition 1.4.9]. The Riemann–Roch theorem gives us a bound on the dimension of this vector space, denoted $\ell(D)$, for every divisor D.

Theorem 2.1.26 (Riemann–Roch)**.** *Let $C/\Bbbk$ be an irreducible non-singular curve over $\Bbbk$. Then, there exists an integer $g \geq 0$ such that for every divisor $D \in \mathrm{Div}(C)$ we have*

$$\ell(D) \geq \deg(D) - g + 1.$$

If $\deg(D) > 2g - 2$ then we have equality.

We are now ready to introduce the two examples of plane curves of most interest to us: elliptic curves (curves of genus 1) and hyperelliptic curves of genus 2.

2.2 Elliptic curves

We start with the genus-1 case: elliptic curves. For simplicity, we consider $\Bbbk$ such that the characteristic $\mathrm{char}\,\Bbbk \neq 2$ or 3. As we focus on cryptographic applications where the characteristic $\mathrm{char}\,\Bbbk$ is a cryptographic-sized prime, we lose nothing with this simplification. In this section, we follow Silverman [292].

To define the group associated to an elliptic curve via the construction of the Picard group in Section 2.1.4, we must look at genus-1 curves that are non-singular and irreducible. We illustrate the difference between singular and non-singular genus 1 curves in Figure 2.1, best illustrated by setting $\Bbbk = \mathbb{R}$.

This leads us to the following definition.

Definition 2.2.1. An *elliptic curve* E defined over field $\Bbbk$ is a non-singular, irreducible projective curve of genus 1 with a distinguished $\Bbbk$-rational point 0_E.

We consider two models of elliptic curves, corresponding to different shapes of the defining equation: short Weierstrass and Montgomery models. Weierstrass curves are the most classical family of elliptic curves, and every elliptic curve can be put in this form. Due to this, the Weierstrass model is often considered to be the canonical way to represent an elliptic curve. As $\mathrm{char}\,\Bbbk \neq 2, 3$, the elliptic curve E defined over $\Bbbk$ can be put in *short* Weierstrass form.

Definition 2.2.2 (Short Weierstrass Curve)**.** A *short Weierstrass curve* E defined over $\Bbbk$ is defined by the equation

$$E\colon Y^2Z = X^3 + aXZ^2 + bZ^3, \tag{2.1}$$

with $a, b \in \Bbbk$ such that the discriminant $\Delta = -16(4a^3 + 27b^2) \neq 0$. The distinguished point of E is $0_E = (0\colon 1\colon 0)$.

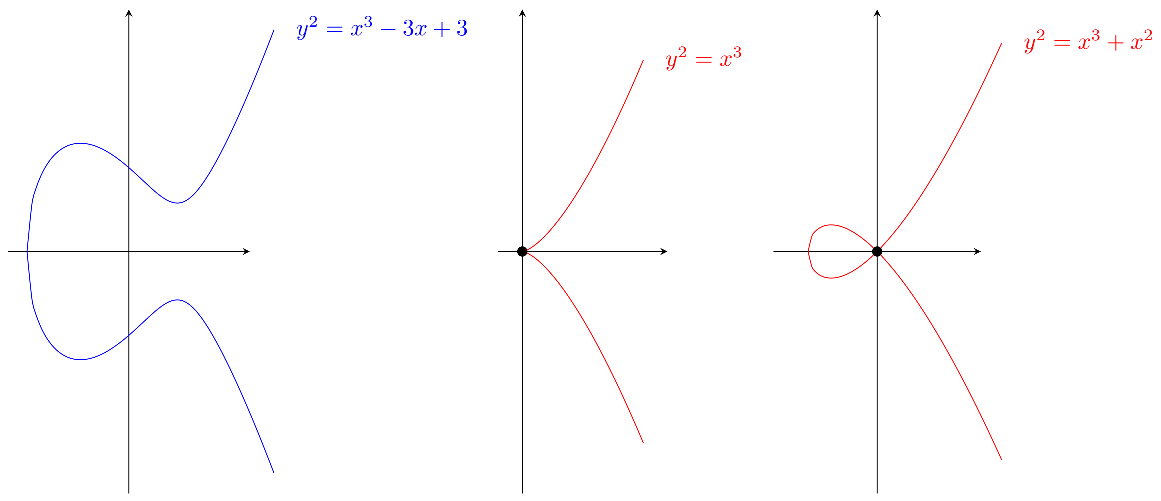


FIGURE 2.1: A depiction of singular and non-singular curves defined over $\mathbb{R}$ following [292, Figure 3.1-3.2]. The curve on the left (in blue) $y^2 = x^3 - 3x + 3$ is non-singular. The other two curves (in red) are singular. The curve $y^2 = x^3$ is a *cusp*, as it has one tangent direction at the singular point $(0,0)$, whereas $y^2 = x^3 + x^2$ has two tangent directions at $(0,0)$ and is called a *node*.

Montgomery [241] introduced another model of curve that is important in the context of cryptography due to its efficient arithmetic.

Definition 2.2.3 (Montgomery Curve)**.** A *Montgomery curve* E over $\Bbbk$ is defined by the equation

$$E\colon BY^2Z = X(X^2 + AXZ + Z^2) \tag{2.2}$$

with $A, B \in \Bbbk$ such that $B \neq 0$ and $A^2 \neq 4$. The distinguished point of E is given by $0_E = (0\colon 1\colon 0)$.

Recall that we use the map $(X, Y, Z) \mapsto (x, y) = (X/Z, Y/Z)$ to obtain the affine model of E. The equations defining the affine short Weierstrass and Montgomery models are:

$$E\colon y^2 = x^3 + ax + b \quad \text{and} \quad E\colon By^2 = x(x^2 + Ax + 1),$$

respectively. All points $P = (X_P\colon Y_P\colon Z_P) \in E(\Bbbk)$ with $P \neq 0_E$ can be mapped to an affine point $(x_P, y_P) := (X_P/Z_P, Y_P/Z_P)$. The *point at infinity* 0_E cannot be represented in these affine coordinates (as its Z-coordinate is zero).

To associate a group to an elliptic curve E, we turn to the Picard group. In this case, there exists a bijection of sets $\operatorname{Pic}^0(E) \to E$ [292, Proposition III.3.4]. In fact, $\operatorname{Pic}^0(E)$ is an abelian group and this bijection induces a group structure on E. As such, E is an abelian group with group law given by the addition of points. We often take the identity of the group law to be 0_E. Refer to, for example, Frey and Lange [164, p. 13.1.1] for explicit formulæ for the group law when E is a short Weierstrass curve. This group law has a geometric interpretation given by the chord-and-tangent rule. We depict this in Figure 2.2 taking $\Bbbk = \mathbb{R}$ for illustrative purposes.

We now give an extended example which will depict properties of elliptic curves and the group law on a specific Montgomery curve.

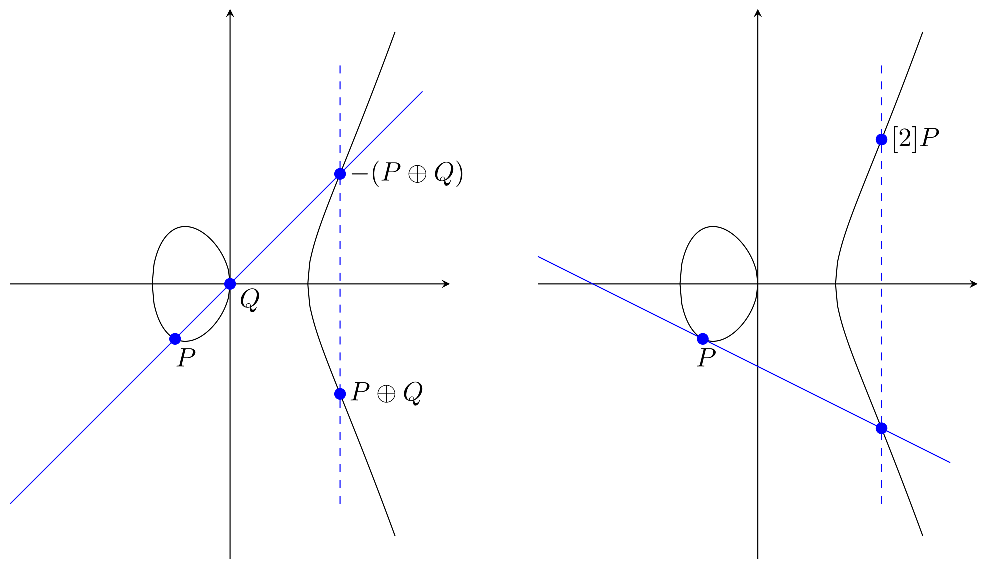


Figure 2.2: Chord-and-tangent rule for point addition (on the left) and point doubling (on the right) on elliptic curves. For illustrative purposes, we take $\Bbbk = \mathbb{R}$.

Example 2.1.16 (continuing from p. 38)**.** Recall $E : y^2 = x^3 - 5x^2 + x$ is an elliptic curve defined over $\Bbbk$ when $\operatorname{char} \Bbbk \neq 2, 3, 7$. In fact, we can now see that it is an elliptic curve in Montgomery form with distinguished point $0_E := (0 : 1 : 0)$. To the curve E, we can associate a group of points $E(\Bbbk)$ with group law given by the addition of points $\oplus$ and identity 0_E. Consider a point $P = (x_P : y_P : 1)$ in $E(\Bbbk)$. The inverse of P under $\oplus$ is a point P' such that $P \oplus P' = 0_E$. We find that $P' = (x_P : -y_P : 1)$. We denote the inverse of P by $-P$. We describe addition of points in affine coordinates. A discussion of how to obtain efficient projective formulæ is given in Section 2.2.1. For now, we remark that by using projective formulæ for the group law, we can avoid all field inversions. Let $P = (x_P, y_P)$ and $Q = (x_Q, y_Q) \neq \pm P$ in $E(\Bbbk)$. Then, $R = P \oplus Q$ is given by (x_R, y_R) where

$$\begin{aligned} x_R &:= \lambda^2 - (x_P + x_Q) + 5, \\ y_R &:= \lambda(x_P - x_R) - y_P, \end{aligned}$$

and $\lambda = (y_P - y_Q)/(x_P - x_Q)$. If $Q = P$, then $R = [2]P$ and

$$(x_{[2]P}, y_{[2]P}) := \left(\frac{(x_P^2 - 1)^2}{2x_P(x_P^2 - 5x_P + 1)}, y_P \cdot \frac{(x_P^2 - 1)(x_P^4 - 10x_P^3 + 6x_P^2 - 10x_P + 1)}{8x_P^2(x_P^2 - 5x_P + 1)} \right).$$

Otherwise, $Q = -P$ and $R = 0_E$. In this way, we see that the group law is given by rational functions. To give a concrete example we consider $\Bbbk = \mathbb{F}_{11}$. Taking $P = (2, 1)$, $Q = (5, 4)$ in $E(\mathbb{F}_{11})$ we have $\lambda = 1$ and $R = (1 - (2 + 5) + 5, (5 - 2) - 1) = (10, 2)$. Similarly, $[2]P = (5, 4)$.

The Montgomery form of an elliptic curve is special in the following sense.

Proposition 2.2.4 ([250])**.** *A Weierstrass elliptic curve $E : y^2 = x^3 + ax + b$ defined over $\Bbbk$ can be put in Montgomery form if and only if:*

- *The group order $\#E(\Bbbk)$ is divisible by* 4*,*
- $x^3 + ax + b$ *has at least one root* $\alpha \in \Bbbk$*, and*
- $3\alpha^2 + a$ *is a quadratic residue in* $\Bbbk$*.*

Although not every curve can be put into Montgomery form, it is a popular choice for cryptography as it is amenable to efficient arithmetic.

2.2.1 x-only arithmetic

Consider a Montgomery elliptic curve $E_{A,B} \subseteq \mathbb{P}^2$ defined over $\Bbbk$ with projective coordinates $(X : Y : Z)$. The quotient map $\mathbf{x}\colon E_{A,B} \mapsto E_{A,B}/\langle \pm 1 \rangle$ defined as $\mathbf{x}(P) = (X_P : 1)$ if $P = (X_P : Y_P : 1)$, and $\mathbf{x}(P) = (1 : 0)$ if $P = (0 : 1 : 0)$. As $E_{A,B}/\langle \pm 1 \rangle$ is isomorphic to $\mathbb{P}^1$ with most points of the form $(X_P : 1)$, we call it the *x-line* (or the *Kummer line*). The quotient map destroys the group law on $E_{A,B}$, and as such $\mathbb{P}^1$ does not inherit the group structure of $E_{A,B}$. However, we still recover a *pseudo-group law.* We can define a *pseudo-addition* on $\mathbb{P}^1$ by

$$(\mathbf{x}(P), \mathbf{x}(Q), \mathbf{x}(P \ominus Q)) \mapsto \mathbf{x}(P \oplus Q),$$

where we require the knowledge of $P \ominus Q$ to compute the addition $P \oplus Q$. We can also define *pseudo-doubling*

$$\mathbf{x}(P) \mapsto \mathbf{x}([2]P),$$

where no additional information is needed (compared to the standard doubling formulæ). Furthermore, we compute scalar-multiplication $\mathbf{x}(P) \mapsto \mathbf{x}([k]P)$, for $k \in \mathbb{Z}$ using, for example, the Montgomery ladder [241].

For cryptographic applications, the pseudo-operations on the x-line are sufficient. We remark that we can construct the x-line associated to any elliptic curve E, including those in Weierstrass form. However, using elliptic curves in Montgomery form gives us very efficient x-line arithmetic. We refer to Costello and Smith [106, §3] for more details on x-only arithmetic on Montgomery curves. Regarding isogeny computation, the explicit formulæ given by Costello and Hisil [102] is the state-of-the-art for evaluating isogenies (of low degree) on Montgomery curves. It is a variation of Vélu's formulæ working entirely on the x-line.

2.2.2 Isomorphism classes of elliptic curves

Rather than considering each individual curve, we study them up to $\overline{\Bbbk}$-isomorphism. Namely, we study equivalence classes of elliptic curves where E and E' defined over $\Bbbk$ are equivalent if and only if they are isomorphic over $\overline{\Bbbk}$. To obtain a representative for each equivalence class, we define the j-invariant.

Definition 2.2.5 (j-invariant)**.** Consider the short Weierstrass curve $E\colon Y^2Z = X^3 + aXZ^2 + bZ^3$ with $a, b \in \Bbbk$. The *j-invariant* is given by

$$j(E) = 1728 \frac{4a^3}{4a^3 + 27b^2}. \tag{2.3}$$

When E is a Montogomery curve, i.e., defined as $E\colon BY^2 = X(X^2 + AXZ + Z^2)$, we have

$$j(E) = \frac{256(A^2-3)^3}{A^2-4}. \tag{2.4}$$

The following proposition tells us that the $\overline{\Bbbk}$-isomorphism class of E is uniquely determined by the j-invariant, thus giving us a suitable label for the equivalence classes.

Proposition 2.2.6. *Two elliptic curves E_1, E_2 defined over $\Bbbk$ are isomorphic over $\overline{\Bbbk}$ if and only if $j(E_1) = j(E_2)$.*

Proof. See [292, Proposition III.1.4(b)]. □

Example 2.2.7. Consider elliptic curves $E_1\colon y^2 = x^2 + 16x + 71$ and $E_2\colon y^2 = x^2 + 7x + 67$ in short Weierstrass form defined over $\mathbb{F}_{103}$. Then $j(E_1) = j(E_2) = 13$ and so they are isomorphic (over $\overline{\mathbb{F}}_{103}$). Indeed, the isomorphism is given by

$$\begin{aligned}\phi\colon E_1 &\to E_2,\\ (x,y) &\mapsto (32x, 23y),\end{aligned}$$

with inverse

$$\begin{aligned}\phi^{-1}\colon E_2 &\to E_1,\\ (x,y) &\mapsto (29x, 9y).\end{aligned}$$

2.2.3 Quadratic twists

Let $\Bbbk$ be a field of characteristic $\operatorname{char}\Bbbk \neq 2$ and let E be an elliptic curve defined over $\Bbbk$ of the form

$$E\colon y^2 = x^3 + ax^2 + bx + c, \quad a, b, c \in \Bbbk.$$

Given $d \neq 0$ not a square in $\Bbbk$, we define the *quadratic twist by d* of E to be the curve E^d defined by $E^d\colon dy^2 = x^3 + ax^2 + bx + c$, or equivalently, $E^d\colon y^2 = x^3 + dax^2 + d^2bx + d^3c$. The curves E and E^d are not isomorphic over $\Bbbk$, but they are isomorphic over $\Bbbk(\sqrt{d})$ via

$$\rho_d : E \xrightarrow{\sim} E^d, \quad (x,y) \mapsto (x, y\sqrt{d}).$$

Example 2.2.8. Consider $E\colon y^2 = x^3 + 60x + 7$ defined over $\mathbb{F}_{103}$. The elliptic curve $E^d\colon y^2 = x^3 + 68x + 33$ is a quadratic twist of E by $d = 102$. Indeed, E and E^d are not isomorphic over $\mathbb{F}_{103}$, but they are isomorphic over $\mathbb{F}_{103^2} = \mathbb{F}_{103}(i)$ where $i^2 = -1$ via the isomomorphism

$$\phi\colon E \to E^d, \quad (x,y) \mapsto (-x, iy)$$

For elliptic curves defined over finite fields, the quadratic twist of an elliptic curve E is unique (up to $\mathbb{F}_q$-isomorphism), as shown by the following proposition. For example, the quadratic twist E^d of E from Example 2.2.8 above is unique up to $\mathbb{F}_{103}$-isomorphism.

Proposition 2.2.9. *Let E be an elliptic curve defined over $\mathbb{F}_q$, and $d, d' \neq 0$ be non-squares in $\mathbb{F}_q$. Then E^d is isomorphic over $\mathbb{F}_q$ to $E^{d'}$.*

Proof. As d, d' are quadratic non-residues in $\mathbb{F}_q$, $d'd^{-1}$ is a quadratic residue, and so there exists some $\alpha \in \mathbb{F}_q$ with $\alpha^2 = d'd^{-1}$. Then the map $(x, y) \mapsto (x, \alpha y)$ gives an isomorphism from E^d to $E^{d'}$ over $\mathbb{F}_q$. □

By Proposition 2.2.9, for an elliptic curve E defined over $\mathbb{F}_q$, we denote the (unique) quadratic twist by E^t.

2.3 Hyperelliptic curves of genus 2

We now introduce another example of plane curves: *hyperelliptic curves of genus* 2. A comprehensive overview of hyperelliptic curves in the context of cryptography can be found in [164, §4.4.2.b] or [170, Chapter 10].

Definition 2.3.1. A non-singular projective curve C defined over $\Bbbk$ of genus g is called a *hyperelliptic curve of genus g* if its function field $\Bbbk(C)$ is separable extension of degree 2 over the rational function field $\Bbbk(x)$, i.e., $[\Bbbk(C) \colon \Bbbk(x)] = 2$.

Following Frey and Lange [164], one can use the Riemann–Roch theorem to find an equation describing an affine part of C. For the purposes of this thesis, it will be sufficient to characterise hyperelliptic curves of genus g by their affine part.

Theorem 2.3.2. *The function field of a hyperelliptic curve of genus g defined over $\Bbbk$ with $\operatorname{char} \Bbbk \neq 2$ is the function field of an affine curve given by an equation*

$$C : y^2 = f(x), \tag{2.5}$$

where $f(x) \in \Bbbk[x]$ with $2g + 1 \leq \deg(f) \leq 2g + 2$.

Proof. See the proof of [164, Theorem 4.122]. □

If C has a $\Bbbk$-rational Weierstrass point, it is given by an affine curve of the form $C \colon y^2 = f(x)$ as in Equation (2.5) with $\deg(f) = 2g + 1$. The homogenization of any such curve has a single point at infinity, denoted ∞. If $\deg(f) = 2g+2$, then there are two points at infinity ∞^- and ∞^+.

The *Weierstrass points of C* are precisely the points $(w_i, 0)$ where w_i is one of the (at most $2g+2$) distinct roots of $f(x)$. When $\deg(f) = 2g+1$, we also consider the (single) point at infinity ∞ to be a Weierstrass point.

Remark 2.3.3. The definition of a hyperelliptic curve of genus g as a projective curve with affine part given by Equation (2.5) in Theorem 2.3.2 means that we can view an elliptic curve as a hyperelliptic curve of genus 1. From this point onwards, we refer to a hyperelliptic curve of genus g as given by Theorem 2.3.2, therefore encompassing the elliptic curve case.

For cryptographic applications, we often work with a particular model of hyperelliptic curve that yields more efficient arithmetic.

Definition 2.3.4. A hyperelliptic curve of genus 2 defined over $\Bbbk$ is in *Rosenhain form* if it is given by

$$C_{\lambda,\mu,\nu}\colon y^2 = x(x-1)(x-\lambda)(x-\mu)(x-\nu),$$

where $\lambda, \mu, \nu \in \Bbbk$.

For a genus-2 hyperelliptic curve $C\colon y^2 = f(x)$, if $f(x)$ has all its roots in $\Bbbk$, then C is $\Bbbk$-isomorphic to a curve $C_{\lambda,\mu,\nu}$ in Rosenhain form, with isomorphism given by, for example, mapping one root to 0, one root to 1, and one root to ∞, as described in [249, §2.1]. For a curve in Rosenhain form, the six Weierstrass points are $w_1 = \infty$, $w_2 = 0$, $w_3 = 1$, $w_4 = \lambda$, $w_5 = \mu$ and $w_6 = \nu$.

Unlike in the elliptic curve case, for general hyperelliptic curves of genus g, the points do not form a group as $\mathrm{Pic}^0(C) \ncong C$. Instead, the abelian group $\mathrm{Pic}^0(C)$ must be considered as a separate object. In this case, we identify the Picard group with the *Jacobian.*

Definition 2.3.5. Let C be a hyperelliptic curve. There exists a (unique) non-singular, irreducible projective variety $\mathcal{J}_C$ defined over $\Bbbk$ such that $\mathcal{J}_C(\Bbbk')$ is isomorphic to $\mathrm{Pic}^0_{\Bbbk'}(C)$ for all intermediate fields $\Bbbk \subseteq \Bbbk' \subseteq \overline{\Bbbk}$ [239, §1]. The variety $\mathcal{J}_C$, also denoted as $\mathrm{Jac}(C)$, is called the *Jacobian* of C.

Throughout this thesis, we work with both the algebraic properties of the Jacobian and the geometric properties of the underlying hyperelliptic curve. We remark that if C is of genus g, then $\mathcal{J}_C$ has dimension equal to g.

For a hyperelliptic curve C of genus g, each divisor class $D \in \mathcal{J}_C$ has a unique representative of the form $\sum_{i=1}^{r}(P_i) - r(0_C)$, with $r \leq g$, and where $P_i \neq 0_C$ for all $i = 1, \ldots, r$ and, writing $P_i = (x_i, y_i)$, we have $(x_i, y_i) \neq (x_j, -y_j)$ for all $i \neq j$. We call such a representative a *reduced* divisor. Mumford [245] introduced a convenient way to represent reduced divisors as D.

Definition 2.3.6. Let C be a hyperelliptic curve of genus 2 with Jacobian $\mathcal{J}_C$. Let $D = \sum_{i=1}^{r}(P_i) - r(0_C) \in \mathcal{J}_C$ be a reduced divisor with $P_i = (x_i, y_i)$. We say D is in *Mumford representation* if $D = (u(x), v(x))$ where $u(x)$ is a monic polynomial with $\deg(u(x)) \leq 2$ satisfying $u(x_i) = 0$, and $v(x)$ with $\deg(v(x)) < \deg(u(x))$ such that $v(x_i) = y_i$, for $1 \leq i \leq r$.

Given two reduced divisors D_1, D_2 in their Mumford representations, Cantor's algorithm [64] allows us to compute the reduced divisor D in Mumford representation such that D is equivalent to $D_1 + D_2$. This allows us to perform addition (and scalar multiplication) of divisor classes in $\mathcal{J}_C$ using the Mumford representation. We explore the group law on $\mathcal{J}_C$ through an example; for a generic exposition of the group law we refer to Lange [213, Chapter 3].

Example 2.3.7. Let $C : y^2 = x^5 + 36x^4 + 42x^3 + 47x^2 + 63x + 10$ be a hyperelliptic curve of genus 2 defined over $\mathbb{F}_{67}$ with point at infinity ∞. Let $\mathcal{J}_C$ be the Jacobian corresponding to C. Consider the points

$$P_1 = (14, 21),\ P_2 = (4, 41),\ P_3 = (46, 27),\ \text{and}\ P_4 = (5, 9),$$

lying on C. Let $D_1 = (P_1) + (P_2) - 2(\infty)$ and $D_2 = (P_3) + (P_4) - 2(\infty)$ be divisors in $\mathcal{J}_C$. We would like to compute $D_1 + D_2$. For $D = (P_1) + (P_2) + (P_3) + (P_4) - 4(\infty) \in \mathcal{J}_C$, there is a unique reduced divisor $\widetilde{D}$ equivalent to D of the form $\widetilde{D} = (Q_1) + (Q_2) - 2(\infty)$. Then, $\widetilde{D} = D_1 + D_2$. We find it in the following way.

We first compute the unique cubic function passing through the points P_1, P_2, P_3, P_4 to be $f : y = 54x^3 + 12x^2 + 50x + 12$. The function f intersects C at two more points, namely

$$P_1' = (28, 10) \text{ and } P_2' = (0, 12).$$

Therefore, we have $\mathrm{div}(f) = (P_1) + (P_2) + (P_3) + (P_4) + (P_1') + (P_2') - 6(\infty)$ and $(P_1) + (P_2) + (P_3) + (P_4) - 4(\infty)$ is equivalent to $-(P_1') - (P_2') + 2(\infty)$. Constructing vertical lines

$$\ell_1 : x = 28 \quad \text{and} \quad \ell_2 : x = 0,$$

which each intersect C at another point $Q_1 = (28, 57)$ and $Q_2 = (0, 55)$, respectively. As such, $\mathrm{div}(\ell_i) = (P_i') + (Q_i) - 2(\infty)$ for $i = 1, 2$. Concluding, we have that $-(P_1') - (P_2') + 2(\infty)$ is equivalent to

$$-(P_1') - (P_2') + 2(\infty) + \mathrm{div}(\ell_1) + \mathrm{div}(\ell_2) = (Q_1) + (Q_2) - 2(\infty) = \widetilde{D}.$$

Although not evident by this example, we remark the points Q_1, and Q_2 need not be defined over $\mathbb{F}_{67}$, but the divisor $\widetilde{D}$ will be. For example, Q_1 and Q_2 can be Galois conjugates as explained in Remark 2.1.25.

What about Mumford coordinates? Consider first the divisor D_1. Let $u_1(x) = (x-14)(x-4) = x^2 + 49x + 56$, and $v_1(x) = 65x + 49$, so that $v_1(14) = 21$ and $v_1(4) = 41$. Therefore, in Mumford coordinates, D_1 given by

$$D_1 = (u_1(x), v_1(x)) = (x^2 + 49x + 56, 65x + 49).$$

Similarly, in Mumford coordinates, D_2 is

$$D_2 = (u_2(x), v_2(x)) = (x^2 + 16x + 29, 56x + 64).$$

Using Cantor's algorithm,[1] we compute $\widetilde{D} = D_1 + D_2$ to be $(x(x + 39), 24x + 55)$. We verify that these are the Mumford coordinates of the divisor $(Q_1) + (Q_2) - 2(\infty)$ where $Q_1 = (28, 57)$ and $Q_2 = (0, 55)$.

2.4 Abelian varieties

Abelian varieties have both an algebraic structure (as an abelian group), and a geometric structure (as a smooth projective curve). In previous examples, we have already seen two examples of *abelian varieties*: elliptic curves and Jacobians of hyperelliptic curves of genus g. We begin with a description of abelian varieties of dimension g, before specialising to dimension 1 and 2. For the discussion on abelian varieties of dimension g we refer to Mumford [244], Lang [212] and Milne [237].

Definition 2.4.1. An *abelian variety* A is a projective algebraic variety A defined over $\overline{\Bbbk}$ that is also an algebraic group, i.e., A has a group law $\oplus : A \times A \to A$ such that $\oplus$ and its inverse $\ominus$

[1]Cantor's algorithm is implemented in `MAGMA`, for example.

are morphisms of varieties (as defined in Definition 2.1.19). We denote by 0_A the identity of the abelian variety under this group law.

As shown by Mumford [244, pg. 44], A is an abelian group. We denote by $A/\Bbbk$ an abelian variety whose defining equations have coefficients in $\Bbbk$, and $A(\Bbbk)$ denotes the group of points defined over $\Bbbk$ in A.

Remark 2.4.2. When clear by context, we often denote the group law morphism $\oplus$ and its inverse $\ominus$ simply as $+$ and $-$, respectively.

Example 2.1.16 (continuing from p. 38)**.** Let E be the projective elliptic curve $E\colon Y^2Z = X^3 - 5X^2Z + XZ^2$ with projective coordinates $(X : Y : Z) \in \mathbb{P}^2$. Letting $I = (-Y^2Z - X^3 + 5X^2Z - XZ^2)$ be an ideal in $\Bbbk[X, Y, Z]$, we have that $E = V(I)$. For the elliptic curve E, we saw that the group law is commutative and given by rational functions at each point on E and with identity $0_E = (0 : 1 : 0)$. In fact, an elliptic curve is an abelian variety of *dimension* 1.

We only deal with abelian varieties of dimension 1 and 2. Abelian varieties of dimension 1 are precisely elliptic curves. An example of abelian varieties of dimension 2 are Jacobians of hyperelliptic curves of genus 2. Abelian varieties arising from elliptic curves and hyperelliptic curves will be the main focus of this thesis.

Remark 2.4.3. In general, we can construct an abelian variety of dimension g by constructing the Jacobian of any hyperelliptic curve of genus g.

2.4.1 Principally polarised abelian varieties

We briefly discuss principal polarisations and dual abelian varieties. We do not aim for this discussion to be complete or to give precise definitions, but rather to give some intuition on what these terms mean within the context of cryptography. We emphasize that for the purposes of this work, it is sufficient to understand the specific examples of principally polarised abelian varieties detailed in Sections 2.2 and 2.5.

In Section 2.1.4 we defined the Picard group associated to a curve C. We can also define divisors on an abelian variety and the Picard group $\mathrm{Pic}^0(A)$. We define the *dual abelian variety* $\widehat{A}$ to be $\mathrm{Pic}^0(A)$. Returning to our primary example of an abelian variety, every elliptic curve E is isomorphic to its dual $\widehat{E}$. However, this fails for higher dimensional abelian varieties, and the concept of a dual therefore becomes more interesting. For our purposes, however, we would like to work only with abelian varieties A that are isomorphic to their dual $\widehat{A}$. For this reason, we omit a precise definition of the dual $\widehat{A}$; such a definition would require introducing heavy machinery from algebraic geometry, and it will not be necessary for the purposes of this thesis. For interested readers, a general definition of the dual abelian variety is given by Milne [237, §9].

Definition 2.4.4. We say that (A, λ) is a *principally polarised (p.p.) abelian variety* if A is an abelian variety and $\lambda\colon A \to \widehat{A}$ is an isomorphism.

By working with principally polarised (p.p.) abelian varieties (A, λ), we ensure that $A \cong \widehat{A}$.

In cryptography, we often take for granted that we have equations defining our elliptic curves in projective space. The existence of coordinates and equations for an elliptic curve E arise from the fact that an elliptic curve has a canonical principal polarisation (i.e., a canonical isomorphism $E \cong \widehat{E}$). We remark that this is not always the case: some abelian varieties A have no principal polarisation [183], and others have more than one (so we need to be careful with our choice). Reassuringly, the abelian varieties constructed from hyperelliptic curves in Section 2.3 also have a canonical principal polarisation [239, Summary 6.11]. The following example of a polarisation will also be important.

Example 2.4.5. Consider the (Cartesian) product $E_1 \times E_2$ of (p.p.) elliptic curves (E_1, λ_1) and (E_2, λ_2) defined over $\Bbbk$. Points on $E_1 \times E_2$ are given as tuples (P_1, P_2) where $P_1 \in E_1(\Bbbk)$ and $P_2 \in E_2(\Bbbk)$, and

$$(P_1, P_2) \oplus (Q_1, Q_2) := (P_1 \oplus Q_1, P_2 \oplus Q_2),$$

where $(P_1, P_2), (Q_1, Q_2) \in (E_1 \times E_2)(\Bbbk)$. The polarisation on $E_1 \times E_2$ is given by the product polarisation $\lambda_1 \times \lambda_2$, which gives an isomorphism $\lambda_1 \times \lambda_2 : E_1 \times E_2 \xrightarrow{\sim} \widehat{E}_1 \times \widehat{E}_2$, and is thus a principal polarisation. We remark, however, that this is not the only polarisation.

When the choice of principal polarisation λ is clear (for example, when there is a canonical choice), we denote the p.p. abelian variety (A, λ) by A.

2.5 Abelian surfaces

Every abelian variety of dimension 1 is an elliptic curve, and so every dimension-1 abelian variety has a (canonical) principal polarisation. In dimension 2, we work with *p.p. abelian surfaces*. Over an algebraically closed field $\overline{\Bbbk}$, there are only two types of p.p. abelian surfaces (see [256] or [177, Theorem 3.1]):

1. The Jacobian $\mathcal{J}_C$ of a hyperelliptic curve C of genus 2; or
2. A product of two elliptic curves $E_1 \times E_2$ with the product polarisation described in Example 2.4.5.

Over a field $\Bbbk$, another type of p.p. abelian surface appears: the Weil restriction of an elliptic curve, but this will not be relevant to the work in this thesis.

2.5.1 Kummer surfaces

Kummer surfaces are the natural analogue of the x-line defined in Section 2.2.1 in dimension 2. As they have more efficient arithmetic than the corresponding Jacobian, they are preferred for constructive cryptographic applications. In Chapter 9, we use Kummer surfaces to construct efficient two-dimensional isogenies.

The Kummer surface $\mathcal{K}$ of the Jacobian $\mathcal{J}_C$ of a genus-2 curve C is the quotient $\mathcal{J}_C/\{\pm 1\}$. The quotient map $\pi\colon \mathcal{J}_C \to \mathcal{K}$ sends a divisor $D_P \in \mathcal{J}_C$ to a point P on the Kummer surface $\mathcal{K}$. Unlike the x-line which involves working with one coordinate in $\mathbb{P}^1$, the Kummer surface is a more complicated object. Geometrically, it has a quartic model in $\mathbb{P}^3$ with sixteen point singularities, called *nodes*. This means that $\mathcal{K}$ has coordinates $(X_1 : X_2 : X_3 : X_4) \in \mathbb{P}^3$. The nodes are the images in $\mathcal{K}$ of the 2-torsion points of $\mathcal{J}_C$, since these are precisely the points fixed by -1.

While $\mathcal{K}$ inherits scalar multiplication from $\mathcal{J}_C$, it loses the group law: the points $P = \pi(\pm D_P)$ and $Q = \pi(\pm D_Q)$ on $\mathcal{K}$ do not uniquely determine $P + Q = \pi(\pm D_{P+Q})$, unless at least one of P and Q is the image of a point in $\mathcal{J}_C[2]$. However, the operation $\{P, Q\} \mapsto \{P + Q, P - Q\}$ is well-defined, so we have a *pseudo-addition* operation $(P, Q, P - Q) \mapsto P + Q$.

Distinct models of hyperelliptic curves of genus 2 give rise to different Kummer surface models. This is analogous to what we observe in the dimension-1 case: whereas a general elliptic curve in (short) Weierstrass form admits x-only arithmetic, this x-only arithmetic is much more efficient for curves in Montgomery form. Similarly, the *general* Kummer surface introduced by Cassels and Flynn [66] can be constructed for any hyperelliptic curve C, yet high-speed cryptography requires the use of the more efficient Kummer surfaces constructed from curves in Rosenhain form. These *fast* Kummer surfaces will be introduced in Chapter 9.

2.5.2 Isomorphism classes of abelian surfaces

As with elliptic curves, we work with abelian surfaces A defined over $\Bbbk$ up to isomorphism. First consider the case where $\mathcal{J}_C$ is the Jacobian of a genus-2 curve $C/\Bbbk$. Recall from Definition 2.1.10 that $\mathbb{P}(1, 2, 3, 5)$ is weighted projective space with weights $1, 2, 3, 5$. We associate to C its Igusa–Clebsch invariants

$$(I_2(C) \colon I_4(C) \colon I_6(C) \colon I_{10}(C)) \in \mathbb{P}(1, 2, 3, 5).$$

Explicitly, if C is a genus-2 curve given by a Weierstrass equation

$$C \colon y^2 = (x - a_0) \cdots (x - a_5),$$

where $a_0, \dots, a_5 \in \overline{\Bbbk}$, we define:

$$\begin{aligned}
I_2(C) &:= \sum_{15} (01)^2(23)^2(45)^2, \quad I_4(C) := \sum_{10} (01)^2(12)^2(20)^2(34)^2(45)^2(53)^2, \\
I_6(C) &:= \sum_{60} (01)^2(12)^2(20)^2(34)^2(45)^2(53)^2(03)^2(14)^2(25)^2, \quad \text{and} \\
I_{10}(C) &:= \prod_{i<j} (a_i - a_j)^2,
\end{aligned}$$

where, for any permutation $\sigma \in S_6$, we let (ij) denote the difference $(a_{\sigma(i)} - a_{\sigma(j)})$. Here the sums are taken over all distinct expressions in the a_i as σ ranges over S_6; the subscripts denote the number of expressions in each sum. Moreover, the isomorphism class of C over $\overline{\Bbbk}$ is uniquely determined by its Igusa–Clebsch invariants (see [188, 233]).

In the case where $E_1 \times E_2$ is the product of elliptic curves E_1, E_2, the isomorphism class of A as a p.p. abelian surface is uniquely determined by the set of j-invariants $\{j(E_1), j(E_2)\}$. Here, we consider the set of invariants, rather than an ordered tuple, as the order of the elliptic curves does not matter: $E_1 \times E_2$ is isomorphic to $E_2 \times E_1$.

Example 2.5.1. Let $\mathbb{F}_{11^2} = \mathbb{F}_{11}(i)$ where $i^2 = -1$. Consider the hyperelliptic curves

$$\begin{aligned}&C_1\colon y^2 = (7i+3)x^6 + x^5 + (2i+7)x^4 + (5i+10)x^3 + 3x^2 + (3i+5)x + 4i + 8, \text{ and}\\ &C_2\colon y^2 = x^5 + x^3 + 9x\end{aligned}$$

We can write the equation defining C_1 as $y^2 = (x-a_0)\cdots(x-a_5)$ where

$$a_0 = 8i+2,\ a_1 = 9i+8,\ a_2 = 7i+10,\ a_3 = i+7,\ a_4 = i+9,\ a_5 = 2i+7.$$

Then, for example,

$$\begin{aligned}I_{10}(C_1) &= \prod_{i<j}(a_i - a_j)^2\\ &= (a_0-a_5)^2\cdot(a_0-a_5)^2\cdot(a_2-a_5)^2\cdot(a_3-a_5)^2\cdot(a_4-a_5)^2\cdot\\ &\quad (a_0-a_4)^2\cdot(a_1-a_4)^2\cdot(a_2-a_4)^2\cdot(a_3-a_4)^2\cdot(a_0-a_3)^2\cdot\\ &\quad (a_1-a_3)^2\cdot(a_2-a_3)^2\cdot(a_0-a_2)^2\cdot(a_1-a_2)^2\cdot(a_0-a_1)^2\\ &= 5.\end{aligned}$$

We can similarly compute $I_2(C_1), I_4(C_1), I_6(C_1)$ using the $a_0,\dots,a_5 \in \mathbb{F}_{11^2}$; we omit it from this example as the formulæ are very long. We find that the Igusa–Clebsch invariants of C_1 are $(4:1:6:5) \in \mathbb{P}(1,2,3,5)(\mathbb{F}_{11^2})$ and of C_2 are $(5i+1 : 2i+4 : 2 : 7i+3) \in \mathbb{P}(1,2,3,5)(\mathbb{F}_{11^2})$. As the invariants lie in weighted projective space, we need to normalise them to see if they are equal. A common choice of normalisation is

$$\left(\frac{I_4(C)}{I_2(C)^2}, \frac{I_2(C)I_4(C)}{I_6(C)}, \frac{I_4(C)I_6(C)}{I_{10}(C)}\right) \in \mathbb{A}^3.$$

This give us normalised invariants $(9,8,10)$ for both C_1 and C_2, and therefore the corresponding Jacobians $\mathcal{J}_1$ and $\mathcal{J}_2$, respectively, are isomorphic over $\overline{\mathbb{F}}_{11}$.

2.6 Isogenies between abelian varieties

Now that we have introduced our objects of interest, we are ready to discuss morphisms between them. As the group structure of abelian varieties is crucial for all cryptographic applications, we restrict to maps that preserve this group structure.

Definition 2.6.1 (Isogeny of abelian varieties)**.** Let A_1 and A_2 be abelian varieties defined over $\Bbbk$. A morphism $\varphi : A_1 \to A_2$ (of abelian varieties) is an *isogeny of abelian varieties* if it is surjective on $\Bbbk$-points and its kernel has finitely many points over $\overline{\Bbbk}$. We say A_1 and A_2 are *isogenous* is there exists an isogeny $\varphi : A_1 \to A_2$.

As an isogeny is a morphism, it is given by rational functions. We say the isogeny $\varphi : A_1 \to A_2$ is defined over $\Bbbk$ if it can be defined as a rational function whose coefficients lie in $\Bbbk$.

Remark 2.6.2. The finite kernel ensures that A_1 and A_2 are of the same dimension g.

An isogeny of abelian varieties $\varphi : A_1 \to A_2$ is a morphism of abelian varieties, and therefore commutes with the group law morphism $\oplus$: $\varphi(P \oplus Q) = \varphi(P) \oplus \varphi(Q)$ for all $P, Q \in A_1(\overline{\Bbbk})$. As such, φ is a *group homomorphism* between the groups $A_1(\Bbbk)$ and $A_2(\Bbbk)$, for any field $\Bbbk$ over which φ is defined. As φ is a group homomorphism, we have

$$\varphi(0_{A_1}) = \varphi(P \oplus -P) = \varphi(P) \oplus \varphi(-P) = \varphi(P) \oplus -\varphi(P) = 0_{A_2}.$$

In practice, this means we can easily recover the identity on the image from the image of 0_{A_1}.

Remark 2.6.3. Consider the morphism of curves $\varphi\colon C_1 \to C_2$. Then, φ is either constant or surjective (see, for example, [292, Theorem 2.3]). Therefore, if $\varphi\colon E_1 \to E_2$ is an isogeny of elliptic curves, the surjectivity of φ implies that it is non-constant. This means we exclude uninteresting morphisms such as the zero map.

Isogenies are said to be either separable or inseparable. We give the definition of these terms below for completeness, but we highlight that we are (mostly) only interested in the former type.

Definition 2.6.4. Let $\varphi : A \to A'$ be an isogeny of abelian varieties. Then, φ induces an embedding $\varphi^* : f \mapsto f \circ \varphi$ of the function field $\Bbbk(A')$ in $\Bbbk(A)$. We say φ is *separable*, *inseparable*, or *purely inseparable* if the finite field extension $\Bbbk(A)/\varphi^*(\Bbbk(A'))$ has the corresponding property.

This field extension also gives us information about the *degree* of our isogeny. The degree will be an important property of our isogenies as it indicates the cost of computing it.

Definition 2.6.5. The *degree* of φ is the degree of the extension $\Bbbk(A)/\varphi^*(\Bbbk(A'))$. We denote the degree of φ by $\deg(\varphi)$.

Remark 2.6.6. Let us briefly consider a field $\Bbbk$ with $\operatorname{char} \Bbbk = p > 0$. As a finite extension of fields in characteristic $p > 0$ is separable whenever its degree is not divisible by p, we see that an isogeny defined over $\Bbbk$ is always separable if its degree is not divisible by $\operatorname{char} \Bbbk$.

The *composition* of isogenies is as we expect: composing two isogenies $\varphi : A \to A'$ and $\psi : A' \to A''$ will give the isogeny $\psi \circ \varphi : A \to A''$. Then, we have that $\deg(\psi \circ \varphi) = \deg(\psi) \cdot \deg(\varphi)$; the degree of isogenies is multiplicative.

Every isogeny can be factorised as $\varphi = \varphi_S \circ \varphi_I$ where $\varphi_I : A \to B$ is an inseparable isogeny and $\varphi_S : B \to A'$ is a separable isogeny [146, Corollary V.5.8]. Furthermore, this isogeny is unique up to isomomorphism in the sense that, if $\varphi = \varphi'_S \circ \varphi'_I$ is another factorisation then there is an isomorphism $\alpha : B \xrightarrow{\sim} B$ such that $\varphi'_S = \varphi_S \circ \alpha$ and $\varphi'_I = \alpha \circ \varphi_I$.

Consider the kernel of $\varphi\colon A_1 \to A_2$ on $\overline{\Bbbk}$-points, namely $\ker(\varphi) \subseteq A_1(\overline{\Bbbk})$. In general, we have $\deg_s(\varphi) = \#\ker(\varphi)$, where $\deg_s(\varphi)$ is the degree of the separable part of the isogeny φ (see, for example, [244, pg. 63]). In particular, when φ is separable, the degree of the isogeny is given precisely by the size of this kernel: $\deg(\varphi) = \#\ker(\varphi)$. From this we can see that the finite subgroup $\ker(\varphi) \subseteq A(\overline{\Bbbk})$ carries a lot of information about the isogeny when it is separable. In fact, we have a correspondence between finite subgroups of the abelian variety A and isogenies with domain A.

Theorem 2.6.7. *Let $A/\Bbbk$ be an abelian variety. Then there is a one-to-one correspondence between:*

- *Finite subgroups $G \subseteq A(\bar{\Bbbk})$,*
- *Separable $\bar{\Bbbk}$-isogenies $\varphi\colon A \to A'$, where two isogenies $\varphi_1\colon A \to A_1$, $\varphi_2\colon A \to A_2$ are considered equal if there is an isomorphism $\iota\colon A_1 \to A_2$ such that $\varphi_2 = \iota \circ \varphi_1$, which has $G = \ker(\varphi)$ and $A' = A/G$.*

Proof. A proof can be found in [244, pg. 72, Theorem 4] for abelian varieties of arbitrary dimension g. For the case $g = 1$, a more down-to-earth proof is given by combining [292, Proposition III.4.12] and [292, Exercise III.3.13]. $\square$

This theorem is crucial for cryptographic applications; it tells us that separable isogenies are completely determined by their kernels. Therefore, rather than representing our isogeny as a rational function, we can instead store a description of a finite group, which is often much more compact.

For many applications, we need to compute separable isogenies of large, composite degrees. Computing these isogenies directly is computationally expensive. The following theorem shows that we can instead factor an isogeny φ of degree $\deg \varphi = \prod_{i=1}^{k} \ell_i$ into a composition of isogenies $\varphi = \varphi_k \circ \cdots \circ \varphi_1$, where $\deg(\varphi_i) = \ell_i$. In this sense, the degree of an isogeny is multiplicative.

Theorem 2.6.8. *Let A_1, A_2, A_3 be abelian varieties of dimension g over $\Bbbk$ and $\varphi : A_1 \to A_2$, $\varphi' : A_1 \to A_3$ be separable isogenies over $\Bbbk$. Suppose that $\ker(\varphi) \subseteq \ker(\varphi')$. Then, there is a unique isogeny $\psi : A_2 \to A_3$ defined over $\Bbbk$ such that $\varphi' = \psi \circ \varphi$.*

Proof. A proof can be found in [244, pg. 111, Theorem 1(A)] for general dimension g. For $g = 1$, a simpler proof is given by Silverman [292, Corollary III.4.11]. $\square$

By Theorem 2.6.8, if we have an isogeny φ of *smooth* degree, i.e., the prime factors of $\deg(\varphi)$ are small, then we can efficiently compute φ by decomposing it into its prime degree parts.

Unless stated otherwise, we will assume our isogenies are separable from this point onwards.

2.6.1 Torsion Points

For $N \in \mathbb{Z}$, let $[N]_A$ be the multiplication-by-N morphism on A, defined as follows. For $N > 0$, we have

$$\begin{aligned} [N]_A : A &\to A, \\ P &\mapsto \underbrace{P + \cdots + P}_{N}. \end{aligned}$$

If $N < 0$ we set $[N](P) = [-N](-P)$, otherwise if $N = 0$, we define $[0]P = 0_A$. We define the *order* of a point $P \in A$ to be the least positive integer k such that $[k]P = 0_A$.

Definition 2.6.9. We define the *N-torsion subgroup* of $A(\Bbbk)$, denoted $A(\Bbbk)[N]$ or simply $A[N]$ if the field of definition is clear, to be the kernel of $[N]_A$ on $\Bbbk$-points. Equivalently, for every $P \in A[N]$ we have $[N]P = 0_A$, so P has order dividing N.

The morphism $[N]_A$ is an isogeny of degree N^{2g}, and it is a separable isogeny if and only if that characteristic $\operatorname{char}\Bbbk$ does not divide N [244, pg. 63]. In this case, $\#\ker[N]_A = \#A(\overline{\Bbbk})[N] = N^{2g}$. In fact, the structure of the N-torsion point group is:

$$A(\overline{\Bbbk})[N] \cong (\mathbb{Z}/N\mathbb{Z})^{2g}. \tag{2.6}$$

See, for example, Mumford [244, pg. 64].

Example 2.6.10. Let E be an elliptic curve defined over $\Bbbk$ with prime characteristic $\operatorname{char}\Bbbk = p$. Suppose $N \in \mathbb{N}$ is not divisible by p. By Remark 2.6.6, the multiplication-by-N map $[N]_E : E \to E$ is separable of degree N^2. Therefore, as $E(\overline{\Bbbk})[N]$ contains exactly the points $P \in E(\overline{\Bbbk})$ such that $[N]_E P = 0_E$, we have $\#E(\overline{\Bbbk})[N] = \#\ker([N]_E) = N^2$. Similarly, for every integer d dividing N, we have $\#E(\overline{\Bbbk})[d] = d^2$. As $E(\overline{\Bbbk})[N]$ is a finite abelian group, it can be written as the product of cyclic groups, and so $E(\overline{\Bbbk})[N] \cong (\mathbb{Z}/N\mathbb{Z})^2$.

2.7 Pairings

We take a short detour from our discussion on isogenies of abelian varieties to introduce *pairings*. Pairings were first used in cryptography as a cryptanalytic tool in elliptic curve cryptography to reduce the complexity of the discrete logarithm problem on some *weak* curves [165, 229]. The use of pairings to build cryptosystems was introduced, most notably, by Joux [196], who constructed a one round protocol for tripartite Diffie–Hellman protocol. Following this, there was an explosion of research constructing new primitives with pairings, such as, identity-based encryption [40], short signatures [42], group signatures [38, 43], and commitment schemes [198]. More recently, pairings have found a new application in accelerating isogeny-based cryptoschemes (for example, [265]). The purpose of this section is to define cryptographic pairings with the lens of isogeny-based cryptography. In particular, we do not go into detail on *how* the pairing maps are computed, but rather focus on their most useful properties. General references for this section are [144, 171, 292]. A brilliant introduction to pairings for beginners is given by Costello [101].

A pairing is a special type of bilinear map on a (additive) abelian group G_1 taking values in some other (multiplicative) abelian group G_T:

$$(\cdot,\cdot)\colon G_1 \times G_1 \to G_T.$$

The bilinearity property means that, for $P, Q, R \in G_1$ we have

$$(P+Q, R) = (P,R)\cdot(Q,R), \text{ and } (P, Q+R) = (P,Q)\cdot(P,R).$$

This bilinearity property of pairings is what makes it so useful for cryptography. Indeed, the group structure of G_1 gives us a way to *add* points $P, Q \in G_1$ using the group law, whereas the bilinearity of the pairing map allows us to perform multiplication in the group G_T. Being able to exploit both this addition and multiplication has proven invaluable for the construction of many advanced cryptosystems.

Remark 2.7.1. We use this notation for elements of the group G_1 as eventually this will be a subgroup of $A(\overline{\mathbb{k}})$, where A is a p.p. abelian variety defined over $\mathbb{k}$.

To encompass a wider range of useful pairings, we relax the condition that the two arguments in the bilinear map come from the same group G_1, and arrive at the following definition of a pairing.

Definition 2.7.2. Let G_1 and G_T be groups of prime order N, and let G_2 be a group where each element has order dividing N.[2] We write G_1, G_2 as additive groups with identity $0_{G_1}, 0_{G_2}$ (respectively), and G_T as a multiplicative group with identity 1_{G_T}. A *cryptographic pairing* e is an efficiently computable function

$$e : G_1 \times G_2 \to G_T,$$

satisfying the following properties:

1. Non-degeneracy: $e(P_1, P_2) = 1_{G_T}$ for all $P_2 \in G_2$ if and only if $P_1 = 0_{G_1}$, and similarly $e(P_1, P_2) = 1_{G_T}$ for all $P_1 \in G_1$ if and only if $P_2 = 0_{G_2}$.

2. Bilinearity: For all $P_1, Q_1 \in G_1$ and $P_2, Q_2 \in G_2$, we have $e(P_1+Q_1, P_2) = e(P_1, P_2)e(Q_1, P_2)$ and $e(P_1, P_2 + Q_2) = e(P_1, P_2)e(P_1, Q_2)$.

Remark 2.7.3. When G_1 and G_2 are cyclic, the bilinearity condition collapses to: for all $P_1 \in G_1$ and $P_2 \in G_2$, we have $e(aP_1, bP_2) = e(P_1, P_2)^{ab}$ for any $a, b \in \mathbb{Z}$.

Remark 2.7.4. The property in Definition 2.7.2 that makes the pairing *cryptographic* is the fact that it is efficiently computable. Otherwise, it would not be useful for practical applications.

We introduce the two main instantiations of pairings for use in cryptography: the Weil and the Tate pairing. The first applications in cryptography used the Weil pairing. However, the Tate pairing and its variants are more commonly used today as they have proven to be more efficient. We detail their construction in detail for Jacobians of hyperelliptic curves of genus g, and defer to other literature for their instantiation when working with an abelian variety.

2.7.1 The Weil pairing

Following Milne [237, §16], we define the N-Weil pairing for any abelian variety. Let μ_N be (multiplicative) group of the N-th roots of unity. For abelian varieties defined over $\overline{\mathbb{k}}$ and $N \geq 2$ coprime to char $\mathbb{k}$, we have a non-degenerate bilinear map

$$A(\overline{\mathbb{k}})[N] \times \widehat{A}(\overline{\mathbb{k}})[N] \to \mu_N.$$

When (A, λ) is principally polarised, the isomorphism $\lambda : A \xrightarrow{\sim} \widehat{A}$ induces the N-Weil pairing

$$e_N : A(\overline{\mathbb{k}})[N] \times A(\overline{\mathbb{k}})[N] \to \mu_N.$$

We detail the construction of the N-Weil pairing in the case where A is the Jacobian $\mathcal{J}_C$ of a hyperelliptic curve C of genus g. For a divisor class $P \in \mathcal{J}_C(\overline{\mathbb{k}})[N]$ of order N, we fix a divisor D_P

[2] G_2 need not be cyclic.

which represents this class. Let $f_{N,P} \in \overline{\Bbbk}(C)$ be a function with divisor $\operatorname{div}(f_{N,P}) = ND_P$. The evaluation of $f_{N,P}$ at a divisor $D = \sum_{Q \in C} n_Q(Q)$ is given by

$$f_{N,P}(D) := \prod_{Q \in C} f_{N,P}(Q)^{n_Q}.$$

The N-Weil pairing

$$e_N \colon \mathcal{J}_C(\overline{\Bbbk})[N] \times \mathcal{J}_C(\overline{\Bbbk})[N] \to \mu_N$$

is given by $e_N(P, Q) := f_{N,P}(D_Q)/f_{N,Q}(D_P)$, when D_Q has disjoint support to D_P.

Remark 2.7.5. When $g = 1$, we obtain a pairing $E(\overline{\Bbbk})[N] \times E(\overline{\Bbbk})[N] \to \mu_N$. In this case, we can take $D_P = (P) - (0_E)$ for $P \in E(\overline{\Bbbk})[N]$. The functions $f_{N,P}$ are called *Miller functions* and can be computed using Miller's algorithm [235] in $O(\log(N))$. An extended discussion on how to compute these functions for $g \geq 1$ can be found in [172].

2.7.2 The Tate pairing

Tate defined a pairing for arbitrary abelian varieties (defined over local fields). An overview of Tate's definition can be found in [144]. Lichtenbaum [219] then showed how this pairing can be computed efficiently for the Jacobian of hyperelliptic curves. Frey and Rück [166] focused on the case of elliptic curves defined over finite fields, framing this pairing within the context of cryptography. Due to its history, the Tate pairing we present here is also referred to in the literature as the Tate–Lichtenbaum pairing or the Frey–Rück pairing. For our description of the Tate pairing, we restrict to a finite field $\Bbbk = \mathbb{F}_q$ of characteristic p.

Let C be a hyperelliptic curve of genus g defined over $\mathbb{F}_q$ with Jacobian $\mathcal{J}_C$. Let $N \neq p$ be a prime dividing $\#\mathcal{J}_C(\mathbb{F}_q)$. Let $P, Q \in J_C(\mathbb{F}_{q^k})$ be such that D_P and D_Q have disjoint support. Let $k > 1$ be the *embedding degree*, defined to be the smallest positive integer such that $N \mid q^k - 1$, i.e., the smallest positive integer such that $\mu_N \subset \mathbb{F}_{p^k}$. The *Tate pairing*

$$T_N \colon \mathcal{J}_C(\mathbb{F}_{q^k})[N] \times \mathcal{J}_C(\mathbb{F}_{q^k})/[N]\mathcal{J}_C(\mathbb{F}_{q^k}) \to \mathbb{F}_{q^k}^{\times}/(\mathbb{F}_{q^k}^{\times})^N$$

is defined as $T_N(P, Q + [N]\mathcal{J}_C(\mathbb{F}_{q^k})) := f_{N,P}(D_Q)(\mathbb{F}_{q^k}^{\times})^N$. Frey and Rück [166] showed that the Tate pairing can be computed in $O(\log(N))$ operations in $\mathbb{F}_{q^k}$.

The output of the Tate pairing lies in an equivalence class. If this pairing is to be useful in practice, different parties must be able to compute the same exact value, rather than the same value under the above notion of equivalence. Therefore, we now adapt the Tate pairing so that its output produces unique values.

When $\mathcal{J}_C(\mathbb{F}_{q^k})$ does not contain any point of order N^2, the Tate pairing can be given by a map

$$\mathcal{J}_C(\mathbb{F}_{q^k})[N] \times \mathcal{J}_C(\mathbb{F}_{q^k})[N] \to \mathbb{F}_{q^k}^{\times}/(\mathbb{F}_{q^k}^{\times})^N.$$

Furthermore, we observe that $\mathbb{F}_{q^k}^{\times}/(\mathbb{F}_{q^k}^{\times})^N$ is isomorphic to the subgroup of N-th roots of unity $\mu_N \subseteq \mathbb{F}_{q^k}^{\times}$ via the map $a(\mathbb{F}_{q^k}^{\times})^N \mapsto a^{(q^k-1)/N}$. Using these two observations, we define a new pairing induced by the Tate pairing, called the *reduced Tate pairing*:

$$t_N \colon \mathcal{J}_C(\mathbb{F}_{q^k})[N] \times \mathcal{J}_C(\mathbb{F}_{q^k})[N] \to \mu_N \subseteq \mathbb{F}_{q^k}^{\times},$$
$$(P,Q) \mapsto T_N(P,Q)^{(q^k-1)/N} = f_{N,P}(D_Q)^{(q^k-1)/N}.$$

2.8 Isogenies of principally polarised abelian varieties

As we work exclusively with principally polarised abelian varieties, we want to restrict to isogenies that respect this polarisation. We make this more precise in the following definition.

Definition 2.8.1. Let (A,λ) and (A',λ') be p.p. abelian varieties. An isogeny $\varphi : A \to A'$ is said to be and *isogeny of principally polarised abelian varieties* if there exists an integer $m \geq 1$ such that $\widehat{\varphi} \circ \lambda' \circ \varphi = [m]\lambda$. We also call such an isogeny a *polarised isogeny.*

Using the Weil pairing, we define maximal isotropic subgroups, which will correspond to kernels of isogenies of p.p. abelian varieties [237, Proposition 16.8].

Definition 2.8.2. Let A be an abelian variety defined over $\overline{\Bbbk}$. We say that a subgroup $G \subseteq A(\overline{\Bbbk})[N]$ is *isotropic* (with respect to the N-Weil pairing) if $e_N(P,Q) = 1$ for all $P,Q \in G$. We say G is *maximal isotropic* if moreover there is no isotropic subgroup G' with $G \subsetneq G' \subseteq A(\overline{\Bbbk})[N]$.

As we can factor an isogeny φ into its prime degree components (see Theorem 2.6.8), we restrict to prime $N \in \mathbb{N}$. Every maximal isotropic subgroup $G \subset A(\overline{\Bbbk})[N]$ is of the form $G \cong (\mathbb{Z}/N\mathbb{Z})^g$, and is therefore generated by g points $P_1,\dots,P_g$ such that $e_N(P_i,P_j) = 1$ for all $i,j = 1,\dots,g$ [159, Proposition 2].

Let A_1 be a p.p. abelian variety defined over $\overline{\Bbbk}$ and let $G \subset A_1(\overline{\Bbbk})[N]$ be a maximal isotropic subgroup. By Theorem 2.6.7, the subgroup $G = \langle P_1,\dots,P_g\rangle$ corresponds to an isogeny $\varphi : A_1 \to A_2 := A_1/G$ with kernel G. As the kernel of φ is maximal isotropic subgroup, φ is an isogeny of p.p. abelian varieties. In fact, every isogeny of p.p. abelian varieties arises from such a maximal isotropic subgroup.

Proposition 2.8.3. *Let (A,λ) be a principally polarised abelian variety of dimension g defined over $\overline{\Bbbk}$. Let G be a finite, proper subgroup of $A(\overline{\Bbbk})$. There exists a p.p. abelian variety A' of dimension-g defined over $\overline{\Bbbk}$, and an isogeny $\varphi : A \to A'$ (of degree coprime to $\operatorname{char}\Bbbk$) with $\ker(\varphi) = G$, if and only if $G \subseteq A(\overline{\Bbbk})[N]$ is a maximal isotropic subgroup for some $N \in \mathbb{N}$.*

Proof. We follow Flynn and Ti [159, Proposition 1]. By Theorem 2.6.7, $\varphi : A \to A/G$ is an isogeny of abelian varieties, so it suffices to consider the polarisation. Letting $\psi = [\deg\varphi] \circ \lambda$ be a polarisation on A so that $\ker\varphi \subseteq \ker\psi$, we can construct a polarisation λ' on A/G following Milne [237, Proposition 16.8, Remark 16.9], where $\deg(\lambda') = 1$. Thus, $(A/G, \lambda')$ is a principally polarised abelian variety. Conversely, suppose $\varphi : A \to A'$ is an isogeny of p.p. abelian varieties. By [237, Proposition 16.8], $\ker(\varphi)$ is a maximal isotropic subgroup of $A(\overline{\Bbbk})[N]$ for some N coprime to $\operatorname{char}\Bbbk$. □

Example 2.8.4. Let $\mathbb{F}_{131^2} = \mathbb{F}_{101}(i)$ where $i^2 = -1$. Consider the hyperelliptic curve

$$C_1 : y^2 = x(x-1)(x-(49i+104))(x-91)(x-(45i+52))$$

of genus 2 (in Rosenhain form) defined over $\mathbb{F}_{131^2}$ with corresponding Jacobian $\mathcal{J}_1$. Let

$$\begin{aligned} D_P &= \left(x^2 + 83x + 77i + 20,\, (6i+73)x + 35i + 126\right), \text{ and} \\ D_Q &= \left(x^2 + (6i+73)x + 31i + 62,\, 83x + 64i + 72\right), \end{aligned}$$

be divisors in $\mathcal{J}_1$ in Mumford coordinates. We have that D_P and $D_Q \in \mathcal{J}_1(\mathbb{F}_{131^2})[3]$ with $e_3(D_P, D_Q) = 1$. Therefore, the subgroup $G = \langle D_P, D_Q\rangle \subset \mathcal{J}_1(\mathbb{F}_{101^2})[3]$ is maximal isotropic (with respect to the 3-Weil pairing). It generates the isogeny $\varphi\colon \mathcal{J}_1 \to \mathcal{J}_2$ where $\mathcal{J}_2$ is the Jacobian of the genus-2 hyperelliptic curve

$$C_2\colon x(x-1)(x-(89i+24))(x-(119i+130))(x-(130i+41)).$$

As we are working with p.p. abelian varieties, this type of isogeny will naturally be of most interest to us, and so we give it a special name that highlights the structure of the kernel.

Definition 2.8.5. Let A be a p.p. abelian variety of dimension g defined over $\overline{\mathbb{k}}$. We call the (separable) isogeny $\varphi : A \to A/G$, where $G \subset A(\overline{\mathbb{k}})[N]$ is a maximal isotropic subgroup, an $(N,\dots,N)$-*isogeny*, and we say that G is an $(N,\dots,N)$-*subgroup*.

Proposition 2.8.6. *Let $\varphi : A_1 \to A_2$ be an $(N,\dots,N)$-isogeny between p.p. abelian varieties A_1, A_2 defined over $\overline{\mathbb{k}}$. There exists a unique dual isogeny (up to isomorphism), denoted $\widehat{\varphi} : A_2 \to A_1$, such that*

$$\widehat{\varphi}\circ\varphi = [\deg(\varphi)]_{A_1}, \text{ and } \varphi\circ\widehat{\varphi} = [\deg(\varphi)]_{A_2}.$$

Proof. We show this for $g = 1$ following Silverman [292, III.6.1(a)]. For $g > 1$, see Mumford [244, Theorem 1, pg. 143]. Let $\varphi\colon A_1 \to A_2$ be an N-isogeny between p.p. abelian varieties A_1, A_2 of dimension 1. Since $\ker(\varphi) = \mathbb{Z}/N\mathbb{Z}$, we have that $\ker(\varphi) \subset A_1(\overline{\mathbb{k}})[N] = \ker([N])$. By Theorem 2.6.8, there is an isogeny ψ with $[N]_{A_1} = \psi\circ\varphi$. To show uniqueness, suppose that $\psi' : A_2 \to A_1$ is another such isogeny. Then

$$(\psi - \psi')\circ\varphi = [N]_{A_1} - [N]_{A_1} = [0].$$

As φ is non-constant, $\psi - \psi'$ must be constant and so $\psi = \psi'$. We denote the isogeny ψ by $\widehat{\varphi}$ and call it the dual isogeny. Finally, we have

$$(\varphi\circ\widehat{\varphi})\circ\varphi = \varphi\circ[N]_{E_1} = [N]_{E_2}\circ\varphi,$$

and so $\varphi\circ\widehat{\varphi} = [N]_{E_2}$, as φ is non-constant. □

Remark 2.8.7. It can also be shown that when $\varphi : E_1 \to E_2$ is an inseparable isogeny between elliptic curves E_1, E_2, there exists a unique dual isogeny $\widehat{\varphi}$. See, for example, [292, Theorem III.6.1(a)].

2.8.1 Isogenies between elliptic curves

Let E be an elliptic curve defined over $\overline{\mathbb{k}}$. Fix a positive integer N not divisible by char $\mathbb{k}$. We specialise the discussion in Section 2.8 to dimension $g = 1$ in order to define N-isogenies. The kernel of such an isogeny is an N-subgroup, namely a cyclic subgroup of $E(\overline{\Bbbk})$ generated by a point $P \in E(\overline{\Bbbk})[N]$. In this way, N-isogenies are examples of *cyclic* isogenies.

Remark 2.8.8. Every point $P \in E[N]$ gives a (not necessarily distinct) isotropic subgroup of $E[N]$ as every point $Q \in \langle P \rangle$ is of the form $[k]P$ for some $1 \leq k \leq N$ and $e_N(P, [k]P) = e_N(P, P)^k = 1$ by the alternating property of the N-Weil pairing [292, Proposition III.8.1(b)].

For an N-subgroup $G = \langle P \rangle$ where $P \in E[N]$, there is an isogeny $\varphi : E \longrightarrow E/G$ with kernel G and image unique up to isomorphism. On input of the point P, computing the N-isogeny with kernel G has complexity $O(N)$ using Vélu's formulas [314]. For large N, it is preferable to compute φ using $\sqrt{\text{élu}}$ formulas [27] with an asymptotic complexity of $\widetilde{O}(\sqrt{N})$. Using these algorithms, the isogeny will be defined over the same field $\Bbbk$ as $P \in E(\Bbbk)[N]$. A discussion on how to compute isogenies when P is defined over an extension field $\Bbbk'/\Bbbk$ but the subgroup $\langle P \rangle \subseteq E(\Bbbk)[N]$ is given in Chapter 11.

We have shown that, given a point $P \in E[N]$, we can compute the isogenies corresponding to the kernel $\langle P \rangle$. However, it is natural to question how many distinct isogenies of degree N have domain E. This corresponds to the number of distinct cyclic subgroups of $E[N] \cong \mathbb{Z}/N\mathbb{Z} \times \mathbb{Z}/N\mathbb{Z}$ (see Section 2.6.1), and thus grows linearly with the degree. More explicitly, if $N = \prod_{i=1}^{n} \ell_i^{e_i}$, there are $D_{N,1}$ N-subgroups where

$$D_{N,1} := \prod_{i=1}^{n} (\ell_i + 1)\ell_i^{e_i - 1}. \tag{2.7}$$

For prime N, the number of outgoing N-isogenies from E is $N + 1$.

Example 2.8.9. Consider the elliptic curve $E_\alpha \colon y^2 = x(x - \alpha)(x - 1/\alpha)$ in Montgomery form defined over $\mathbb{F}_{31^2}$ where $\alpha = 11i + 16$. The point $(\alpha, 0) \in E_\alpha(\mathbb{F}_{31^2})$ has order 2, and therefore generates the kernel of a 2-isogeny $\varphi \colon E_\alpha \to E_{\alpha'}$ where $E_{\alpha'} \colon y^2 = x(x - \alpha')(x - 1/\alpha')$ has $\alpha' = 14i + 2$.

2.8.1.1 Modular Polynomials

The classical modular polynomial $\Phi_N(X, Y) \in \mathbb{Z}[X, Y]$ of level N parameterises pairs of elliptic curves with cyclic N-isogeny in terms of their j-invariants. More precisely, Φ_N is the unique (up to scaling) bivariate polynomial such that $\Phi_N(j_1, j_2) = 0$ if and only if j_1 and j_2 are the j-invariants of N-isogenous elliptic curves. Sutherland's database [301] contains $\Phi_N(X, Y)$ for all $N \leq 300$ and for all primes $N \leq 1000$, computed using techniques from various joint works of theirs [59, 302].

The polynomial Φ_N is symmetric in X and Y, i.e., $\Phi_N(X, Y) = \Phi_N(Y, X)$. If $j(E) \neq 0, 1728$ (more details on these exceptions in Section 4.2), there are exactly $D_{N,1}$ N-isogenies emanating from E, where $D_{N,1}$ is as defined in Equation (2.7). Therefore, $\Phi_N(X, j(E))$ must be of degree $D_{N,1}$ in X so that there are $D_{N,1}$ roots corresponding to the elliptic curves isogenous to E. By symmetry, the degree of $\Phi_N(X, Y)$ in Y must also be $D_{N,1}$.

The difficulty in computing $\Phi_N(X, Y)$ is in the size of its coefficients, rather than the number of them. As discussed by Sutherland [302], storing $\Phi_N(X, Y)$ requires $O(N^3 \log N)$ bits, which corresponds to several gigabytes for $N \approx 1000$ and many terabytes for $N \approx 10^4$. Fortunately, the modular polynomials already contained in Sutherland's database are more than sufficient for the purposes of this thesis. In Chapter 8, we use modular polynomials in the context of cryptanalysis of the supersingular isogeny problem over a fixed finite field $\mathbb{F}_{p^2}$, meaning we can reduce all the large coefficients modulo p as a precomputation. Indeed, $\Phi_N(X,Y) \in \mathbb{Z}[X,Y]$ can be preprocessed into

$$\Phi_{N,p}(X,Y) \in \mathbb{F}_p[X,Y],$$

where we note the additional subscript, defined by reducing all coefficients of $\Phi_N(X,Y)$ modulo p. By the symmetry of $\Phi_N(X,Y)$, this means we must store around $D_{N,1}^2/2$ coefficients in $\mathbb{F}_p$, requiring only $O(N^2 \log p)$ bits.

Example 2.8.9 (continuing from p. 62)**.** The classical modular polynomial of level 2 modulo 31 is given by

$$\Phi_{2,31}(X,Y) = X^3 + 30X^2Y^2 + 6X^2 + 5XY + 12X + Y^3 + 6Y^2 + 12Y + 8.$$

As expected, it is of degree 3 in X and Y. Let E_α be as before with $j(E_\alpha) = 8i + 17$. We have

$$\Phi_2(X, j(E_\alpha)) = X^3 + (7i + 29)X^2 + (9i + 4)X + (30i + 18),$$

which has roots $j_1 = 19$, $j_2 = 8i + 27$ and $j_2 = 16i + 18$ in $\mathbb{F}_{103^2}$. These j-invariants correspond to the three 2-isogenous neighbours of E_α. For example, with $E_{\alpha'}$ as before we have $j(E_{\alpha'}) = 16i + 18$.

2.8.1.2 The endomorphism ring

An important object, particularly in the context of isogeny-based cryptography, is the endomorphism ring of an elliptic curve.

Definition 2.8.10 (Endomorphism)**.** An isogeny $\varphi : E \to E$ is called an *endomorphism.*

The set of endomorphisms of E together with the zero map $[0]_E$ form a ring $\mathrm{End}(E)$ under the operations of pointwise addition and composition. The invertible elements of $\mathrm{End}(E)$ form the *automorphism group* of E, which is denoted by $\mathrm{Aut}(E)$. In this way, an *automorphism* is an isomorphism $\varphi : E \to E$.

Example 2.8.11. The multiplication-by-N morphism $[N]_E$ on E, for $N \in \mathbb{Z}$, is an example of an endomorphism. Therefore, we have that $\mathbb{Z} \subseteq \mathrm{End}(E)$. In fact, it forms a subring of $\mathrm{End}(E)$.

In Section 3.1, we look closer at endomorphism rings and show their connection with quaternion algebras.

Example 2.8.12. Another important example of an endomorphism arises when we consider an elliptic curve defined over a finite field $\mathbb{F}_q$ of characteristic p. The q-power Frobenius endomorphism π on an elliptic curve $E \subseteq \mathbb{P}^2$ is defined as

$$\pi : E \to E, \quad (X : Y : Z) \mapsto (X^q : Y^q : Z^q).$$

For an elliptic curve E defined over $\mathbb{F}_q$ we have $\mathbb{Z}[\pi] \subseteq \mathrm{End}(E)$.

2.8.2 Isogenies between abelian surfaces

We look now at p.p. abelian surfaces defined over $\overline{\Bbbk}$ and the (N,N)-isogenies between them. As before, we fix $N \in \mathbb{N}$ coprime to $\operatorname{char} \Bbbk$.

The (N,N)-subgroups of $A_1[N]$ are generated by two points $P, Q \in A_1[N]$ with $e_N(P,Q) = 1$. For every (N,N)-subgroup $G \subset A_1[N]$ there is an isogeny $\varphi : A_1 \longrightarrow A_2 := A_1/G$, with kernel $G = \langle P, Q\rangle$ and image unique up to isomorphism (as a polarised abelian variety). The number of (N,N)-subgroups $D_{N,2}$ is given by Castryck and Decru [69] (see also Costello and Smith [107, Lemma 2] and Flynn and Ti [159, Proposition 3(2)] when N is a prime not equal to $\operatorname{char} \Bbbk$, respectively prime power):

$$D_{N,2} := N^3 \prod_{\substack{\text{primes} \\ \ell \mid N}} \frac{1}{\ell^3}(\ell+1)(\ell^2+1). \tag{2.8}$$

When N is prime $D_{N,2} = (N^2+1)(N+1)$, and the number of isogenies emanating from A_1 grows as $O(N^3)$.

Example 2.8.13. Let $\mathbb{F}_{83^2} = \mathbb{F}_{83}(i)$ where $i^2 = -1$. Consider the following genus-2 hyperelliptic curve:

$$C_1 : y^2 = (13i+36)x^6+(4i+7)x^5+(75i+44)x^4+(69i+18)x^3+(31i+60)x^2+(5i+18)x+62i+74,$$

with corresponding Jacobian $\mathcal{J}_1$. There exists a $(2,2)$-isogeny $\varphi\colon \mathcal{J}_1 \to \mathcal{J}_2$ where $\mathcal{J}_2$ is the Jacobian of hyperelliptic curve

$$C_2\colon y^2 = (38i+10)x^6+(76i+32)x^5+(61i+53)x^4+(74i+66)x^3+(39i+24)x^2+(69i+18)x+60i+1.$$

There exists another $(2,2)$-isogeny from $\mathcal{J}_1$, namely $\psi\colon \mathcal{J}_1 \to E_1 \times E_2$ where

$$\begin{aligned} E_1\colon y^2 &= x^3 + (73i+45)x^2 + (34i+79)x + (69i+77), \text{ and} \\ E_2\colon y^2 &= x^3 + (27i+59)x^2 + (38i+34)x + (66i+47) \end{aligned}$$

are elliptic curves.

The example above highlights that there are different types of (N,N)-isogenies. In fact, we can consider three types of (N,N)-isogenies depending on whether A_1 or A_2 are Jacobians or products of elliptic curves:

1. An (N,N)-isogeny between Jacobians: $\varphi : \mathcal{J}_1 \to \mathcal{J}_2$;
2. A *splitting* (N,N)-isogeny: $\varphi : \mathcal{J}_1 \to E_1 \times E_2$;
3. A *gluing* (N,N)-isogeny: $\varphi : E_1 \times E_2 \to \mathcal{J}_2$.

We will primarily be considering the first two types of isogenies. Developing a method to efficiently detect whether a Jacobian is (N,N)-isogenous to a product of elliptic curves will be a large focus of Chapter 8.

2.9 Elliptic curves over finite fields

Up until now, we have looked at abelian varieties defined over a perfect field $\Bbbk$. In cryptography, however, we usually work with finite fields $\mathbb{F}_q$ with prime characteristic p. This section is dedicated to exploring special properties of elliptic curves over finite fields that we will use.

Given two elliptic curves E_1 and E_2 defined over $\mathbb{F}_q$, the first question one may ask is:

Are these elliptic curves isogenous?

A landmark result by Tate [306] gives us a way to answer this question.

Theorem 2.9.1 (Tate's Isogeny Theorem)**.** *Elliptic curves E_1 and E_2 defined over $\mathbb{F}_q$ are isogenous over $\mathbb{F}_q$ if and only if $\#E_1(\mathbb{F}_q) = \#E_2(\mathbb{F}_q)$.*

Therefore, to answer this question we need to find the order of the elliptic curve group $E(\mathbb{F}_q)$. For an elliptic curve $E/\mathbb{F}_q$, the group $E(\mathbb{F}_q)$ is finite, and Hasse's theorem gives us a bound on the size of $E(\mathbb{F}_q)$.

Theorem 2.9.2 (Hasse's Theorem)**.** *Let E be an elliptic curve defined over finite field $\mathbb{F}_q$ with characteristic p. We have that*

$$| \, \#E(\mathbb{F}_q) - q - 1 \, | \leq 2\sqrt{q}. \tag{2.9}$$

Proof. This is [292, Theorem 1.1]. We give the proof here for completeness. For $P \in E(\overline{\mathbb{F}}_p)$, we have that $P \in E(\mathbb{F}_q)$ if and only if $\pi_q(P) = P$, where π_q is the q-power Frobenius morphism on E. Therefore, $E(\mathbb{F}_q) = \ker(1 - \pi_q)$. As $1 - \pi_q$ is separable (see [292, Corollary 5.5]), we have that $\#E(\mathbb{F}_q) = \deg(1 - \pi_q)$. Using a variant of the Cauchy–Schwarz inequality (see [292, Lemma 1.2]) and the fact that $\deg \pi_q = q$, we get the statement of the theorem. $\square$

Remark 2.9.3. Let $E/\mathbb{F}_q$ be an elliptic curve. The quantity $a = q + 1 - \#E(\mathbb{F}_q)$ is called the *trace of Frobenius.*[3]

We now have efficient point counting algorithms for elliptic curves over finite fields, such as the SEA algorithm [278], making the decisional question of whether two elliptic curves are isogenous easy in this case.

Remark 2.9.4. Suppose $\varphi : E_1 \to E_2$ is a $\mathbb{F}_q$-rational (separable) isogeny of degree $N > 1$, i.e., $\# \ker(\varphi) = N$. A common point of confusion is how $E_1/\mathbb{F}_q$ and $E_2/\mathbb{F}_q$ can have the same number of $\mathbb{F}_q$-points, as guaranteed by Tate's theorem. Indeed, if all points in $\ker(\varphi)$ are $\mathbb{F}_q$-rational, there are $\#E_1(\mathbb{F}_q) - N$ elements in $E_1(\mathbb{F}_q)$ that are mapped to non-zero points in $E_2(\mathbb{F}_q)$. As $N > 1$, this seems to suggest that $\#E_2(\mathbb{F}_q) < \#E_1(\mathbb{F}_q)$. However, this is resolved by the fact that points in higher extension fields, i.e., points in $E(\mathbb{F}_{q^k})$ for $k > 1$, map to $E_2(\mathbb{F}_q)$ under φ. The same thing happens in the reverse direction when mapping points from E_2 to E_1 via the dual isogeny.

[3]Indeed, it is equal to the trace of the q-power Frobenius map considered as a linear transformation of the *ℓ-adic Tate module of E* [292, Remark V.2.6].

The proof of this Tate's theorem is not constructive. Given the existence of an isogeny $\varphi : E_1 \to E_2$, we need extra information to compute φ. Theorem 2.6.7 tells us that it is sufficient to know the kernel of φ. Explicitly constructing such an isogeny (without the extra information) is conjectured to be hard, as we will discuss in more detail in Section 4.3.

There are two distinct types of elliptic curve over $\mathbb{F}_q$: supersingular and ordinary. This distinction is important for this thesis: in Section 4.1 we will argue that supersingular elliptic curves (and their higher-dimensional analogue) are the correct choice within the context of isogeny-based cryptography.

Definition 2.9.5. Let E be an elliptic curve defined over finite field $\mathbb{F}_q$ of prime characteristic p. We say E is *supersingular* if $E[p] = \{0_E\}$. Otherwise, we have $E[p] = \mathbb{Z}/p\mathbb{Z}$ and E is *ordinary*.

Later in Lemma 4.1.1 from Section 4.1, we will give equivalent definitions of supersingularity which will help us generalise this property to higher dimensional p.p. abelian varieties.

Chapter 3

Quaternion Algebras

Deuring [133] showed the connection between the geometric world of supersingular curves and the arithmetic world of quaternion algebras. This connection has proved to be a powerful constructive and cryptanalytic tool in isogeny-based cryptography, and it will be essential for the work in this thesis. In Section 3.1 we give the necessary background on quaternion algebras and then depict the Deuring correspondence in Section 3.2. Throughout, we refer to Voight [316] and Leroux [217].

3.1 Quaternion algebras

A quaternion algebra is a non-commutative algebra, defined as follows.

Definition 3.1.1 (Quaternion algebra)**.** An algebra $\mathcal{B}$ over a field $\Bbbk$ is a *quaternion algebra* if there exist $i, j \in \mathcal{B}$ such that $\mathcal{B}$ is a $\Bbbk$-vector space with basis $1, i, j, ij$ satisfying and

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

for some $a, b \in \Bbbk^\times$. We usually denote ij by k.

We restrict to quaternion algebras defined over $\Bbbk = \mathbb{Q}$. For every prime p, we can define the field of p-adic numbers $\mathbb{Q}_p$, as defined in Section 1.2. The extension of scalars $\mathcal{B} \otimes_{\mathbb{Q}} \mathbb{Q}_p$ (as defined by Bourbaki [51, §II.5]) is either a matrix ring or a division algebra (see Example 1.2.1). We say that $\mathcal{B}$ is *unramified* (or *split*) at p if $\mathcal{B} \otimes_{\mathbb{Q}} \mathbb{Q}_p$ is the matrix ring $\mathrm{M}_2(\mathbb{Q}_p)$, and otherwise $\mathcal{B}$ is *ramified* at p. Similarly, we say that $\mathcal{B}$ is *unramified* (or *split*) at ∞ if $\mathcal{B} \otimes_{\mathbb{Q}} \mathbb{R}$ is the matrix ring $\mathrm{M}_2(\mathbb{R})$. Otherwise, $\mathcal{B}$ is *ramified* at ∞.

We focus on a particular quarternion algebra: namely the quaternion algebra over $\mathbb{Q}$ ramified at a prime p and ∞ and unramified elsewhere, denoted $\mathcal{B}_{p,\infty}$. When we introduce the Deuring correspondence in Section 3.2, the prime p will equal the characteristic of the finite field over which our elliptic curves are defined.

For an element $\alpha = x + yi + zj + wk \in \mathcal{B}_{p,\infty}$, the *conjugate* $\bar{\alpha}$ of α is defined as

$$\bar{\alpha} := x - yi - zj - wk,$$

where $x, y, z, w \in \mathbb{Q}$. We define the *reduced trace* on $\mathcal{B}_{p,\infty}$ by

$$\mathrm{trd}\colon \mathcal{B}_{p,\infty} \to \mathbb{Q}, \quad \alpha \mapsto \alpha + \bar{\alpha}. \tag{3.1}$$

The reduced trace trd is a $\mathbb{Q}$-linear map, meaning that for $\alpha, \beta \in \mathcal{B}_{p,\infty}$ we have $\mathrm{trd}(\alpha + \beta) =$

$\mathrm{trd}(\alpha) + \mathrm{trd}(\beta)$. Similarly, we define the *reduced norm* as

$$\mathrm{nrd}\colon \mathcal{B}_{p,\infty} \to \mathbb{Q}, \quad \alpha \mapsto \alpha\bar{\alpha}. \tag{3.2}$$

The reduced norm nrd is multiplicative, i.e., $\mathrm{nrd}(\alpha\beta) = \mathrm{nrd}(\alpha)\,\mathrm{nrd}(\beta)$.

3.1.1 Orders and Ideals

We are mainly interested in *lattices* of the quaternion algebra $\mathcal{B}_{p,\infty}$.

Definition 3.1.2. A *lattice* of $\mathcal{B}_{p,\infty}$ is a $\mathbb{Z}$-submodule of rank 4. If a lattice $\mathcal{O} \subset \mathcal{B}_{p,\infty}$ is also a subring of $\mathcal{B}_{p,\infty}$, then $\mathcal{O}$ is said to be an *order* of $\mathcal{B}_{p,\infty}$. An order is *maximal* if it is not properly contained in another order.

Remark 3.1.3. The ring of integers $\mathcal{O}_{\Bbbk}$ is an order in a quadratic field $\Bbbk = \mathbb{Q}(\sqrt{d})$. In fact, it is the *unique maximal order* in $\mathbb{Q}(\sqrt{d})$. However, this construction does not work in the non-commutative setting: the set of all integral elements (i.e., $\alpha \in \mathcal{B}_{p,\infty}$ satisfying a monic polynomial with integer coefficients) does not form an order, see [316, Example 10.1.1]. Furthermore, there are many maximal orders in $\mathcal{B}_{p,\infty}$: if $\mathcal{O}$ is a maximal order and $\alpha \in \mathcal{B}_{p,\infty}^{\times}$[1] then $\alpha\mathcal{O}\alpha^{-1} \subseteq \mathcal{B}_{p,\infty}$ is a maximal order, and we may have $\alpha\mathcal{O}\alpha^{-1} \neq \mathcal{O}$ due to non-commutativity.

Example 3.1.4. Consider the quaternion algebra $\mathcal{B}_{p,\infty}$ with $p \equiv 3 \bmod 4$. The lattice $\mathcal{O} = \mathbb{Z} + i\mathbb{Z} + j\mathbb{Z} + k\mathbb{Z}$ in $\mathcal{B}_{p,\infty}$ is closed under multiplication and so defines an order. However, it is not a maximal order. Indeed, $\mathcal{O} \subsetneq \mathbb{Z} + i\mathbb{Z} + \frac{i+j}{2}\mathbb{Z} + \frac{1+k}{2}\mathbb{Z}$.

Let $I \subseteq \mathcal{B}_{p,\infty}$ be a lattice and define

$$\mathcal{O}_L(I) := \{\alpha \in \mathcal{B}_{p,\infty} \colon \alpha I \subseteq I\}.$$

Then, $\mathcal{O}_L(I)$ is an order, called the *left order* of I. We define the *right order* of I analogously. If $\mathcal{O}$ is the left order of I, and $\mathcal{O}'$ is its right order, then we say that I is a *connecting* $(\mathcal{O}, \mathcal{O}')$*-ideal*. We sometimes denote I by $I(\mathcal{O}, \mathcal{O}')$ to highlight this property. Note that $\mathcal{O}$ and $\mathcal{O}'$ need not be distinct. This choice in terminology will become clear in Section 3.2 when we show the link between such ideals and isogenies that 'connect' two elliptic curves.

Example 3.1.5. Let I be a *principal* ideal, meaning that $I = \mathcal{O}_L(I)\alpha$ for some $\alpha \in \mathcal{B}_{p,\infty}^{\times}$. Then, $\mathcal{O}_R(I) = \alpha^{-1}\mathcal{O}_L(I)\alpha$. Therefore, $I = \alpha\mathcal{O}_R(I)$ and $\mathcal{O}_R(I)$ and $\mathcal{O}_L(I)$ are isomorphic.

Definition 3.1.6. For an order $\mathcal{O}$, we define a *left fractional* $\mathcal{O}$*-ideal* to be a lattice $I \subset \mathcal{B}_{p,\infty}$ such that $\mathcal{O} \subseteq \mathcal{O}_L(I)$. Furthermore, we say I is integral if $I \subseteq \mathcal{O}_L(I)$. We define *right* integral and fractional $\mathcal{O}$-ideals similarly.

As with elements of the quaternion algebra, we can define the conjugate and reduced norm of an ideal.

Definition 3.1.7. The conjugate of an ideal I is given by $\bar{I} := \{\bar{\alpha} \colon \alpha \in I\}$. The *reduced norm* $\mathrm{nrd}(I)$ of an ideal I is the largest positive rational number such that $\mathrm{nrd}(\alpha) \in \mathrm{nrd}(I)\mathbb{Z}$ for any $\alpha \in I$.

[1]In this thesis, we use the $R^{\times}$ to denote the invertible elements of R.

Remark 3.1.8. Any ideal I in $\mathcal{B}_{p,\infty}$ can be written as $I = \mathcal{O}_L(I)\alpha + \mathcal{O}_L(I)\operatorname{nrd}(I)$ for some $\alpha \in \mathcal{O}_L(I)$, and similarly for $\mathcal{O}_R(I)$. We simplify this notation by writing $\mathcal{O}\alpha + \mathcal{O}N = \mathcal{O}\langle \alpha, N\rangle$ for any order $\mathcal{O}$ [316, Chapter 16, Exercise 6].

Henceforth, we only deal with integral left fractional ideals (unless otherwise stated) and therefore simply refer to them as ideals. The choice of left ideals over right ideals is an arbitrary choice; we can switch from left to right ideals simply by taking conjugates as $\mathcal{O}_L(\bar{I}) = \mathcal{O}_R(I)$.

3.1.1.1 Invertible Ideals

Before defining invertible ideals, we discuss quaternion ideal multiplication.

Definition 3.1.9. Given two ideals I, J in a quaternion algebra $\mathcal{B}_{p,\infty}$, we say they are *compatible* if $\mathcal{O}_R(I) = \mathcal{O}_L(J)$.

We remark that this compatibility relation is neither symmetric nor transitive. When two ideals I, J in $\mathcal{B}_{p,\infty}$ are compatible, their *product* $IJ := \{\alpha\beta \colon \alpha \in I, \beta \in J\}$ is also an ideal with left order $\mathcal{O}_L(I)$ and right order $\mathcal{O}_R(J)$. Furthermore, $\operatorname{nrd}(IJ) = \operatorname{nrd}(I)\cdot\operatorname{nrd}(J)$.

Definition 3.1.10. An ideal I is *invertible* if there exists another ideal J such that

$$IJ = \mathcal{O}_L(I) = \mathcal{O}_R(J) \text{ and } JI = \mathcal{O}_L(J) = \mathcal{O}_R(I).$$

We denote J by I^{-1}.

When the ideal I is invertible, I and its conjugate satisfy the following relation

$$I\bar{I} = \operatorname{nrd}(I)\mathcal{O}_L(I), \tag{3.3}$$

which recovers a similar equation to Equation (3.2) which holds for elements of a quaternion algebra. Using this, we can compute the inverse as

$$I^{-1} = \frac{1}{\operatorname{nrd}(I)}\bar{I}. \tag{3.4}$$

When $\mathcal{O}$ is a maximal order in $\mathcal{B}_{p,\infty}$, every $\mathcal{O}$-ideal is invertible [316, Proposition 16.1.2].

Example 3.1.11. If I is a principal ideal, then $I = \mathcal{O}_L(I)\alpha$ for some $\alpha \in \mathcal{B}_{p,\infty}^{\times}$, and I is invertible. Letting $\mathcal{O} := \mathcal{O}_L(I)$, we have $I^{-1} = \alpha^{-1}\mathcal{O}$: we have

$$(\mathcal{O}\alpha)(\alpha^{-1}\mathcal{O}) = \mathcal{O}(\alpha\alpha^{-1})\mathcal{O} = \mathcal{O},$$

and by Example 3.1.5

$$(\alpha^{-1}\mathcal{O})(\mathcal{O}\alpha) = \alpha\mathcal{O}\alpha^{-1} = \mathcal{O}_R(I),$$

as required.

3.1.1.2 Equivalent Ideals

We define an equivalence relation $\sim$ on (left) $\mathcal{O}$-ideals by right scalar multiplication. Two ideals I and J are *(right-)equivalent* if $I = J\alpha$ for some $\alpha \in \mathcal{B}_{p,\infty}^{\times}$. We denote this by $I \sim J$. If $I \sim J$, then we have $\mathcal{O}_L(I) = \mathcal{O}_L(J)$ and $\mathcal{O}_R(I) = \alpha^{-1}\mathcal{O}_R(J)\alpha$.

Following Kohel, Lauter, Petit, and Tignol [203], for any invertible ideal J we can construct a map χ_J that sends quaternion elements in $\mathcal{B}_{p,\infty}^{\times}$ to ideals. On input $\alpha \in \mathcal{B}_{p,\infty}^{\times}$, it is defined as

$$\chi_J(\alpha) := J\frac{\bar{\alpha}}{\operatorname{nrd}(J)}. \tag{3.5}$$

Then, $I = \chi_J(\alpha)$ is an ideal of norm $\operatorname{nrd}(\alpha)/\operatorname{nrd}(J)$. For $\alpha \in J\backslash\{0\}$, the ideal I is equivalent to J. In fact, ideals equivalent to J are precisely the ideals $\chi_J(\alpha)$ with $\alpha \in J\backslash\{0\}$.

3.2 The Deuring Correspondence

In 1941, Deuring [133] showed that, for a supersingular elliptic curve E defined over $\bar{\mathbb{F}}_p$, the endomorphism ring of E is isomorphic to a maximal order $\mathcal{O}$ in $\mathcal{B}_{p,\infty}$. In this section, we describe this correspondence in more detail. For a modern reformulation of the Deuring correspondence and proofs of the statements in this section, we refer to Voight [316, Chapter 42] or Waterhouse [317].

We first demonstrate the Deuring correspondence through an example.

Example 3.2.1. Let $p \equiv 3 \bmod 4$ and consider the elliptic curve $E_0\colon y^2 = x^3 + x$ defined over $\mathbb{F}_{p^2}$ with $j(E_0) = 1728$. Then $\operatorname{End}(E_0)$ is isomorphic to the maximal order $\mathcal{O}_0 = \left\langle 1, i, \frac{i+j}{2}, \frac{1+k}{2} \right\rangle$ in $\mathcal{B}_{p,\infty}$ via the isomorphism

$$\begin{aligned} \mathcal{O}_0 &\xrightarrow{\sim} \operatorname{End}(E_0) \\ \alpha = x + yi + zj + wk &\longmapsto \theta = x + y\iota + z\pi + w\iota\pi \end{aligned}$$

where $x, y, z, w \in \mathbb{Z}$, $\pi\colon (x, y) \mapsto (x^p, y^p)$ is the p-power Frobenius endomorphism on E_0, and $\iota\colon (x, y) \mapsto (-x, \sqrt{-1}y)$.

The elliptic curve E_0 is somewhat special as we were able to explicitly compute its endomorphism ring $\operatorname{End}(E_0)$ and construct the isomorphism to a maximal order $\mathcal{O}_0$ in $\mathcal{B}_{p,\infty}$. In general, given a supersingular elliptic curve, it is conjecturally hard to compute its endomorphism ring $\operatorname{End}(E)$. In fact, the hardness of this problem underlies the security of many cryptographic schemes constructed with isogenies. We discuss this in greater depth in Section 4.3. Conversely, given a maximal order $\mathcal{O}$, we can recover the corresponding elliptic curve E in polynomial time by taking advantage of the few supersingular curves whose endomorphism rings are known. For example, we could use [147, Algorithm 12].

The Deuring correspondence extends beyond this. It is in fact an equivalence of categories, meaning that there is furthermore a correspondence between the morphisms of each object: isogenies and ideals. Given a supersingular curve E, we can associate each pair (E', φ), where E' is another supersingular elliptic curve and $\varphi : E \to E'$ is an isogeny, to a left integral $\mathcal{O}_0$-ideal, where $\mathrm{End}(E_0) \cong \mathcal{O}_0$. Furthermore, every such ideal arises in this way.

Remark 3.2.2. We identify $\alpha \in \mathcal{O}_0$ as an endomorphism of E_0 via the explicit isomorphism $\mathcal{O}_0 \cong \mathrm{End}(E_0)$ given in Example 3.2.1. When given an isogeny $\varphi : E_0 \to E$ corresponding to a $(\mathcal{O}_0, \mathcal{O})$-ideal I, the isomorphism $\mathcal{O}_0 \cong \mathrm{End}(E_0)$ naturally induces an isomorphism $\mathcal{O} \cong \mathrm{End}(E)$. In these cases, we do not explicitly give the isomorphism and use the implicitly defined isomorphism $\mathcal{O} \cong \mathrm{End}(E)$.

The correspondence is defined using kernel ideals (see, for example, [317, §3.2]). Given a $\mathcal{O}_0$-ideal I of norm coprime to p, we define

$$E[I] := \{P \in E \mid \alpha(P) = 0_E \ \forall \alpha \in I\} = \bigcap_{\alpha \in I} \ker \alpha \subset E,$$

to be the kernel of I. By Remark 3.1.8, we can write $I = \mathcal{O}\langle \alpha, N\rangle$ where $\mathrm{nrd}(I) = N$, and we have $E[I] = \ker(\alpha) \cap E[N]$ [316, §42.2.1]. Given such an I, we denote the corresponding separable isogeny with kernel $E[I]$ by φ_I. Conversely, given a separable isogeny φ from E, the corresponding ideal is

$$I_\varphi = \{\alpha \in \mathcal{O} \mid \alpha(P) = 0_E \ \forall P \in \ker \varphi\}.$$

The isogeny φ is an endomorphism if and only if I_φ is a principal ideal.

This correspondence is also compatible with a number of properties of ideals and isogenies. Firstly, it is compatible with the 'size' of the maps, in the sense that $\deg(\varphi) = \mathrm{nrd}(I_\varphi)$ [317, Theorem 3.15]. Further to this, letting $\varphi\colon E_1 \to E_2$ be an isogeny, we have $I_{\widehat{\varphi}} = \overline{I_\varphi}$. If $\psi : E_1 \to E_2$ is another isogeny, then $I_{\psi\circ\varphi} = I_\varphi \cdot I_\psi$. As a consequence, if we have two isogenies $\varphi : E \to E'$ and $\psi : E \to E'$ then the corresponding ideals are equivalent $I_\varphi \sim I_\psi$ (as defined in Section 3.1.1.2). The converse statement can be found in [316, Lemma 42.2.7].

The correspondence given above is largely theoretical, and it still remains unclear whether it can be made *effective*. To build cryptographic schemes using the Deuring correspondence, we need efficient algorithms that will translate between isogenies of supersingular elliptic curves and ideals connecting maximal orders in $\mathcal{B}_{p,\infty}$. The following two sections will be dedicated to exhibiting algorithms that enable us to perform the translation efficiently. We introduce them within the context of the isogeny-based signature scheme SQIsign [125, 126], but note that the ideas in these algorithms appeared already in [147, 174, 203].

3.2.1 Converting $\mathcal{O}_0$-ideals to isogenies

The simplest translation from ideals to isogenies is when I is a $\mathcal{O}_0$-ideal, and the corresponding isogenies can be defined over $\mathbb{F}_{p^2}$ using both quadratic twists of E_0. We detail the algorithm $\mathsf{IdealToIsogeny}_D$ for ideals of norm dividing D fully in Algorithm 3.1. It terminates in $O(\sqrt{D})$ operations over the field of definition of $E_0[D]$ [217, §4.2.1], where $\mathcal{O}_0 \cong \mathrm{End}(E_0)$, which for this section we assume to be $\mathbb{F}_{p^2}$. In Chapter 11, we will adapt this algorithm to handle cases where $E_0[D]$ is defined over an extension field of $\mathbb{F}_{p^2}$.

Suppose we would like to translate the $\mathcal{O}_0$-ideal $I = \mathcal{O}_0\langle \alpha, D\rangle$. The main idea of the algorithm is to find the action of the endomorphism

$$\alpha = x_2 + x_2 i + x_3 \frac{i+j}{2} + x_4 \frac{1+k}{2}, \ x_i \in \mathbb{Z},$$

on a fixed basis of $P_D, Q_D \in E[D]$, from which we can recover $\ker(\alpha) \cap E[D]$. Explicitly, let $\mathbf{M}_i, \mathbf{M}_{\frac{i+j}{2}}, \mathbf{M}_{\frac{1+k}{2}}$ be the action of i, $\frac{i+j}{2}$, $\frac{1+k}{2}$, respectively. We recover the action $\mathrm{M}_\alpha \in \mathrm{M}_2(\mathbb{Z}/D\mathbb{Z})$ as

$$\mathbf{M}_\alpha := x_1 \mathbf{I} + x_2 \mathbf{M}_i + x_3 \mathbf{M}_{\frac{i+j}{2}} + x_4 \mathbf{M}_{\frac{1+k}{2}},$$

and then compute some vector $[a,b]^T \in \ker \mathbf{M}_\alpha$. This will give the corresponding generator $[a]P_D + [b]Q_D$ of the kernel, from which we compute the isogeny φ_I.

Algorithm 3.1 IdealToIsogeny$_D(I)$

Input: I, a left $\mathcal{O}_0$-ideal of norm dividing D.
Output: The corresponding isogeny φ_I.

```
1: Compute $\alpha$ such that $I = \mathcal{O}_0\langle \alpha, \mathrm{nrd}(I) \rangle$
2: Let $\mathbf{A} = [1, i, \frac{i+j}{2}, \frac{1+k}{2}]$ denote a basis of $\mathcal{O}_0$
3: Compute $\mathbf{v}_\alpha := [x_1, x_2, x_3, x_4]^T \in \mathbb{Z}^4$ such that $\mathbf{A}\mathbf{v}_\alpha = \alpha$
4: $\mathbf{M}_\alpha := x_1 \mathbf{I} + x_2 \mathbf{M}_i + x_3 \mathbf{M}_{\frac{i+j}{2}} + x_4 \mathbf{M}_{\frac{1+k}{2}}$
5: Compute $[a,b]^T \in \ker \mathbf{M}_\alpha$
6: $K_D := [a]P_D + [b]Q_D$
7: Set $\phi_I$ to be the isogeny generated by the point $K_D$.
8: return $\varphi_I$
```

3.2.2 Converting $\mathcal{O}$-ideals to isogenies

The algorithm we presented in Section 3.2.1 works only for special orders, and is therefore quite restrictive. However, the ideal-to-isogeny algorithm IdealToIsogeny can be extended to work for a $\mathcal{O}$-ideal of ℓ-power norm for a prime ℓ. This algorithm, IdealToIsogenyEichler, was introduced by De Feo, Leroux, Longa, and Wesolowski [126, §4] (as an improvement to [125, Alg. 8]). It is used in the key generation and signing algorithms of SQIsign.

To fix an isomorphism $\mathcal{O} \cong \mathrm{End}(E)$, we require the knowledge of a connecting $(\mathcal{O}_0, \mathcal{O})$-ideal and the corresponding isogeny, which will allow us to transfer the known isomorphism $\mathcal{O}_0 \cong \mathrm{End}(E_0)$, as described in Remark 3.2.2. Given a left $\mathcal{O}$-ideal I of large prime power norm ℓ^e, a $(\mathcal{O}_0, \mathcal{O})$-ideal J of ℓ-power norm and corresponding isogeny $\varphi_J : E_0 \to E$, IdealToIsogenyEichler outputs φ_I of degree ℓ^e.

When the norm ℓ^e is large, we may not have $\mathbb{F}_{p^2}$-rational ℓ^e-torsion on E (or its quadratic twist). As a result, the elliptic curve operations in the ideal-to-isogeny translation would need to be done over a field extension. Instead, we perform the translation in steps of size ℓ^f, with $e = gf$ and where the ℓ^f-torsion is $\mathbb{F}_{p^2}$-rational. Namely, we split up the ideal I into g ideals I_i of norm ℓ^f and translate these ideals to isogenies φ_{I_i} such that $\varphi_I = \varphi_{I_g} \circ \cdots \circ \varphi_{I_2} \circ \varphi_{I_1}$.

To translate left $\mathcal{O}$-ideals I of norm ℓ^f to an isogeny, we need to understand how a special endomorphism $\theta \in \mathrm{End}(E)$ acts. We choose this endomorphism in such a way that P and $\theta(P)$

generate the ℓ^f-torsion for some input point $P \in E[\ell^f]$, and the kernel generator of φ_I can be constructed from these generators. The precise conditions placed on θ for this to be satisfied are given in [126, Lemma 8]. Most notably, the endomorphism θ is of order T^2 where $T \approx p^{5/4}$ is coprime to ℓ. Viewing θ as an element of $\mathcal{O}$, to evaluate θ we decompose it into two ideals of norm T, and convert these into isogenies of degree T using two executions of $\mathsf{IdealToIsogeny}_T$. For precise details of this algorithm and a proof of its correctness, we refer to [126, §4].

The main takeaways from this section are as follows. Firstly, the only subroutine involving elliptic curve operations is $\mathsf{IdealToIsogeny}_T$. This will be important for Chapter 11 when we look at how to modify this to allow our elliptic curve points to be defined over field extensions $\mathbb{F}_{p^{2k}}$. Secondly, the bottleneck of this procedure is the computation of $\mathbb{F}_{p^2}$-rational T-isogenies, where T is as before. This will be important for Chapter 10 when this condition will impose restrictions on SQIsign-friendly parameters.

3.2.3 Converting isogenies to ideals

Translating isogenies to their corresponding ideals is comparatively easy. Consider the elliptic curve E with endomorphism ring $\mathrm{End}(E) \cong \mathcal{O}$. We want to find the $\mathcal{O}$-ideal corresponding to the isogeny $\varphi : E \to E/\langle P \rangle$ with kernel generated by a point $P \in E[N]$. To do so, we first compute a basis $\{\theta_1, \theta_2, \theta_3, \theta_4\}$ of $\mathrm{End}(E)$, where each θ_i has norm coprime to N. We then find i, j such that $\theta_i(P), \theta_j(P)$ is a basis of $E[N]$. Taking $k \neq i, j$, we compute a, b such that

$$\theta_k(P) = [a]\theta_i(P) + [b]\theta_j(P).$$

The endomorphism $\alpha = \theta_j - a\theta_i - b\theta_j$ sends the point P to the identity, and $I = \mathcal{O}\langle \alpha, N\rangle$ is the $\mathcal{O}$-ideal corresponding to φ, where we view α as an element in $\mathcal{B}_{p,\infty}$ via the isomorphism $\mathrm{End}(E) \cong \mathcal{B}_{p,\infty}$. For a proof of this fact, see [217, Lemma 4.2.2].

Example 3.2.3. Consider the supersingular elliptic curve $E_0 : y^2 = x^3 + x$ with $j(E_0) = 1728$, defined over $\mathbb{F}_{23^2} = \mathbb{F}_{23}(i)$ with $i^2 + 1 = 0$. Then, as shown in Example 3.2.1, $\mathrm{End}(E_0) \cong \mathcal{O}_0$ where $\mathcal{O}_0 = \langle 1, i, \frac{i+j}{2}, \frac{1+k}{2}\rangle$. In this case, we fix the choice $\theta = j + \frac{1+k}{2}$ and $\eta = i$.

As we need the maximal order $\mathcal{O} \cong \mathrm{End}(E)$ to perform this conversion, if $j(E) \neq 1728$ then again we require knowledge of an isogeny $\varphi : E_0 \to E$. We summarise the effective Deuring correspondence in Figure 3.1.

$$\begin{tikzcd}
\text{Maximal order } \mathcal{O} \text{ in } \mathcal{B}_{p,\infty} \arrow[r, bend left] & E/\mathbb{F}_{p^2} \text{ supersingular} \arrow[l, bend left] \\
\begin{array}{c}\text{Integral ideal } \mathcal{O}_1\text{-ideal } I \\ \text{and right } \mathcal{O}_2\text{-ideal}\end{array} \arrow[r, bend left] & \begin{array}{c}\text{Isogeny } \varphi : E_1 \to E_2 \\ \text{with } \mathrm{End}(E_i) \cong \mathcal{O}_i\end{array} \arrow[l, bend left] \\
\begin{array}{c}\mathrm{nrd}\,(I) \\ \bar{I} \\ I_\varphi \cdot I_\psi\end{array} \arrow[r, leftrightarrow] & \begin{array}{c}\deg(\varphi) \\ \widehat{\varphi} \\ \psi \circ \varphi\end{array}
\end{tikzcd}$$

FIGURE 3.1: A summary of the effective Deuring correspondence using the current state-of-the-art algorithms. Arrows in green indicate when there is an efficient algorithm to compute the correspondence. The arrow in red is only computable in exponential time.

Chapter 4

Isogenies in cryptography

The use of isogenies in post-quantum cryptography was introduced by Couveignes [108] and independently rediscovered by Rostovtsev and Stolbunov [275], who considered ordinary elliptic curves defined over $\mathbb{F}_q$ (for q some power of prime p) and isogenies between them. Later, De Feo, Jao, and Plût [123] introduced a key-exchange protocol *Supersingular Isogeny Diffie–Hellman* (SIDH), and corresponding key encapsulation mechanism SIKE. Here, a different setting was used: supersingular elliptic curves over $\mathbb{F}_{p^2}$, first introduced to cryptography by Charles, Goren and Lauter [76]. It offers two main advantages. Firstly, every supersingular elliptic curve is isomorphic to a curve E defined over $\mathbb{F}_{p^2}$. Secondly, for any supersingular elliptic curve E, given a prime N, we can straightforwardly enforce the N-isogenies to be $\mathbb{F}_{p^2}$-rational. In this way, all arithmetic is efficiently computable over $\mathbb{F}_{p^2}$ (and no larger extension).

In this thesis, we focus solely on the supersingular setting. More generally, we work with *superspecial* principally polarised abelian varieties, a natural generalisation of supersingular elliptic curves to higher dimensions. In this setting, the security of isogeny-based cryptography rests on the hardness of finding an isogeny between two superspecial p.p. abelian varieties, conjectured to be hard for both classical and quantum computers. As such, isogenies are a promising candidate for the construction of post-quantum secure cryptoschemes.

Throughout this chapter, we fix $\Bbbk$ to be a finite field $\mathbb{F}_q$ of prime characteristic $p > 5$.

4.1 Superspecial abelian varieties

We follow the methodology introduced by Castryck, Decru, and Smith [72, §2], who argue that, for cryptographic applications involving higher dimensional p.p. abelian varieties, the correct generalisation of supersingularity is *superspeciality*. The aim of this section is to introduce superspecial p.p. abelian varieties from a cryptographic perspective. For a more extensive survey of superspecial abelian surfaces, see Brock's thesis [52].

We motivate the notion of superspeciality by first considering the dimension-1 case. Recall from Definition 2.9.5, that an elliptic curve $E/\mathbb{F}_q$ is supersingular if it has no non-trivial p-torsion. We observe, however, that there are many alternative definitions of supersingularity.

Lemma 4.1.1. *Let E be an elliptic curve defined over $\mathbb{F}_q$ of prime characteristic p. The following are equivalent:*

(a) $\operatorname{tr}\pi_{\mathrm{E}} \equiv 0 \bmod p$, *where π_E is the q-power Frobenius endomorphism.*

(b) $E[p] = \{0_E\}$.

(c) $[p] : E \to E$ *is purely inseparable.*

(d) The endomorphism ring $\mathrm{End}(E)$ *is a maximal order in the quaternion algebra* $\mathcal{B}_{p,\infty}$*.*

If an elliptic curve E satisfies the properties above, we say E is supersingular*.*

Proof. A proof of the equivalence (b) $\iff$ (c) $\iff$ (d) can be found in [292, Theorem V.3.1]. The equivalence (a) $\iff$ (b) is in the proof of [292, Theorem V.4.1]. □

The alternative definitions for supersingularity in Lemma 4.1.1 generalise to distinct properties when considering abelian varieties of higher dimension.

Definition 4.1.2. Let $E/\overline{\mathbb{F}}_p$ be a supersingular elliptic curve. We say a p.p. abelian variety $A/\mathbb{F}_q$ of dimension g is *supersingular* if it is $\overline{\mathbb{F}}_p$-isogenous (as an unpolarised abelian variety) to $E^g := E \times \cdots \times E$. We say A is *superspecial* if it is $\overline{\mathbb{F}}_p$-isomorphic (as an unpolarised abelian variety) to E^g. For $g = 2$, we say a genus-2 curve C is *supersingular* or *superspecial*, if its Jacobian $\mathcal{J}_C$ is.

An alternative definition for a superspecial p.p. abelian variety $A/\mathbb{F}_q$ of dimension g is that its Hasse–Witt matrix $\mathbf{M} \in \mathbb{F}_q^{g\times g}$ vanishes identically [52, Theorem 2.3A]. The matrix $\mathbf{M}$ characterises how the Frobenius map acts on A.[1] For $g = 1$, $\mathbf{M}$ is a 1×1 matrix consisting of an element m_1, where $m_1 \equiv a \bmod p$ for a the trace of Frobenius π_A. In this way, this definition is a natural generalisation of the property given in Item (a) of Lemma 4.1.1 for elliptic curves. For genus-2 curves $y^2 = f(x)$, we have $\mathbf{M} = (m_{ij})_{1\le i,j\le 2}$ with

$$m_{ij} = f_{pi-j}^{(p-1)/2} \bmod p,$$

where f_k^n denotes the coefficient of x^k in $f(x)^n$ [52, Corollary 2.11]. If A is superspecial, then it is supersingular. The converse is false in dimension $g \ge 2$. For dimension $g \le 2$, the supersingularity condition is equivalent to A having no non-trivial p-torsion, thus generalising the property given in Item (b) of Lemma 4.1.1 [255, §4]. However, supersingularity becomes a stronger notion in dimension $g > 2$.

In isogeny-based cryptography, we mostly work with superspecial abelian varieties of dimension $g = 1$ and 2, and isogenies between them. This is enabled by the fact that being superspecial is a property which is invariant under (polarised) isogeny. Namely, if $A_1/\mathbb{F}_q$ is a superspecial p.p. abelian variety, and $\varphi : A_1 \to A_2$ is an isogeny (of p.p. abelian varieties) of degree coprime to p, then A_2 is also superspecial. We prove the case $g = 1$ below. The case $g = 2$ may be found in [52, Lemma 2.2A].

Proposition 4.1.3. *Let $\varphi : E_1 \to E_2$ be an isogeny of elliptic curves E_1, E_2 defined over $\mathbb{F}_q$ of characteristic p. Then E_1 is supersingular if and only if E_2 is supersingular.*

Proof. Suppose E_1 is supersingular. By Lemma 4.1.1, we have $\mathrm{tr}_{E_1} \equiv 0 \bmod p$, and so Remark 2.9.3 tells us $\#E_1(\mathbb{F}_q) \equiv 1 \bmod p$. By Tate's theorem, E_2 has order $\#E_2(\mathbb{F}_q) = \#E_1(\mathbb{F}_q) \equiv 1 \bmod p$, and is therefore supersingular (again by Lemma 4.1.1). Noting that $\widehat{\varphi}\colon E_2 \to E_1$ is an isogeny, changing the roles of E_1 and E_2, we get the converse implication: E_1 is supersingular if E_2 is. □

[1]More precisely, $\mathbf{M}$ represents the action of the Frobenius operator on the cohomology group $H^1(A, \mathcal{O}_A)$ with respect to some basis, where $\mathcal{O}_A$ is the structure sheaf.

With the construction of efficient protocols in mind, the reason for working with superspecial p.p. abelian varieties A in cryptography is two-fold. Firstly, the following theorem shows us that every superspecial p.p. abelian variety admits a model over $\mathbb{F}_{p^2}$, a necessary condition for the arithmetic to be performed over $\mathbb{F}_{p^2}$.

Theorem 4.1.4. *Let $A/\mathbb{F}_{p^k}$ be superspecial p.p. abelian variety of dimension g. Then A is $\overline{\mathbb{F}}_p$-isomorphic to a superspecial p.p. abelian variety defined over $\mathbb{F}_{p^2}$.*

Proof. We follow Silverman [292, Theorem V.3.1] to prove this for the case $g = 1$. For $g \geq 2$, a proof can be found in [186, Theorem 1]. Let $A/\mathbb{F}_{p^k}$ be a supersingular elliptic curve. Let $\pi : A \to A^{(p)}$ be the p-power Frobenius, and $\widehat{\pi} : A^{(p)} \to A$ its dual (called the *Verschiebung*). As A is supersingular, the Verschiebung $\widehat{\pi}$ is purely inseparable. By [292, Corollary II.2.12], it factors as

$$\begin{tikzcd} A^{(p)} \arrow[r, "\pi'"] \arrow[rr, bend right, "\widehat{\pi}"'] & A^{(p^2)} \arrow[r, "\psi"] & A \end{tikzcd}$$

where π' is the p-power Frobenius map on $A^{(p)}$ and ψ is a (seperable) isogeny of degree one. Therefore, ψ is an isomorphism and so $j(A) = j(A^{(p^2)}) = j(A)^{p^2}$, implying that $j(A) \in \mathbb{F}_{p^2}$. Let $j = j(A) \in \mathbb{F}_{p^2}$ and consider the elliptic curve

$$E\colon y^2 = x^3 + \frac{3j}{1728 - j}x + \frac{2j}{1728 - j},$$

when $j \neq 0, 1728$, $E\colon y^2 = x^3 + 1$ for $j = 0$, and $E\colon y^2 = x^3 + x$ for $j = 1728$. Then, E is defined over $\mathbb{F}_{p^2}$ and $j(E) = j$, so $A \cong E$ over $\overline{\mathbb{F}}_p$. □

A second important feature of superspecial p.p. abelian varieties is that we can straightforwardly write down the order $\#A(\mathbb{F}_q)$. We concentrate first on exhibiting this in the case $g = 1$.

Proposition 4.1.5. *Let E be a supersingular elliptic curve defined over $\mathbb{F}_{p^2}$ with $p > 5$. Then we have*

$$\#E(\mathbb{F}_{p^k}) = \begin{cases} p^k + 1, \text{ when } k \text{ is odd} \\ p^k + 1,\ p^k \pm p^{k/2} + 1 \text{ or } (p^{k/2} \pm 1)^2, \text{ when } k \text{ is even} \end{cases}.$$

Proof. We have $\#A(\mathbb{F}_{p^k}) = p^k + 1 - a$ where a is the trace of the p^k-power Frobenius endomorphism on A. By a result of Waterhouse [317, Theorem 4.1]: (a) if k is even, then $a = \pm 2\sqrt{p^k}$, or $a = \pm\sqrt{p^k}$ and $p \not\equiv 1 \bmod 3$, or $a = 0$ and $p \not\equiv 1 \bmod 4$; (b) if k is odd, then $a = 0$. From this we get the values for $\#A(\mathbb{F}_{p^k})$ in the statement of the proposition. □

Every supersingular elliptic curve admits a model such that the p^2-power Frobenius endomorphism π is equal to the multiplication-by-$(-p)$ map. Such elliptic curves E are $\mathbb{F}_{p^2}$-isogenous to elliptic curves defined over $\mathbb{F}_p$ [292, Ex. 5.15] and satisfy

$$E(\mathbb{F}_{p^{2k}}) = \mathbb{Z}/(p^k - (-1)^k)\mathbb{Z} \oplus \mathbb{Z}/(p^k - (-1)^k)\mathbb{Z}, \tag{4.1}$$

while their quadratic twist over $\mathbb{F}_{p^{2k}}$, denoted E_k^t, satisfies

$$E_k^t(\mathbb{F}_{p^{2k}}) = \mathbb{Z}/(p^k + (-1)^k)\mathbb{Z} \oplus \mathbb{Z}/(p^k + (-1)^k)\mathbb{Z}. \tag{4.2}$$

For any prime $N \mid p^k \pm 1$, the full N-torsion group $E[N]$ is defined over $\mathbb{F}_{p^{2k}}$, either on the curve or on its twist. As a result, we can ensure that isogenies of degree N are $\mathbb{F}_{p^{2k}}$-rational. For example, say we start with a supersingular elliptic curve $E_1/\mathbb{F}_{p^2}$ with $\#E_1(\mathbb{F}_{p^2}) = (p+1)^2$. Choosing p such that $3 \mid p+1$, we can compute an $\mathbb{F}_{p^2}$-rational 3-isogeny $\varphi : E_1 \to E_2$. By Tate's isogeny theorem in Theorem 2.9.1, $\#E_2(\mathbb{F}_{p^2}) = (p+1)^2$, and so we can compute $\mathbb{F}_{p^2}$-rational 3-isogenies from E_2, and so on. In this way, a simple choice of p ensures we can compute chains of $\mathbb{F}_{p^2}$-rational isogenies from any starting curve with order $(p+1)^2$.

Costello [99] gives a framework to work on a supersingular elliptic curve and its twist, meaning that we can compute $\mathbb{F}_{p^2}$-rational isogenies for any prime N dividing $p \pm 1$. Let γ be a non-square in $\mathbb{F}_{p^2}$, and let $\delta^2 = \gamma$ so that $\mathbb{F}_{p^4} = \mathbb{F}_{p^2}(\delta)$. Recall from Section 2.2.3 that $E(\mathbb{F}_{p^4})$ is isomorphic to $E^t(\mathbb{F}_{p^4})$ via the map $(x, y) \mapsto (x, \delta y)$, which leaves the x-coordinate unchanged. Costello [99, §3] details how x-only arithmetic can therefore be used to perform isogeny computations on a curve and its twist at no extra cost.

We now consider dimension $g = 2$. Every superspecial p.p. abelian variety has a model defined over $\mathbb{F}_{p^2}$, see [186, Theorem 1]. Furthermore, by definition of superspeciality, the p.p. abelian surface A is $\overline{\mathbb{F}}_p$-isomorphic to $E \times E$ for some supersingular curve E (as an unpolarised abelian variety). As such, every superspecial abelian surface admits a model $A/\mathbb{F}_{p^2}$ with $\#A(\mathbb{F}_{p^2}) = (p+1)^4$, and

$$A(\mathbb{F}_{p^2}) \cong (\mathbb{Z}/(p+1)\mathbb{Z})^4 .$$

If $N \mid p+1$ for prime N, this implies A has full $\mathbb{F}_{p^2}$-rational N-torsion. It follows that when $p \neq 2$, A automatically has full $\mathbb{F}_{p^2}$-rational 2-torsion. In particular, if A is the Jacobian of a genus-2 hyperelliptic curve C, this means that C can be put in Rosenhain form.

From the discussion above, we can construct efficient protocols from superspecial p.p. abelian varieties of dimension 1 and 2 and their isogenies using arithmetic solely over $\mathbb{F}_{p^2}$, rather than taking arbitarily large extensions of $\mathbb{F}_p$.

Beyond efficiency, for cryptographic applications we require a *hard* problem from which to base the security of our protocols. Superspecial abelian varieties also naturally offer such a problem: the *dimension-g superspecial isogeny problem.*

Problem. *Given a pair of superspecial p.p. abelian varieties A_1 and A_2 of dimension g defined over $\mathbb{F}_{p^2}$, find a polarised $\overline{\mathbb{F}}_p$-isogeny $A_1 \to A_2$.*

We will revisit this problem in Section 4.3 and study it in greater depth. Before doing so, we introduce *the superspecial isogeny graph.*

4.2 The superspecial isogeny graph

An important object in isogeny-based cryptography is the dimension-g superspecial isogeny graph, whose vertex set $\mathcal{S}_g(\overline{\mathbb{F}}_p)$ is the set of isomomorphism classes of superspecial p.p. abelian varieties of dimension g defined over $\overline{\mathbb{F}}_p$, and its edge set contains $(N, \ldots, N)$-isogenies (up to composition by an isomomorphism). Throughout we will only be considering the case where N is coprime to p. By solely studying the expansion properties of this graph, we can obtain some insight into the hardness of finding isogenies between p.p. abelian varieties. However, as we will show in

Section 4.3.1 and Section 4.3.2, by exploiting the special structure of this graph arising from the geometric nature of abelian varieties we can obtain new, more efficient attacks against the isogeny problem. Improving these attacks will be the main focus of Chapters 7 and 8. In this section, we explore the superspecial isogeny graph and discuss what is already known about its structure. We first motivate the definitions in this section through an example in dimension 1.

Example 4.2.1. Let $g = 1$ and consider $\mathcal{X}_1(\overline{\mathbb{F}}_{127}, 2)$, the supersingular 2-isogeny graph for $p = 127 = 2^7 - 1$, given in Figure 4.1. We label each edge by the j-invariant corresponding to the $\overline{\mathbb{F}}_p$-isomorphism class of supersingular elliptic curves defined over $\mathbb{F}_{127^2} = \mathbb{F}_{127}(i)$ where i is a root of $x^2 + 1 \in \mathbb{F}_{127}[x]$. In this case, there are 11 distinct j-invariants.

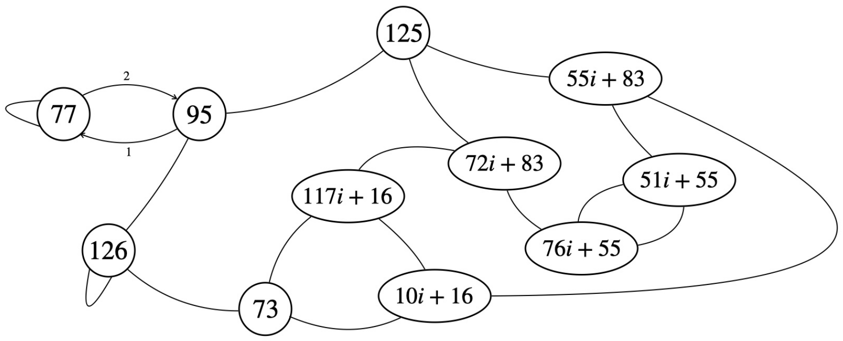


Figure 4.1: The graph $\mathcal{X}_1(\overline{\mathbb{F}}_{127}, 2)$, where $\mathbb{F}_{127^2} = \mathbb{F}_{127}(i)$ where i is a root of $x^2 + 1 \in \mathbb{F}_{127}[x]$.

We first observe that there exists a path between any two vertices. We will see that this remains a property for general degree N in dimensions $g = 1$ and 2: the graph $\mathcal{X}_g(\overline{\mathbb{F}}_p, N)$ is *connected*. Furthermore, for all but one vertex, the graph is 3-regular and undirected: given a 2-isogeny $\phi : E \to E'$, there exists a 2-isogeny from E' to E given by the dual isogeny $\widehat{\phi} : E' \to E$.

The exception occurs at the vertex with label $j = 1728 \equiv 77 \bmod 127$ due to extra automorphisms on the corresponding elliptic curve. All supersingular elliptic curves $E \in \mathcal{S}_1(\overline{\mathbb{F}}_p)$ have the automorphism $[-1] : (x, y) \mapsto (x, -y)$. However, the elliptic curve $E_{77} \colon y^2 = x^3 + x$ with $j(E_{77}) = 77$ has an extra automorphism $\iota \colon (x, y) \mapsto (-x, iy)$.

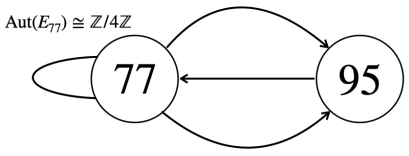


Figure 4.2: A look at the node $j = 77$, corresponding to elliptic curve E_{77}, in the graph $\mathcal{X}_1(\overline{\mathbb{F}}_{127}, 2)$.

In Figure 4.2, we zoom into the neighbourhood of this vertex and observe the following. There are two edges from $j(E_{77})$ to $j(E_{95})$, corresponding to the 2-isogenies

$$E_{77} \to E_{77}/\langle (i, 0) \rangle \cong E_{95}, \text{ and } E_{77} \to E_{77}/\langle (-i, 0) \rangle \cong E_{95}.$$

The extra automorphism of E_{77} maps the 2-torsion point $(i, 0)$ to a distinct point $(-i, 0)$ of order 2. As $E_{77}/\langle\iota(i, 0)\rangle \cong E_{77}/\langle(i, 0)\rangle$, the automorphism ι gives rise to two distinct isogenies $E_{77} \to E_{95}$. However, there is only one edge from $j(E_{95})$ to $j(E_{77})$, corresponding to the 2-isogeny $E_{95} \to E_{95}/\langle(0, 0)\rangle \cong E_{77}$. Indeed, E_{95} only has non-trivial automorphism $[-1]$, but $[-1]\langle(0, 0)\rangle = \langle(0, 0)\rangle$ and so it does not generate a distinct isogeny.

To account for this imbalance, we assign weights on the edges of the graph; in Figure 4.1, all edges have weight 1, except for the edge $E_{77} \to E_{96}$ which has weight 2.

For general dimension g and degree N, Florit and Smith [155] show that automorphisms play an important role in the structure of the graph $\mathcal{X}_g(\overline{\mathbb{F}}_p, N)$. Indeed, automorphisms are the source of vertices where the isogeny graph is directed, which in turn results in non-trivial weights on edges in the neighbourhood of this vertex.

Keeping the previous example in mind, we give a more precise definition of the superspecial isogeny graph.

Definition 4.2.2. Fix a positive integer g and a prime p. The *superspecial* $(N, \ldots, N)$*-isogeny graph*, denoted $\mathcal{X}_g(\overline{\mathbb{F}}_p, N)$, is the directed weighted multigraph defined as follows.

- The vertices are isomomorphism classes of superspecial p.p. abelian varieties of dimension g defined over $\overline{\mathbb{F}}_p$, denoted $\mathcal{S}_g(\overline{\mathbb{F}}_p)$.
- The edges are isomorphism classes of $(N, \ldots, N)$-isogenies, weighted by the number of distinct kernels yielding isogenies in the class. We denote the weight of an edge φ by $w(\varphi)$.

Since every superspecial p.p. abelian variety admits a model over $\mathbb{F}_{p^2}$, the set $\mathcal{S}_g(\overline{\mathbb{F}}_p)$ is finite. In particular, $\#\mathcal{S}_1(\overline{\mathbb{F}}_p) = O(p)$ and $\#\mathcal{S}_2(\overline{\mathbb{F}}_p) = O(p^3)$. Define the *out-degree* of a vertex $A \in \mathcal{S}_g(\overline{\mathbb{F}}_p)$ to be the total sum of the weights of edges leaving A. The graph $\mathcal{X}_g(\overline{\mathbb{F}}_p, N)$ is regular, meaning that every vertex has the same out-degree. Jordan and Zaytman [194] showed further that $\mathcal{X}_g(\overline{\mathbb{F}}_p, N)$ is connected, i.e., there exists a path between any two vertices in the graph. We remark that a proof of this was already given implicitly in a different language by Oort [254].

We now discuss more formally how automorphisms give rise to non-trivial weights in the isogeny graph. We first observe that every p.p. abelian variety has a non-trivial automorphism $[-1]$. However, $[-1]$ fixes every kernel and commutes with every isogeny. Therefore, it does not have an impact on the edge weights in the isogeny graph. Therefore, we can simplify by considering the *reduced automorphism group* defined as $\mathrm{RA}(A) := \mathrm{Aut}(A)/\langle\pm 1\rangle$.

Let $\varphi : A \to A/K$ be an isogeny of p.p. abelian varieties and $\alpha \in \mathrm{RA}(A)$ of degree N coprime to p. Suppose $K \neq \alpha(K)$ and let $\varphi' : A \to A/\alpha(K)$. The automorphism α induces an isomorphism $\alpha_* : A/K \xrightarrow{\sim} A/\alpha(K)$. Let $\varphi_\alpha = \alpha_*^{-1} \circ \varphi'$. Then, φ and φ_α are isogenies of degree N with the same domain and codomain, but distinct kernel. Therefore, the edge $A \to A' \cong A/K$ has non-trivial weight. On the other hand, $\widehat{\varphi}$ and $\widehat{\varphi}_\alpha$ have the same kernel, and so the automorphism $\alpha \in \mathrm{Aut}(A)$ has no impact on the weight of the edge $A' \to A$.

Example 4.2.3. To illustrate the definitions introduced above, we look more deeply at $g = 2$ and follow the example of the graph $\mathcal{X}_2(\overline{\mathbb{F}}_{11}, 2)$ due to Florit and Smith [155, Example 3.3]. There are 5 vertices (up to $\overline{\mathbb{F}}_{11}$-isomorphism):

- $A_1 = \mathcal{J}_{C_1}$, the Jacobian of $C_1 \colon y^2 = x^6 - 1$ with $\#\mathrm{RA}(A_1) = 12$,
- $A_2 = \mathcal{J}_{C_2}$, the Jacobian of $C_2 \colon y^2 = (x^3 - 1)(x^3 - 3)$ with $\#\mathrm{RA}(A_2) = 6$,
- $A_3 = E_{1728} \times E_{1728}$ with $\#\mathrm{RA}(A_3) = 16$,
- $A_4 = E_0 \times E_0$ with $\#\mathrm{RA}(A_4) = 36$,
- $A_5 = E_0 \times E_{1728}$ with $\#\mathrm{RA}(A_5) = 12$.

We illustrate this in Figure 4.3.

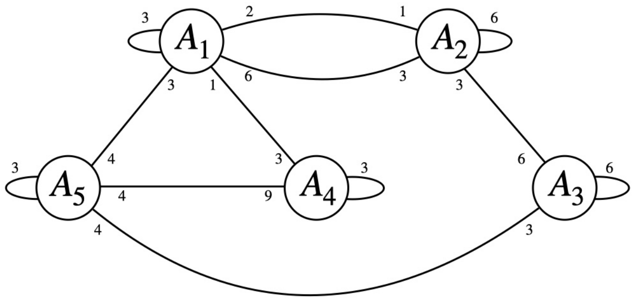


Figure 4.3: The graph $\mathcal{X}_2(\bar{\mathbb{F}}_{11}, 2)$ with weights on the edges. Note that the out-degree of each vertex is 15.

Though not apparent in the example above (due to p being very small), for cryptographic-sized primes p the number of vertices A with non-trivial $\mathrm{RA}(A)$ is negligible for low dimension cases $g = 1$ and 2. Therefore, for our purposes it suffices to treat $\mathcal{X}_g(\bar{\mathbb{F}}_p, N)$ as an undirected, regular graph.

4.2.1 The graph in dimension one

Consider the set of $\bar{\mathbb{F}}_p$-isomorphism classes of supersingular elliptic curves $\mathcal{S}_1(\bar{\mathbb{F}}_p)$ labelled by their j-invariant. For supersingular elliptic curve E we have $j(E) \in \mathbb{F}_{p^2}$, therefore $\mathcal{S}_1(\bar{\mathbb{F}}_p)$ is finite. The precise size of this set is given by the following theorem due to Igusa [189].

Theorem 4.2.4. *For $p \geq 5$, we have $\#\mathcal{S}_1(\bar{\mathbb{F}}_p) = \lfloor p/12 \rfloor + b$ where*

$$b = \begin{cases} 0 \ \textit{if}\ p \equiv 1 \bmod 12 \\ 1 \ \textit{if}\ p \equiv 5, 7 \bmod 12 \\ 2 \ \textit{if}\ p \equiv 11 \bmod 12 \end{cases}.$$

Proof. We follow the proof of [292, Theorem V.4.1(c)]. For $p \geq 5$, every elliptic curve is isomorphic to one defined by the equation $E \colon y^2 = x(x - 1)(x - \alpha)$, where $\alpha \in \bar{\mathbb{F}}_p$ with $\alpha \neq 0, 1$. By [292, Theorem V.4.1(b)], we see that E is supersingular if and only if $H_p(\alpha) = 0$ where

$$H_p(t) = \sum_{i=0}^{(p-1)/2} \binom{m}{i}^2 t^i.$$

By direct computation, we find that the roots of $H_p(t)$ are distinct, and each root α gives a supersingular elliptic curve E_α. It remains to determine which of the resulting E_α are isomorphic. The j-invariant map

$$\alpha \mapsto j(E_\alpha)$$

is six-to-one except over $j = 0$ and $j = 1728$ where it is two-to-one and three-to-one, respectively [292, Theorem III.1.7]. Furthermore, if $H_p(\alpha) = 0$, then for every α' satisfying $j(\alpha) = j(\alpha')$ we have $H_p(\alpha') = 0$, since $E_\alpha \cong E_{\alpha'}$. Defining

$$\delta_p(j) := \begin{cases} 1, \text{ if an elliptic curve } E \text{ with } j = j(E) \text{ is supersingular} \\ 0, \text{ if an elliptic curve } E \text{ with } j = j(E) \text{ is ordinary} \end{cases},$$

we have $\#\mathcal{S}_1(\bar{\mathbb{F}}_p) = \frac{1}{6}\left(\frac{p-1}{2} - 2\delta_p(0) - 3\delta_p(1728)\right) = \frac{p-1}{12} - \frac{1}{3}\delta_p(0) - \frac{1}{2}\delta_p(1728)$. We compute that

$$\delta_p(0) = \begin{cases} 1, \text{ if } p \equiv 2 \bmod 3 \\ 0, \text{ if } p \equiv 1 \bmod 3 \end{cases} \quad \text{and} \quad \delta_p(1728) = \begin{cases} 1, \text{ if } p \equiv 3 \bmod 4 \\ 0, \text{ if } p \equiv 1 \bmod 4 \end{cases}.$$

For example, see [292, Example V.4.4] and [292, Example V.4.5], respectively. Taking the four possibilities for p mod 12, we get the statement of the theorem. □

For $p > 3$, if $E/\bar{\mathbb{F}}_p$ has $j(E) \neq 0, 1728$ then $\#\mathrm{RA}(E) = 0$ [292, Theorem III.10.1]. Therefore, the only elliptic curves that have non-trivial reduced automorphism groups are those with j-invariant 0 or 1728. The proof of Theorem 4.2.4 shows that the values of p mod 12 also indicate which supersingular isogeny graphs may have vertices $j(E)$ with non-trivial $\mathrm{RA}(E)$. Consider first $E_0 : y^2 = x^3 + 1$ with $j(E_0) = 0$. The curve E_0 has an extra automorphism $(x, y) \mapsto (\omega^2 x, -y)$, where ω is a primitive third root of unity, and so $\mathrm{RA}(E_0) \cong \mathbb{Z}/3\mathbb{Z}$. Similarly, recall from Example 4.2.1, that $E_{1728} : y^2 = x^3 + x$ with $j(E_{1728}) = 1728$ has an extra automorphism $(x, y) \mapsto (-x, iy)$ and $\mathrm{RA}(E_{1728}) \cong \mathbb{Z}/2\mathbb{Z}$. Therefore, it suffices to determine when E_0 and E_{1728} are supersingular. By the proof of Theorem 4.2.4, E_0 is supersingular if and only if $p \equiv 2 \bmod 3$ and E_{1728} if and only $p \equiv 3 \bmod 4$.

In summary, except for the vertices $j \in \{0, 1728\}$, which occur depending on the value p mod 12, and their edges, the graph $\mathcal{X}_1(\bar{\mathbb{F}}_p, N)$ is undirected. As $\#\mathcal{S}_1(\bar{\mathbb{F}}_p) = O(p)$, the 'directedness' only occurs at one or two vertices in $\mathcal{S}_1(\bar{\mathbb{F}}_p)$, and for our purposes, we can ignore it. For example, when discussing the distribution of random walks on our graph, we treat $\mathcal{X}_1(\bar{\mathbb{F}}_p, N)$ as an undirected graph, and study the distribution in a neighbourhood of these special vertices separately.

The graph $\mathcal{X}_1(\bar{\mathbb{F}}_p, N)$ is connected [232] and $D_{N,1}$-regular where $D_{N,1}$ is as defined in Equation (2.7). Furthermore, the graph $\mathcal{X}_1(\bar{\mathbb{F}}_p, N)$ has optimal expansion.

Theorem 4.2.5. *For prime $N \neq p$, the graphs $\mathcal{X}_1(\bar{\mathbb{F}}_p, N)$ are Ramanujan, i.e., the second largest eigenvalue of its adjacency matrix is smaller (in absolute value) than $2\sqrt{N}$.*

Proof. See [261, Proposition 3]. □

Theorem 4.2.5 shows that $\mathcal{X}_1(\bar{\mathbb{F}}_p, N)$ is an expander graph as $p \to \infty$. This is an important property in cryptography as it means we have fast mixing; concretely, we only have to take a relatively short walk (of length $O(\log(p))$) until we land on a near uniform random node.

With Chapter 7 in mind, we note that the set $\mathcal{S}_1(\bar{\mathbb{F}}_p)$ can be partitioned into two distinct subsets:

$$\begin{aligned} \mathcal{S}_{p^2} &:= \{j \mid j = j(E) \text{ for } E \in \mathcal{S}_1(\bar{\mathbb{F}}_p) \text{ and } j \in \mathbb{F}_{p^2} \backslash \mathbb{F}_p\}, \\ \mathcal{S}_p &:= \{j \mid j = j(E) \text{ for } E \in \mathcal{S}_1(\bar{\mathbb{F}}_p) \text{ and } j \in \mathbb{F}_p\}. \end{aligned} \tag{4.3}$$

As demonstrated by the following proposition, we have $\#\mathcal{S}_p = \widetilde{O}(p^{1/2})$.

Proposition 4.2.6. *For a prime $p > 3$ we have*

$$\#\mathcal{S}_p = \begin{cases} \frac{1}{2}h(-4p) \text{ if } p \equiv 1 \bmod 4 \\ h(-p) \text{ if } p \equiv 7 \bmod 8 \\ 2h(-p) \text{ if } p \equiv 3 \bmod 8 \end{cases},$$

where $h(d)$ is the class number of the imaginary quadratic field $\mathbb{Q}(\sqrt{d})$. In particular, $\#\mathcal{S}_p = \widetilde{O}(p^{1/2})$.

Proof. When E is a supersingular elliptic curve, Proposition 4.1.5 tells us that $\#E(\mathbb{F}_p) = p + 1$. The formula for $\#\mathcal{S}_p$ then follows from [111, Theorem 14.14] with $a = 0$.

The class number of an imaginary quadratic field $\mathbb{Q}(\sqrt{d})$ is bounded as $h(d) \leq \frac{1}{\pi}\sqrt{D}\ln(D)$ [84, Exercise 5.27]. Here $D = d$ if $d \equiv 1 \bmod 4$, and $D = 4d$, otherwise. This gives $\#\mathcal{S}_p = \widetilde{O}(p^{1/2})$. □

For Chapter 7, it will be important to understand the ratio of nodes in $\mathcal{S}_p$ to nodes visited while performing a random walk on $\mathcal{X}_1(\bar{\mathbb{F}}_p, N)$. Using the sizes of $\mathcal{S}_{p^2}$ and $\mathcal{S}_p$, and the optimal expansion of the graph, the expected number of randomly chosen elements in $\mathcal{S}_{p^2}$ we would have to sample before finding one in $\mathcal{S}_p$ is $\widetilde{O}(p^{1/2})$.

4.2.2 The graph in dimension two

We now turn to the dimension-2 superspecial isogeny graph. The vertex set $\mathcal{S}_2(\bar{\mathbb{F}}_p)$ is composed of $\bar{\mathbb{F}}_p$-isomorphism classes of superspecial p.p. abelian surfaces. As described in Section 2.5, every p.p. abelian surface is isomorphic to either the Jacobian of a curve of genus 2, or to a product of two elliptic curves. In the latter case, if the abelian surface is superspecial, then the elliptic curves will be supersingular. Therefore, $\mathcal{S}_2(\bar{\mathbb{F}}_p)$ is equal to the disjoint union of the following two sets:

$$\begin{aligned} \mathcal{J}_2(\bar{\mathbb{F}}_p) &:= \{A \in \mathcal{S}_2(\bar{\mathbb{F}}_p) \ : \ A \cong \mathrm{Jac}(C) \text{ for some genus-2 curve } C\}, \\ \mathcal{E}_2(\bar{\mathbb{F}}_p) &:= \{A \in \mathcal{S}_2(\bar{\mathbb{F}}_p) \ : \ A \cong E_1 \times E_2 \text{ for some } E_1, E_2 \in \mathcal{S}_1(\bar{\mathbb{F}}_p)\}, \end{aligned} \tag{4.4}$$

where the isomorphisms are of p.p. abelian varieties over $\bar{\mathbb{F}}_p$. As $\mathcal{S}_2(\bar{\mathbb{F}}_p)$ is finite, so are the sets $\mathcal{J}_2(\bar{\mathbb{F}}_p)$ and $\mathcal{E}_2(\bar{\mathbb{F}}_p)$.

Proposition 4.2.7. *Let $p > 5$. With notation as above, we have*

$$\#\mathcal{J}_2(\overline{\mathbb{F}}_p) = \frac{1}{2880}p^3 + O(p^2) \quad \text{and} \quad \#\mathcal{E}_2(\overline{\mathbb{F}}_p) = \frac{1}{288}p^2 + O(p).$$

Proof. We follow Castryck, Decru, and Smith [72, Proposition 2]. From Theorem 4.2.4, we know that up to $\overline{\mathbb{F}}_p$-isomorphism, the number of supersingular elliptic curves is $\frac{p-1}{12} + b$ for $b \in [0, \frac{7}{6}]$ (depending only on $p \bmod 12$). Therefore,

$$\#\mathcal{E}_2(\overline{\mathbb{F}}_p) = \frac{1}{2} \cdot \frac{p-1}{12} \cdot \frac{p-1}{12} + O(p) = \frac{p^2}{288} + O(p),$$

as required. Then, by counting the number of superspecial genus-2 curves with certain reduced automorphism groups, Ibukiyama, Katsura, and Oort [187, Theorem 3.3] show $\#\mathcal{J}_2(\overline{\mathbb{F}}_p) = \frac{1}{2880}p^3 + O(p^2)$, as required. □

The graph $\mathcal{X}_2(\overline{\mathbb{F}}_p, N)$ is $D_{N,2}$-regular where $D_{N,2}$ is as defined in Equation (2.8). Though cryptographic primitives constructed using superspecial p.p. abelian surfaces, such as the Castryck–Decru–Smith hash function [72], assume the rapid convergence of random walks in the graphs $\mathcal{X}_2(\overline{\mathbb{F}}_p, N)$ to the uniform distribution, it is important to note that these expansion properties are not well understood. As we have discussed previously, the graphs $\mathcal{X}_2(\overline{\mathbb{F}}_p, N)$ do not fit into the definition of an expander graph as they are directed multigraphs. To understand the directed nature of the graph, recall that we must understand the reduced automorphism groups of superspecial p.p. abelian surfaces.

Proposition 4.2.8. *Let A be a superspecial p.p. abelian surface. The number of A with $\#\mathrm{RA}(A) = 0$ is $\frac{p^3}{2880} + O(p^2)$. If, however, $A \in \mathcal{E}_2(\overline{\mathbb{F}}_p)$ then $\mathbb{Z}/2\mathbb{Z} \subseteq \mathrm{RA}(A)$. The number of products A such that $\mathrm{RA}(A) = \mathbb{Z}/2\mathbb{Z}$ is $\frac{p^2}{288} + O(p)$.*

Proof. The fact that $C_2 \subseteq \mathrm{RA}(A)$ when $A = E_1 \times E_2$ follows from the fact that $\mathrm{RA}(E_1 \times E_2)$ always contains the involution $[1]_{E_1} \times [-1]_{E_2}$ [155, §5.2]. The counts follow from Ibukiyama, Katsura, and Oort [187]. □

As before, the set of surfaces A with non-trivial $\mathrm{RA}(A)$ is negligible in the vertex set $\mathcal{S}_2(\overline{\mathbb{F}}_p)$, and for our purposes it will be sufficient to simply ignore the directed nature of the graph. Florit and Smith [155, §4.1-4.2] discuss formally how this can be achieved to obtain upper bounds on the eigenvalues of these graphs to determine whether $\mathcal{X}_2(\overline{\mathbb{F}}_p, N)$ is Ramanujan.

Jordan and Zaytman [194] show that $\mathcal{X}_2(\overline{\mathbb{F}}_{11}, 2)$ from Example 4.2.3 is not Ramanujan. Florit and Smith provide evidence that the same is true for $\mathcal{X}_2(\overline{\mathbb{F}}_p, 2)$ where $11 \leq p \leq 201$, therefore suggesting that the superspecial $(2,2)$-isogeny graph fails to be Ramanujan as for larger primes p [155, Appendix A]. Despite this, Florit and Smith conjecture [155, Conjecture 4.10] that the family $\mathcal{X}_2(\overline{\mathbb{F}}_p, N)$ still has "good enough" expansion for cryptographic purposes:

Conjecture 4.2.9. *Let λ_N be the second largest eigenvalue of $\mathcal{X}_2(\overline{\mathbb{F}}_p, N)$. Then there exists a fixed $L < 1$ such that $\lambda_N \leq L$, for every prime $p \geq 5$. In particular, for every prime $p \geq 41$ we have*

$$\frac{11}{15} \leq \lambda_2 \leq \frac{12}{15}.$$

To make "good enough" more precise, Florit and Smith [155, Theorem 6.1] show that, assuming this conjecture, a random walk of length $n \geq 4.5m\log(p)+9$ approximates the stationary distribution on $\mathcal{X}_2(\overline{\mathbb{F}}_p, 2)$ with an error of $1/p^m$. More recently, Aikawa, Tanaka, and Yamauchi [2] showed that $\mathcal{X}_2(\overline{\mathbb{F}}_p, N)$ are expander graphs.

As we will see in Chapter 8, it is important to understand the ratio of nodes $A \in \mathcal{E}_2(\overline{\mathbb{F}}_p)$ to nodes visited while performing a random walk in $\mathcal{X}_2(\overline{\mathbb{F}}_p, N)$. A natural first guess would be that this ratio matches the proportion of such nodes in the entire graph, computed to be $10/p+O(1/p^2)$ using Proposition 4.2.7. However, from Proposition 4.2.8 we see that all but $O(p)$ of the products of elliptic curves have reduced automorphism group of order 2, and deduce that the expected proportion of products in a random walk is $\sim \frac{1}{2} \cdot \frac{10}{p} = \frac{5}{p}$ [155, §6.2].

4.3 The general superspecial isogeny problem

The most general problem underlying the security of isogeny-based cryptography is the *dimension-g superspecial isogeny problem*, defined as follows.

Problem 4.3.1 (The dimension-g superspecial isogeny problem)**.** Given a pair of superspecial p.p. abelian varieties A_1 and A_2 of dimension g defined over $\mathbb{F}_{p^2}$, find a $\overline{\mathbb{F}}_p$-isogeny $A_1 \to A_2$.

Alternatively, we can view the isogeny problem of dimension g as a path finding problem in the superspecial isogeny graph $\mathcal{X}_d(\overline{\mathbb{F}}_p)$.

Problem 4.3.2 (The dimension d isogeny path problem)**.** Given nodes A_1 and A_2 in $\mathcal{S}_g(\overline{\mathbb{F}}_p)$, find a path in $\mathcal{X}_d(\overline{\mathbb{F}}_p)$ connecting them.

For $g = 1$, the graphs $\mathcal{X}_1(\overline{\mathbb{F}}_p, N)$ with $p \nmid N$ are Ramanujan. Viewing Problem 4.3.2 purely as a graph theoretic problem, without any extra structure coming from the supersingular elliptic curves and their isogenies, we expect the path-finding problem to be hard to solve. In light of results due to Aikawa, Tanaka, and Yamauchi [2], we conjecture the same to be true for $\mathcal{X}_2(\overline{\mathbb{F}}_p, N)$.

4.3.1 Algorithms for dimension 1

In dimension 1, Problem 4.3.1 underpins the security of all primitives in isogeny-based cryptography constructed using supersingular elliptic curves over $\mathbb{F}_{p^2}$. It is often referred to in the literature as the general supersingular isogeny problem, or simply the general isogeny problem.

The best classical attack against this problem is due to Delfs and Galbraith [132] and runs in $\widetilde{O}(p^{1/2})$. We want to find an isogeny $\varphi\colon E \to E'$ between two supersingular curves, $E/\mathbb{F}_{p^2}$ and $E'/\mathbb{F}_{p^2}$ with j-invariants in $\mathcal{S}_{p^2}$. The key observation that leads to the Delfs–Galbraith algorithm is that it is easier to find an isogeny between *subfield curves*, i.e., elliptic curves with j-invariant in $\mathcal{S}_p$. Therefore, rather than finding the isogeny φ directly, we search for subfield curves.

More explicitly, we perform simple non-backtracking random walks in $\mathcal{X}_1(\overline{\mathbb{F}}_p, N)$ until hitting an elliptic curve with j-invariant in $\mathcal{S}_p$. Finding a walk from $E/\mathbb{F}_{p^2}$ to $E_1/\mathbb{F}_p$ yields an isogeny $\psi\colon E \to E_1$, and finding a walk from $E'/\mathbb{F}_{p^2}$ to $E_1'/\mathbb{F}_p$ yields an isogeny $\psi'\colon E' \to E_1'$. A full isogeny $\varphi\colon E \to E'$ is then found as the composition $\varphi = (\widehat{\psi'} \circ \phi \circ \psi)$, where $\phi\colon E_1 \to E_1'$ is the subfield isogeny that can be computed in $\widetilde{O}(p^{1/4})$ using [132, Algorithm 1].

From the discussion in Section 4.2, the number of elliptic curves with $j \in \mathcal{S}_{p^2}$ we expect to search over before finding one with j-invariant in $\mathcal{S}_p$ is $\widetilde{O}(p^{1/2})$. Following Delfs and Galbraith [132, Section 4], the steps taken in $\mathcal{X}_1(\bar{\mathbb{F}}_p, N)$ are non-backtracking, meaning that one stores the current j-invariant, j_c, and the previous j-invariant j_p. To take the next step, one then chooses one of the $D_{N,1} - 1$ roots of

$$\Phi_N(X, j_c)/(X - j_p)$$

at random, where Φ_N is the modular polynomial of level-N as defined in Section 2.8.1.1. Since N and $D_{N,1}$ are fixed and small, it follows that the asymptotic complexity of the search for subfield j-invariants is $\widetilde{O}(p^{1/2})$. Therefore, the bottleneck in the Delfs–Galbraith algorithm is finding the paths from curves with $j \in \mathcal{S}_{p^2} \setminus \mathcal{S}_p$ to the curves with $j \in \mathcal{S}_p$. In Chapter 7, we delve into the Delfs–Galbraith algorithm in greater detail and improve its concrete complexity.

The best quantum attack, due to Biasse, Jao, and Sankar [33], runs in $\widetilde{O}(p^{1/4})$. The authors adapt the Delfs–Galbraith algorithm in the context of quantum computation using the techniques of Childs, Jao, and Soukharev [82]. More explicitly, by recasting the first step of finding an isogeny between E and E_1 as a search problem and applying Grover's algorithm, they achieve a quadratic speed-up.

4.3.2 Algorithms for dimension $g \geq 2$

For $g \geq 2$, the best classical and quantum attacks are due to Costello and Smith [107], and they run in $\widetilde{O}(p^{g-1})$ [107, Theorem 1] and $\widetilde{O}(\sqrt{p^{g-1}})$ [107, Theorem 2], respectively.

We will depict the algorithm for $g = 2$, as this will be sufficient for our purposes. In this case, Costello and Smith exploit the fact that finding isogenies between products of elliptic curves is easier than the general problem. Therefore, rather than finding an isogeny between abelian surfaces A and $A' \in \mathcal{J}_2(\bar{\mathbb{F}}_p)$ directly, we search for surfaces in $\mathcal{E}_2(\bar{\mathbb{F}}_p)$.

Given two $\bar{\mathbb{F}}_p$-isomorphism classes of p.p. abelian surfaces A and $A' \in \mathcal{J}_2(\bar{\mathbb{F}}_p)$, we find paths $\varphi\colon A \to E_1 \times E_2$ and $\varphi'\colon A' \to E_1' \times E_2'$, where $E_1 \times E_2, E_1' \times E_2' \in \mathcal{E}_2(\bar{\mathbb{F}}_p)$. As $\#\mathcal{J}_2(\bar{\mathbb{F}}_p) = O(p^3)$ and $\#\mathcal{E}_2(\bar{\mathbb{F}}_p) = O(p^2)$, we expect to complete both of these steps using $\widetilde{O}(p)$ operations. We then solve the dimension-1 isogeny problem using the Delfs–Galbraith algorithm on input E_1 and E_1' and on input E_2 and E_2' to output the paths $\psi_1\colon E_1 \to E_1'$ and $\psi_2\colon E_2 \to E_2'$ in $\mathcal{X}_1(\bar{\mathbb{F}}_p)$. Both of these steps terminate in $\widetilde{O}(p^{1/2})$.

If $\mathrm{length}(\psi_1) \equiv \mathrm{length}(\psi_2) \bmod 2$, we can use these to construct a product path $\pi\colon E_1 \times E_2 \to E_1' \times E_2'$, as described in [107, Lemma 3]. Otherwise, we fail and return $\perp$. The desired path between $\mathcal{A}$ and $\mathcal{A}'$ is then $\phi := \widehat{\varphi'} \circ \pi \circ \varphi$.

Remark 4.3.3. Only three runs of this procedure are required to successfully return path ϕ. Indeed, if we instead run the algorithm to find paths $\psi_1 : E_1 \to E_1'$, $\psi_{2,1} : E_2 \to E$, and $\psi_{2,2} : E \to E_2'$, where $E : y^2 = x^3 + x$ has an endomorphism of degree 2, say τ, then we can set $\psi_2 = \psi_{2,2} \circ \psi_{2,1}$ if $\mathrm{length}(\psi_1) \equiv \mathrm{length}(\psi_{2,1} \circ \psi_{2,2}) \bmod 2$ and $\psi_2 = \psi_{2,2} \circ \tau \circ \psi_{2,1}$, otherwise.

Overall, the cost of the algorithm is $\widetilde{O}(p)$ bit operations. Chapter 8 will focus on the Costello–Smith algorithm for $g = 2$, and improve its concrete complexity. Applying Grover's algorithm analogously to the dimension-1 case, we obtain a quantum algorithm that achieves a quadratic speed-up in runtime.

Setting our prime to be of size $p \approx 2^\lambda$ where λ is the security parameter, we see that the dimension-g superspecial isogeny problem is a good candidate for a cryptographic hard problem.

4.3.3 The isogeny problem with fixed degree

In many cryptographic schemes, additional information is often known and the security of the schemes is based on variants of Problem 4.3.1. A common restriction is requiring the solution to have a specific degree (see for example [19, Problem 7]). The guaranteed existence of a solution satisfying this additional constraint supplies more information about the problem to the attacker, which may make it easier to solve.

Problem 4.3.4 (The dimension-g superspecial N-isogeny problem)**.** Given a pair of isogenous superspecial p.p. abelian varieties A_1 and A_2 of dimension g defined over $\mathbb{F}_{p^2}$, find a $\bar{\mathbb{F}}_p$-isogeny $A_1 \to A_2$ of degree N.

We first consider the case $g = 1$. A common classical attack strategy is the generic meet-in-the-middle attack. For simplicity of exposition, we assume we are given supersingular elliptic curves $E_1, E_2/\mathbb{F}_{p^2}$ connected by an isogeny of degree N^k in $\mathcal{X}_1(\bar{\mathbb{F}}_p)$ for some prime N. The more general case, where the isogeny has smooth degree, follows similarly. This attack proceeds as follows. Start by tabulating all possible walks of length $\lfloor k/2 \rfloor$ starting from E_1. Then, iterate over the walks of length $\lceil k/2 \rceil$ starting from E_2, until finding a collision. We expect to find a collision, and therefore the isogeny $\varphi : E_1 \to E_2$, in $O(N^{k/2})$ and with $O(N^{k/2})$ storage. Regarding quantum algorithms, Tani's claw finding algorithm [305] solves Problem 4.3.4 for sufficiently smooth degrees N. However, this algorithm assumes an unrealistic cost of accessing quantum memory, which makes this algorithm more expensive than its classical counterpart, as argued by Jaques and Schanck [192].

Letting the degree of the isogeny be $p^{1/2+\epsilon}$ for $1/2 \leq \epsilon \leq 3/4$, Bencina, Kutas, Merz, Petit, Stopar, and Weitkämper [22] exhibit a new classical attack, which outperforms meet-in-the-middle. The authors also improve the best quantum attack in the range $0 < \epsilon < 5/2$. Furthermore, both these algorithms are essentially memory-free.

How the hardness of this problem relates to the general supersingular isogeny problem for dimensions $g > 1$ is unknown.

4.3.4 The isogeny problem as an endomorphism ring problem

The endomorphism ring problem asks, given a supersingular elliptic curve E, to compute its endomorphism ring $\mathrm{End}(E)$. Wesolowski [318] showed that the endomorphism ring problem and the dimension-1 supersingular isogeny problem are equivalent under reductions of polynomial expected time, assuming the generalised Riemann hypothesis (GRH), improving upon known reductions that relied on a variety of heuristic assumptions [148]. Finding efficient algorithms to compute the endomorphism ring of a supersingular elliptic curve has been the focus of several works [14, 148, 169, 204]. We highlight in particular, the work by Eisenträger, Hallgren, Leonardi, Morrison, and Park [148] which gives an algorithm for computing the endomorphism ring of a supersingular curve that terminates in time $O((\log p)^2 p^{1/2}) = \widetilde{O}(p^{1/2})$ (under certain heuristics). Fuselier, Iezzi, Kozek, Morrison, and Namoijam [169] remove these heuristics and obtain an algorithm running in

$O((\log p)^2(\log\log p)^3p^{1/2}) = \widetilde{O}(p^{1/2})$. Due to the result by Wesolowski, any improvements to these algorithms will immediately impact the security of the dimension 1 supersingular isogeny problem.

Chapter 5

Quaternions in cryptography

This chapter explores how the Deuring correspondence, introduced in Section 3.2, can be used in cryptography. We begin by translating the general isogeny problem to a problem involving ideals and maximal orders in a quaternion algebra. Interestingly, this quaternionic problem can be solved efficiently in polynomial time using the *KLPT algorithm*. Though the KLPT algorithm was first developed for cryptanalysis, it has since found many constructive applications. Most notably, SQIsign, which we introduce in Section 5.2, is an isogeny-based signature scheme built using the Deuring correspondence and (variants of) the KLPT algorithm.

As before, throughout this chapter, we fix $\Bbbk$ to be a finite field $\mathbb{F}_q$ of prime characteristic $p > 5$.

5.1 The isogeny problem as a quaternion problem

Using the Deuring correspondence, we can rephrase the isogeny problem as a problem involving maximal orders and ideals in a quaternion algebra.

Problem 5.1.1 (Quaternion $\mathcal{N}$-isogeny path problem)**.** Let $\mathcal{N} \subset \mathbb{N}$. Given a maximal order $\mathcal{O}$ and a left $\mathcal{O}$-ideal I, find $J \sim I$ of norm in $\mathcal{N}$. When $\mathcal{N} = \{N\}$ we recover Section 4.3.3.

5.1.1 The KLPT algorithm

In their seminal 2014 paper, Kohel, Lauter, Petit, and Tignol [203] give a probabilistic algorithm (usually referred to as the KLPT algorithm) which solves the quaternion $\mathcal{N}$-isogeny path problem when $\mathcal{N} = \{\ell^e : \ell \text{ prime}, e \in \mathbb{N} \text{ where } e > \frac{7}{2}\log(p) + \epsilon \text{ for } \epsilon > 0\}$ in expected polynomial time, subject to reasonable heuristics on expected distributions of primes. A variant of this algorithm where $\mathcal{N}$ includes powersmooth norms was also introduced in the original KLPT paper [203, §4.7], and described in detail by Galbraith, Petit, and Silva [174, §4.3]. The general case is treated by Leroux [217]. For an excellent exposition of the different KLPT-like algorithms, we refer to [217].

Though it is a purely quaternionic algorithm, it has seen a variety of applications in isogeny-based cryptography due to the Deuring correspondence. A variant of the KLPT algorithm which provably terminates in polynomial time (assuming GRH) was the core to showing the equivalence of the endomorphism ring problem and the isogeny problem [318]. Other versions of the algorithm have also shown to be a key algorithmic tool in constructive applications, such as the GPS signature [174] and SQIsign [125, 126], amongst others.

At a high-level, to solve Problem 5.1.1, we need to find the solution to norm equations in ideals. The brilliant idea behind the KLPT algorithm is the observation that finding solutions to norm equations in ideals of *special extremal orders* is easier. Special extremal orders are defined generally in, for example, [203, §2.3]. For our purposes, it will suffice to know the following example.

Example 5.1.2. Consider $p \equiv 3 \bmod 4$. In this case, we have the special extremal order $\mathcal{O}_0 = \langle 1, i, \frac{i+j}{2}, \frac{1+k}{2} \rangle$ with $i^2 = -1, j^2 = -p$ and $k = ij$. The order $\mathcal{O}_0$ is isomorphic to the endomorphism ring $\operatorname{End}(E_0)$ of the elliptic curve with j-invariant 1728 (see Example 3.2.1). This special order will play an important role throughout this thesis, and we therefore fix the notation $\mathcal{O}_0$ and its corresponding elliptic curve E_0. Examples of special extremal orders for other primes appear in [203].

Remark 5.1.3. When applying the KLPT algorithm throughout this thesis, we set $p \equiv 3 \bmod 4$ so that we can fix our special extremal order to be $\mathcal{O}_0$ as in Example 5.1.2.

Given a $\mathcal{O}_0$-ideal I and some fixed set $\mathcal{N}$, the KLPT algorithm finds an equivalent ideal $J \sim I$ of norm in $\mathcal{N}$. For a $\mathcal{O}_0$-ideal I and $\beta \in I\backslash\{0\}$, the map given in Equation (3.5) outputs an equivalent ideal $J \sim I$ of norm $\operatorname{nrd}(J) = \operatorname{nrd}(\beta)/\operatorname{nrd}(I)$. So, it suffices to find a non-zero element $\beta \in I$ with norm in $\operatorname{nrd}(I)\mathcal{N} := \{\operatorname{nrd}(I)n \colon n \in \mathcal{N}\}$. Therefore, the main goal of the KLPT algorithm is to solve the norm equation $\operatorname{nrd}(\beta) = \operatorname{nrd}(I)n$ for $n \in \mathcal{N}$ and $\beta \in I$. An in-depth understanding of how to find this element β will not be needed for the work in this thesis, and we refer a reader to [203] or [217, Chapter 3] for more details.

Though the original KLPT algorithm is presented as a probabilistic algorithm, Leroux describes how it can be made deterministic by fixing an order on the random choices made (see [217, Remark 3.2.8]). For the purposes of this thesis, it will be sufficient to know that KLPT algorithm is an efficient deterministic algorithm and to know the size of its output, as given by the following theorem.

Theorem 5.1.4. *The KLPT algorithm on input I and $\mathcal{N} = \{\ell^e : \ell$ prime, $e \in \mathbb{N}\}$ is a deterministic algorithm which runs in heuristic polynomial time in $\log(p)$ and $\log(\operatorname{nrd} I)$ (under certain heuristic assumptions). Furthermore, the norm of the output ideal J satisfies $\log(\operatorname{nrd} J) \approx \frac{7}{2}\log_\ell(p)$.*

Proof. This is given by, for example, Leroux [217, Proposition 3.2.7]. For more details on the heuristic assumptions, see [203]. □

An improvement due to Petit and Smith [260] allows us to reduce the size of the output J to $\log(\operatorname{nrd} J) \approx 3\log_\ell(p)$. We usually take $\ell = 2$.

Remark 5.1.5. Looking at Theorem 5.1.4, we remark that a big drawback is that the norm of the output ideal will be large compared to the norm of the optimal solution (which can a priori be estimated as $O(p)$). In fact, it is a large source of inefficiency in constructive applications of the KLPT algorithm, as the norm of the output ideal directly corresponds to the degree of an isogeny that needs to be computed.

Kohel, Lauter, Petit, and Tignol [203] show that their algorithm can be extended to work for a $\mathcal{O}$-ideal where $\mathcal{O}$ is not necessarily a special extremal order. Take a connecting $(\mathcal{O}_1, \mathcal{O}_2)$-ideal I, where neither $\mathcal{O}_1$ nor $\mathcal{O}_2$ are special extremal. To find an equivalent ideal $J \sim I$, it suffices to find $J_i \sim I_i$ where I_i is a connecting $(\mathcal{O}_0, \mathcal{O}_i)$-ideal, for $i = 1, 2$. Note that I_i exists because all maximal orders are connected,[1] and the J_i can be computed using the KLPT algorithm.

While this algorithm is satisfactory for cryptanalysis, the norm of the output ideal is too large to be used for constructive purposes. The original version of the KLPT algorithm was used to construct the GPS signature [174], but it proved to be impractical. This leads to a new generic KLPT algorithm introduced by De Feo, Kohel, Leroux, Petit, and Wesolowski [125], and later improved by De Feo, Leroux, Longa, and Wesolowski [126], with the main objective of constructing SQIsign, a practical signature scheme. For any maximal order $\mathcal{O}$, given a $\mathcal{O}$-ideal I and a $(\mathcal{O}_0, \mathcal{O})$-ideal K of prime norm coprime to $\operatorname{nrd}(I)$, the generic KLPT algorithm finds an equivalent ideal $J \sim I$ of norm in $\mathcal{N}$. It does so by solving norm equations in the order $\mathcal{O} \cap \mathcal{O}_0 = \mathbb{Z} + I$.[2] For more algorithmic details see [217, §3.3]; for our purposes it is sufficient to understand the runtime and output size of the generic KLPT algorithm.

Proposition 5.1.6. *On input I, a left $\mathcal{O}$-ideal and $\mathcal{N} = \{\ell^e : \ell \text{ prime}, e \in \mathbb{N}\}$, the generic KLPT algorithm is a deterministic algorithm which runs in heuristic polynomial time in $\log(p)$, $\log(\operatorname{nrd} I)$ and the bound on coefficients of a basis for $\mathcal{O}$. Furthermore, the norm $\operatorname{nrd}(J)$ of the output ideal J satisfies $\log_\ell(\operatorname{nrd} J) \approx \frac{15}{4}\log_\ell(p)$.*

Proof. This follows from [217, §3.3]. □

5.2 SQIsign: a signature scheme from isogenies

We now describe the isogeny-based signature scheme SQIsign. SQIsign was introduced by De Feo, Kohel, Leroux, Petit, and Wesolowski [125], and later improved by De Feo, Leroux, Longa, and Wesolowski [126]. In June 2023, it was submitted to NIST's alternate call for post-quantum secure signature schemes [78].

SQIsign is constructed from a Σ-protocol that proves knowledge of a secret (non-scalar) endomorphism $\alpha \in \operatorname{End}(E)$ for some public curve E. At its core, the prover shows this knowledge by being able to compute an isogeny ϕ from E to some random curve E'.

5.2.1 Identification Protocol

Following the blueprint laid out in Sections 1.1.3 and 1.1.4, we construct an identification protocol for the following relation

$$(E, \alpha) \in R \iff \alpha \text{ is a cyclic smooth endomorphism of } E. \tag{5.1}$$

This relation is *hard*, as per Definition 1.1.12, assuming the hardness of the supersingular smooth endomorphism problem.

[1] This follows from the fact that all maximal orders have the same *genus*, as discussed by Leroux [217, pg. 34] or Voight [316, §17.1].

[2] Such an order $\mathcal{O} \cap \mathcal{O}_0$ is called an *Eichler order* as defined in [316, Definition 23.4.1], and the a proof of the equality with $\mathbb{Z} + I$ can be found in [125, Proposition 1].

Problem 5.2.1. The *supersingular smooth endomorphism problem* asks, given a supersingular elliptic curve E defined over $\mathbb{F}_{p^2}$, to find a (non-trivial) cyclic endomorphism of E of smooth degree.

Random endomorphisms in E can be constructed by taking a random walk $E \to E'$, then finding a non-zero cyclic endomorphism of E'. Adapting the algorithm by Page and Wesolowski [257], we can reduce the endomorphism ring problem to Problem 5.2.1. In Section 4.3, we saw the equivalence of the endomorphism ring problem and the dimension-1 supersingular isogeny problem, whose best classical attack runs in $\widetilde{O}(p^{1/2})$ and best quantum attack runs in $\widetilde{O}(p^{1/4})$. Based on this evidence, the relation R is a hard relation (as defined in Definition 1.1.12).

We first give a high-level description of the identification protocol (constructed as a Σ-protocol for the hard relation R) between a prover P and verifier V below. Fix a prime number $p \approx 2^\lambda$, for security parameter λ, and supersingular elliptic curve $E_0/\mathbb{F}_{p^2}$ with known endomorphism ring $\mathcal{O}_0$. Fix $e \in \mathbb{N}$. The ID scheme proceeds as follows:

- $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{Gen}(1^\lambda)$: Compute an $\varphi_{\text{sk}} : E_0 \to E_{\text{pk}}$, with corresponding public key E_{pk}. Output key pair $\mathsf{pk} := E_{\text{pk}}$ and $\mathsf{sk} := \varphi_{\text{sk}}$.
- $\mathsf{P}(\mathsf{pk}, \mathsf{sk}) \leftrightharpoons \mathsf{V}(\mathsf{pk})$: A 3-move interactive protocol between P and V defined as follows.
  1. P generates a random commitment isogeny $\varphi_{\text{com}} : E_0 \to E_{\text{com}}$ of smooth degree D_{com}, and sends E_{com} to V.
  2. V computes a random challenge isogeny $\varphi_{\text{chall}} : E_{\text{com}} \to E_{\text{chall}}$ of smooth degree D_{chall}, and sends φ_{chall} to P.
  3. P uses the knowledge of φ_{com} and φ_{chall} to compute the response isogeny $\varphi_{\text{resp}} : E_{\text{pk}} \to E_{\text{chall}}$ of degree 2^e.

  V outputs `accept` if φ_{resp} is an isogeny from E_{pk} to E_{chall} of degree 2^e, and $\widehat{\varphi_{\text{chall}}} \circ \varphi_{\text{resp}}$ is cyclic. Otherwise, V outputs `reject`.

$$\begin{tikzcd}
E_0 \arrow[r, "\varphi_{\text{com}}"'] \arrow[d, dashed, "\varphi_{\text{sk}}"] & E_{\text{com}} \arrow[d, "\varphi_{\text{chall}}"'] \\
E_{\text{pk}} \arrow[r, "\varphi_{\text{resp}}"] & E_{\text{chall}}
\end{tikzcd}$$

FIGURE 5.1: The SQIsign protocol with three phases: commitment φ_{com}, challenge φ_{chall}, and response φ_{resp}.

We now describe each phase in more detail. For the setup, we take $p \equiv 3 \bmod 4$ and use the curve $E_0 : y^2 = x^3 + x$, as in Example 3.2.1.

Key Generation. Sample a random secret left $\mathcal{O}_0$-ideal I_{sk} of secret prime norm D_{sk}. This ideal will be used by P in the response phase. Next, we compute an equivalent ideal $J_{\text{sk}} \sim I_{\text{sk}}$ whose norm is a power-of-2 using $\mathsf{KeyGenKLPT}$, a variant of the original KLPT algorithm from Theorem 5.1.4 which follows a similar structure but is specialised to the application of key generation. For more details, we refer to Algorithm 14 in [78]. Translate J_{sk} to the corresponding isogeny $\varphi_{\text{sk}} : E_0 \to E_{\text{pk}}$ using the ideal-to-isogeny procedure from Section 3.2.1. The prover P outputs $\mathsf{sk} := (I_{\text{sk}}, J_{\text{sk}}, \varphi_{\text{sk}})$ and $\mathsf{pk} := E_{\text{pk}}$.

Commitment, Challenge, Response. We first compute the commitment ideal I_{com} of norm D_{com} and the corresponding isogeny $\varphi_{\text{com}} : E_0 \to E_{\text{com}}$, using an execution of the algorithm $\mathsf{IdealToIsogeny}_{D_{\text{com}}}$ (Algorithm 3.1). After receiving the challenge isogeny φ_{chall}, we use the isogeny-to-ideal routine from Section 3.2.3 to obtain the corresponding ideal I_{chall}. The response phase proceeds as follows. We construct an equivalent ideal $J \sim \overline{I_{\text{sk}}} \cdot I_{\text{com}} \cdot I_{\text{chall}}$ of norm 2^e using $\mathsf{SigningKLPT}$. The algorithm $\mathsf{SigningKLPT}$ is a variant of the generic KLPT algorithm from Proposition 5.1.6 (with $\ell = 2$), stemming from security considerations: (a) the output in $\mathsf{SigningKLPT}$ has constant norm 2^e (so that it does not depend on the secret); and (b) randomisation is introduced into $\mathsf{SigningKLPT}$ to ensure a good distribution of the output ideals. For more details, we refer to Algorithm 17 in [78]. We fix $e \approx \frac{15}{4}\log(p)$ in the setup, as this is the expected output size of the generic KLPT algorithm. Explicitly, the NIST submission takes $e = \lceil \frac{15}{4}\log(p) \rceil + 25$ [78, §7.2.3]. This procedure is repeated until J corresponds to a cyclic isogeny.[3] Afterwards, the prover P computes the corresponding isogeny $\varphi_{\text{resp}} : E_{\text{pk}} \to E_{\text{chall}}$ with the ideal-to-isogeny algorithm from Section 3.2.2, using knowledge of an isogeny $\varphi_{\text{sk}} : E_0 \to E_{\text{pk}}$ and ideal J_{sk}. Finally, P outputs response φ_{resp}.

Verification. The main bottleneck of verification is the computation of the response isogeny φ_{resp}, which has degree 2^e where $e \approx \frac{15}{4}\log(p)$. For example, for NIST Level I security we have $e \approx 900$. To ensure isogeny computations remain over $\mathbb{F}_{p^2}$, we decompose φ_{resp} as $\varphi_g \circ \cdots \circ \varphi_1$, where each φ_i is of degree (dividing) 2^f, $g = \lceil e/f \rceil$, and $f \in \mathbb{N}$ is maximal such that $2^f \mid p^2 - 1$. We delay a more detailed discussion of verification to Chapter 11, where we explore new ways to accelerate it.

Correctness follows straightforwardly from the correctness of the algorithms involved in the key generation and interactive protocol. Most notably, the correctness of $\mathsf{SigningKLPT}$ follows similarly to the correctness of the generic KLPT algorithm in Proposition 5.1.6. For more details see [217, Proposition 5.3.5].

5.2.1.1 Special soundness

We show that the underlying Σ-protocol is special sound, as defined in Definition 1.1.9. We follow De Feo, Kohel, Leroux, Petit, and Wesolowski [125, §3.2].

Proposition 5.2.2. *Assuming the hardness of Problem 5.2.1 (when E is restricted to be the public key of the ID scheme), the identification scheme described above is special sound.*

Proof. Suppose we are given two transcripts $(D_{\text{com}}, \varphi_{\text{chall}}, \varphi_{\text{resp}})$ and $(D_{\text{com}}, \varphi'_{\text{chall}}, \varphi'_{\text{resp}})$. If $\varphi_{\text{chall}} \neq \varphi'_{\text{chall}}$ then we construct $\alpha = \widehat{\varphi'_{\text{resp}}} \circ \varphi'_{\text{chall}} \circ \widehat{\varphi_{\text{chall}}} \circ \varphi_{\text{resp}} : E_{\text{pk}} \to E_{\text{pk}}$. As $\widehat{\varphi_{\text{chall}}} \circ \varphi_{\text{resp}}$ and $\widehat{\varphi'_{\text{chall}}} \circ \varphi'_{\text{resp}}$ are non-cyclic by [125, Lemma 2], α is a non-scalar endomorphism of E_{pk} of smooth degree, and is therefore a witness for the relation in Equation (5.1), where E a public key of the ID scheme. As $\varphi_{\text{chall}} = \varphi'_{\text{chall}}$ with negligible probability, we get the statement of the theorem. $\square$

[3]The left $\mathcal{O}$-ideal J corresponds to a cyclic isogeny if for all primes N we have $J \not\subseteq N\mathcal{O}$.

We remark that to reduce the general isogeny problem to the smooth endomorphism problem in Problem 5.2.1 when E is restricted to be the public key of the ID scheme, we must assume that the public keys are computationally indistinguishable from the uniform distribution on supersingular elliptic curves [257, §8.2].

5.2.1.2 Honest verifier zero-knowledge

Proving honest verifier zero-knowledge is harder and relies on the output distribution of the $\mathsf{SigningKLPT}$. Indeed, the KLPT algorithm is needed for computing the response: although setting $\varphi_{\text{resp}} = \varphi_{\text{chall}} \circ \varphi_{\text{com}} \circ \widehat{\varphi_{\text{sk}}}$ gives an isogeny from E_{pk} to E_{chall}, this leaks the secret φ_{sk}. Further, this is not a valid response, since the composition with $\widehat{\varphi_{\text{chall}}}$ is not cyclic. More precisely, zero-knowledge relies on the assumption that the distribution of signatures is computationally indistinguishable from random isogenies of the same degree from the curve E_{pk}. This is a new, more ad-hoc assumption, see Problem 14 in [126]. We omit an in-depth discussion on zero-knowledge as it will not be necessary for this thesis, and refer an interested reader to the original $\mathsf{SQIsign}$ articles [125, 126]. We simply remark that there have, thus far, been no specialised attacks that exploit this newer assumption that are better than simply recovering the secret isogeny [125, Appendix B].

5.2.2 SQIsign: the signature scheme

We now detail how the identification protocol of Section 5.2.1 can be transformed into a signature scheme using the Fiat–Shamir transform, as detailed in Section 1.1.4. The resulting signature scheme is called $\mathsf{SQIsign}$.

We first require a cryptographic hash function that outputs an isogeny of degree $D = \prod_{i=1}^{n} \ell_i^{e_i}$. It is constructed as follows. Let $\mathsf{H} : \{0,1\}^* \to [1, \mu(D)]$ be a cryptographic hash function, where $\mu(D) = \prod_{i=1}^{n} \ell_i^{e_i - 1}(\ell_i + 1)$. From the output of H, say $s \in [1, \mu(D)]$, we use the function $F_D(E, s)$ to map s to non-backtracking sequences of isogenies of total degree D starting at E. This function is a generalisation of the CGL hash function [76] given by De Feo, Saint Guilhem, Fouotsa, Kutas, Leroux, Petit, Silva, and Wesolowski [128, §3.1].

The signature scheme $\mathsf{SQIsign}$ is defined as the triple $(\mathsf{KGen}, \mathsf{Sign}, \mathsf{Verify})$ of polynomial time algorithms defined (at a high-level) as follows. For the setup, fix $p \equiv 3 \bmod 4$, supersingular curve $E_0 : y^2 = x^3 + x$, and $e \in \mathbb{N}$ with $e \approx \frac{15}{4} \log(p)$.

- $(\mathsf{vk}, \mathsf{sk}) \overset{\$}{\leftarrow} \mathsf{KGen}(1^\lambda)$: Fix a cryptographic hash function $\mathsf{H} : \{0,1\}^* \to [1, \mu(D_{\text{chall}})]$. Run the key generation algorithm $(E_{\text{pk}}, \varphi_{\text{sk}}) \leftarrow \mathsf{Gen}(1^\lambda)$ of the underlying ID scheme, and output verification key $\mathsf{vk} := (\mathsf{H}, E_{\text{pk}})$ and signing key $\mathsf{sk} := (\varphi_{\text{sk}}, \mathsf{H}, E_{\text{pk}})$.

- $\sigma \overset{\$}{\leftarrow} \mathsf{Sign}(\mathsf{sk}, m)$: The signing algorithm performs a run of the underlying Σ-protocol by parsing sk as $(\varphi_{\text{sk}}, \mathsf{H}, E_{\text{pk}})$ and computing the commitment $\varphi_{\text{com}} : E_0 \to E_{\text{com}}$. Then, it computes the challenge φ_{chall} of degree D_{chall} by running $F_{D_{\text{chall}}}(E_{\text{com}}, s)$ where $s = \mathsf{H}(j(E_{\text{com}}), m)$. Finally, it computes the response isogeny φ_{resp} as in the ID scheme, and returns $\sigma := (E_{\text{com}}, \varphi_{\text{resp}})$ as a signature on m.

- $\texttt{accept/reject} \leftarrow \mathsf{Verify}(\mathsf{vk}, \sigma, m)$: The verification algorithm parses vk as $(\mathsf{H}, E_{\text{pk}})$ and σ as $(E_{\text{com}}, \varphi_{\text{resp}})$, and uses $s' = \mathsf{H}(j(E_{\text{com}}), m)$ to compute $\varphi'_{\text{chall}} = F_{D_{\text{chall}}}(E_{\text{com}}, s') : E_{\text{com}} \to E'_{\text{chall}}$. It will return $\texttt{accept}$ if φ_{resp} is an isogeny from E_{pk} to E'_{chall} of degree 2^e, and $\widehat{\varphi_{\text{chall}}} \circ \varphi_{\text{resp}}$ is cyclic. Otherwise, it outputs $\texttt{reject}$.

As $\mathsf{SQIsign}$ is constructed from a Σ-protocol for a hard relation (and therefore a passively secure ID scheme by Theorem 1.1.13), by Theorem 1.1.16 we have that $\mathsf{SQIsign}$ is correct and is unforgeable under chosen-message attacks in the random oracle model assuming the hardness of Problem 5.2.1 and the problem underlying the zero-knowledge property, given more explicitly in [126, Problem 14].

In the discussion above, we have not detailed how we represent the signature $(E_{\text{com}}, \varphi_{\text{resp}})$. $\mathsf{SQIsign}$ uses supersingular elliptic curves of Montgomery form $E\colon y^2 = x(x^2 + Ax + 1)$. Therefore, after normalising the elliptic curve so that we fix representative of the $\overline{\mathbb{F}}_p$-isomorphism class, we can represent E_{com} by its coefficient $A \in \mathbb{F}_{p^2}$. One way to represent φ_{resp} is by the kernel points $K_1, \dots, K_g$ generating isogenies $\varphi_1, \dots, \varphi_g$, respectively, such that $\varphi_{\text{resp}} = \varphi_g \circ \cdots \circ \varphi_1$, and computing the corresponding isogenies with Vélu's formulæ or $\sqrt{\text{élu}}$. This will be sufficient for uncompressed signatures, however there are techniques we can use to compress the signatures further, as we will detail in Chapter 11.

5.2.2.1 Attack avenues

The possible avenues of attack for $\mathsf{SQIsign}$ are as follows [78, §9]:[4]

- Endomorphism ring and general isogeny computation: We discuss this extensively in Section 4.3.1 and Section 4.3.4. The best classical and quantum attacks run in $\widetilde{O}(p^{1/2})$ and $\widetilde{O}(p^{1/4})$, respectively.

- Key recovery: In $\mathsf{SQIsign}$, the secret isogeny φ_{sk} has secret prime degree bounded by B_{sk}. We set $B_{\text{sk}} \approx p^{1/4}$ so that the number of such isogenies is close to $p^{1/2}$. As B_{sk} is small enough, the best attack is an exhaustive search, i.e., an attacker computes all isogenies of degree smaller than B_{sk} and compares their codomain curve with E_{pk}. Classicaly, this runs in $\widetilde{\Theta}(p^{1/2})$. Quantumly, we can apply Grover's algorithm which yields a quadratic speed-up at best.

  We could also recover the secret key by first computing the commitment isogeny $\varphi_{\text{com}}\colon E_0 \to E_{\text{com}}$. With φ_{com}, the challenge isogeny φ_{chall} and the signature isogeny φ_{resp}, we can obtain an isogeny from E_0 to E_{pk}, giving us an *equivalent* key that is sufficient to sign. As D_{com} is smooth, we run a meet-in-the-middle strategy which costs $\widetilde{O}(D_{\text{com}}^{1/2})$, though with high memory requirements. Applying Grover's algorithm to this, we obtain a quantum attack costing at least $\widetilde{O}(D_{\text{com}}^{1/4})$.

- Soundness/Forgery attacks: As discussed in Section 5.2.1.1, breaking soundness of the underlying ID protocol reduces to finding one non-trivial endomorphism of E_{pk}. The best classical attack costs $\widetilde{O}(p^{1/2})$, and best quantum attack runs in $\widetilde{O}(p^{1/4})$.

[4]We do not consider side-channel attacks in this thesis.

An attacker can forge a signature by first generating a random isogeny $\sigma : E_{\mathrm{pk}} \to E_{\mathrm{chall}}$ of expected degree of the signature and a random isogeny $\widehat{\varphi} : E_{\mathrm{chall}} \to E_{\mathrm{com}}$ of degree D_{chall}, and hoping that $\mathsf{H}(j(E_{\mathrm{com}}), m)$ is consistent with the challenge isogeny. The cost of this attack is $\widetilde{O}(D_{\mathrm{chall}})$ classically and $\widetilde{O}(D_{\mathrm{chall}}^{1/2})$ quantumly.

- Zero-knowledge: To attack zero-knowledge of the underlying ID scheme, the best we can do is recover the secret isogeny. Using this secret key, one can then trivially distinguish signatures from random isogenies of the same degree originating at E_{pk}. As we have already argued, the cost of computing the secret isogeny is $\widetilde{O}(p^{1/2})$ classically and $\widetilde{O}(p^{1/4})$ quantumly.

5.2.2.2 Parameter selection

Let us turn now to the parameter selection of SQIsign. Taking into account the possible avenues of attack, we want to choose SQIsign parameters that offer at least λ bits of security against all classical attacks and $\lambda/2$ bits of security against all quantum attacks. We therefore have the following requirements for security level λ:

- The prime p satisfies $\log(p) \approx 2\lambda$ with $p \equiv 3 \bmod 4$.
- The secret isogeny φ_{sk} is chosen with (secret) degree a prime bounded by $B_{\mathrm{sk}} = 2^{\lambda/2}$.
- The degree of φ_{com} should be of size roughly $2^{2\lambda} \approx p$.
- The degree of φ_{chall} should be of size roughly $2^{\lambda} \approx p^{1/2}$.

There are more restrictions imposed to increase the efficiency of SQIsign. The bottleneck of SQIsign is the computation of isogenies. By Equation (4.1) and Equation (4.2), for a supersingular curve $E/\mathbb{F}_{p^2}$, we have $\#E(\mathbb{F}_{p^2}) = (p \pm 1)^2$ and $\#E^t(\mathbb{F}_{p^2}) = (p \mp 1)^2$. To use x-only arithmetic over $\mathbb{F}_{p^2}$, SQIsign restricts to computing isogenies of *smooth* degree $N \mid (p^2 - 1)$. Finding SQIsign-friendly primes reduces to finding primes p with $p^2 - 1$ divisible by a large, smooth number. More explicitly, for a security level λ, we also have the following restrictions:

- We perform signing in $\lceil e/f \rceil$ isogeny blocks of degree 2^f. The bottleneck of this is the generation of kernel points for each of the 2^f-isogenies, and so we require f to be as large as possible such that $2^f \mid p + 1$ to reduce the number of kernel point generations.
- Recall from Section 3.2.2 that for signing we need a smooth odd factor $T \mid (p^2 - 1)$ of size roughly $p^{5/4}$.
- The degrees D_{com} and D_{chall} should be coprime. We usually fix that $D_{\mathrm{com}} \mid T$ and $D_{\mathrm{chall}} \mid 2^f 3^g$ for the efficiency of challenge generation.

To achieve NIST Level I, III, and V security, we set the security parameter as $\lambda = 128, 192, 256$, respectively. Concretely, this means that, for each of these security parameters, we have $\log(p) \approx 256, 384, 512$, and $\log(T) \approx 320, 480, 640$, with f as large as possible given the above restrictions. Finding SQIsign-friendly parameters is a difficult problem that will be the focus of Chapter 10.

Chapter 6

Overview of Literature and Contributions

In this chapter, we review the relevant literature and summarise the main contributions of this thesis.

Remark 6.0.1. We acknowledge this literature review is appearing unusually late in the thesis. This choice was made due to the fact that the literature relevant to this thesis is quite (mathematically) technical for a general cryptographic audience, and can be more easily discussed after the preliminaries.

6.1 Foundations

The use of isogenies in cryptography dates back to elliptic-curve cryptography (ECC), which constructs public-key cryptography based on elliptic curves over finite fields. ECC started in the 1980s by Miller [236] and Koblitz [202] who first suggested the use of elliptic curves to construct a Diffie-Hellman key exchange protocol with security based on the hardness of the discrete logarithm problem in elliptic curves groups. This field gained popularity after Schoof [280] developed an efficient algorithm to compute the order of elliptic curves over $\mathbb{F}_q$. This enabled the construction of an elliptic curve E so that the discrete logarithm problem in $E(\mathbb{F}_q)$ is hard. Schoof's algorithm was later improved by Elkies [149] and Atkins [279] from $O(\log(q)^{5+\epsilon})$ to $O(\log(q)^{4+\epsilon})$, giving the Schoof–Elkies–Atkins algorithm. The key tool to this improvement are isogenies. Isogenies have also seen applications in the cryptanalysis of ECC [191].

The real power of isogenies in cryptography, however, was seen through the advent of isogeny-based cryptography. Though invented in the 2000s, this field only gained traction in the late 2010s due to increased interest in constructing post-quantum secure cryptoschemes. Isogeny-based cryptography is a promising type of post-quantum secure cryptography, which gives schemes that have small communication cost.

6.1.1 Computation of isogenies

We begin by providing an overview of work that deal with the efficient computation of isogenies. We focus in particular on the case most relevant to cryptography: A is a p.p. abelian variety (of dimension 1 or 2) defined over a finite field $\mathbb{F}_q$ of prime characteristic p. In light of the correspondence between finite subgroups $G \subseteq A(\bar{\mathbb{F}}_p)[N]$ and N-isogenies with domain A, it is natural to ask whether this correspondence is effective.

Question 6.1.1. *Given the description of a subgroup $G \subseteq A(\bar{\mathbb{F}}_p)[N]$, where N is coprime to p, can we efficiently compute the corresponding $(N, \ldots, N)$-isogeny $\varphi : A \to A/G$ with kernel G?*

As we can decompose isogenies into their prime degree components, it suffices to answer this question for prime N. When A is an elliptic curve in Weierstrass form, this question was answered by Vélu. Given an N-torsion point P generating the kernel of an N-isogeny, Vélu [314] gives an $O(N)$ algorithm to compute the image of the isogeny as well as push points through the isogeny. This was improved for large enough N by Bernstein, De Feo, Leroux, and Smith [27] who present an algorithm, called $\sqrt{\text{é}}\text{lu}$ that runs in $\widetilde{O}(\sqrt{N})$. Other works have optimised N-isogeny formulæ for different forms of elliptic curves, including Montgomery curves [102, 266], twisted Edwards curves [234, 243], and Hessian curves [54, 114].

Alternative approaches to computing N-isogenies have been considered that do not require a point of order N that generates the kernel, for example, by using modular polynomials or division polynomials. Castryck, Decru, and Vercauteren [73] introduced a new approach using elliptic curves in Tate normal form to give *radical formulæ* for computing isogenies. This method is fully deterministic and completely avoids generating N-torsion points. The authors give explicit formulæ for the coordinates of an N-torsion point P' on the codomain of a cyclic N-isogeny $\varphi : E \to E'$, such that composing φ with $E' \to E'/\langle P' \rangle$ yields a cyclic N^2-isogeny, for $N \leq 13$. As such, this method is particularly useful when computing chains of N-isogenies. Chi-Domínguez and Reijnders [80] tailor this method for use in cryptography by making the formulæ fully projective (therefore avoiding field inversions) and constant time. Onuki and Moriya [251] give radical isogenies for elliptic curves in Montgomery form for $N = 3, 4$. Castryck, Decru, Houben, and Vercauteren [71] extend to $N \leq 17$ and all prime powers in $18 \leq N \leq 37$. More recently, Decru [129] provides explicit radical N-isogeny formulæ for all odd integers N, giving a highly efficient method to compute a long chain of N-isogenies.

Moving to dimension 2, the situation is more difficult. For small N, there has been extensive research into the computation of (N, N)-isogenies between Jacobians of genus-2 curves. The case $N = 2$ is the most well understood. Explicit formulæ for computing $(2, 2)$-isogenies date back to Richelot [269], and were re-developed in modern language by Bost and Mestre [50] and Cassels and Flynn [66, §3].

For larger N, the most efficient methods arise from working instead on Kummer surfaces. In this setting, Bruin, Flynn, and Testa [58], Nicholls [248], and Flynn [157] gave formulæ for $N = 3$, 4, and 5, respectively. Flynn and Ti revisited the formulæ for $N = 3$ in a cryptographic context [159], and Decru and Kunzweiler [130] further optimised these formulæ. A line of works by Bisson, Cosset, Lubicz, and Robert [35, 96, 224, 272, 273] uses the theory of theta functions [245] to provide efficient algorithms for any odd N and arbitrarily high dimensional abelian varieties. The `AVIsogenies` software package based on their results is publicly available [36]. Dartois, Maino, Pope, and Robert [117] revisited these techniques in the context of cryptography to efficiently compute chains of $(2, 2)$-isogenies between products of elliptic curves in the theta model. In Chapter 9, we delve deeper into literature surrounding the topic of computing (N, N)-isogenies, where we exhibit a general method for computing (N, N)-isogenies between *fast* Kummer surfaces. In particular, Chapter 9 focuses on the case $N = 3$: we develop a highly optimised algorithm for computing chains of $(3, 3)$-isogenies that outperforms those in the literature by at least a factor of 8.

6.1.2 Hard Homogeneous Spaces

The use of isogenies in cryptography was first introduced independently by Couveignes [108] and Rostovtsev and Stolbunov [275]. Couveignes introduced the notion of a *hard homogeneous space* (HHS) [108]. At a high level, a HHS is a commutative group G that *acts* on the set X, i.e., given an element $\alpha \in G$ we can efficiently compute its action $[\alpha] \star x$ for all $x \in X$, but inverting this action is computationally hard for any x. Couveignes instantiated a HHS using ordinary elliptic curves E defined over $\mathbb{F}_q$, observing that the class group $\mathrm{Cl}(\mathcal{O})$ of the ring of integers $\mathcal{O}$ of imaginary quadratic field $\mathbb{Q}(\sqrt{d})$ (for some $d < 0$) acts commutatively on the set of ordinary elliptic curves with endomorphism ring $\mathcal{O}$. Couveignes then showed how to construct a Diffie–Hellman-like non-interactive key exchange from this HHS. Rostovtsev and Stolbunov [275] proposed a public key encryption scheme based on the same HHS, later followed by Stolbunov [299] who proposed a non-interactive key exchange, similar to that described by Couveignes. The commutative group action leads to a sub-exponential classical attack due to Childs, Jao, and Soukharev [82], which adapts the Kuperberg's quantum algorithm for the hidden shift problem. As such, protocols built from this HHS are very inefficient, even after implementing the speed-ups due to De Feo, Kieffer, and Smith [124].

Following this, Castryck, Lange, Martindale, Panny, and Renes [74] constructed a more practical HHS using supersingular elliptic curves defined over $\mathbb{F}_p$ called CSIDH. As CSIDH is a *restricted* HHS, i.e., the structure of the group G is not known, the group action can only be efficiently evaluated for a subset of elements. Beullens, Kleinjung, and Vercauteren [32] address this by constructing another isogeny-based HHS, called CSI-FiSh, by explicitly computing the structure of G for the NIST Level I parameter set. However, both CSIDH and CSI-FiSh are still vulnerable to sub-exponential quantum attacks. Recent strides in the quantum cryptanalysis of CSIDH by Bonnetain and Schrottenloher [44] and Peikert [258] have called the original CSIDH parameter sets into question, meaning larger parameters may have to be used [77]. A lot of research in the field has thus focused on making CSIDH more practical [68, 73, 234]. More recently, a new isogeny-based HHS called SCALLOP [120] aims to obtain a (non-restricted) HHS for higher security levels using *oriented* elliptic curves.

Alamati, De Feo, Montgomery, and Patranabis [3] introduce a framework to formalise HHSs, and instead use the terminology *cryptographic group action*. The authors demonstrate the utility of this framework by using it to construct several new primitives, such as a Naor–Reingold style pseudorandom function. HHSs have also been used extensively (both directly and indirectly) to construct cryptographic primitives such as signatures [32, 74, 112, 122, 131], secret sharing [7, 8, 30, 127], ring signatures [31], oblivious pseudorandom functions [41, 276], and blind signatures [199]. This framework will not be pertinent to this work as we focus instead on supersingular curves defined over $\mathbb{F}_{p^2}$, where we do not have this commutative group action.

6.1.3 Supersingular isogeny Diffie–Hellman

Supersingular elliptic curves were introduced to cryptography by Charles, Goren, and Lauter [76], who created a hash function based on the Ramanujan property of the supersingular isogeny graph. To combat the disadvantages of the CRS scheme, De Feo, Jao, and Plût [123] proposed a key exchange scheme called *Supersingular Isogeny Diffie–Hellman* (SIDH). SIDH follows in the footsteps of the CGL hash function and uses supersingular elliptic curves over $\mathbb{F}_{p^2}$ to create a Diffie–Hellman-like key exchange protocol. As the endomorphism rings of supersingular elliptic curves are maximal orders in the quaternion algebra $\mathcal{B}_{p,\infty}$, there is no commutative group action and SIDH is not susceptible to the sub-exponential classical attack [82]. Due to this non-commutativity, the image of certain points under the isogeny needs to be provided as auxiliary input during the key exchange. As a result, the security of SIDH is based on a variant of the general isogeny problem, called the supersingular computational Diffie–Hellman problem, whose security was not well-understood at the time it was proposed.

Problem 6.1.2. Fix a base curve E_0 with $j(E_0) = 1728$.[1] Let $\varphi_A : E_0 \to E_A$ be an isogeny whose kernel is generated by $K_A = [m_A]P_A + [n_A]Q_A$, where m_A, n_A chosen at random from $\mathbb{Z}/N_A^{e_A}\mathbb{Z}$ and not both divisible by N_A. We similarly define an isogeny $\varphi_B : E_0 \to E_B$ with kernel generator $K_B = [m_B]P_B + [n_B]Q_B$. Given the curves E_A, E_B and the torsion points images

$$\phi_A(P_B),\ \phi_A(Q_B),\ \phi_B(P_A),\ \phi_B(Q_A),$$

find the j-invariant of $E_{A,B} = E_0/\langle K_A, K_B\rangle$. We depict this in Figure 6.1.

$$\begin{tikzcd}
E_0 \arrow[r, "\varphi_A"] \arrow[d, "\varphi_B"'] & E_A = E_0/\langle K_A\rangle \arrow[d] \\
E_B = E_0/\langle K_B\rangle \arrow[r] & E_{A,B} = E_0/\langle K_A, K_B\rangle
\end{tikzcd}$$

Figure 6.1: A depiction of Problem 6.1.2. The data in red is secret, in black is public and in blue is the data the problem asks for.

Galbraith, Petit, Shani, and Ti [173] gave a polynomial-time active attack against SIDH with static keys using the additional torsion point information revealed during the key exchange. To protect against this attack, a variant of the Fujisaki–Okamoto transformation [167] due to Hofheinz, Hövelmanns, and Kiltz [181] is applied to SIDH to make it a secure key encapsulation mechanism called SIKE [10]. The initial proposal by De Feo, Jao and Plût was mostly theoretical in nature, taking around 50ms per key exchange. Costello, Longa, and Naehrig [104] introduced new algorithms for SIDH that significantly improved its performance and illustrated its potential as a practical post-quantum key exchange candidate. This lead to the submission of SIKE to NIST's standardisation effort [310] for post-quantum secure key encapsulation mechanisms and signature schemes.

Following this, there was an increase in research on supersingular elliptic curves for cryptography. A large part of this research tackled the inefficiency drawbacks of SIDH compared to other types of post-quantum secure KEMs, such as the lattice-based alternative Kyber [47] that has since been standardised by NIST. We highlight in particular a work by Costello [99], which introduced a variant of SIDH, called B-SIDH, built using a new framework for instantiating isogeny-based

[1]Recall that such curve has known endomorphism ring – see Example 3.2.1.

cryptography to work with both an elliptic curve and its quadratic twist. Costello noted that restricting operations to the x-line on both sets of twists allows all arithmetic to be carried out over $\mathbb{F}_{p^2}$ as usual. We remark that this idea was first mentioned in De Feo's habilitation [118].

6.2 SIDH is broken

The security of SIDH rests upon a variant of the general isogeny problem, where the attacker knows both the degree of the secret isogenies and the images of certain points under the secret isogenies. Revealing this torsion point information proved to be a great weakness to the security of SIDH.

The first line of works that exhibited this was pioneered by Petit [259], and exploits the torsion point images to give new classical attacks against unbalanced variants of SIDH. Within the context of SIDH, the degrees of Alice and Bob's secret isogenies in Problem 6.1.2 are $A \approx B \approx \sqrt{p}$. When the degrees of the isogenies are unbalanced, namely $B > A^4 > p^4$, Petit [259, Prop. 2] showed that one can exploit the torsion point information to recover one of the secret isogenies. Martindale and Panny [228] present a number of avenues towards breaking SIDH that proved to be unsuccessful. One such direction was attempting to apply Petit's attack to balanced SIDH parameters. In 2021, Quehen, Kutas, Leonardi, Martindal, Panny, Petit, and Stange [262] improve and extend Petit's attacks. Séta [128] is a public-key encryption scheme that is constructed from a new family of trapdoor one-way functions, where the inversion algorithm uses these torsion attacks on SIDH. Kutas, Merz, Petit, and Weitkämper [211] further showed that unbalanced variants of SIDH were susceptible to a subexponential quantum attack using the torsion point information revealed in the protocol.

In August 2022, polynomial attacks against Problem 6.1.2 were announced independently by Castryck and Decru [67] (who rely on the special starting curve used in SIKE), and Maino and Martindale, later joined by Panny, Pope and Wesolowski [227] (who give a sub-exponential attack when the starting curve is arbitrary). The attacks were then extended by Robert [271], who gives a polynomial time attack for arbitrary starting curve and for abelian varieties of arbitrary dimension. In particular, after this wave of attacks, any protocol whose security relied on the isogeny problem with revealed torsion point information was completely broken. This includes SIDH/SIKE and its variants such as B-SIDH, and Séta.

The attacks heavily rely on the extra information provided to the adversary in Problem 6.1.2, namely: the degree of the secret isogenies and the torsion point images. Preliminary countermeasures have been proposed by Fouotsa, Moriya, and Petit [160], later improved by Basso and Fouotsa [18], which prevent the attacks by masking one of these pieces of data in the SIDH protocol. However, the countermeasures come at the expense of a decrease in efficiency and increase in communication cost.

More recently, De Feo, Fouotsa, and Panny [121] explore *modular* isogeny problems where the torsion information is masked by the action of a group of 2×2 matrices, giving reductions between these problems and classifying them by their hardness. For example, when the matrix is the identity matrix, we recover the SIDH hardness assumption.

6.3 Constructions

It is important to emphasize that the hardness of the general superspecial isogeny problem in dimension g, where no torsion point information is revealed, is unaffected by these attacks. Therefore, the security of isogeny-based signature SQIsign, introduced in Section 5.2, and variants is unaffected. Recall from Section 4.3 that the best classical and quantum attacks against the more general problem in dimension 1 run in $\widetilde{O}(p^{1/2})$ [132] and $\widetilde{O}(p^{1/4})$ [33], respectively. In Chapter 7, we study the concrete complexity of the Delfs–Galbraith algorithm, and give a new classical attack SuperSolver with improved concrete complexity.

6.3.1 SQIsign

Tangential to the line of work focusing on SIDH and CSIDH, Galbraith, Petit, and Silva [174] give the first constructive application of the Deuring correspondence and the KLPT algorithm [203]: an identification protocol whose security is based on the hardness of the endomorphism ring problem. Using the Fiat–Shamir transform, one can obtain a post-quantum secure signature scheme, often referred to as the GPS signature. Though this signature showed the feasibility of such a construction, it remains mainly theoretical with no accompanying implementation (to the best of this author's knowledge). A primary obstacle to the practicality of the identification scheme is the binary challenge space $\{0, 1\}$, as this means the protocol must be repeated $O(\lambda)$ times to achieve negligible soundness error, where λ is the security parameter.

Following in the footsteps of the GPS signature, De Feo, Kohel, Leroux, Petit, and Wesolowski [125] present SQIsign, a new KLPT-based signature scheme. Whilst the GPS signature relies on the original KLPT algorithm that is efficient for special maximal orders (see Theorem 5.1.4), SQIsign uses a generic variant of the KLPT algorithm that works for arbitrary maximal orders and with smaller output size than the original proposal (see Proposition 5.1.6). Using these new tools, the authors are able to construct an identification scheme with an exponentially sized challenge space, and from it obtain a practical signature scheme from quaternions and isogenies. SQIsign signatures are an order of magnitude smaller than all other post-quantum isogeny-based signature schemes in terms of combined signature and public key sizes. For NIST-I security, key generation and verification times are reasonable, taking around 0.6s and 50ms respectively. On the other hand, signing requires around 2.5s. The inefficiency of signing is largely due to the conversion from ideals of a maximal order in the quaternion algebra $\mathcal{B}_{p,\infty}$ to isogenies between supersingular elliptic curves, which is polynomial time but expensive in practice. De Feo, Leroux, Longa, and Wesolowski [126] develop new algorithms for this translation, obtaining a two-fold improvement on signing time (see Section 3.2.2). Lin, Wang, Xu, and Zhao [221] introduce further improvements to the SQIsign software, which lead to a speed-up of 5.5%, 8.8% and 25.3% for key generation, signature and verification, respectively. However, SQIsign still remains far slower than other post-quantum secure signatures based on, for example, lattices [140, 161]. In Chapter 11, we accelerate SQIsign verification to make it as competitive as possible. In particular, we make verification 2.1 times faster, or up to 3.4 times when using size-speed trade-offs, compared to the state-of-the-art implementation of SQIsign [126], without majorly degrading the performance of signing.

SQIsign suffers from another major drawback: the complicated parameter selection. As discussed in Section 5.2.2, for the algorithms involved in SQIsign signing and verification to be as efficient as possible, we place heavy restrictions on the prime p. At its essence, the problem reduces to finding a prime p such that $p^2 - 1$ has a large enough smooth factor. However, this has proven to be a difficult task that has spurred a new line of research [99, 105, 128]. Finding such primes was also required to find parameters for (the now broken) B-SIDH and Séta, and was the main focus of these three earlier works. We extend this work in Chapter 10, targetting SQIsign more specifically. In particular, we introduce a new method for prime finding combining the CHM algorithm [86] with a 'boosting' method, exploiting the fact that SQIsign does not require p^2-1 to be fully smooth. More recently, Sterner [296] obtains primes p with $p^2 - 1$ achieving the smallest smoothness bound in the literature. However, the parameters given in this work are not suitable for SQIsign as it is hard to enforce $p^2 - 1$ to be divisible by a large power of 2 *and* 3, as needed for the degree of the challenge and response isogeny. In joint work with Eriksen, Meyer, and Rodríguez-Henríquez [277], we focus particularly on obtaining SQIsign-friendly parameters for all NIST security levels.

Algorithmically, SQIsign is very different from SIDH/SIKE in its reliance on quaternionic algorithms and exploiting the effective Deuring correspondence. With respect to security, the identification protocol underlying SQIsign is sound under the endomorphism problem, a fundamental problem in isogeny-based cryptography which is equivalent to the general supersingular isogeny problem, as shown by Wesolowski [318]. In contrast, its zero-knowledge property relies on the hardness of a new ad-hoc problem based on the output distribution of the KLPT variant used in signing (see Section 5.2). Despite this, as no torsion point information is released in the construction of SQIsign, neither of these problems are susceptible to the SIDH/SIKE attacks. In 2023, SQIsign was submitted to NIST's new call for post-quantum secure signature schemes; SQIsign (NIST) is currently the only isogeny-based candidate that has progressed to Round 2.

6.3.2 Moving to higher dimensions

While almost all research in isogeny-based cryptography has focused on elliptic curves and the isogenies between them, there has been some interest in using higher dimensional varieties in cryptography.

The conjectured hardness of the general superspecial isogeny problem (see Problem 4.3.1 with $g \geq 2$) underlies the security of higher dimensional isogeny-based cryptography. The best classical and quantum attacks are due to Costello and Smith [107] and run in $\widetilde{O}(p^{g-1})$ and $\widetilde{O}(p^{(g-1)/2})$, respectively. In Chapter 8, we give an optimised implementation of the classical attack for $g = 2$, and determine its concrete complexity. Further to this, we introduce SplitSearcher, a new classical attack with reduced concrete complexity. We remark that our new dimension-1 attack SuperSolver from Chapter 7 and SplitSearcher follow the same general strategy. At the beginnning of Part II, we formalise this strategy, paving the way for future work in this area.

In 2018, Takashima [303, §4.2] made a concrete proposal for a CGL-like hash function in dimension 2 built from Jacobians of supersingular genus-2 curves and $(2, 2)$-isogenies between them. From each Jacobian, there are 14 possible non-backtracking isogenies one can take. As pointed out by Flynn and Ti [159, §2.3], however, the hash function is not collision-resistant due to an inherent presence of small cycles in the corresponding isogeny graph. This was repaired by Castryck, Decru, and Smith [72] by introducing the concept of a *good extension*. This reduces the number of possible outgoing isogenies to 8. Furthermore, they argue that the correct generalisation of supersingular elliptic curves to higher dimensions is superspecial p.p. abelian varieties. Decru and Kunzweiler [130, §6] and Castryck and Decru [70] proposed new variants of this hash function by using $(3,3)$-isogenies. In Chapter 9, we improve on the efficiency of these proposals by introducing a new hash function KuHash, built from our efficient $(3,3)$-isogenies between fast Kummer surfaces.

6.3.3 Constructions from the SIDH attack

Remarkably, the SIDH/SIKE attacks have shown to be a powerful *constructive* tool and have demonstrated that higher dimensional isogenies are an indispensable tool in constructing isogeny-based cryptosystems. We conclude this chapter by discussing a new wave of research using these attacks constructively, signalling a new era for isogeny-based cryptography from superspecial p.p. abelian varieties.

The formulation of the SIDH/SIKE attack due to Robert [271] exhibited a powerful technique, namely how to embed an isogeny of elliptic curves of large degree into an isogeny of smoother degree between higher dimensional abelian varieties. A key parameter for the efficiency of this embedding is the dimension: Robert showed that 8 dimensions is always sufficient. As the cost of computing isogenies increases exponentially with dimension, applications aim to limit the dimension to 2 or 4. As consequence of this breakthrough result, we now have a new way to represent isogenies; it is the first representation that can describe non-smooth isogenies between any two elliptic curves. With impressive strides towards optimising the implementation of $(2,2)$-isogenies in dimension two [117] and $(2,2,2,2)$-isogenies in dimension four [115] using formulas derived from theta structures, this new representation is efficient.

Existing protocols have received considerable improvements by relying on higher dimension representations, such as SCALLOP-HD [79] and SQIsignHD [116]. Most pertinent to this manuscript are the improvements to SQIsign. SQIsignHD uses 4-dimensional isogenies to drastically improve the signing time and combined signature/public-key sizes in SQIsign, in addition to a number of other benefits, but at the cost of a slowdown in verification. A new ideal-to-isogeny algorithm introduced by Onuki and Nakagawa [252] uses two-dimensional isogenies. When applied to SQIsign, key generation and the signing procedures are at least twice as fast as those in SQIsign (NIST) for NIST security Level I, with the advantage becoming more significant at higher security levels, and no degradation to verification time. Even more substantial improvements are seen with new 2-dimensional variants of SQIsign, called SQIsign2D [17, 142, 247]. These achieve sizes comparable to SQIsignHD, slightly slower signing than SQIsignHD but still much faster than SQIsign, and the fastest verification of any known variant of SQIsign. We further highlight that the security of the rigorous variants of SQIsignHD and SQIsign2D in [17] rely solely on the endomorphism ring problem, provided access to an isogeny sampling oracle. In this way, the ad-hoc assumption for zero-knowledge is removed. We note also that for these new variants, SQIsignHD and SQIsign2D, the restrictions on the prime p are lifted, and parameter selection is a much simpler task.

New schemes built from higher-dimensional representations have also since been constructed. FESTA [19] is an isogeny-based public-key encryption (PKE) scheme constructed from a trapdoor function, where the SIDH attacks are used to invert the one-way function. By constructing a new algorithm to compute an isogeny of non-smooth degree using quaternion algebras and the SIDH attack, FESTA was improved by Nakagawa and Onuki [246] to give QFESTA. Following this, Basso [16] introduced POKE, a new framework to build cryptographic protocols from irrational isogenies using higher dimensional representations. Basso uses this framework to construct, for example, a new PKE scheme. Duparc, Fouotsa, and Vaudenay [143] construct an updatable public key encryption scheme (UPKE) using the Deuring correspondence and the SIDH attacks, overcoming the limitations of SIDH-based UPKE highlighted by Eaton, Jao, Komlo, and Mokrani [145]. Leroux [218] proposed a class of verifiable delay functions where the evaluation uses algorithms for the Deuring correspondence requiring isogenies of dimension d_1, and the verification algorithm requires the computation of d_2-dimensional isogeny representations.

Part II

On the concrete complexity of the isogeny problem

Attacking the isogeny problem

In Part II, we study the general supersingular isogeny problem. Recall that this problem asks, given superspecial p.p. abelian varieties A_1, A_2 of dimension g defined over $\mathbb{F}_{p^2}$, to find an isogeny $\varphi : A_1 \to A_2$. Throughout Chapter 7 and Chapter 8, we analyse the concrete complexity of the best classical attacks against this problem in dimensions 1 and 2, respectively. In the case where $g = 1$, the best attack is due to Delfs and Galbraith [132] and runs in $\widetilde{O}(\sqrt{p})$. Increasing dimension to $g > 1$, the best attack runs in $\widetilde{O}(p^{g-1})$, and is due to Costello and Smith [107]. As a first step, we give an optimised implementation of each attack to obtain a theoretical grasp of their concrete complexity. We then extend this by developing new tools that improve their concrete complexity. The improvements made to each algorithm are best motivated by observing that the Delfs–Galbraith and Costello–Smith attack follow a common strategy.

The general strategy

Superspecial abelian varieties $A/\mathbb{F}_{p^2}$ of dimension $g \geq 1$ are nodes in the graph $\mathcal{X}_g(\bar{\mathbb{F}}_p)$. We denote the set of nodes by $\mathcal{S}_g(\bar{\mathbb{F}}_p)$. Our goal is to find a path connecting two nodes A_1 and A_2. Before giving the general strategy to find such a path, we define a *special subset* $X_g \subset \mathcal{S}_g(\bar{\mathbb{F}}_p)$ so that finding a path between nodes in this set is asymptotically less costly than the general problem of finding paths in $\mathcal{X}_g(\bar{\mathbb{F}}_p)$.

Definition 6.3.1. Consider the subset $X_g \subset \mathcal{S}_g(\bar{\mathbb{F}}_p)$, where the best attack against the general isogeny problem between two abelian varieties A_1, $A_2 \in X_g$ has asymptotic complexity $\widetilde{O}(p^a)$ for $a > 0$. Then, X_g is defined to be *special* if $\#\mathcal{S}_g(\bar{\mathbb{F}}_p)/\#X_g = \widetilde{O}(p^\epsilon)$ for $\epsilon \geq a$.

Let X_g be a special subset of $\mathcal{S}_g(\bar{\mathbb{F}}_p)$. Our strategy to solve the isogeny problem in $\mathcal{X}_g(\bar{\mathbb{F}}_p)$ given $A_1, A_2 \in \mathcal{S}_g(\bar{\mathbb{F}}_p)$, is as follows. Fix a prime number $N \neq p$.

1. Take a non-backtracking walk in $\mathcal{X}_g(\bar{\mathbb{F}}_p, N)$ from a node A_1 until landing on a node $A_1' \in X_g$, thus obtaining an isogeny $\phi_1 : A_1 \to A_1'$. Similarly, for A_2, we obtain an isogeny $\phi_2 : A_2 \to A_2'$, for $A_2' \in X_g$. Starting from a node in $\mathcal{S}_g(\bar{\mathbb{F}}_p)$, we expect to take $\widetilde{O}(p^\epsilon)$ steps in $\mathcal{X}_g(\bar{\mathbb{F}}_p, N)$ before finding an abelian variety in the special subset X_g. Thus, we find isogenies ϕ_1 and ϕ_2 in $\widetilde{O}(p^\epsilon)$.

2. Find a path $\varphi : A_1' \to A_2'$ between these nodes in X_g. This can be done with $\widetilde{O}(p^a)$ bit operations.

3. Output the isogeny $\widehat{\phi}_2 \circ \varphi \circ \phi_1 : A_1 \to A_2$.

As X_g is special, we have that $\epsilon \geq a$. Therefore, the first step in this attack is the *bottleneck* step of the algorithm. Assuming that taking a step in the graph is polylogarithmic in p, the algorithm will terminate with $\widetilde{O}(p^\epsilon)$ bit operations. We now see how the attacks against the general isogeny problem in dimension $g = 1$ and 2 fit within this framework.

Example 6.3.2 (The Delfs–Galbraith algorithm)**.** We first aim to solve the dimension-1 supersingular isogeny problem. Recall from Problem 4.3.1 that this asks to find an isogeny $\varphi \colon E_1 \to E_2$

between two supersingular curves, $E_1/\mathbb{F}_{p^2}$ and $E_2/\mathbb{F}_{p^2}$, whose j-invariants lie in S_{p^2} (see Equation (4.3)). The best classical attack against this problem is the Delfs–Galbraith algorithm, described in Section 4.3. Using the notation as above, the special subset is S_p:

$$X_1 := S_p = \{E : j(E) \in \mathbb{F}_p\} \subset \mathcal{S}_1(\overline{\mathbb{F}}_p).$$

The best attack against the isogeny problem for curves in X_1 has asymptotic complexity $\widetilde{O}(p^{1/4})$ (i.e., $a = 1/4$). As $\#\mathcal{S}_1(\overline{\mathbb{F}}_p)/\#X_1 = \widetilde{O}(p^{1/2})$, we have $\epsilon = 1/2$. As discussed in Section 4.3, to take a step in $\mathcal{X}_1(\overline{\mathbb{F}}_p, N)$, we compute one of the $D_{N,1} - 1$ roots of

$$\Phi_N(X, j_c)/(X - j_p)$$

with cost polynomial in N and $\log p$. For N fixed and small, the asymptotic complexity of the Delfs–Galbraith algorithm is $\widetilde{O}(p^{1/2})$.

Example 6.3.3 (The Costello–Smith algorithm)**.** Consider now the dimension-2 superspecial isogeny problem that asks to find an isogeny $\varphi\colon A_1 \to A_2$ between two p.p. abelian surfaces $A_1/\mathbb{F}_{p^2}$ and $A_2/\mathbb{F}_{p^2}$ lying in $\mathcal{J}_2(\overline{\mathbb{F}}_p)$ as defined by Equation (4.4). As explored in Section 4.3, the best classical attack against this problem is the Costello–Smith algorithm, which takes the special set to be the set of elliptic products $\mathcal{E}_2(\overline{\mathbb{F}}_p)$:

$$X_2 := \mathcal{E}_2(\overline{\mathbb{F}}_p) = \{A \in \mathcal{S}_2(\overline{\mathbb{F}}_p)\colon A \cong E_1 \times E_2\} \subset \mathcal{S}_2(\overline{\mathbb{F}}_p).$$

The best attack against the isogeny problem for curves in X_2 has asymptotic complexity $\widetilde{O}(p^{1/2})$ (i.e., $a = 1/2$), and $\#\mathcal{S}_2(\overline{\mathbb{F}}_p)/\#X_2 = \widetilde{O}(p)$ so $\epsilon = 1$. Taking steps in $\mathcal{X}_g(\overline{\mathbb{F}}_p, N)$ amounts to computing (N, N)-isogenies between p.p. abelian surfaces, which has quasi-linear complexity in N (see, for example, [224]). Therefore, for N fixed and small, the asymptotic complexity of the Costello–Smith algorithm is $\widetilde{O}(p)$.

Improving the detection of the special subset

When attacking the general isogeny problem, we walk around the graph $\mathcal{X}_g(\overline{\mathbb{F}}_p, N)$ for some fixed N. Most commonly, we take $N = 2$ as this minimises the cost of taking a step in the graph. Indeed, taking a step corresponds to computing a $(N, \dots, N)$-isogeny, whose cost grows as N grows.

For $g = 1$ and 2, let $D_{N,g}$ be the number of outgoing edges from a node $A \in \mathcal{S}_g(\overline{\mathbb{F}}_p)$ as defined in Equations (2.7) and (2.8). At each step, we inspect *at most* $D_{N,g}$ nodes and detect whether they lie in the special subset X_g. The main idea to improve the concrete complexity of the attacks is to improve the detection of the special subset X_g. Explicitly, we develop methods to detect whether a node $A \in \mathcal{S}_g(\overline{\mathbb{F}}_p)$ is N-isogenous to a node in X_g for any $N \geq 2$ and $g = 1$ and 2. Therefore, while walking around $\mathcal{X}_g(\overline{\mathbb{F}}_p, 2)$, we can also inspect nodes in $\mathcal{X}_g(\overline{\mathbb{F}}_p, N)$ for any N, thus scanning a larger proportion of the graph at each step, increasing the probability of finding a special node. If the detection is not too costly, one can hope that this will lead to an improvement of the concrete complexity of path finding. Introducing an efficient detection method whose complexity grows only with N, and not p, for $g = 1$ and 2 will be the key to the works presented in Chapter 7 and Chapter 8, respectively.

Chapter 7

The isogeny problem in dimension 1

In this chapter, we study the general supersingular isogeny problem in dimension 1, and determine the concrete complexity of the best classical attack against it: the Delfs–Galbraith algorithm. Using a new efficient method to detect whether a polynomial in $\mathbb{F}_{p^2}[x]$ has roots in $\mathbb{F}_p$, we improve the concrete complexity of this attack. For cryptographic sized primes $p \approx 2^{256}$, we show that there is a decrease in complexity by around a factor of 8. The chapter is based on the paper

Accelerating the Delfs–Galbraith algorithm with fast subfield root detection

which was joint work with Craig Costello and Jia Shi, published at CRYPTO 2022 [90]. This chapter presents the paper as published, except for the following alterations:

- At the time of publication, understanding the concrete complexity of the dimension 1 isogeny problem had applications to better understanding the security of B-SIDH, a variant of SIDH. However, B-SIDH is now broken (see Section 6.2), and so this application is no longer of interest. We have added disclaimers throughout the chapter when these schemes are mentioned. We emphasize that this problem still underlies the security of all isogeny-based cryptoschemes using supersingular elliptic curves over $\mathbb{F}_{p^2}$, and so the results in this chapter still remain of interest to the community.

- We add Remark 7.4.5 to present an alternative method for inspecting the graph that was pointed out to us by Travis Morrison. We note, however, that it does not outperform the detection method given in this work.

- We fixed a small error in the code which led to slightly inaccurate experimental costs in Table 7.3 and suboptimal choices of sets $\mathcal{N}_b$ for primes of bitsize ≥ 100. This affected the experimental results reported in Tables 7.1, 7.6 and 7.7. We emphasise that these corrections do not invalidate any of the experimental results in the published paper, but rather lead to an improved choice of optimal set $\mathcal{N}_b$ and thus lead to a lower concrete complexity for SuperSolver than what was reported. For example, for the SQIsign prime, the approximate number of $\mathbb{F}_p$-multiplications per node inspected at each step is now 55.7 rather than 58.0 (around a 1% improvement).

- We provide a more in-depth exposition on the algorithm used to compute square roots in $\mathbb{F}_{p^2}$ in Algorithm 7.1 and to take a step in the graph $\mathcal{X}_1(\bar{\mathbb{F}}_p, 2)$ in Algorithm 7.2, with cost given by Lemma 7.2.3. In Section 7.4.2, we give a tighter theoretical upper bound on the cost cost_N of inspecting N-isogenous neighbours. We also formalise a heuristic in Remark 7.4.2 that was implicitly stated in the published paper. These additions, along with Heuristic 7.4.6, lead to a (reasonably tight) theoretical bound for the concrete cost of our implementation of the Delfs–Galbraith algorithm Solver and our improvement SuperSolver in Section 7.4.4.

- Small editorial changes, such as notation changes to match the preliminaries.

Introduction

In its most general form, the *supersingular isogeny problem* asks an adversary to find an isogeny

$$\phi \colon E_1 \to E_2$$

between two given supersingular elliptic curves $E_1/\mathbb{F}_{p^2}$ and $E_2/\mathbb{F}_{p^2}$. We emphasize that this is the general problem, where we do not assume knowledge of the degree of the isogeny, or any torsion point information. The best known classical attack against the supersingular isogeny problem is the Delfs–Galbraith algorithm [132]. We recall Section 4.3 to fix notation. The first step computes random walks in the N-isogeny graph (for some choice of N) to find isogenies $\varphi_1 \colon E_1 \to E_1'$ and $\varphi_2 \colon E_2 \to E_2'$, such that $E_1'/\mathbb{F}_p$ and $E_2'/\mathbb{F}_p$ are *subfield curves* in $\widetilde{O}(p^{1/2})$ bit operations. The second step searches for a *subfield* isogeny $\varphi' \colon E_1' \to E_2'$ that connects ϕ_1 and ϕ_2, and it requires $\widetilde{O}(p^{1/4})$ bit operations. It follows that the entire algorithm runs in $\widetilde{O}(p^{1/2})$ operations on average, with the cost dominated by the first step, i.e., the search for paths to subfield curves. By Example 6.3.2, this algorithm follows the general strategy described at the beginning of Part II with special subset $X_1 \subset \mathcal{S}_1(\overline{\mathbb{F}}_p)$ being $\mathcal{S}_p$. In this work, we investigate and improve the concrete complexity of the Delfs–Galbraith algorithm.

Contributions

Solver. To the best of our knowledge, a precise complexity analysis of the Delfs–Galbraith algorithm has not been conducted. We fill this gap by presenting an optimised implementation of the Delfs–Galbraith algorithm, called Solver, and conducting experiments over many thousands of instances of the subfield search problem to determine its concrete complexity. Though Solver finds the full path, we focus on the optimisation and complexity of the bottleneck step: finding subfield curves. These optimisations include:

- *Choice of N.* In their high-level description of the algorithm, Delfs and Galbraith do not specify which N-isogeny graph to walk in. Framing the problem of taking a step in the N-isogeny graph as computing the roots of a polynomial of degree N, in Solver we chose the simplest and most efficient choice at the time of publication: $N = 2$.

- *Fast square root finding in $\mathbb{F}_{p^2}$.* We use the techniques presented by Scott [281, §5.3] to construct an optimised algorithm for finding square roots in $\mathbb{F}_{p^2}$.

- *Random walks in the 2-isogeny graph.* We implement a depth-first search to find subfield nodes in the 2-isogeny graph and give a precise complexity analysis on the number of $\mathbb{F}_p$ operations required.

SuperSolver. The main contribution of this chapter is a new state-of-the-art algorithm for solving the general supersingular isogeny problem, called SuperSolver. This is a variant of the Delfs–Galbraith algorithm that exploits a combination of our new *subfield root detection* algorithm and the use of modular polynomials. We show that we can efficiently determine whether a polynomial $f \in \Bbbk'[X]$ has a root in a subfield $\Bbbk \subset \Bbbk'$, without finding any roots explicitly. Though this algorithm works for general fields and polynomials (and may be of use in other contexts), we apply it to the case where $f = \Phi_{N,p}(X, j) \in \mathbb{F}_{p^2}[X]$, i.e., where f is the N-th modular polynomial evaluated at a supersingular j-invariant. This provides a means of quickly determining whether there is an N-isogeny connecting the corresponding elliptic curve to a subfield curve: we develop this NeighbourInFp subroutine in Section 7.3, and use it as the core of our SuperSolver algorithm in Section 7.4.

Experiments and cryptographic applications. In Section 7.6, we conduct extensive experiments using both our Solver and SuperSolver libraries, all of which show that SuperSolver performs much faster than Solver. In Table 7.1, we give a taste of the types of improvements we see in searching for subfield nodes over supersingular sets of various sizes, taking a number of primes from the isogeny-based literature. These primes were specifically chosen because the Delfs–Galbraith algorithm for the general supersingular isogeny problem is the best known classical attack against the cryptosystems they target.[1]

The work in this chapter has implications on the classical bit-security of any supersingular isogeny-based scheme for which the Delfs–Galbraith algorithm is the best known attack; this includes the key exchange scheme B-SIDH [99],[2] the GPS signature scheme [174, §4], and the signature scheme SQIsign [125]. For any proposed instantiation of such schemes, our SuperSolver suite allows the analysis in Section 7.6 to be conducted on input of any prime p, and determines a precise estimate on the number of operations required (on average) to solve the corresponding supersingular isogeny problem. Furthermore, in Section 7.4.4 we give a theoretical upper bound for the cost of Solver and SuperSolver, subject to certain heuristics. This is especially accurate when the cardinality of the associated class group is known (e.g., in the oriented case), which has recently been shown to be feasible for primes up to 512 bits [32].

Availability of Software

Our Solver and SuperSolver algorithms are written in SageMath [311] and Python and can be found at:

https://github.com/microsoft/SuperSolver.

Outline

We give the preliminaries in Section 7.1. In Section 7.2, we present our optimised instantiation of the traditional Delfs–Galbraith algorithm, called Solver. In Section 7.3, we construct an efficient algorithm to detect whether a polynomial has a root in a subfield. We use this algorithm to build SuperSolver in Section 7.4. In Section 7.5, we present a worked example to highlight the differences

[1] At the time of publication, the best attack against B-SIDH was the Delfs–Galbraith algorithm.
[2] At the time of publication, B-SIDH was still secure.

	Solver		SuperSolver		
Prime p	Nodes inspected	$\mathbb{F}_p$-mults. per node	Fastest Sets $\mathcal{N}$	Nodes inspected	$\mathbb{F}_p$-mults. per node
B-SIDH-p247 [99]	248,913	402	{3,5,7,9,11}	1,716,751	55.6
			{3,5,7,9,11,13}	1,727,601	56.0
			{3,5,7,11,13}	1,731,625	57.8
TwinSmooth-p250 [105]	233,507	427	{3,5,7,9,11}	1,680,337	59.1
			{3,5,7,9,11,13}	1,699,825	59.1
			{3,5,7,11,13}	1,697,769	59.8
SQISign-p256 [125]	248,915	403	{3,5,7,9,11}	1,716,751	55.7
			{3,5,7,9,11,13}	1,727,601	55.9
			{3,5,7,11,13}	1,731,625	57.8
TwinSmooth-p384 [105]	163,331	610	{3,5,7,9,11,13}	1,529,025	63.1
			{3,5,7,9,11}	1,464,709	65.1
			{3,5,7,9,11,13,17}	1,487,919	65.4
TwinSmooth-p512 [105]	127,511	784	{3,5,7,9,11,13}	1,397,761	69.1
			{3,5,7,9,11,13,17}	1,391,645	70.0
			{3,5,7,9,11,13,19}	1,351,509	72.1

Table 7.1: The number of nodes inspected per 10^8 field multiplications and for primes targeting schemes where Delfs–Galbraith is the best known classical attack. The Solver column corresponds to optimised Delfs–Galbraith walks in $\mathcal{X}_1(\bar{\mathbb{F}}_p, 2)$ – see Section 7.2. The SuperSolver columns correspond to enabling our fast subfield root detection algorithm with the three fastest sets $\mathcal{N}$ of N's (top to bottom) – see Section 7.4. We also give the approximate number of $\mathbb{F}_p$-multiplications per node inspected at each step, as computed during the precomputation phase that predicts which sets $\mathcal{N}$ will perform fastest; for Solver we have $\mathcal{N} = \{\}$.

between both algorithms, and in Section 7.6 we present a number of implementation results that illustrate the concrete improvements offered by SuperSolver.

7.1 Preliminaries

For this chapter, we require knowledge of supersingular elliptic curves and their isogenies (see Chapter 2 and Section 4.1), the dimension 1 supersingular isogeny problem and the best attack against it (see Section 4.3.1), the supersingular isogeny graph (see Section 4.2.1), and modular polynomials (see Section 2.8.1.1). We now introduce requisite background for this chapter that was not introduced in the preliminaries in Part I.

7.1.1 Factoring polynomials over finite fields

Let $f(X) \in \mathbb{F}_q[X]$ be a monic polynomial of degree N with $q = p^k$ for a prime p, and for the purposes of this paper, assume that p is very large (i.e., cryptographically sized) and N is relatively small (i.e., $N < 100$). The literature contains a number of methods for finding the irreducible factors of f in $\mathbb{F}_q[x]$, and we briefly mention the most applicable and well-known algorithms for our scenario. Berlekamp's algorithm [23] factors f using an expected number of $O(N^3 + N^2 \log N \log q)$

operations in $\mathbb{F}_q$ [289, Theorem 20.12]. This appears to be superior to the Cantor–Zassenhaus algorithm [65], which uses an expected number of $O(N^3 \log q)$ operations in $\mathbb{F}_q$ [289, Theorem 20.9], however one can take advantage of certain time-memory trade-offs to implement Cantor–Zassenhaus so that it requires $O(N^3 + N^2 \log q)$ operations in $\mathbb{F}_q$ [289, Exercise 20.13]. Note that both of these big-O complexities hide a number of subtleties, that $\mathbb{F}_q$-inversions are included as $\mathbb{F}_q$ operations, and moreover that both of these algorithms are probabilistic. Their deterministic variants have worse complexities [289, §20.6].

7.1.2 Polynomial gcd

Euclid's integer gcd algorithm is easily adapted to compute polynomial gcd's [289, §17.3]. Computing the gcd of two polynomials $g, h \in \mathbb{F}_q[x]$ requires $O(\deg(g) \cdot \deg(h))$ operations in $\mathbb{F}_q$. Again, here each $\mathbb{F}_q$ inversion is counted as a $\mathbb{F}_q$ operation. In order to make our algorithms run as fast as possible, one of the necessary subroutines we derive in Section 7.3 is an inversion-free polynomial gcd algorithm, for which we state an upper bound on the concrete complexity.

7.1.3 Measuring complexity

Throughout this paper we will avoid stating asymptotic (i.e., big-O-style) complexities in favour of stating concrete ones. One of our goals in Section 7.2 and Section 7.4.4 is to replace the $\widetilde{O}(p^{1/2})$ complexity of the original Delfs–Galbraith algorithm with a closed formula that can be used to give precise estimates on the classical security of the relevant cryptographic instantiations. We will use the metric of $\mathbb{F}_p$-multiplications as convention, noting that it is relatively straightforward to convert this into a more fine-grained metric (e.g., bit operations, machine operations, cycle counts, gate counts, circuit depth, etc.) depending on the context and on the implementation of the $\mathbb{F}_p$ arithmetic. For simplicity, we will count $\mathbb{F}_p$-squarings as multiplications and ignore additions. We justify this by noting that, roughly speaking, the ratio of multiplications to additions in all the algorithms in this work are similar, and the complexity of $\mathbb{F}_p$-additions have a minimal impact on any of the aforementioned metrics.

7.1.4 Subfield search complexity determines concrete bit security

Both the Solver implementation detailed in Section 7.2 and the SuperSolver implementation detailed in Section 7.4 solve *all* instances of the general supersingular isogeny problem. On input of any prime p and any two supersingular j-invariants in $\mathcal{S}_{p^2}$, both implementations will always terminate with an isogeny that solves the corresponding problem. We emphasise that henceforth our sole focus is on the $\widetilde{O}(p^{1/2})$ subfield search phase of the Delfs–Galbraith algorithm. Finding a path between subfield nodes requires $\widetilde{O}(p^{1/4})$ operations, which is negligible in both the asymptotic sense and in the sense of obtaining cryptographic security estimates. To see this, suppose the asymptotic $\widetilde{O}(p^{1/2})$ complexity of the first phase is replaced by a concrete complexity of $c_p \cdot p^{1/2}$, and the asymptotic $\widetilde{O}(p^{1/4})$ complexity of the second phase is replaced by a concrete complexity of $d_p \cdot p^{1/2}$, where c_p and d_p are polynomials in $\log p$. The total complexity of the Delfs–Galbraith algorithm is then

$$c_p \cdot p^{1/2} + d_p \cdot p^{1/4}.$$

For primes of cryptographic size, small changes in c_p have an immediate influence on the total runtime of the algorithm, while much larger changes in d_p will not play a part in the bit security of the problem. For $p > 2^{200}$, a factor 2 change in c_p changes the bit security of the problem by 1, while d_p would have to change by a factor of at least 2^{50} to have the same impact on the bit security.

7.2 An optimised implementation of the Delfs–Galbraith algorithm

We begin the exploration into attacking the dimension one superspecial isogeny problem by presenting Solver, an optimised implementation of the first, bottleneck step of the Delfs–Galbraith algorithm: searching for subfield curves in $\mathcal{X}_1(\bar{\mathbb{F}}_p, N)$. In Section 7.2.4, we experimentally determine the concrete complexity of our optimised algorithm Solver, which will set the stage for the improvements given in Section 7.4 on the concrete complexity of the Delfs–Galbraith algorithm.

7.2.1 Efficient square root computation in finite fields

To take steps in the graph $\mathcal{X}_1(\bar{\mathbb{F}}_p, 2)$, we need to compute square roots in $\mathbb{F}_{p^2}$. Optimal computation of square roots in extension fields of large characteristic requires careful attention to detail. A 2013 paper by Adj and Rodríguez-Henríquez [230] cost the process of computing square roots in $\mathbb{F}_{p^2}$ at two $\mathbb{F}_p$ residuosity tests, two $\mathbb{F}_p$ square roots, and one $\mathbb{F}_p$ inversion, for a total of five exponentiations in $\mathbb{F}_p$. In [281, §5.3], Scott shows that these operations can be combined in a clever way to significantly reduce this cost. The inputs into the Tonelli-Shanks $\mathbb{F}_p$ square root algorithm [231, Algorithm 3.34] can be tweaked in such a way that the two residuosity tests are absorbed into the two square roots. Moreover, Scott shows that most of the inversion cost can also be absorbed by application of Hamburg's combined 'square-root-and-inversion' trick [180]. In addition, there are a handful of $\mathbb{F}_p$-multiplications and additions (whose precise number depends on the maximum integer e such that $2^e \mid p-1$) that either update the Tonelli-Shanks outputs depending on the residuosity outcomes or collect and combine the results according to the formula in [281, §5.3].

We follow Scott to construct a general square root algorithm Sqrt that is highly optimised with respect to the number of $\mathbb{F}_p$ operations it incurs. For more details on how this algorithm is constructed, refer to [281]. Let $\mathbb{F}_{p^2} = \mathbb{F}_p(\alpha)$. Given $a \in \mathbb{F}_{p^2}$, it will compute $b \in \mathbb{F}_{p^2}$ such that $b^2 = a$ (assuming such a b exists), using two subroutines:

- `consts` $\leftarrow$ TonelliShanksConstants(p): on input a prime p, this algorithm computes the following constants:
  - $\beta := \alpha^2$, a quadratic non-residue in $\mathbb{F}_p$.
  - e, the maximum integer such that $2^e \mid p-1$ (so $e < \log(p)$);
  - $z := \beta^{(p-1)/2^e}$, a precomputed 2^e-th root of unity;
  - $c := (p - 2^e - 1)/2^{e+1}$, so that the *progenitor* of x is $y := x^c$ (as defined by Scott [281, §1.3]).

The algorithm then outputs

$$\texttt{consts} = \{\beta, 2^{-1}, e, z, c, \beta^c, 2^e - 1, e+1, \beta^{-1}\},$$

where the inverses are computed modulo p.

- $s, \texttt{b} \leftarrow \mathsf{TonelliShanks}(x, y, \texttt{consts})$: on input x, $y \in \mathbb{F}_p$, and $\texttt{consts}$ defined as above, this algorithm computes $\texttt{b} := (xy^2)^{2^{e-1}}$ to determine if x is a quadratic residue (see [281, §2]). If $\texttt{b} = 1$, then x is a quadratic residue, and the algorithm runs the Tonelli–Shanks algorithm (see, for example, [281, pg. 2] for pseudocode) on input $(x, y, \texttt{consts})$ to compute $s \in \mathbb{F}_p$ such that $s^2 = x$. Otherwise, if $\texttt{b} \neq 1$, then x is a quadratic non-residue, and the algorithm runs the Tonelli–Shanks algorithm on input $(x\beta, y\beta^c, \texttt{consts})$ to compute $s \in \mathbb{F}_p$ such that $s^2 = x\beta$ (note β and β^c are precomputed and stored as the first and sixth elements of $\texttt{consts}$). The algorithm outputs s and $\texttt{b}$. Each call to $\mathsf{TonelliShanks}$ costs at most $\frac{1}{2}(e^2 + 7e + 4)$ $\mathbb{F}_p$-multiplications.

Algorithm 7.1 Sqrt: given $a \in \mathbb{F}_{p^2}$ where $\mathbb{F}_{p^2} = \mathbb{F}_p(\alpha)$, compute $b \in \mathbb{F}_{p^2}$ such that $b^2 = a$ (if it exists).

Input: $a \in \mathbb{F}_{p^2}$
Output: $b \in \mathbb{F}_{p^2}$ with $b^2 = a$.

```
1: $\texttt{consts} \leftarrow \mathsf{TonelliShanksConstants}(p)$
2: $\beta, 2^{-1}, e, z, c, \beta^c, 2^e - 1, e+1, \beta^{-1} \leftarrow \texttt{consts}$
3: $a_0 + a_1\alpha \leftarrow a$
4: $\delta \leftarrow a_0^2 - \beta a_1^2$
5: $y \leftarrow \delta^c$
6: $s_\delta, \texttt{b} \leftarrow \mathsf{TonelliShanks}(\delta, y, \texttt{consts})$
7: if $\texttt{b} \neq 1$ then
8:     return $\perp$                              {$a$ is a quadratic non-residue}
9: end if
10: $x \leftarrow 2^{-1}(a_0 + s_\delta)$
11: $y \leftarrow x^c$
12: $s_x, \texttt{b} \leftarrow \mathsf{TonelliShanks}(x, y, \texttt{consts})$
13: $x_1 \leftarrow x^{2^e - 1}$
14: $a_1(2x)^{-1} \leftarrow x_1 \cdot a_1 y^{2^{e+1}} \cdot 2^{-1}$          {compute $y^{2^{e+1}}$ with repeated squarings}
15: if $\texttt{b} \neq 1$ then
16:     $b_0 \leftarrow s_x \cdot a_1(2x)^{-1}$
17:     $b_1 \leftarrow s_x \cdot \beta^{-1}$
18: else
19:     $b_0 \leftarrow s_x$
20:     $b_1 \leftarrow s_x \cdot a_1(2x)^{-1}$
21: end if
22: return $b_0 + b_1\alpha$
```

The algorithm has applications beyond taking steps in $\mathcal{X}_1(\bar{\mathbb{F}}_p, 2)$. Indeed, we will reuse this algorithm in both Chapter 8 and Chapter 11.

The two calls to $\mathsf{TonelliShanks}$ costs $(e^2 + 7e + 4)$ $\mathbb{F}_p$-multiplications. In Lines 5, 11 we compute exponentiation by c, and in Line 13 exponentiation by $2^e - 1$. The other lines of Algorithm 7.1

requires at most $(e+10)$ $\mathbb{F}_p$-multiplications. We remark that Line 1 is often done as a precomputation as TonelliShanksConstants depends only on p, so we do not include it in the cost. In total, Sqrt requires at most 3 $\mathbb{F}_p$-exponentiations and $(e^2+8e+14)$ $\mathbb{F}_p$-multiplications where $e < \log(p)$.

Remark 7.2.1. The fixed exponentiations that take place in Lines 5, 11 and 13 could be further optimised for a specific p by tailoring a larger window or a different addition chain. For our purposes in this chapter, the impact of this improvement would be minor.

7.2.2 Taking a step in $\mathcal{X}_1(\bar{\mathbb{F}}_p, 2)$

We take steps in the isogeny graph $\mathcal{X}_1(\bar{\mathbb{F}}_p, N)$ using the modular polynomial $\Phi_N(X,Y)$ of level N, defined in Section 2.8.1.1. Recall from Section 4.3.1 that we can take steps in $\mathcal{X}_1(\bar{\mathbb{F}}_p, N)$ by choosing one of the $D_{N,1}-1$ roots of $\Phi_N(X, j_c)/(X-j_p)$, at random, where j_c is the current j-invariant, j_p is the previous j-invariant, and

$$D_{N,1} := \prod_{i=1}^{n} (\ell_i+1)\ell_i^{e_i-1}, \text{ for } N = \prod_{i=1}^{n} \ell_i^{e_i}.$$

The first parameter that we specify is N, i.e., the isogeny graph to walk around in. Considering $D_{N,1}$ and the complexity of factorisation algorithms given in Section 7.1.1, we chose $N=2$ to obtain the most efficient and simplest choice where we are able to take advantage of fast explicit methods for computing square roots in $\mathbb{F}_{p^2}$.

After stepping from $j_p \in \mathbb{F}_{p^2}$ to $j_c \in \mathbb{F}_{p^2}$, a non-backtracking walk in $\mathcal{X}(\bar{\mathbb{F}}_p, 2)$ will step to one of two new nodes: j_0 and j_1. These are computed by solving the quadratic equation that arises from the modular polynomial $\Phi_N(X,Y)$ with $N=2$:

$$\begin{aligned}\Phi_2(X,Y) = -X^2Y^2 + X^3 + Y^3 + 1488 \cdot (X^2Y + Y^2X) - 162000 \cdot (X^2+Y^2) \\ + 40773375 \cdot XY + 8748000000 \cdot (X+Y) - 157464000000000.\end{aligned}$$

The three neighbours of j_c in $\mathcal{X}(\bar{\mathbb{F}}_p, 2)$ are j_p, j_0, and j_1, meaning that $\Phi_2(X, j_c)$ factorises as

$$\Phi_2(X, j_c) = (X-j_p)(X-j_0)(X-j_1).$$

This yields a quadratic equation, whose solutions are j_0, j_1, defined by $X^2 + \alpha X + \beta = 0$, where

$$\begin{aligned}\alpha &= -j_c^2 + 1488 \cdot j_c + j_p - 162000, \\ \beta &= j_p^2 - j_c^2 j_p + 1488 \cdot (j_c^2 + j_c j_p) + 40773375 \cdot j_c - 162000 \cdot j_p + 8748000000.\end{aligned}$$

Computing these coefficients costs a small, constant number of $\mathbb{F}_p$ operations, so the process of computing both j_0 and j_1 from j_p and j_c boils down to solving the quadratic equation, which essentially requires one $\mathbb{F}_{p^2}$ square root, for which we use the algorithm Sqrt described in Section 7.2.1. We summarise the algorithm to take a step in $\mathcal{X}_1(\bar{\mathbb{F}}_p, 2)$ in Algorithm 7.2.

Remark 7.2.2. In practice, in Algorithm 7.2, we check if the input to Sqrt (namely $\delta := \alpha^2 - 4\beta$) is $\mathbb{F}_p$ or in $\mathbb{F}_{p^2} \backslash \mathbb{F}_p$ before computing the square root. If it lies in $\mathbb{F}_p$, we compute this square root in

Algorithm 7.2 Given a pair of j-invariants (j_p, j_c) take a non-backtracking step in $\mathcal{X}_1(\bar{\mathbb{F}}_p, 2)$.

Input: Previous j-invariants $j_p \in \mathbb{F}_{p^2}$, current j-invariant $j_c \in \mathbb{F}_{p^2}$
Output: $j \in \mathbb{F}_{p^2}$ neighbouring j_c with $j \neq j_p$.

```
1: $\alpha \leftarrow j_c(j_c + 1488) - j_p + 162000$
2: $\beta \leftarrow j_p^2 - j_c^2 j_p + 1488 \cdot (j_c^2 + j_c j_p) + 40773375 \cdot j_c - 162000 \cdot j_p + 8748000000$
3: $\delta \leftarrow \alpha^2 - 4\beta$
4: $\delta' \leftarrow \mathsf{Sqrt}(\delta)$
5: $j \leftarrow 2^{-1}(\alpha + \delta')$
6: **return** $j$
```

$\mathbb{F}_p$ and the search for a subfield node has terminated. Otherwise, we run Sqrt on input $\delta \in \mathbb{F}_{p^2} \backslash \mathbb{F}_p$ to obtain $\delta' \in \mathbb{F}_{p^2}$ such that $(\delta')^2 = \delta$, if it exists.

To understand the concrete complexity of Solver and SuperSolver, presented in Section 7.4, it is important to determine the concrete cost of taking a step in $\mathcal{X}_1(\bar{\mathbb{F}}_p, 2)$ using Algorithm 7.2.

Lemma 7.2.3. *The number of $\mathbb{F}_p$-multiplications required to take a step in $\mathcal{X}_1(\bar{\mathbb{F}}_p, 2)$ using Algorithm 7.2 is $cost_{step}$ where*

$$cost_{step} \leq e^2 + 8e + 31 + 2\mathsf{Exp}(c) + \mathsf{Exp}(2^e - 1),$$

e is the maximum positive integer such that $2^e \mid p - 1$, $c = (p - 2^e - 1)/2^{e+1}$ and $\mathsf{Exp}(\lambda)$ is the cost of exponentiation of an element in $\mathbb{F}_p$ by $\lambda \in \mathbb{F}_p$.

Proof. Lines 1, 2 and 3 can be done with at most 16 $\mathbb{F}_p$-multiplications as shown in the implementation accompanying the chapter. As discussed in Section 7.2.1, our implementation of the square root algorithm Sqrt used in Line 4 requires $(e^2 + 8e + 14)$ $\mathbb{F}_p$-multiplications, 2 $\mathbb{F}_p$-exponentiations by c and 1 $\mathbb{F}_p$-exponentiation by $2^e - 1$. As $2^{-1} \in \mathbb{F}_p$ can be precomuted for a fixed p, Line 5 costs 1 $\mathbb{F}_p$-multiplication. $\square$

Remark 7.2.4. Note $e = \lambda \log(p)$ for $0 \leq \lambda \leq 1$. When $\lambda = 1$, we only require 1 exponentiation by $2^e - 1$, and we compute the exponentiation by c instead using a handful of multiplications. Similarly, if $\lambda = 0$, we only compute 2 exponentiations. In this way, the precise cost of taking a step depends heavily on e.

Remark 7.2.5. There is no traditional elliptic curve arithmetic found in either Solver or SuperSolver. All the steps taken within $\mathcal{X}_1(\bar{\mathbb{F}}_p, 2)$ and the rapid inspections conducted in $\mathcal{X}_1(\bar{\mathbb{F}}_p, 2)$ use the modular polynomials. We point out there *may* be specific instances of p where one could perform walks faster than repeatedly solving the quadratic polynomial $\Phi_{2,p}(X, j)$ by, say, employing Vélu's formulas [314] with the optimal strategies of De Feo, Jao, and Plût [123]. For example, with a prime $p = 2^a 3^b - 1$, the price of computing a 2^a-isogeny (i.e., walking through a nodes in $\mathcal{X}_1(\bar{\mathbb{F}}_p, 2)$) in this way may be cheaper than the price of computing a square roots in $\mathbb{F}_{p^2}$ (note that the latter reveals 2 nodes each time). However, computing general $(\prod N_i^{e_i})$-isogenies from kernel elements is much more expensive than N^e-isogenies when $N \in \{2, 3\}$, and one covers fewer nodes in $\mathcal{S}_{p^2}$ per isogeny when the N_i grow larger. Therefore, particularly in the case of general primes, it is unlikely that using Vélu's formulas will be competitive with the binary tree depth-first search in $\mathcal{X}_1(\bar{\mathbb{F}}_p, 2)$. For the primes in Table 7.1, further investigation maybe be warranted.

Remark 7.2.6 (Radical isogenies)**.** Another alternative to solving the quadratic equation that arises from $\Phi_2(X, j_c)/(X - j_p)$ is to instead take steps in $\mathcal{X}_1(\bar{\mathbb{F}}_p, 2)$ using formulas for radical isogenies [73]. For example, given a supersingular Montgomery curve parameterised as $E_A : y^2 = x^3 + Ax^2 + x$ or as $E_\alpha = x(x - \alpha)(x - 1/\alpha)$, one can compute non-backtracking chains of 2-isogenies as either $A \to A' \to A'' \ldots$, or as $\alpha \to \alpha' \to \alpha'' \ldots$, rather than computing the chain of j-invariants $j \to j' \to j'' \ldots$, as we do. Computing the next value in all of these chains requires one square root (which dominates the cost for primes of cryptographic size) and a small handful of additional field operations, the number of which depends on the choice of chain. In the case of computing the chains $A \to A' \to A'' \ldots$ or $\alpha \to \alpha' \to \alpha'' \ldots$, the number of additional operations are fewer (see [68] and [62]) than those which we incur using the modular polynomial, however we have not opted to exploit this minor speed-up for the following reasons. Firstly, it is not true in general that $j(E_A) \in \mathbb{F}_p$ implies $A \in \mathbb{F}_p$ or that $j(E_\alpha) \in \mathbb{F}_p$ implies $\alpha \in \mathbb{F}_p$. Since $j(E_A) = 256(A^2 - 3)^3/(A^2 - 4)$, in general there are six values of A corresponding to a given j. Similarly, since $j(E_\alpha) = 256(\alpha^4 - \alpha^2 + 1)^3/(\alpha^4(\alpha^2 - 1)^2)$, in general there are twelve values of α corresponding to a given j. For large primes it is typically the case that most (or all) of the A's and α's corresponding to a given $j \in \mathcal{S}_p$ are not defined over $\mathbb{F}_p$. Thus, if radical isogenies were used to compute chains of α's or A's in the context of Delfs–Galbraith, we would need to compute a value that determines whether the corresponding j lies in $\mathbb{F}_p$. We note that this can be achieved without inverting the denominators in the expressions for $j(E_\alpha)$ or $j(E_A)$, i.e., $(a + b \cdot \beta)/(c + d \cdot \beta)$ is in $\mathbb{F}_p$ if and only if $ad = bc$ for $a, b, c, d \in \mathbb{F}_p$ and $\mathbb{F}_{p^2} = \mathbb{F}_p(\beta)$. For these reasons, the original Delfs–Galbraith walk in $\mathcal{X}_1(\bar{\mathbb{F}}_p, 2)$ is likely to save a small, fixed number of multiplications per 2-isogeny by computing chains of A's or α's instead of j's. In future work, it would therefore be interesting to conduct a more in-depth exploration of the benefits of using radical isogenies in Solver. We emphasize, however, that when invoking our fast subfield root detection in the sections that follow, it is critical (for Algorithm 7.4) that the j-invariants of each node are computed explicitly, so that the higher degree-$D_{N,1}$ modular polynomials can be used to probe for N-isogenous subfield neighbours. This subsequent computation of the j-invariant seems to require an additional field exponentiation (we could not see a way to merge the square roots and inversions into one exponentiation in these instances), which would kill the potential advantage of radical isogenies in the optimised SuperSolver.

Remark 7.2.7 (Alternative modular functions)**.** There are several well-known modular functions other than the j-function – see [302]. A natural question in the context of this paper is whether any such functions can be used to make the search for subfield nodes in supersingular isogeny graphs more efficient. For example, the modular polynomials for Weber's f-function [301] are the same degree as those of the j-function, but have *much* smaller coefficients, many of which are zero. If these more compact modular polynomials could be used in the same way as those for the j-function, the practical gains would be significant. However, their applicability in the context of SuperSolver appears to be hampered by reasons similar to those discussed in Remark 7.2.6. Weber's f is related to j via $j = (f^{24} - 16)^3/f^{24}$, meaning there can be as many as 72 f's corresponding to a single j-invariant, and it is not true in general that given $j \in \mathbb{F}_p$, the corresponding Weber invariant f lies in $\mathbb{F}_p$. Although this makes the Weber polynomials unreliable replacements in the context of the SuperSolver algorithm, our search for alternative modular functions that would be compatible with SuperSolver was far from exhaustive, and it is likely that the j-function is not optimal across all of them. We leave any further investigations in this direction as future work.

7.2.3 The depth-first search in $\mathcal{X}_1(\bar{\mathbb{F}}_p, 2)$

Repeating the process described above allows us to perform the search for subfield nodes in $\mathcal{X}_1(\bar{\mathbb{F}}_p, 2)$. Recalling from Section 4.2.1 that $\mathcal{X}_1(\bar{\mathbb{F}}_p, 2)$ is an expander graph, there is a short path (of length approximately $\log(p)$) between any two nodes. Therefore, we use a depth-first search in a binary tree with d levels to navigate the supersingular isogeny graph. We write $j_{m,n}$ for the n-th node at level m, where $0 \leq m \leq d$ and $0 \leq n \leq 2^m - 1$. The first three levels are depicted in Figure 7.1.

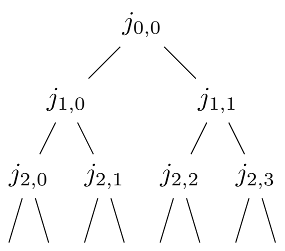


Figure 7.1: Levels 0, 1, and 2 of the binary tree in the depth first search of $\mathcal{X}_1(\bar{\mathbb{F}}_p, 2)$.

We initialise the root node $j_{0,0}$ as the target $j \in \mathbb{F}_{p^2} \backslash \mathbb{F}_p$, and set $j_{1,0}$ and $j_{1,1}$ as two of its three neighbours[3] in $\mathcal{X}_1(\bar{\mathbb{F}}_p, 2)$. The depth first search starts by setting $j_c = j_{1,0}$ and $j_p = j_{0,0}$. We then solve the quadratic equation above to obtain $j_{2,0}$ and $j_{2,1}$, and repeat this procedure with $j_c = j_{i+1,0}$ and $j_p = j_{i,0}$ for $1 \leq i \leq d-1$ until the leftmost leaf $j_{d,0}$ is computed, and the `path` stack is fully initialised as

$$\texttt{path} = [j_{0,0},\ j_{1,0}, \ldots, j_{d-1,0},\ j_{d,0}].$$

To avoid any waste, we also maintain a stack of the other solution to the quadratic equations that were computed along the way, which we call sibling nodes

$$\texttt{siblings} = [j_{1,1}, \ldots, j_{d-1,1},\ j_{d,1}].$$

The algorithm then proceeds back up the levels by popping `path` until its last element is the root of a subtree that has not been checked in its entirety. At this point `siblings` is popped and pushed into `path`. When the last element of `path` is the root of a subtree that has not been exhausted, we initialise the process of solving quadratic equations, pushing one of the two solutions into `path` and the other into `siblings` until `path` contains $d+1$ elements. Each time the quadratic equation solver is called, the two roots (i.e., j-invariants) are immediately checked; if either of them lie in $\mathbb{F}_p$, it is added to `path` and the process is terminated. Otherwise, the process is repeated recursively

[3]Initially we do not have a j_p, so all three neighbours can be computed using generic root finding; our code does this during the setup phase.

until `path` $= [j_{0,0}]$, in which case the $2^{d+1} - 1$ nodes in the tree have been exhausted without finding a solution. To guarantee that a solution is found, one could increase d and start again, but our code proceeds by simply storing the first (leftmost) leaf and its parent in separate memory so that the process can restart here and avoid recomputing any prior j's. As Delfs and Galbraith point out, setting the depth $d = \frac{1}{2}\log p$ should be enough. Since the number of nodes in the tree is 2^d, increasing d by ϵ makes the failure probability diminish by $1/2^\epsilon$. Setting $\epsilon = 10$ was sufficient in all of our experiments.

Finally, this process parallelises perfectly [132, §4]. For P processors, one can simply compute a binary tree of depth $\lceil \log P \rceil$ during setup and distribute P of the leaf nodes as individual starting points.

7.2.4 Concrete complexity

Table 7.2 reports on experiments conducted using Solver, the optimised instantiation of the traditional Delfs–Galbraith walk. For each bitlength between 21 and 40, we solved 10,000 instances of the subfield search. In each case we chose 100 random primes and, for each prime, 100 pseudo-random j-invariants in $\mathcal{S}_{p^2}$. The numbers in each column report the averages (as base-2 logarithms) of these search complexities. We visualise this information in Figure 7.2.

In all cases the number of $\mathbb{F}_p$-multiplications is found to be

$$\#(\mathbb{F}_p \text{ mults.}) = c \cdot \sqrt{p} \cdot \log p,$$

with $0.75 \leq c \leq 1.05$. In Section 7.4.4, we shed more light on the concrete complexity of both Solver and SuperSolver and put these experimental observations on theoretical footing.

Bitlengths of primes p	21	22	23	24	25	26	27	28	29	30
Av. number of nodes visited	8.8	9.4	10.0	10.3	10.9	11.4	11.9	12.3	13.1	13.5
Av. number of $\mathbb{F}_p$-multiplications	14.5	15.0	15.7	16.0	16.7	17.2	17.8	18.2	19.0	19.5
Bitlengths of primes p	**31**	**32**	**33**	**34**	**35**	**36**	**37**	**38**	**39**	**40**
Av. number of nodes visited	13.5	14.2	14.7	15.3	15.8	16.3	17.1	17.3	17.6	18.1
Av. number of $\mathbb{F}_p$-multiplications	19.5	20.5	20.8	21.3	21.9	22.4	23.2	23.6	24.1	24.6

TABLE 7.2: The concrete cost of the subfield search phase of the Delfs–Galbraith over small fields of various bitlengths. Further explanation in text.

7.3 Fast subfield root detection

In this section we derive a method for determining whether a polynomial $f(X) = a_n X^n + ... + a_1 X + a_0 \in \mathbb{F}_{q^d}[X]$ with $d \geq 2$ has a root lying in the subfield $\mathbb{F}_q$, where q is a power of prime p. Though this can be achieved by factoring the polynomial, the methods described in Section 7.1.1 become too costly for our purposes; the number of $\mathbb{F}_q$ operations required depends on the size of q, which hampers their relative efficiency as q grows large. Our aim in this section is to detail a much faster algorithm that detects whether a root lies in a subfield and show that the number of

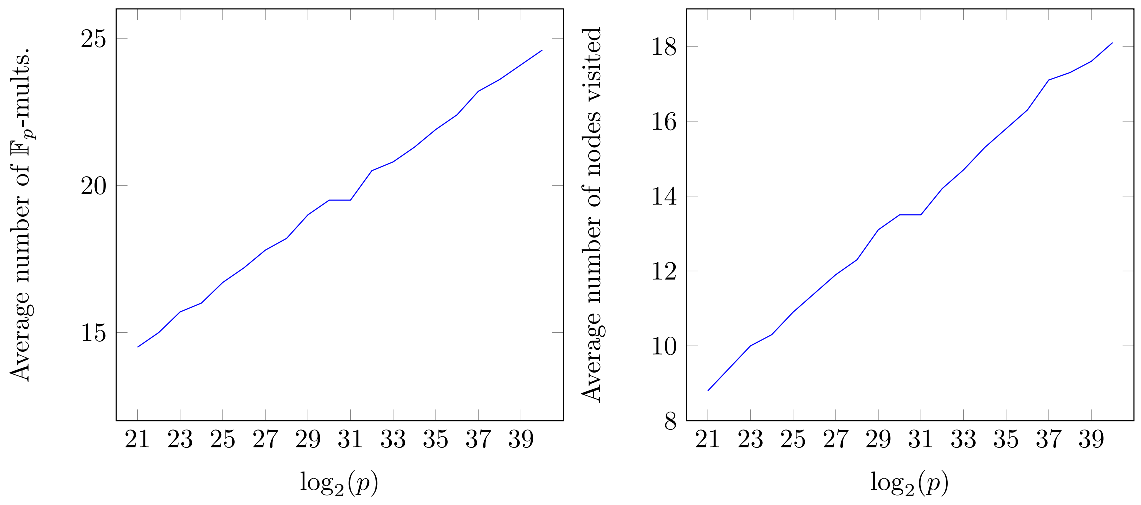


FIGURE 7.2: Visualising the data from Table 7.2 on the concrete cost of the subfield search phase of the Delfs–Galbraith over small fields of various bitlengths. The graph on the left gives the average number of $\mathbb{F}_p$-mulitplications required to run the subfield search in the Delfs–Galbraith algorithm for various primes p, whereas the graph on the right shows the average number of nodes visited in this search. See more information in text.

$\mathbb{F}_q$ operations required by our algorithm only depends on the degree of f and the degree of the extension d.

As the algorithms in this section may be of independent interest, we leave them as general as possible before specialising back to the application at hand in Section 7.4. The results up to Proposition 7.3.3 are presented for general finite field extensions of the form $\mathbb{F}_{q^d}/\mathbb{F}_q$, but we will later specialise to the quadratic extensions of prime fields, i.e., where $q = p$ and $d = 2$. The inversion-free gcd in Algorithm 7.3 is derived for an arbitrary polynomial ring $\Bbbk[X]$, but we will only need to use it in $\mathbb{F}_p[X]$.

For a polynomial in $\mathbb{F}_{q^d}[X]$, we will reduce the problem of detecting a root in $\mathbb{F}_q$ to computing the greatest common divisor of d related polynomials $g_1, ..., g_d$. In the case where $d > 2$, we will need to compute the gcd of more than two polynomials. This can be done by recursively computing the gcd of two polynomials and using the following identity:

$$\gcd(g_1, g_2, ..., g_d) = \gcd(g_1, \gcd(g_2, ..., g_d)). \tag{7.1}$$

As we aim to minimise the number of $\mathbb{F}_q$ multiplications needed to compute the gcd, we construct these polyomials so that they are defined over $\mathbb{F}_q$ (rather than over $\mathbb{F}_{q^d}$). To achieve this, we will need two results. The first is a theorem by Lidl and Niederreiter [220, Theorem 2.24], which we reproduce here for completeness.

Theorem 7.3.1. *Let $\Bbbk'$ be a finite extension of a finite field $\Bbbk$, both considered as vector spaces over $\Bbbk$. Then the linear transformations from $\Bbbk'$ into $\Bbbk$ are exactly the mappings $L_\beta(\alpha)$, for $\beta \in \Bbbk'$, where $L_\beta(\alpha) = \mathrm{Tr}_{\Bbbk'/\Bbbk}(\beta\alpha)$ for all $\alpha \in \Bbbk'$, where Tr is the trace map of the extension $\Bbbk'/\Bbbk$. Furthermore, we have $L_\beta \neq L_\gamma$ whenever β, γ are distinct elements of $\Bbbk'$.*

The second result we will need is the following lemma.

Lemma 7.3.2. *For $n \in \mathbb{N}$, let $f_1, ..., f_n \in \mathbb{F}_{q^d}[X]$ be polynomials and $A \in \mathrm{GL}_n(\mathbb{F}_{q^d})$. Defining $(g_1, ..., g_n) := A \cdot (f_1, ..., f_n)$, we have*

$$\gcd(f_1, ..., f_n) = \gcd(g_1, ..., g_n).$$

Proof. If a polynomial $h \in \mathbb{F}_{q^d}[X]$ divides $f_1, ..., f_n$, then h divides any linear combination of the $f_1, ..., f_n$. Therefore, h divides $g_1, ..., g_n$. Since A is invertible, by swapping the roles of g_i and f_i we see that the converse holds. □

We are now ready to present the main result of this section.

Proposition 7.3.3. *For some $d \geq 2$, let π be the q-power Frobenius endomorphism in $\mathrm{Gal}(\mathbb{F}_{q^d}/\mathbb{F}_q)$ and consider a polynomial $f(X) = a_nX^n + ... + a_1X + a_0 \in \mathbb{F}_{q^d}[X]$. Define the action of π on such a polynomial by $\pi(f) := \pi(a_n)X^n + ... + \pi(a_1)X + \pi(a_0) \in \mathbb{F}_{q^d}[X]$. Let β be a primitive element of the extension $\mathbb{F}_{q^d}/\mathbb{F}_q$, in the sense that the field extension is generated by a single element β, i.e., $\mathbb{F}_q(\beta) = \mathbb{F}_{q^d}$. For $i = 1, .., d$, define the following polynomials over $\mathbb{F}_{q^d}$:*

$$g_i := \sum_{j=0}^{d-1} \pi^j(\beta^{i-1}f).$$

Then $g_i(X) \in \mathbb{F}_q[X]$, and $\gcd(g_1, ..., g_d)$ divides f. In particular, if $\gcd(g_1, ..., g_d)$ is of degree 1, then f has a root in $\mathbb{F}_q$. Furthermore, if $\gcd(g_1, ..., g_d) = 1$, then $f(X)$ does not have any roots in $\mathbb{F}_q$.

Proof. Using the notation in Theorem 7.3.1, we have

$$\begin{aligned} g_i(X) &= [(\beta^{i-1}a_n + \pi(\beta^{i-1}a_n) + ... + \pi^{d-1}(\beta^{i-1}a_n))X^n + ... + (\beta^{i-1}a_0 + ... + \pi^{d-1}(\beta^{i-1}a_0))] \\ &= \sum_{m=0}^{n} L_{\beta^{i-1}}(a_m)X^m. \end{aligned}$$

By Theorem 7.3.1, for all $i = 1, ..., d$ and $m = 0, \ldots n$, we have $L_{\beta^{i-1}}(a_m) \in \mathbb{F}_q$, implying that $g_i(X) \in \mathbb{F}_q[X]$. Setting $(d \times d)$ matrix A to be

$$A = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ \beta & \pi(\beta) & \cdots & \pi^{d-1}(\beta) \\ \vdots & \vdots & \ddots & \vdots \\ \beta^{d-1} & \pi(\beta^{d-1}) & \cdots & \pi^{d-1}(\beta^{d-1}) \end{bmatrix} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ \beta & \beta^q & \cdots & \beta^{q^{d-1}} \\ \vdots & \vdots & \ddots & \vdots \\ \beta^{d-1} & (\beta^{d-1})^q & \cdots & (\beta^{d-1})^{q^{d-1}} \end{bmatrix},$$

we have $(g_1, ..., g_d) := A \cdot (f, \pi(f) ..., \pi^{d-1}(f))$. As for Vandermonde matrices [182, §6.2], we find $\det(A) = \prod_{0 \leq i < j \leq d-1} (\beta^{q^j} - \beta^{q^i})$, which is non-zero for β a primitive element of the extension $\mathbb{F}_{q^d}/\mathbb{F}_q$ and so $A \in \mathrm{GL}_d(\mathbb{F}_{q^d})$. By Lemma 7.3.2, we have

$$\gcd(f, \pi(f), ..., \pi^{d-1}(f)) = \gcd(g_1, ..., g_d),$$

therefore $\gcd(g_1, ..., g_d) \mid f$. If $\gcd(g_1, ..., g_d)$ is of degree 1, then $(X - r) \mid f$ for some $r \in \mathbb{F}_q$, and so f has a root in $\mathbb{F}_q$.

We further note that $\gcd(f, \pi(f), ..., \pi^{d-1}(f))$, and therefore $\gcd(g_1, ..., g_d)$, is precisely the largest divisor of f that is defined over $\mathbb{F}_q$. As a result, if $\gcd(g_1, ..., g_d) = 1$, then $f(X)$ does not have any roots in $\mathbb{F}_q$. □

7.3.1 Detecting subfield nodes

The proof of Proposition 7.3.3 tells us that $\gcd(g_1, ..., g_d)$ is precisely the largest divisor of $f \in \mathbb{F}_{q^d}[X]$ that is defined over $\mathbb{F}_q[X]$. In our target application of searching for subfield nodes in large supersingular isogeny graphs, i.e., when $d = 2$ and $q = p$, we will most commonly encounter $\gcd(g_1, g_2) = 1$, which immediately rules out subfield neighbours in the N-isogeny graph. Non-trivial gcd's will, with overwhelmingly high probability, be of degree 1 and reveal a single subfield node; this is why our implementation of Algorithm 7.3 below terminates and returns `true` when the degree of the gcd is 1.

For large supersingular isogeny graphs, the only way for the degree of $\gcd(g_1, g_2)$ to be larger than 1 is when a given j-invariant is N-isogenous to multiple subfield nodes, or when a given j-invariant is N-isogenous to conjugate j-invariants in $\mathbb{F}_{p^2}$. A real-world attack should check any non-trivial gcd, since either of these scenarios are a win for the cryptanalyst; the latter case reveals information about the secret endomorphism ring of the target isomorphism class (see [216, §5.3]), and the former case gives multiple solutions to the subfield search problem.

In our scenario where $d = 2$, we see that $\pi(\beta) + \beta = 0$, meaning that $\pi^k(\beta) = (-1)^k\beta$. As a result, to detect a subfield root, we compute $\gcd(g_1, \beta g_2)$ where $g_1 = f + \pi(f)$ and $g_2 = f - \pi(f)$. In this case we do not need to calculate any more powers of β, and we only need to do one gcd computation.

7.3.2 Inversion-free polynomial GCD

To complete the detection of roots in a subfield, we must compute the gcd of polynomials in polynomial ring $\Bbbk[X]$, where $\Bbbk$ is a field. In Algorithm 7.3, we modify Euclid's polynomial-adapted algorithm [289, §17.3] to compute the gcd of two polynomials $g, h \in \Bbbk[X]$ while avoiding inversions in $\Bbbk$. We use $\mathrm{LC}(f)$ to denote the leading coefficient of the polynomial f. Note that, for the purposes of incorporating it into our target application of subfield searching in the next section, the algorithm outputs the boolean `true` when the gcd has degree 1 in $\Bbbk[X]$.

Proposition 7.3.4. *Given input $g, h \in \Bbbk[X]$ such that $\deg g \geq \deg h$, Algorithm 7.3 terminates using at most*

$$\frac{1}{2}(\deg g + \deg h - 1)(\deg g + \deg h + 6)$$

multiplications in $\Bbbk$.

Proof. Line 1 incurs at most $\deg g + \deg h + 2$ multiplications in $\Bbbk$. Setting $r_0 := r, s_0 := s$, we define this to be loop 0. For $i \geq 1$, we denote by r_i, s_i (where $\deg s_i \geq \deg r_i$) the polynomials in loop i of Line 2 to Line 8. Using this notation, we move to Line 9 when $\deg r_i \leq 1$ or $r_i = s_i$. Now, in loop $i \geq 1$ we replace r_i by $r_i - X^{\deg r_i - \deg s_i} s_i$, meaning $\deg r_{i-1} - \deg r_i \geq 1$, and compute $r_i \cdot LC(s_i)$ and $s_i \cdot LC(r_i)$. This requires $\deg r_i + \deg s_i + 2$ multiplications in $\Bbbk$. In the worst case,

Algorithm 7.3 InvFreeGCD: Inversion-free gcd

Input: Polynomials $g, h \in \Bbbk[X]$, such that $\deg g \geq \deg h$.
Output: A boolean indicating if g, h have a non-trivial gcd of degree 1.

```
1: Initialise $r, s \leftarrow \mathrm{LC}(h) \cdot g, \mathrm{LC}(g) \cdot h$
2: while $\deg r \geq 1$ and $r \neq s$ do
3:     $r \leftarrow r - X^{\deg r - \deg s} \cdot s$
4:     $r, s \leftarrow \mathrm{LC}(s) \cdot r, \mathrm{LC}(r) \cdot s$
5:     if $\deg r \leq \deg s$ then
6:         $r, s \leftarrow s, r$
7:     end if
8: end while
9: return $\neg(\deg r = 1$ and $r \neq s)$
```

we have $\deg r_{i-1} - \deg r_i = 1$ for $i \geq 1$, where the number of multiplications will decrease by exactly 1 after each loop. In the final loop we have $\deg r_i, \deg s_i = 1$, so we compute 4 multiplications in $\Bbbk$. In summary, in the worst case we begin with $\deg g + \deg h + 2$ multiplications, decreasing by 1 until we get to 4. Therefore, the total number of multiplications is at most $\sum_{n=4}^{\deg g + \deg h + 2} n$, which is the bound in the statement of the proposition. $\square$

In summary, Proposition 7.3.3 shows that detecting subfield roots of $f \in \mathbb{F}_{q^d}[X]$ amounts to computing the gcd of d related polynomials in $\mathbb{F}_q[X]$. We showed that computing this gcd is simpler when $d = 2$. Proposition 7.3.4 gives an upper bound on the number of $\mathbb{F}_q$ multiplications required to compute such a gcd in $\mathbb{F}_q[X]$. In the next section we use these tools to build a faster algorithm for finding subfield nodes in supersingular isogeny graphs.

7.4 SuperSolver

SuperSolver is an algorithm which, given two j-invariants in $\mathbb{F}_{p^2}$ corresponding to two supersingular curves $E_1/\mathbb{F}_{p^2}$ and $E_2/\mathbb{F}_{p^2}$, will, on average, solve the supersingular isogeny problem with lower concrete complexity than the traditional Delfs–Galbraith Solver algorithm described in Example 6.3.2. As in the Delfs–Galbraith algorithm, SuperSolver takes non-backtracking walks in $\mathcal{X}_1(\bar{\mathbb{F}}_p, 2)$ until they hit a j-invariant in $\mathbb{F}_p$, i.e., a j in the special subset X_1 as defined in Definition 6.3.1. However, at each step of the random walk, SuperSolver also inspects $\mathcal{X}_1(\bar{\mathbb{F}}_p, N)$, for carefully chosen $N > 2$, to efficiently detect whether j has any N-isogenous neighbours in X_1. Traditionally, inspecting $\mathcal{X}_1(\bar{\mathbb{F}}_p, N)$ for a subfield neighbour requires fully factoring a degree-$D_{N,1}$ polynomial (where we recall the definition of $D_{N,1}$ from Equation (2.7)) and determining whether any of the roots lie in $\mathbb{F}_p$. Performing this for each N would require $O(N^3 + 2N^2 \log p)$ operations in $\mathbb{F}_{p^2}$ using the modified Cantor–Zassenhaus algorithm (see Section 7.1.1), which is prohibitively costly.

Following the results from Section 7.3, however, SuperSolver conducts the inspection of $\mathcal{X}_1(\bar{\mathbb{F}}_p, N)$ with $O(N^2)$ multiplications in $\mathbb{F}_p$. We make this count precise later in this section. Crucially, the number of $\mathbb{F}_p$ operations is no longer dependent on the size of p, and this means that as p grows large, the set of N's that are optimal to use also grows, and the more profitable (relatively speaking)

SuperSolver becomes.

We reiterate that, although both Solver and SuperSolver return the full isogeny between $E_1/\mathbb{F}_{p^2}$ and $E_2/\mathbb{F}_{p^2}$, our discussion focusses on the bottleneck problem of finding an isogeny from $E_1/\mathbb{F}_{p^2}$ (resp. $E_2/\mathbb{F}_{p^2}$) to $E_1'/\mathbb{F}_p$ (resp. $E_2/\mathbb{F}_p$). If, at some node j, we detect an N-isogenous neighbour in $\mathbb{F}_p$, SuperSolver will then factorise the degree-$D_{N,1}$ polynomial $\Phi_{N,p}(X,j)$ to determine the subfield j-invariant. We view this as a post-computation step, since we are only interested in the concrete complexity of the average step taken in the walk (which we assume does not find a subfield node). Note that the paths between $E_1/\mathbb{F}_{p^2}$ and $E_2/\mathbb{F}_{p^2}$ returned by both Solver and SuperSolver both look the same: in general, both start and finish with a chain of 2-isogenies that is connected in the middle by a chain of different prime-degree isogenies. The main difference, as the results in Section 7.6 illustrate, is that 2-isogeny chains returned by SuperSolver at each end are *much* shorter.

Recall that in the original Delfs–Galbraith algorithm, each step consists of finding the roots of a quadratic equation in $\mathbb{F}_{p^2}[X]$, which reveals two neighbouring nodes in $\mathcal{X}_1(\bar{\mathbb{F}}_p, 2)$. In SuperSolver, after forming a list of carefully chosen $N > 2$, each step will also include the rapid inspection of $\mathcal{X}_1(\bar{\mathbb{F}}_p, N)$ for every N in this list. Though the inspection of the neighbours in $\mathcal{X}_1(\bar{\mathbb{F}}_p, N)$ increases the total number of $\mathbb{F}_p$-multiplications at each step, more nodes are checked. We first describe the process of taking a step in SuperSolver, and then move to describing how to choose the list of $N > 2$ in order to minimise the number of $\mathbb{F}_p$-multiplications per node inspected.

Remark 7.4.1 (Odd N only)**.** With the exception of the leaf nodes in the last level of the binary tree, it is redundant to perform rapid node inspections in $\mathcal{X}_1(\bar{\mathbb{F}}_p, 2N)$ if rapid inspections in $\mathcal{X}_1(\bar{\mathbb{F}}_p, N)$ are also part of the routine, since the latter inspections will detect (or exclude) subfield nodes at the next level of the walk down the tree. We therefore find it optimal to only include odd N_i in the lists constructed at the end of this section. Note that there is no redundancy in including odd composite N_i's in our lists, even if they have proper divisors that are also in the list.

7.4.1 Rapid inspection of the N-isogenous neighbours

We describe the algorithm NeighbourInFp given in Algorithm 7.4. On input of N, $j \in \mathbb{F}_{p^2}$ and p, it outputs `true` if j is N-isogenous to a $j' \in \mathbb{F}_p$, and `false` otherwise.

Recall that the degree of $\Phi_{N,p}$ in X and Y is $D_{N,1}$. The first subroutine of NeighbourInFp is EvalModPolyj(N, j, p): it evaluates $\Phi_{N,p}(X,Y)$ at $Y = j$ by computing $j^2, ..., j^{D_{N,1}}$, and then multiplying these by the corresponding coefficients of $\Phi_{N,p}$, returning the coefficients $a_{D_{N,1}}, ..., a_0$ of X in $\Phi_{N,p}(X,j)$. Note that, since we typically have a list of multiple N, i.e., $N_1 < \cdots < N_k$, the powers of j (up to D_{N_k}) are computed once-and-for-all at every j, and recycled among the $N_i < N_k$. We follow Section 7.3 to detect whether $\Phi_{N,p}(X,j) \in \mathbb{F}_{p^2}[X]$ has a root in $\mathbb{F}_p$. Letting $\beta \in \mathbb{F}_{p^2}$ be such that $\mathbb{F}_{p^2} = \mathbb{F}_p(\beta)$, we first compute the related polynomials

$$\begin{aligned} g_1 &:= (1/2) \cdot [\Phi_{N,p}(X,j) + \pi(\Phi_{N,p}(X,j))] \qquad \text{and} \\ g_2 &:= (-\beta/2) \cdot [\Phi_{N,p}(X,j) - \pi(\Phi_{N,p}(X,j))], \end{aligned}$$

where $\pi \in \mathrm{Gal}(\mathbb{F}_{p^2}/\mathbb{F}_p)$ is the Frobenius endomorphism. By Proposition 7.3.3, we have $g_1, g_2 \in$

$\mathbb{F}_p[X]$ and

$$\deg\left(\gcd(g_1, g_2)\right) = 1 \implies \Phi_{N,p}(X, j) \text{ has a root in } \mathbb{F}_p.$$

We then complete the inspection of $\mathcal{X}_1(\overline{\mathbb{F}}_p, N)$ by using Algorithm 7.3 to calculate $\gcd(g_1, g_2)$. If $\gcd(g_1, g_2) \neq 1$, then (for large enough p) it is overwhelmingly likely that $\deg\left(\gcd(g_1, g_2)\right) = 1$, as detailed in the following remark.

Remark 7.4.2. As discussed in Section 7.3.1, if E has a neighbour with j-invariant in $\mathbb{F}_p$, then Algorithm 7.4 will return `true` with high probability. Indeed, otherwise E is N-isogenous to multiple subfield curves, which (heuristically) has probability at most $O(p^{-1/4})$, or E is N-isogenous to an elliptic curve with j-invariant $\pi(j(E)) \in \mathbb{F}_{p^2}$, which (assuming GRH) has probability $O(p^{-1/2})$ [148, Theorem 3.9].

Due to this, our implementation uses the degree of the gcd as the criterion for terminating the subfield search. Another possibility is to terminate whenever $\gcd(g_1, g_2)$ is non-constant, and then to inspect the higher degree gcd according to the two possible scenarios discussed in Section 7.3.1.

Note that if we have a polynomial $f(X) = a_n X^n + a_{n-1} X^{n-1} + \ldots + a_1 X + a_0 \in \mathbb{F}_{p^2}[X]$ then

$$\frac{1}{2}[f + \pi(f)] = \mathrm{Re}(a_n)X^n + \mathrm{Re}(a_{n-1})X^{n-1} + \ldots + \mathrm{Re}(a_1)X + \mathrm{Re}(a_0) \in \mathbb{F}_p[X],$$
$$\frac{-\beta}{2}[f - \pi(f)] = \mathrm{Im}(a_n)X^n + \mathrm{Im}(a_{n-1})X^{n-1} + \ldots + \mathrm{Im}(a_1)X + \mathrm{Im}(a_0) \in \mathbb{F}_p[X],$$

where, for $a + b\beta \in \mathbb{F}_{p^2}$, $\mathrm{Re}(a + b\beta) = a$ and $\mathrm{Im}(a + b\beta) = b$, in analogy with the notation used for complex numbers. As a result, we can obtain g_1 and g_2 directly from $f = \Phi_{N,p}$ by computing

$$g_1 = X^{D_{N,1}} + \ldots + \mathrm{Re}(a_0), \quad \text{and} \quad g_2 = \mathrm{Im}(a_{D_{N,1}-1})X^{D_{N,1}-1} + \ldots + \mathrm{Im}(a_0).$$

This avoids having to compute any $\mathbb{F}_{p^2}$-multiplications to calculate the related polynomials g_1, g_2.

Algorithm 7.4 NeighbourInFp: Detect whether $j \in \mathbb{F}_{p^2}$ is N-isogenous to a $j' \in \mathbb{F}_p$

Input: Positive integer $N \geq 2$, j-invariant j, and prime p.
Output: A boolean indicating if j-invariant j has an N-isogenous neighbour $j' \in \mathbb{F}_p$.

```
1: $a_{D_{N,1}}, \ldots, a_0 \leftarrow$ EvalModPolyj$(N, j, p)$
2: $g_1 \leftarrow X^{D_{N,1}} + \ldots + \mathrm{Re}(a_0)$
3: $g_2 \leftarrow \mathrm{Im}(a_{D_{N,1}-1})X^{D_{N,1}-1} + \ldots + \mathrm{Im}(a_0)$
4: return InvFreeGCD$(g_1, g_2)$
```

Remark 7.4.3. This author would like to thank the examiners for pointing out the algorithm by Stehlé and Zimmerman [295, Figure 7] for computing the GCD of two n-bit integers in quasi-linear time. More precisely, the maximum of the number of bit operations performed by this algorithm is $\approx \frac{19}{4} M(n) \log(n)$, where $M(n)$ is the asymptotic time required to multiply two n-bit integers [295, Theorem 12]. To improve the performance of NeighbourInFp, it would be interesting to adapt Stehlé and Zimmerman's algorithm to compute polynomial GCDs rather than InvFreeGCD.

7.4.2 Cost of inspecting the N-isogeny graph

In this section, we determine the cost of inspecting the N-isogeny graph using NeighbourInFp, denoted by cost_N.

For our application in SuperSolver, at each step the algorithm NeighbourInFp is run for each N_i in a list $\mathcal{N} = \{N_1, \ldots, N_k\}$. As remarked in Section 7.4.1, for a fixed j, the powers $j^2, \ldots, j^{D_{N_k}}$ are computed once-and-for-all, and recycled among the $N_i < N_k$. Therefore, we disregard this cost when determining the cost for each N.

As such, constructing $g_1, g_2 \in \mathbb{F}_p[X]$ following Lines 1 to 3 in Algorithm 7.4 requires at most $2\mathsf{mons}(N)$ multiplications in $\mathbb{F}_p$, where $\mathsf{mons}(N)$ is the number of monomials of the form X^aY^b for $a, b \in \mathbb{Z}_{\geq 0}$ in $\Phi_{N,p}(X,Y)$ such that $b \neq 0$. Here we assume that multiplying a $\mathbb{F}_{p^2}$ element with a $\mathbb{F}_p$ element is equivalent to 2 $\mathbb{F}_p$-multiplications. By Proposition 7.3.4, as $g_1, g_2 \in \mathbb{F}_p[X]$, we compute InvFreeGCD (g_1, g_2) with at most $(2D_{N,1}^2 + 3D_{N,1} - 5)$ $\mathbb{F}_p$-multiplications. Therefore, for a fixed N, the number of $\mathbb{F}_p$-multiplications needed to inspect $\mathcal{X}_1(\bar{\mathbb{F}}_p, N)$ with NeighbourInFp is

$$\begin{aligned}\text{cost}_N &:= \#\mathbb{F}_p \text{ multiplications needed to inspect } N\text{-isogenous neighbours}\\ &\leq 2D_{N,1}^2 + 3D_{N,1} - 5 + 2\mathsf{mons}(N),\end{aligned}$$

and $\text{nodes}_N := D_{N,1}$, both of which depend only on N. This means that, for each N, the ratio $R_N := \frac{\text{cost}_N}{\text{nodes}_N}$ can be computed once for all primes. In Table 7.3, we present the N with the lowest such ratio, ordering them in increasing order from left to right. We present both the theoretical upper bound for R_N given above, and its value computed experimentally. We observe that our bound is exact.

N	3	5	7	9	11	13	17	19	15	23
$D_{N,1}$	4	6	8	12	12	14	18	20	24	24
$\mathsf{mons}(N)$	13	31	57	133	133	183	307	381	553	553
Theoretical R_N	16.3	24.5	32.6	48.8	48.8	56.8	72.8	80.9	96.9	96.9
Experimental R_N	16.3	24.5	32.6	48.8	48.8	56.8	72.8	80.9	96.9	96.9
N	25	29	21	31	27	37	41	43	33	35
$D_{N,1}$	30	30	32	32	36	38	42	44	48	48
$\mathsf{mons}(N)$	871	871	993	993	1261	1407	1723	1893	2257	2257
Theoretical R_N	120.9	120.9	128.9	128.9	144.9	152.9	168.9	177.0	192.9	192.9
Experimental R_N	120.9	120.9	128.9	128.9	144.9	152.9	168.9	177.0	192.9	192.9

TABLE 7.3: The cost of inspecting N-isogenous neighbours for N ordered by increasing cost from left to right. We give both the theoretical upper bound for the ratio R_N, and the $\mathbb{F}_p$-multiplications per node computed experimentally, showing that they match.

Remark 7.4.4. The ratio R_N depends only on $D_{N,1}$. Therefore, if $N, M \in \mathbb{N}$ have $D_{N,1} = D_{M,1}$, their ratio will be the same, namely $R_N = R_M$.

The important takeaway from Table 7.3 is that the number of $\mathbb{F}_p$-multiplications incurred by our algorithm does not grow with p. This count is fixed and depends only on N. Looking back at the root solving algorithms in Section 7.1.1, we see a stark difference in expected performance. Those algorithms have many constants hidden by the big-O, have a leading N^3 term (compared to our N^2 term), and, importantly, the number of field operations they incur grows as the field grows due to their implicit dependency on $\log(p)$. Moreover, the complexities cited are for *probabilistic* root finding algorithms. Their deterministic variants have even worse complexities [289, §20.6].

Remark 7.4.5. An alternative method for inspecting $\mathcal{X}_1(\bar{\mathbb{F}}_p, N)$, pointed out to this author by Travis Morrison, is to compute $\gcd(\Phi_N(j,Y), Y^p - Y)$ as follows. Compute the image of Y^p in $\mathbb{F}_{p^2}[Y]/(\Phi_N(j,Y))$, say $f(Y)$, via a square-and-multiply algorithm. Then

$$\gcd(\Phi_N(j,Y), Y^p - Y) = \gcd(\Phi_N(j,Y), f(Y) - Y),$$

which costs $O(\log p)$ multiplications in $\mathbb{F}_p$, assuming the degree of $\Phi_N(j,Y)$ is $O(1)$. Though this is more efficient than computing roots of the modular polynomial for large N, it is still dependent on p. As such, our NeighbourInFp is more efficient, with its advantage growing as p grows. A more in-depth analysis at the concrete improvement of NeighbourInFp over this alternative method is left as future work.

7.4.3 Choosing the N_i to minimise the cost of a step

We consider the cost of each step in SuperSolver, which we denote by $\mathrm{cost}_{\mathcal{N}}$, running detection on a list $\mathcal{N} = \{N_1, \ldots, N_k\}$ with Algorithm 7.4. Let $\mathrm{cost}_{\mathrm{powers}}$ be the cost of computing the powers $j^2, \ldots, j^{D_{N_k}}$ of j-invariant j. If $\mathcal{N} = \{\}$, then $\mathrm{cost}_{\mathrm{powers}} = 0$. Otherwise, $\mathrm{cost}_{\mathrm{powers}} \le D_{N_k,1} - 1$. The cost $\mathrm{cost}_{\mathcal{N}}$ is given by

$$\mathrm{cost}_{\mathcal{N}} = \mathrm{cost}_{\mathrm{step}} + \mathrm{cost}_{\mathrm{powers}} + \sum_{N \in \mathcal{N}} \mathrm{cost}_N. \tag{7.2}$$

We seek to find a list that minimises the ratio $R_{\mathcal{N}} = \frac{\mathrm{cost}_{\mathcal{N}}}{\mathrm{nodes}_{\mathcal{N}}}$, where $\mathrm{nodes}_{\mathcal{N}}$ is the total number of nodes revealed using set $\mathcal{N}$ at each step.

Recall from Table 7.3 that the N's that give the cheapest cost per node inspected are (from left to right)

$$[3, 5, 7, 9, 11, 13, 17, 19, 15, \ldots]. \tag{7.3}$$

We use $\mathcal{N}_b$ to denote each list of N_i, where the bit representation of b specifies the set of N's from Equation (7.3); the least significant bit of b determines if 3 is included, the second least significant bit of b determines if 5 is included, and so on. For example, $\mathcal{N}_0 = \{\}$, $\mathcal{N}_2 = \{5\}$, and $\mathcal{N}_7 = \{3, 5, 7\}$.

Each step will always include revealing 2 neighbours in $\mathcal{X}_1(\bar{\mathbb{F}}_p, 2)$. Indeed, Algorithm 7.2 can compute both neighbouring j-invariants at no extra cost. Therefore, for a node j we have for each step: $\mathrm{cost}_{\mathcal{N}} \ge \mathrm{cost}_{\mathrm{step}}$ and $\mathrm{nodes}_{\mathcal{N}} \ge 2$. Here, equality holds only when we take the list to be $\mathcal{N}_0$, which corresponds to the original Delfs–Galbraith algorithm.

To compute the exact nodes revealed at each step of our walk in the 2-isogeny graph, we assume the following heuristic.

Heuristic 7.4.6. Let $E_0 \to E_1 \to E_2 \to \ldots$ be a walk in the 2-isogeny graph. Any elliptic curve that is N-isogenous to E_n is not M-isogenous to E_m for some $m \geq 0$.

To justify this heuristic, note that we perform inspection for small values of N, and our walks have length $O(\log(p))$ (see Section 7.2.3), so this would imply an endomorphism of degree $O(\log(p))$. In Appendix B in the full version of work by Love and Boneh [222], the proportion of such curves is $O(\log(p)^{3/2}/p)$, and is therefore negligible as $p \to \infty$. Under this heuristic, we have

$$\text{nodes}_{\mathcal{N}} = 2 + \sum_{N \in \mathcal{N}} D_{N,1}. \tag{7.4}$$

Minimising the ratio $R_{\mathcal{N}}$ is a non-trivial task. We first restrict the $\mathcal{N}_b$ to only contain N such that $R_N < R_{\mathcal{N}_0}$, otherwise it would be more advantageous to take another step by moving to a neighbouring node in $\mathcal{X}_1(\overline{\mathbb{F}}_p, 2)$. We emphasise that $\text{cost}_{\mathcal{N}_0}$ grows with p, whereas cost_N stays fixed. This signifies that the condition on N becomes less restrictive as p increases. Suppose that, imposing this condition we get $\mathcal{N}_b \subseteq \{N_1, ..., N_k\}$. We then exhaust all $b < 2^k$, corresponding to subsets of $\{N_1, ..., N_k\}$, to determine the $\mathcal{N}_b$ that minimise this ratio. It is important to note that, as this optimisation depends only on the prime p, the set $\mathcal{N}_b$ can be determined in the precomputation.

7.4.4 A bound on the cost of the SuperSolver algorithm

We now discuss a heuristic upper bound for the concrete cost of finding a subfield curve using the SuperSolver algorithm combined with our optimised algorithm for walking in $\mathcal{X}_1(\overline{\mathbb{F}}_p, 2)$.

Let $\text{cost}_{\text{step}}$ be the cost of taking a step in $\mathcal{X}_1(\overline{\mathbb{F}}_p, 2)$, as given by Lemma 7.2.3. The concrete cost of running a step of SuperSolver with list $\mathcal{N} = \{N_1, \ldots, N_k\}$ is given by

$$\begin{aligned}\text{cost}_{\mathcal{N}} &= \text{cost}_{\text{step}} + \text{cost}_{\text{powers}} + \sum_{N \in \mathcal{N}} \text{cost}_N \\ &\leq \begin{cases} \text{cost}_{\text{step}}, & \text{for } \mathcal{N} = \{\}, \\ \text{cost}_{\text{step}} + D_{N_k,1} - 1 + \sum_{N \in \mathcal{N}} \left(2D_{N,1}^2 + 3D_{N,1} - 5 + 2\mathsf{mons}(N)\right), & \text{for } \mathcal{N} = \{N_1, \ldots, N_k\}. \end{cases}\end{aligned}$$

Combining our estimate of $\text{cost}_{\mathcal{N}}$ above with Equation (7.4), given a prime p we can obtain a heuristic upper bound on the ratio $R_{\mathcal{N}}$. We demonstrate this through an example.

Example 7.4.7. Consider the prime $p = 2^{250} - 207$ with $\log(p) = 250$. In this case, we follow Section 7.4.3 to calculate that $\mathcal{N} = \{3, 5, 7, 9, 11, 13\}$ is the optimal set, meaning that $\text{cost}_{\text{powers}} \leq 13$. By Lemma 7.2.3, the cost $\text{cost}_{\text{step}}$ depends on the maximum positive integer e such that $2^e \mid p-1$, which in this case is 4. Therefore, $\mathsf{Exp}(2^e - 1) = 15$ $\mathbb{F}_p$-multiplications, and $\mathsf{Exp}(c) \leq 2\log(p) - 2$ $\mathbb{F}_p$-multiplications (using the double-and-add method for exponentiation), and so $\text{cost}_{\text{step}} \leq 90 + 4\log(p)$. For the set $\mathcal{N} = \{3, 5, 7, 9, 11, 13\}$, we have that

$$\sum_{N \in \mathcal{N}} \text{cost}_N \leq 65 + 147 + 261 + 2 \cdot 585 + 795 = 2438.$$

Therefore, running SuperSolver with $\mathcal{N} = \{3, 5, 7, 9, 11, 13\}$, we have an upper bound of

$$R_{\mathcal{N}} \leq \frac{2541 + 4\log(p)}{56} = 63.2$$

$\mathbb{F}_p$-multiplications per node revealed, assuming the heuristic in Remark 7.4.2. Therefore, with 10^8 $\mathbb{F}_p$-multiplications we expect to reveal at least 1582278 nodes. If we specialise instead to $\mathcal{N}_0 = \{\}$, we obtain the concrete cost of Solver. In this case, we find that Solver requires at most 545 $\mathbb{F}_p$-multiplications per node revealed, thus revealing at least 183486 nodes with 10^8 $\mathbb{F}_p$-multiplications. This is confirmed by our experiments in Table 7.6 of Section 7.6, which furthermore show that these bounds are quite tight.

Due to the expansion properties of $\mathcal{X}_1(\overline{\mathbb{F}}_p, 2)$, the expected proportion of subfield nodes in a random walk in the graph is $\#\mathcal{S}_{p^2}/\#\mathcal{S}_p$. Using this, we would expect the number of $\mathbb{F}_p$-multiplications needed to run SuperSolver until finding a subfield node using the set $\mathcal{N}$ to be

$$R_{\mathcal{N}} \cdot \left(\frac{\#\mathcal{S}_{p^2}}{\#\mathcal{S}_p}\right). \tag{7.5}$$

From Theorem 4.2.4, we have $\#\mathcal{S}_{p^2} = \frac{p}{12} + O(1)$. Additionally, combining Proposition 4.2.6 and [216, pg. 97], we find that

$$\#\mathcal{S}_p \geq \begin{cases} \frac{1}{2}F(4p) \text{ if } p \equiv 1 \bmod 4 \\ F(p) \text{ if } p \equiv 7 \bmod 8 \\ 2F(p) \text{ if } p \equiv 3 \bmod 8 \end{cases}$$

where $F(D) := \frac{\pi}{12e^{\gamma}\log\log D}\sqrt{D}$, e is Euler's number and γ is the Euler–Mascheroni constant. Therefore,

$$\frac{\#\mathcal{S}_{p^2}}{\#\mathcal{S}_p} \leq \frac{e^{\gamma}\log\log(p)}{\pi}\sqrt{p} \approx (0.6\log\log(p) + 0.2)\sqrt{p}. \tag{7.6}$$

As detailed above, for a given prime p, we can compute an upper bound for $R_{\mathcal{N}}$ explicitly. Therefore, we can combine Equation (7.6) with Equation (7.5) to obtain a (heuristic) upper bound for the number of $\mathbb{F}_p$-multiplications needed to run SuperSolver until finding a subfield node.

Remark 7.4.8. Interestingly, in Section 7.6 we observe that the average number of nodes visited in the optimised Delfs–Galbraith walk through $\mathcal{X}_1(\overline{\mathbb{F}}_p, 2)$ is more than the expected $\#\mathcal{S}_{p^2}/\#\mathcal{S}_p$. This is due to the clustering of $\mathcal{X}_1(\mathbb{F}_p, 2)$ in $\mathcal{X}_1(\overline{\mathbb{F}}_p, 2)$, as we detail in Remark 7.6.1.

7.5 A worked example

We now use a worked example to illustrate how the Solver and SuperSolver algorithms solve the supersingular isogeny problem, and to highlight the differences between them. Our SuperSolver suite is written in SageMath/Python and a boolean variable `supersolver` specifies whether Solver or SuperSolver is used. For a prime p, and two supersingular j-invariants j_1 and j_2 defined over

$\mathbb{F}_{p^2} = \mathbb{F}_p(\beta)$, Solver runs by entering

$$\texttt{Solver(p, j10, j11, j20, j21, false)}$$

and SuperSolver runs by calling

$$\texttt{Solver(p, j10, j11, j20, j21, true)},$$

where $\texttt{j10} = \mathrm{Re}(j_1)$, $\texttt{j11} = \mathrm{Im}(j_1)$ and similarly for $\texttt{j20}$, $\texttt{j21}$.

We picked $p = 2^{20} - 3$, the smallest of the primes from Table 7.4 of Section 7.6, and generated two pseudo-random[4] j-invariants in $\mathbb{F}_{p^2} \backslash \mathbb{F}_p$:

$$j_1 = 129007\beta + 818380 \qquad \text{and} \qquad j_2 = 97589\beta + 660383.$$

Preprocessing. The preprocessing phase of both programs starts by constructing the extension field $\mathbb{F}_{p^2} = \mathbb{F}_p(\beta)$, where β^2 is the first non-square in the sequence $-1, -2, 2, -3, 3, \ldots$. It then computes a list of constants for the Tonelli-Shanks subroutine, as detailed in Section 7.2.1. The preprocessing phase then computes a set of integers $N \geq 3$ on which we perform the fast inspection according to the optimisations in Section 7.4, fetches the associated files (originally from Sutherland's database [301]) containing $\Phi_N(X, Y) \in \mathbb{Z}[X, Y]$ and reduces all the coefficients to store a set of new, more compact files containing elements of $\mathbb{F}_p$ that define each of the $\Phi_{N,p}(X, Y) \in \mathbb{F}_p[X, Y]$. This is done for both Solver and SuperSolver, since both of these programs use the original Delfs–Galbraith subfield path algorithm [132, Algorithm 1] after the searches for subfield nodes is complete. It is important to note, especially in the cryptanalytic context, that all of these preprocessing steps only depend on p and can therefore be done without knowledge of j_1 and j_2.

Solver. The optimised walk in $\mathcal{X}_1(\bar{\mathbb{F}}_p, 2)$ proceeds exactly as described in Section 7.2, i.e., using the depth first search through the binary trees rooted at j_1 and j_2, until both searches find the subfield nodes $j_1' \in \mathbb{F}_p$ and $j_2' \in \mathbb{F}_p$. In the case of our example, paths were found to $j_1' = 760776$ and $j_2' = 35387$, depicted in Figure 7.3 and Figure 7.4. They correspond to $\phi_1 \colon E_1 \to E_1'$ and $\phi_2 \colon E_2 \to E_2'$, where $j(E_1) = j_1$, $j(E_1') = j_1'$, $j(E_2) = j_2$, and $j(E_2') = j_2'$.

Solver then computes a connecting path between the subfield nodes following Delfs and Galbraith [132, Algorithm 1]. This is depicted in Figure 7.5. Solver simply reverses the steps in ϕ_2 to obtain its dual, $\widehat{\phi_2}$, and outputs the full path as $\phi \colon E_1 \to E_2$ as $\phi = \widehat{\phi_2} \circ \phi' \circ \phi_1$.

SuperSolver. With $p = 2^{20} - 3$, the preprocessing phase determined that SuperSolver is optimal with $\mathcal{N}_3 = \{3, 5\}$ (see also Table 7.4 in the next section). Before departing the starting node $j_1 = 129007\beta + 818380$, SuperSolver performs the rapid inspection of its 3- and 5-isogenous neighbours as described in Section 7.4. It then takes steps in $\mathcal{X}_1(\bar{\mathbb{F}}_p, 2)$ as in Section 7.2, but at each new node it performs the rapid inspection of the 3- and 5-isogenous neighbours. In our example, both walks found a subfield node after 2 steps in $\mathcal{X}_1(\bar{\mathbb{F}}_p, 2)$. The walk from j_1 found a 3-isogenous neighbour and the walk from j_2 found a 5-isogenous neighbour. The final step that finds ϕ' is implemented in SuperSolver exactly as it was for Solver. The three isogenies ϕ_1, ϕ_2, and ϕ', comprising the full

[4]We do this by taking long walks in $\mathcal{X}_1(\bar{\mathbb{F}}_p, 3)$ away from a known subfield curve.

$$\begin{tikzcd}
\phi_{\mathbf{1}} : j_1 \arrow[r, "2"] & 219247\beta + 863507 \arrow[r, "2"] & 489342\beta + 132142 \arrow[d, "2"] \\
174188\beta + 794346 \arrow[d, "2"] & 291380\beta + 146098 \arrow[l, "2"'] & 148602\beta + 24450 \arrow[l, "2"'] \\
263095\beta + 184707 \arrow[r, "2"] & 37438\beta + 90559 \arrow[r, "2"] & 1027930\beta + 498080 \arrow[d, "2"] \\
612554\beta + 208821 \arrow[d, "2"] & 994015\beta + 681197 \arrow[l, "2"'] & 206051\beta + 982009 \arrow[l, "2"'] \\
649416\beta + 751358 \arrow[r, "2"] & 203489\beta + 43055 \arrow[r, "2"] & 393773\beta + 1028490 \arrow[d, "2"] \\
318158\beta + 140927 \arrow[d, "2"] & 175225\beta + 937858 \arrow[l, "2"'] & 971263\beta + 725197 \arrow[l, "2"'] \\
348684\beta + 935077 \arrow[r, "2"] & 341898\beta + 405481 \arrow[r, "2"] & 274229\beta + 367729 \arrow[d, "2"] \\
 & & j_1' = 760776
\end{tikzcd}$$

FIGURE 7.3: A walk through $\mathcal{X}_1(\bar{\mathbb{F}}_p, 2)$ for $p = 2^{20} - 3$ during Solver. The walk starts at $j_1 = 129007\beta + 818380 \in \mathbb{F}_{p^2} \backslash \mathbb{F}_p$ and finds the subfield node $j_1' = 760776 \in \mathbb{F}_p$ after 21 steps.

$$\begin{tikzcd}
\phi_{\mathbf{2}} : j_2 \arrow[r, "2"] & 867493\beta + 220256 \arrow[r, "2"] & 252807\beta + 1011175 \arrow[d, "2"] \\
657423\beta + 286117 \arrow[d, "2"] & 440840\beta + 706619 \arrow[l, "2"'] & 953362\beta + 11601 \arrow[l, "2"'] \\
734841\beta + 660440 \arrow[r, "2"] & 919529\beta + 442520 \arrow[r, "2"] & 219960\beta + 646080 \arrow[d, "2"] \\
638727\beta + 940073 \arrow[d, "2"] & 219719\beta + 594710 \arrow[l, "2"'] & 619876\beta + 961666 \arrow[l, "2"'] \\
407014\beta + 868179 \arrow[r, "2"] & 535787\beta + 1046047 \arrow[r, "2"] & 138865\beta + 8726 \arrow[d, "2"] \\
1016378\beta + 696447 \arrow[d, "2"] & 289439\beta + 170877 \arrow[l, "2"'] & 665078\beta + 700037 \arrow[l, "2"'] \\
895198\beta + 793471 \arrow[r, "2"] & 562302\beta + 547814 \arrow[r, "2"] & 68076\beta + 946405 \arrow[d, "2"] \\
 & & j_2' = 35387
\end{tikzcd}$$

FIGURE 7.4: A walk through $\mathcal{X}_1(\bar{\mathbb{F}}_p, 2)$ for $p = 2^{20} - 3$ during Solver. The walk starts at $j_2 = 97589\beta + 660383 \in \mathbb{F}_{p^2} \backslash \mathbb{F}_p$ and finds the subfield node $j_2' = 35387 \in \mathbb{F}_p$ after 21 steps.

$$\begin{tikzcd}
\phi' : j_1' \arrow[r, "31"] & 815910 \arrow[r, "17"] & 848568 \arrow[r, "31"] & 157399 \arrow[r, "29"] & 451011 \arrow[r, "31"] & 820763 \arrow[d, "31"] \\
 & & & j_2' & 286978 \arrow[l, "17"'] & 76159 \arrow[l, "37"']
\end{tikzcd}$$

FIGURE 7.5: A path connecting two subfield j-invariants by taking steps in $\mathcal{X}_1(\overline{\mathbb{F}}_p, N)$ using SuperSolver with $N \in \{17, 29, 31, 37\}$. The walk starts at $j_1' = 760776 \in \mathbb{F}_p$ and connects to $j_2' = 35387 \in \mathbb{F}_p$ after 8 steps.

$$\begin{tikzcd}
\phi_{\mathbf{1}} : \ j_1 \arrow[r, "2"] & 219247\beta + 863507 \arrow[r, "2"] & 489342\beta + 132142 \arrow[r, "\mathbf{3}"] & j_1' = 35387
\end{tikzcd}$$

$$\begin{tikzcd}
\phi_{\mathbf{2}} : \ j_2 \arrow[r, "2"] & 867493\beta + 220256 \arrow[r, "2"] & 252807\beta + 1011175 \arrow[r, "\mathbf{5}"] & j_2' = 292917
\end{tikzcd}$$

$$\begin{tikzcd}
\phi' : \ j_1' \arrow[r, "17"] & 658300 \arrow[r, "29"] & 343840 \arrow[r, "31"] & 560315 \arrow[d, "17"] \\
 & & j_2' & 439276 \arrow[l, "37"']
\end{tikzcd}$$

FIGURE 7.6: The three paths found comprising an isogeny from E_1 to E_2 as found by SuperSolver.

isogeny $\phi = \widehat{\phi}_2 \circ \phi' \circ \phi_1$ are depicted in Figure 7.6.

To illustrate the core idea in this paper, we focus on the isogeny ϕ_1 depicted at the top of Figure 7.6 and walk through the steps of the NeighbourInFp algorithm. Evaluating the third modular polynomial at the intermediate j-invariants (Step 1 of Algorithm 7.4) yields

$$\begin{aligned}
\Phi_{3,p}(X,\, 219247\beta + 863507) &= X^4 + (212814\beta + 479338)X^3 + (408250\beta + 920025)X^2 \\
&\quad + (811739\beta + 93038)X + 942336\beta + 847782; \\
\Phi_{3,p}(X,\, 489342\beta + 132142) &= X^4 + (872004\beta + 13960)X^3 + (1031755\beta + 822066)X^2 \\
&\quad + (969683\beta + 747785)X + 813010\beta + 255391.
\end{aligned}$$

Though the theory tells us that these two polynomials split over $\mathbb{F}_{p^2}[X]$, to the naked eye there is no way to distinguish which (if any) of these polynomials has a root in $\mathbb{F}_p$. In both cases, setting $g_1 = 1/2 \cdot (\Phi_{3,p} + \pi(\Phi_{3,p}))$ (Step 2 of Algorithm 7.4) and $g_2 = -\beta/2 \cdot (\Phi_{3,p} - \pi(\Phi_{3,p}))$ (Step 3 of Algorithm 7.4) respectively yields

$$\begin{aligned}
g_1 &= X^4 + 479338X^3 + 920025X^2 + 93038X + 847782; \\
g_2 &= 425628X^3 + 816500X^2 + 574905X + 836099,
\end{aligned}$$

and

$$g_1 = X^4 + 13960X^3 + 822066X^2 + 747785X + 255391;$$
$$g_2 = 695435X^3 + 1014937X^2 + 890793X + 577447.$$

In the first case, Step 4 of Algorithm 7.4 outputs $\gcd(g_1, g_2) = 1$, meaning that $\Phi_{3,p}(X, 219247\beta + 863507)$ has no subfield roots. In the second case, we see $\gcd(g_1, g_2) = X + 1013186$, meaning that $-1013186 = 35387$ is a subfield root. In our example, we note that the total number of steps between j_1 and j_2 returned by SuperSolver is 10, which is much shorter than the 50 steps taken by Solver. Since the middle subfield path finding algorithm is the same in both routines, there is no guarantee that the total path will always be smaller for SuperSolver. It is worth pointing out, however, that the two *outer* paths from j-invariants in $\mathbb{F}_{p^2}\backslash\mathbb{F}_p$ to j-invariants in the special subset X_1 (i.e., ϕ_1 and ϕ_2) returned by SuperSolver will never be longer than those returned by Solver. Indeed, Solver can be viewed as a special case of SuperSolver where the list of N's is chosen to be $\mathcal{N}_0$. Finally, we note that both Solver and SuperSolver always conclude by checking the correctness of the full path from j_1 to j_2.

7.6 Implementation results

In this section we present some experimental results highlighting the efficacy of SuperSolver. The experiments focus solely on the search for subfield nodes (i.e., the bottleneck step of Delfs–Galbraith algorithm) and come in two flavours: many j-invariants over small primes, and one j-invariant over a large, cryptographic prime.

Small primes and many walks. Table 7.4 and Table 7.5 report experiments that were run on the largest primes of the 30 bitlengths from 20 to 49. We started at 5000 pseudo-random[5] supersingular j-invariants in $\mathcal{S}_{p^2} \setminus \mathcal{S}_p$ for the primes of bitlengths 20-24, at 1000 j's for the primes of bitlengths 25-29, at 500 j's for the primes of bitlengths 30-34, at 100 j's for the primes of bitlengths 35-39, at 50 j's for the primes of bitlengths 40-44, and at 10 j's for the primes of bitlengths 45-49. For every j, we ran both Solver and SuperSolver (with the five sets of N's that were predicted to perform best during preprocessing) until all walks hit a subfield j-invariant. Throughout, we will denote these fast sets of N's by $\mathcal{N}_b$, as in Section 7.4. In all cases we counted the exact number of $\mathbb{F}_p$-multiplications, squarings and additions required to find the subfield node. Following our metric in Section 7.1, Table 7.5 reports the average number of $\mathbb{F}_p$-multiplications by counting squarings as multiplications, and highlights in **bold** which of the five predicted sets of N's performed best on average.

Table 7.4 reports the average number of nodes visited in each of the walks, along with the ratio $\lceil \#\mathcal{S}_{p^2}/\#\mathcal{S}_p \rceil$, the expected number of random elements in $\mathcal{S}_{p^2}$ that would need to be sampled to find a subfield element in $\mathcal{S}_p$, and R, the ratio of the number of nodes we visited on average using Solver against the number of elements we would expect to draw at random from $\mathcal{S}_{p^2}$ before finding one in $\mathcal{S}_p$ (see Remark 7.6.1). Here, the primes are small enough that $\mathcal{S}_p$ can be computed

[5]Just as in Section 7.5, we used long walks in $\mathcal{X}_1(\bar{\mathbb{F}}_p, 3)$ away from a known starting curve to achieve uniformity in $\mathcal{S}_{p^2}$.

precisely (i.e., the class group can be computed). For each prime, Table 7.4 highlights in **bold** the column that matches up with the least multiplications reported in Table 7.5. Note that, for SuperSolver, the number of nodes visited is the number of nodes that are actually walked onto in $\mathcal{X}_1(\bar{\mathbb{F}}_p, 2)$, not the number of nodes inspected using our fast subfield detection algorithm. Thus, in general, the lowest average number of nodes visited does not correspond to the lowest average number of multiplications. Indeed, the walks with fewer N's spend less compute inspecting N-isogenous neighbours and therefore move onto new nodes faster, but do not cover as much of the supersingular set during the fast inspection.

						Solver	SuperSolver				
Prime p	p mod 8	$\left\lceil \frac{\#\mathcal{S}_{p^2}}{\#\mathcal{S}_p} \right\rceil$	R	$\mathbb{F}_p$-mults. per step	Fastest $\mathcal{N}_j$'s $[\mathcal{N}_{(i)} \dots, \mathcal{N}_{(v)}]$	$\mathcal{N} = \{\}$	$\mathcal{N}_{(i)}$	$\mathcal{N}_{(ii)}$	$\mathcal{N}_{(iii)}$	$\mathcal{N}_{(iv)}$	$\mathcal{N}_{(v)}$
$2^{20} - 3$	5	530	1.5	54	$[\mathcal{N}_3, \mathcal{N}_1, \mathcal{N}_7, \mathcal{N}_5, \mathcal{N}_2]$	812	**127**	257	76	107	193
$2^{21} - 9$	7	156	2.9	53	$[\mathcal{N}_3, \mathcal{N}_1, \mathcal{N}_7, \mathcal{N}_5, \mathcal{N}_2]$	459	**86**	218	53	87	111
$2^{22} - 3$	5	584	1.5	60	$[\mathcal{N}_3, \mathcal{N}_7, \mathcal{N}_1, \mathcal{N}_5, \mathcal{N}_6]$	885	170	108	**288**	146	145
$2^{23} - 15$	1	583	1.4	71	$[\mathcal{N}_3, \mathcal{N}_7, \mathcal{N}_5, \mathcal{N}_6, \mathcal{N}_1]$	838	**172**	106	169	121	430
$2^{24} - 3$	5	1277	1.5	64	$[\mathcal{N}_3, \mathcal{N}_7, \mathcal{N}_5, \mathcal{N}_1, \mathcal{N}_6]$	1897	**318**	209	311	618	273
$2^{25} - 39$	1	1231	1.5	71	$[\mathcal{N}_3, \mathcal{N}_7, \mathcal{N}_5, \mathcal{N}_1, \mathcal{N}_6]$	1873	**360**	223	359	933	259
$2^{26} - 5$	3	732	1.9	62	$[\mathcal{N}_3, \mathcal{N}_7, \mathcal{N}_1, \mathcal{N}_5, \mathcal{N}_6]$	1362	352	**194**	691	271	233
$2^{27} - 39$	1	2348	1.5	73	$[\mathcal{N}_3, \mathcal{N}_7, \mathcal{N}_5, \mathcal{N}_6, \mathcal{N}_1]$	3455	917	**438**	579	497	1766
$2^{28} - 57$	7	2965	3.3	64	$[\mathcal{N}_3, \mathcal{N}_7, \mathcal{N}_1, \mathcal{N}_5, \mathcal{N}_6]$	9748	1788	1022	**3065**	1314	1306
$2^{29} - 3$	5	2953	1.5	74	$[\mathcal{N}_3, \mathcal{N}_7, \mathcal{N}_5, \mathcal{N}_6, \mathcal{N}_1]$	4384	1053	**526**	712	603	2161
$2^{30} - 35$	5	3965	1.4	75	$[\mathcal{N}_3, \mathcal{N}_7, \mathcal{N}_5, \mathcal{N}_6, \mathcal{N}_1]$	5555	1443	749	**961**	849	2825
$2^{31} - 1$	7	9009	3.0	75	$[\mathcal{N}_3, \mathcal{N}_7, \mathcal{N}_5, \mathcal{N}_6, \mathcal{N}_1]$	27103	4501	**2602**	3755	3136	8794
$2^{32} - 5$	3	5142	2.0	75	$[\mathcal{N}_3, \mathcal{N}_7, \mathcal{N}_5, \mathcal{N}_6, \mathcal{N}_1]$	10149	2520	**1445**	2108	1702	5335
$2^{33} - 9$	7	6638	3.1	77	$[\mathcal{N}_3, \mathcal{N}_7, \mathcal{N}_5, \mathcal{N}_6, \mathcal{N}_1]$	20387	**3832**	2342	3756	2676	10562
$2^{34} - 41$	7	10526	3.1	78	$[\mathcal{N}_3, \mathcal{N}_7, \mathcal{N}_5, \mathcal{N}_6, \mathcal{N}_1]$	32640	6443	3790	6094	**4531**	16320
$2^{35} - 31$	1	117571	1.3	99	$[\mathcal{N}_7, \mathcal{N}_3, \mathcal{N}_5, \mathcal{N}_6, \mathcal{N}_{23}]$	150101	**14893**	27873	23076	20921	9850
$2^{36} - 5$	3	29040	2.2	83	$[\mathcal{N}_3, \mathcal{N}_7, \mathcal{N}_5, \mathcal{N}_6, \mathcal{N}_{23}]$	63384	15929	**9127**	11974	10807	5249
$2^{37} - 25$	7	70328	3.1	84	$[\mathcal{N}_3, \mathcal{N}_7, \mathcal{N}_5, \mathcal{N}_6, \mathcal{N}_{23}]$	218775	**26241**	16098	29226	24153	10405
$2^{38} - 45$	3	100268	2.2	86	$[\mathcal{N}_3, \mathcal{N}_7, \mathcal{N}_5, \mathcal{N}_6, \mathcal{N}_{23}]$	217145	43595	21343	**27187**	26982	14897
$2^{39} - 7$	1	174817	1.3	96	$[\mathcal{N}_7, \mathcal{N}_3, \mathcal{N}_5, \mathcal{N}_6, \mathcal{N}_{23}]$	230235	28802	48488	**36770**	38318	19677
$2^{40} - 87$	1	266662	1.5	95	$[\mathcal{N}_7, \mathcal{N}_3, \mathcal{N}_5, \mathcal{N}_6, \mathcal{N}_{23}]$	394908	49855	**80764**	66646	56901	28016
$2^{41} - 21$	3	205227	2.2	92	$[\mathcal{N}_7, \mathcal{N}_3, \mathcal{N}_5, \mathcal{N}_6, \mathcal{N}_{23}]$	448887	52656	105639	69940	62212	**27395**
$2^{42} - 11$	5	557046	1.3	99	$[\mathcal{N}_7, \mathcal{N}_3, \mathcal{N}_5, \mathcal{N}_6, \mathcal{N}_{23}]$	720206	93920	189498	147651	**102116**	64309
$2^{43} - 57$	7	198777	3.5	95	$[\mathcal{N}_7, \mathcal{N}_3, \mathcal{N}_5, \mathcal{N}_6, \mathcal{N}_{23}]$	705224	**69021**	153095	95778	81922	44112
$2^{44} - 17$	7	307870	2.6	98	$[\mathcal{N}_7, \mathcal{N}_3, \mathcal{N}_5, \mathcal{N}_6, \mathcal{N}_{23}]$	808057	131220	285136	**145263**	142750	72964
$2^{45} - 55$	1	3120225	0.7	108	$[\mathcal{N}_7, \mathcal{N}_3, \mathcal{N}_5, \mathcal{N}_6, \mathcal{N}_{23}]$	2298828	301730	**410169**	579449	404520	226542
$2^{46} - 21$	3	2759728	3.3	102	$[\mathcal{N}_7, \mathcal{N}_3, \mathcal{N}_5, \mathcal{N}_6, \mathcal{N}_{23}]$	9075335	516826	**788898**	957832	730020	382101
$2^{47} - 115$	5	4234340	1.2	108	$[\mathcal{N}_7, \mathcal{N}_3, \mathcal{N}_5, \mathcal{N}_6, \mathcal{N}_{23}]$	5182631	650377	866413	**650377**	801837	781907
$2^{48} - 59$	5	2706129	2.5	111	$[\mathcal{N}_7, \mathcal{N}_3, \mathcal{N}_5, \mathcal{N}_6, \mathcal{N}_{23}]$	6739857	**546014**	899553	756358	651990	491312
$2^{49} - 81$	7	1239417	2.9	107	$[\mathcal{N}_7, \mathcal{N}_3, \mathcal{N}_5, \mathcal{N}_6, \mathcal{N}_{23}]$	3582205	288124	660449	**326050**	319641	252270

TABLE 7.4: The average number of nodes visited in the search for subfield j-invariants with Solver and SuperSolver using the five fastest sets $\mathcal{N}_{(i)} \dots, \mathcal{N}_{(v)}$. See further explanation in text.

The key trend to highlight is that, relatively speaking, SuperSolver gains more advantage over Solver as the primes get larger. This is not as evident for the small primes in Tables 7.4 and 7.5 as it is for the larger primes below.

Remark 7.6.1 ($\mathcal{X}_1(\mathbb{F}_p, 2)$ clusters in $\mathcal{X}_1(\bar{\mathbb{F}}_p, 2)$)**.** An interesting trend to highlight in Table 7.4 is that the average number of nodes visited in the optimised Delfs–Galbraith walk through $\mathcal{X}_1(\bar{\mathbb{F}}_p, 2)$ is significantly more than the expected number of elements one would need to select randomly

				Solver		SuperSolver			
Prime p	$\left\lceil\frac{\#\mathcal{S}_{p^2}}{\#\mathcal{S}_p}\right\rceil$	$\mathbb{F}_p$-mults. per step	Fastest $\mathcal{N}_j$'s $[\mathcal{N}_{(i)}\ldots,\mathcal{N}_{(v)}]$	$\mathcal{N}=\{\}$	$\mathcal{N}_{(i)}$	$\mathcal{N}_{(ii)}$	$\mathcal{N}_{(iii)}$	$\mathcal{N}_{(iv)}$	$\mathcal{N}_{(v)}$
$2^{20}-3$	530	54	$[\mathcal{N}_3,\mathcal{N}_1,\mathcal{N}_7,\mathcal{N}_5,\mathcal{N}_2]$	44848	**20601**	22585	22235	23459	24951
$2^{21}-9$	156	53	$[\mathcal{N}_3,\mathcal{N}_1,\mathcal{N}_7,\mathcal{N}_5,\mathcal{N}_2]$	24187	**13648**	18578	15453	18770	14064
$2^{22}-3$	584	60	$[\mathcal{N}_3,\mathcal{N}_7,\mathcal{N}_1,\mathcal{N}_5,\mathcal{N}_6]$	52385	28062	31962	**26410**	32555	38348
$2^{23}-15$	583	71	$[\mathcal{N}_3,\mathcal{N}_7,\mathcal{N}_5,\mathcal{N}_6,\mathcal{N}_1]$	59691	**30508**	32883	39703	33370	44556
$2^{24}-3$	1277	64	$[\mathcal{N}_3,\mathcal{N}_7,\mathcal{N}_5,\mathcal{N}_1,\mathcal{N}_6]$	112878	**53900**	62725	70482	59206	73117
$2^{25}-39$	1231	71	$[\mathcal{N}_3,\mathcal{N}_7,\mathcal{N}_5,\mathcal{N}_1,\mathcal{N}_6]$	128703	**63021**	68210	83333	94434	70878
$2^{26}-5$	732	62	$[\mathcal{N}_3,\mathcal{N}_7,\mathcal{N}_1,\mathcal{N}_5,\mathcal{N}_6]$	85437	59484	**58286**	65813	61261	62216
$2^{27}-39$	2348	73	$[\mathcal{N}_3,\mathcal{N}_7,\mathcal{N}_5,\mathcal{N}_6,\mathcal{N}_1]$	251304	164036	**135672**	136633	137780	185819
$2^{28}-57$	2965	64	$[\mathcal{N}_3,\mathcal{N}_7,\mathcal{N}_1,\mathcal{N}_5,\mathcal{N}_6]$	631157	305345	308003	**298049**	299314	351102
$2^{29}-3$	2953	74	$[\mathcal{N}_3,\mathcal{N}_7,\mathcal{N}_5,\mathcal{N}_6,\mathcal{N}_1]$	326888	199985	**171489**	173335	177986	235902
$2^{30}-35$	3965	75	$[\mathcal{N}_3,\mathcal{N}_7,\mathcal{N}_5,\mathcal{N}_6,\mathcal{N}_1]$	412457	260188	232753	**228089**	236360	301541
$2^{31}-1$	9009	75	$[\mathcal{N}_3,\mathcal{N}_7,\mathcal{N}_5,\mathcal{N}_6,\mathcal{N}_1]$	1998840	809040	**807306**	889210	871068	934319
$2^{32}-5$	5142	75	$[\mathcal{N}_3,\mathcal{N}_7,\mathcal{N}_5,\mathcal{N}_6,\mathcal{N}_1]$	758637	455571	**449889**	501335	474549	572203
$2^{33}-9$	6638	77	$[\mathcal{N}_3,\mathcal{N}_7,\mathcal{N}_5,\mathcal{N}_6,\mathcal{N}_1]$	1564701	**700390**	733705	900515	751310	1153911
$2^{34}-41$	10526	78	$[\mathcal{N}_3,\mathcal{N}_7,\mathcal{N}_5,\mathcal{N}_6,\mathcal{N}_1]$	2537688	1184024	1191084	1467113	**1276654**	1799292
$2^{35}-31$	117571	99	$[\mathcal{N}_7,\mathcal{N}_3,\mathcal{N}_5,\mathcal{N}_6,\mathcal{N}_{23}]$	15272705	**5037679**	5790782	6109529	6396752	6213090
$2^{36}-5$	29040	83	$[\mathcal{N}_3,\mathcal{N}_7,\mathcal{N}_5,\mathcal{N}_6,\mathcal{N}_{23}]$	5244914	3006618	**2913909**	2942626	3099020	3211580
$2^{37}-25$	70328	84	$[\mathcal{N}_3,\mathcal{N}_7,\mathcal{N}_5,\mathcal{N}_6,\mathcal{N}_{23}]$	18322417	**4979176**	5155517	7211517	6950196	6375918
$2^{38}-45$	100268	86	$[\mathcal{N}_3,\mathcal{N}_7,\mathcal{N}_5,\mathcal{N}_6,\mathcal{N}_{23}]$	18402937	8315681	6856578	**6735588**	7791309	9143526
$2^{39}-7$	174817	96	$[\mathcal{N}_7,\mathcal{N}_3,\mathcal{N}_5,\mathcal{N}_6,\mathcal{N}_{23}]$	22505327	9627241	9879376	**9587856**	11562406	12332858
$2^{40}-87$	266662	95	$[\mathcal{N}_7,\mathcal{N}_3,\mathcal{N}_5,\mathcal{N}_6,\mathcal{N}_{23}]$	38602102	16664021	**16455546**	17377853	17169885	17559520
$2^{41}-21$	205227	92	$[\mathcal{N}_7,\mathcal{N}_3,\mathcal{N}_5,\mathcal{N}_6,\mathcal{N}_{23}]$	41185437	17284297	20890068	17817383	18399209	**17006068**
$2^{42}-11$	557046	99	$[\mathcal{N}_7,\mathcal{N}_3,\mathcal{N}_5,\mathcal{N}_6,\mathcal{N}_{23}]$	70760036	31439715	38704883	38573868	**30864574**	40337712
$2^{43}-57$	198777	95	$[\mathcal{N}_7,\mathcal{N}_3,\mathcal{N}_5,\mathcal{N}_6,\mathcal{N}_{23}]$	66820000	**22863425**	30733754	24686759	24474388	27514948
$2^{44}-17$	307870	98	$[\mathcal{N}_7,\mathcal{N}_3,\mathcal{N}_5,\mathcal{N}_6,\mathcal{N}_{23}]$	79795521	43991657	58381667	**38022655**	43217829	45803624
$2^{45}-55$	3120225	108	$[\mathcal{N}_7,\mathcal{N}_3,\mathcal{N}_5,\mathcal{N}_6,\mathcal{N}_{23}]$	247697962	103871110	**87674099**	156886333	126109617	144250917
$2^{46}-21$	2759728	102	$[\mathcal{N}_7,\mathcal{N}_3,\mathcal{N}_5,\mathcal{N}_6,\mathcal{N}_{23}]$	923415913	174816651	**163893709**	198006728	205399202	220786555
$2^{47}-115$	4234340	108	$[\mathcal{N}_7,\mathcal{N}_3,\mathcal{N}_5,\mathcal{N}_6,\mathcal{N}_{23}]$	550653552	222915969	183895536	**175113230**	248769348	272706341
$2^{48}-59$	2706129	111	$[\mathcal{N}_7,\mathcal{N}_3,\mathcal{N}_5,\mathcal{N}_6,\mathcal{N}_{23}]$	729589278	**188237971**	192728454	205161765	251091015	310387211
$2^{49}-81$	1239417	107	$[\mathcal{N}_7,\mathcal{N}_3,\mathcal{N}_5,\mathcal{N}_6,\mathcal{N}_{23}]$	385982057	99186957	141171527	**88278361**	99648273	160633132

TABLE 7.5: The average number of $\mathbb{F}_p$-multiplications used to search for subfield j-invariants with Solver and SuperSolver using the five fastest sets $\mathcal{N}_{(i)}\ldots,\mathcal{N}_{(v)}$. See further explanation in text.

from $\mathcal{S}_{p^2}$ in order to find an element of $\mathcal{S}_p$. The reason for this is that components of $\mathcal{X}_1(\mathbb{F}_p,2)$ *cluster together* in $\mathcal{X}_1(\bar{\mathbb{F}}_p,2)$. Thus, with respect to finding subfield nodes, walks in $\mathcal{X}_1(\bar{\mathbb{F}}_p,2)$ are significantly different from selecting nodes at random from $\mathcal{S}_{p^2}$. The types of clusterings in $\mathcal{X}_1(\bar{\mathbb{F}}_p,2)$ depend on the value of $p \bmod 8$ [132, Theorem 2.7], which is why this value is given alongside p in each row. Write R for the ratio of the number of nodes we visited on average using Solver (i.e., the $\mathcal{N}=\{\}$ column) against the number of elements we would expect to draw at random from $\mathcal{S}_{p^2}$ before finding one in $\mathcal{S}_p$ (i.e., $\#\mathcal{S}_{p^2}/\#\mathcal{S}_p$). Table 7.4 shows that (i) when $p \equiv 1 \bmod 4$, we typically see $1 \le R \le 2$; (ii) when $p \equiv 3 \bmod 8$, we typically see $2 \le R \le 3$; and (iii) when $p \equiv 7 \bmod 8$, we often see $R > 3$. Indeed, when $p \equiv 1 \mod 4$, the graph $\mathcal{X}_1(\mathbb{F}_p,2)$ is less connected, meaning that short paths are more likely to go through nodes in $\mathcal{S}_p$. In contrast, when $p \equiv 7 \bmod 8$, the graph can be highly connected. This is exhibited in [5, §3.5.3], which presents the number of connected components of $\mathcal{X}_1(\mathbb{F}_p,2)$, depending on the value of $p \bmod 8$. For more experimental data illustrating this phenomenon, see [5, §4.3]. In practice, we do not see the $N > 1$ as enough of a reason to incur the significant overhead of walking in $\mathcal{X}_1(\bar{\mathbb{F}}_p,N)$ for $N > 2$ instead.

In any case, the method of fast subfield root detection proposed in this paper will work regardless of the N-isogenies that are used to take steps in a given walk. In fact, if walking in $\mathcal{X}_1(\bar{\mathbb{F}}_p, N)$ for $N > 2$ results in better concrete performance than for $N = 2$, the greater cost of taking a step in $\mathcal{X}_1(\bar{\mathbb{F}}_p, N)$ is likely to increase the size of the set of "fast N's" and the relative efficacy of invoking subfield root detection.

Large primes and optimal node coverage. Table 7.6 illustrates the increased efficacy of SuperSolver over Solver as the supersingular isogeny graphs get larger. Recall that we reported some of the results from this table up front in the introduction, namely from the experiments using primes from the isogeny literature. We chose the largest prime below 2^k for $k \in \{50, 100, \ldots 800\}$, and started from a pseudorandom j-invariant in $\mathcal{S}_{p^2} \setminus \mathcal{S}_p$ as usual. Since these instances are too large to actually run the full subfield search until it terminates, in each case we ran both Solver and SuperSolver (for the three sets of N's that were predicted to perform best during preprocessing) until the number of $\mathbb{F}_p$-multiplications used exceeded 10^8, and then immediately stopped. The numbers reported in bold in Table 7.6 are the total number of nodes covered (i.e., both walked onto *and* inspected) during these walks. For the smallest prime $p = 2^{50} - 27$, SuperSolver covers around 3 times the number of nodes that Solver does; for the largest prime $p = 2^{800} - 105$, SuperSolver covers around 17 times the number of nodes. Though primes beyond this size are unlikely to be of cryptographic interest, it is worth pointing out that this trend continues: the larger p grows, the more profitable it becomes to keep adding N's in the fast subfield inspection algorithm.

Storing and accessing the reduced modular polynomials. The unreduced modular polynomials $\Phi_N(X, Y) \in \mathbb{Z}[X, Y]$ require a significant amount of storage, but recall that the preprocessing phase immediately reduces all the coefficients into $\mathbb{F}_p$ to produce $\Phi_{N,p}(X, Y) \in \mathbb{F}_p[X, Y]$. This can be done once-and-for-all for a specific prime, and this makes the storage and access of the $\Phi_{N,p}(X, Y)$ a non-issue. Storing $\Phi_{N,p}(X, Y)$ requires at most $(D_{N,1}^2/2) \cdot \log(p)$ bits. For example, the largest $\Phi_{N,p}(X, Y)$ for the 250-bit prime above is $\Phi_{13,p}(X, Y)$, which requires the storage of at most $D_{13,1}^2/2 = 14^2/2 = 98$ elements of $\mathbb{F}_p$, around 3KB. The largest $\Phi_{N,p}(X, Y)$ for the 800-bit prime above requires the storage of at most $D_{19,1}^2/2 = 20^2/2 = 200$ elements of $\mathbb{F}_p$, around 20KB. Any of these would comfortably fit into the L1 cache on a modern CPU.

Concrete security of the supersingular isogeny problem. Our SuperSolver suite makes it straightforward to go beyond the heuristic estimates in Section 7.4.4 and obtain precise estimates on the concrete classical security offered by the general supersingular isogeny problem in $\mathcal{S}_{p^2}$, for any prime p. Combining a small experiment (like those reported in Table 7.6) with the expected number of nodes one must cover before reaching a subfield node allows us to obtain accurate counts on the expected number of $\mathbb{F}_p$-multiplications, squarings and additions that must be carried out during a full cryptanalytic attack. It is then a matter of costing these $\mathbb{F}_p$ operations with respect to the appropriate metric, whether that be bit operations, cycle counts, gate counts, or circuit depth.

Take, for example, the SQIsign 256-bit prime

$$p = 73743043621499797449074820543863456997944695372324032511999999999999999999999$$

	Solver		SuperSolver			
Prime p	Nodes inspected	$\mathbb{F}_p$-mults. per node	Fastest Sets $\mathcal{N}$	Nodes inspected	$\mathbb{F}_p$-mults. per node	**Improv. Factor**
$2^{50} - 27$	883,007	113	{3,5,7}	2,859,221	35.0	**2.8 - 3.2x**
			{3,5}	2,736,601	36.5	
			{3,7}	2,533,945	39.4	
$2^{100} - 15$	443,951	223	{3,5,7}	2,165,681	46.0	**4.5 - 4.9x**
			{3,5,7,11}	2,121,313	47.1	
			{3,5,7,9}	1,988,731	47.1	
$2^{150} - 3$	317,215	315	{3,5,7,9,11}	1,847,413	51.8	**5.6 - 6.0x**
			{3,5,7,11}	1,895,201	52.9	
			{3,5,7,9}	1,776,751	52.9	
$2^{200} - 75$	241,987	415	{3,5,7,9,11}	1,700,749	56.2	**7.0 - 7.1x**
			{3,5,7,9,11,13}	1,715,449	56.3	
			{3,5,7,11,13}	1,716,767	58.3	
$2^{250} - 207$	191,115	521	{3,5,7,9,11,13}	1,607,145	60.1	**8.2 - 8.4x**
			{3,5,7,9,11}	1,561,645	61.1	
			{3,5,7,11,13}	1,586,495	63.0	
$2^{300} - 153$	164,273	610	{3,5,7,9,11,13}	1,531,993	63.0	**8.0 - 8.4x**
			{3,5,7,9,11}	1,468,279	65.0	
			{3,5,7,9,11,13,17}	1,489,991	65.3	
$2^{350} - 113$	141,097	709	{3,5,7,9,11,13}	1,452,529	66.5	**9.7 - 10.3x**
			{3,5,7,9,11,13,17}	1,432,345	68.0	
			{3,5,7,9,11}	1,372,351	70.0	
$2^{400} - 593$	123,649	809	{3,5,7,9,11,13}	1,380,849	70.0	**10.8 - 11.2x**
			{3,5,7,9,11,13,17}	1,378,991	70.6	
			{3,5,7,9,11,13,19}	1,339,805	72.8	
$2^{450} - 501$	110,407	906	{3,5,7,11,13,9,17}	1,330,891	73.2	**11.9 - 12.1x**
			{3,5,7,11,13,9}	1,317,849	73.3	
			{3,5,7,11,13,9,17,19}	1,309,703	74.8	
$2^{500} - 863$	97,209	1032	{3,5,7,9,11,13,17}	1,274,503	76.4	**12.8 - 13.1x**
			{3,5,7,9,11,13,17,19}	1,266,275	77.3	
			{3,5,7,9,11,13}	1,245,721	77.5	
$2^{550} - 5$	90,031	1111	{3,5,7,9,11,13,17}	1,239,501	78.6	**13.3 - 13.8x**
			{3,5,7,9,11,13,17,19}	1,238,921	79.0	
			{3,5,7,9,11,13}	1,201,873	80.3	
$2^{600} - 95$	81,253	1230	{3,5,7,9,11,13,17,19}	1,200,945	81.5	**14.4 - 14.8x**
			{3,5,7,9,11,13,17}	1,191,549	81.7	
			{3,5,7,9,11,13,19}	1,166,297	83.6	
$2^{650} - 611$	76,207	1314	{3,5,7,9,11,13,17,19}	1,176,411	83.3	**14.9 - 15.4x**
			{3,5,7,9,11,13,17}	1,161,061	83.9	
			{3,5,7,9,11,13,19}	1,137,873	85.7	
$2^{700} - 1113$	71,037	1408	{3,5,7,9,11,13,17,19}	1,148,963	85.2	**15.8 - 16.2x**
			{3,5,7,9,11,13,17}	1,127,317	86.4	
			{3,5,7,9,11,13,17,19,23}	1,123,125	87.5	
$2^{750} - 161$	66,237	1510	{3,5,7,9,11,13,17,19}	1,121,045	87.4	**16.5 - 16.9x**
			{3,5,7,9,11,13,17}	1,093,351	89.0	
			{3,5,7,9,11,13,17,19,23}	1,101,767	89.3	
$2^{800} - 105$	62,163	1610	{3,5,7,9,11,13,17,19}	1,095,195	89.4	**16.2 - 17.6x**
			{3,5,7,9,11,13,17,19,23}	1,081,825	90.9	
			{3,5,7,9,11,13,17,19,15}	1,008,481	90.9	

Table 7.6: The number of nodes inspected per 10^8 field multiplications for the largest primes of various bitlengths. The Solver column corresponds to optimised Delfs–Galbraith walks in $\mathcal{X}_1(\bar{\mathbb{F}}_p, 2)$ – see Section 7.2. The SuperSolver columns correspond to enabling our fast subfield root detection algorithm with the three fastest sets of N's (left to right) – see Section 7.4. Numbers in round brackets are the approximate number of $\mathbb{F}_p$-multiplications per node inspected, as computed during the precomputation phase that determines which sets of N's will perform fastest. The improvement factor column corresponds to the number of nodes inspected with SuperSolver divided by the number of nodes inspected by Solver.

given by De Feo, Kohel, Leroux, Petit, and Wesolowski [125] to illustrate how our software can be used to obtain precise security estimates. The precomputation phase of SuperSolver, which takes a few seconds on input of p, reveals that taking an optimised step in $\mathcal{X}_1(\overline{\mathbb{F}}_p, 2)$ costs 403 multiplications in $\mathbb{F}_p$. Based on this cost, the precomputation further determines that the fastest set of N's to proceed with are

$$N \in \{3, 5, 7, 9, 11\}.$$

On average, the combination of this set of N's and Algorithm 7.4 reduces the cost of the subfield search from 403 multiplications in $\mathbb{F}_p$ per node to 55.7 multiplications in $\mathbb{F}_p$ per node (see Table 7.1). Thus, on average, solving the supersingular isogeny problem costs

$$55.7 \cdot \left(\frac{\#\mathcal{S}_{p^2}}{\#\mathcal{S}_p} \right)$$

$\mathbb{F}_p$-multiplications. Since $p \equiv 7 \bmod 12$, we have $\#\mathcal{S}_{p^2} = \lfloor p/12 \rfloor + 1$ (see Theorem 4.2.4), and since $p \equiv 7 \bmod 8$, $\#\mathcal{S}_p$ is exactly the class number of the imaginary quadratic field $\mathbb{Q}(\sqrt{-p})$ (see Proposition 4.2.6). We suppose this class number is h, i.e., $\#\mathcal{S}_p = h$. Writing $h = 2^k$, where k is correct to 3 decimal places, we would obtain that the average cost of breaking this instance of SQIsign is $2^{257.564-k}$ multiplications in $\mathbb{F}_p$. Note that, for large, cryptographic sized primes p, computing class numbers is very computationally expensive. Indeed, a recent class group computation for a 512-bit prime terminated in approximately 52 core years. Alternatively, we can use our theoretical upper bound on $\#\mathcal{S}_{p^2}/\#\mathcal{S}_p$ from Equation (7.6).

In Table 7.7 we give average counts for the cost of breaking the supersingular isogeny problem using SuperSolver for a number of primes underlying either B-SIDH[6] or SQIsign.

Prime p	p mod 8	Average number of $\mathbb{F}_p$-mults. per node	$\#\mathcal{S}_{p^2}$	$\#\mathcal{S}_p$	Average cost of SuperSolver
B-SIDH-p247 [99]	7	55.6	$2^{242.559}$	2^{k_1}	$2^{248.356-k_1}$
TwinSmooth-p250 [105]	1	59.1	$2^{246.220}$	2^{k_2-1}	$2^{252.105-k_2}$
SQISign-p256 [125]	7	55.7	$2^{251.764}$	2^{k_3}	$2^{257.564-k_3}$
TwinSmooth-p384 [105]	1	63.1	$2^{379.735}$	2^{k_4-1}	$2^{385.715-k_4}$
TwinSmooth-p512 [105]	5	69.1	$2^{507.896}$	2^{k_5-1}	$2^{514.007-k_5}$

TABLE 7.7: The average number of $\mathbb{F}_p$-multiplications required to solve the supersingular isogeny problem using SuperSolver. When $p \equiv 1 \bmod 4$, we assume that $N = 2^k$ is the class number of $\mathbb{Q}(\sqrt{-4p})$, where k correct to 3 decimal places. Otherwise, it is the class number of $\mathbb{Q}(\sqrt{-p})$. As N varies for each prime, we will index N and k by the row in the table, i.e., $N_i = 2^{k_i}$ will be the class number of the i-th prime in the table. The number of $\mathbb{F}_p$-multiplications per node using SuperSolver is taken from Table 7.1.

Remark 7.6.2. Converting the number of $\mathbb{F}_p$-multiplications into a clock cycle count depends heavily on the prime p. Therefore, to get a rough idea on how the $\mathbb{F}_p$-multiplication counts for SuperSolver translate into clock cycles, we turn to Scott's optimised C implementation of $\mathbb{F}_p$ arithmetic for the pseudo-Mersenne prime $2^{255} - 19$ (see [282] for more details). Running the same experiment as those reported in Table 7.6 on this prime, we find that 53.0 $\mathbb{F}_p$-mutiplications are needed per node revealed (using the optimal set computed to be $\{3, 5, 7, 9, 11, 13\}$). In Table

[6]B-SIDH is now broken.

2 from [282], Scott reports that at most 62 Skylake x86-64 clock cycles are needed for an $\mathbb{F}_p$-multiplication, thus giving us 3286 clock cycles per node revealed.

Chapter 8

The isogeny problem in dimension 2

In this chapter, we study the general supersingular isogeny problem in dimension 2, and determine the concrete complexity of the Costello–Smith attack. Using explicit parametrisations of the moduli space of genus-2 curves whose Jacobians are split by an (N, N)-isogeny, we give an improved attack with lower concrete complexity. For cryptographic-sized primes $p \approx 2^{128}$, we achieve an improvement by around a factor of 25. The chapter is based on the paper

> *An algorithm for efficient detection of (N, N)-splittings and its application to the isogeny problem in dimension 2*

which is joint work with Craig Costello and Sam Frengley, published at PKC 2024 [88], where it was given the Best Paper Award. The chapter is for all intents and purposes the same as the published, barring minor editorial changes and the removal of repeated preliminaries.

Introduction

Let C and C' be genus-2 curves with superspecial Jacobians. The general dimension-2 *superspecial isogeny problem* asks us to find an isogeny

$$\phi \colon \mathcal{J}_C \to \mathcal{J}_{C'},$$

of p.p. abelian surfaces, where $\mathcal{J}_C$ and $\mathcal{J}_{C'}$ are the Jacobians of C and C' respectively.

We say that the Jacobian $\mathcal{J}_C$ of genus-2 curve C is *split* (over $\Bbbk$) if there exists a separable (polarised) $\Bbbk$-isogeny of p.p. abelian surfaces $\mathcal{J}_C \to E_1 \times E_2$ where $E_1/\Bbbk$ and $E_2/\Bbbk$ are elliptic curves.

The best known algorithm for solving the superspecial isogeny problem is due to Costello and Smith [107] and consists of two stages. We recall its description from Section 4.3.2. The first stage computes pseudorandom walks away from the two input Jacobians to find paths to products of two supersingular elliptic curves, i.e., $\varphi \colon \mathcal{J}_C \to E_1 \times E_2$ and $\varphi' \colon \mathcal{J}_{C'} \to E_1' \times E_2'$. Assuming the pseudorandom walks quickly converge to the uniform distribution, the first stage runs in $\widetilde{O}(p)$ classical bit operations, since the proportion of superspecial abelian surfaces that are isomorphic to a product of elliptic curves is $O(1/p)$. The second stage calls the $\widetilde{O}(p^{1/2})$ Delfs–Galbraith algorithm (see Section 4.3.1) to find paths between E_1 and E_1' and between E_2 and E_2'. These are then glued together to obtain the path $\pi \colon E_1 \times E_2 \to E_1' \times E_2'$ connecting φ and φ' in order to output the full solution $\phi := \widehat{\varphi'} \circ \pi \circ \varphi$. It follows that the entire algorithm runs in $\widetilde{O}(p)$ classical bit operations on average, with the cost dominated by the first step: finding paths to products of elliptic curves.

Looking at Example 6.3.3, this algorithm follows the general strategy given at the start of Part II, where the special subset $X_2 \subset \mathcal{S}_2(\bar{\mathbb{F}}_p)$ in this case is $\mathcal{E}_2(\bar{\mathbb{F}}_p)$.

Isogeny-based cryptography in dimension 2. The product-finding algorithm that we accelerate in this work solves the general superspecial isogeny problem, which underlies the security of various isogeny-based protocols in dimension 2. An example of such a scheme is the dimension-2 analogue of the CGL hash function [76], which was proposed by Takashima [303] and later extended by Castryck, Decru, and Smith [72].

The 2022 breaks of SIDH and SIKE [67, 227, 271] revealed that understanding higher dimensional isogenies is essential to navigate the isogeny graphs in dimension 1. More recently there has been a line of works leveraging the techniques used in the attacks to propose new cryptosystems that exploit isogeny computations in higher dimensions, for example SQIsignHD [116], FESTA [19], and SCALLOP-HD [79]. Although the hard problems underlying these schemes are not directly impacted by the algorithm that is optimised in this chapter, we believe the trend towards instantiating schemes in higher dimensions will only make the dimension-2 supersingular isogeny problem more relevant to practitioners as the field of isogeny-based cryptography continues to mature.

Based on the present knowledge of attacks in dimension 2, we believe it is reasonable to speculate that the complexity of the product-finding algorithm may eventually be used as an upper-bound on the classical hardness of attacking many schemes that are currently conceivable, even when the underlying instances of the isogeny problem are special cases of its general formulation (provided no superior algorithm for the special problem is found, of course). For example, consider the dimension-2 analogue of the Σ-protocol that proves knowledge of an isogeny degree of a specified length (see [119] for the latest on this protocol). In dimension 1, the best known classical attack on this protocol is the van Oorschot–Wiener (vOW) meet-in-the-middle algorithm [253]. In dimension 2, however, the $\widetilde{O}(p)$ product-finding algorithm will solve the general problem at least as fast as the van Oorschot–Wiener meet-in-the-middle algorithm [253], and is likely to become the preferred algorithm[1] for large enough p.

Contributions

We begin with an implementation of the algorithm described above for finding paths to products of elliptic curves. This includes a streamlined version of the Takashima–Yoshida algorithm [304, §5.5] for computing chains of Richelot isogenies. With this optimised algorithm, we provide a toolbox for exploring the $(2, 2)$-isogeny graph. The expansion properties of the $(2, 2)$-isogeny graph are not well understood, and our implementation is well suited to exploring this. For example, one could hope to provide evidence towards Conjecture 4.2.9. Understanding the expansion properties of this graph is crucial to gaining a deeper insight into the hardness of the general isogeny problem in dimension 2.

This lays the foundation for the main contribution of this work: a new algorithm that speeds up the search for paths to products of elliptic curves. At the heart of our algorithm is the work

[1]For a fixed memory bound w and single processor, taking $n = p^{3/4}$ [159, §4.1] in [253, Equation 4] gives an asymptotic runtime of $O(p^{9/8})$ on a single core. Moreover, parallel processors running vOW must read from, and write to, the huge central storage database (which hampers parallel performance in practice), while product-finding is memory-free and parallelises *perfectly*.

	Walks in $\mathcal{X}_2(\bar{\mathbb{F}}_p, 2)$ without additional searching [107] (optimised in Section 8.2)	Walks in $\mathcal{X}_2(\bar{\mathbb{F}}_p, 2)$ w. split searching in $\mathcal{X}_2(\bar{\mathbb{F}}_p, N)$ **This work**		
p (bits)	$\mathbb{F}_p$-mults. per node	set $N \in \{\dots\}$	$\mathbb{F}_p$-mults. per node	**improv. factor**
50	579	$\{2,3\}$	35	**16.5x**
100	1176	$\{2,3\}$	48	**24.5x**
150	1575	$\{3,4\}$	54	**29.2x**
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
950	9772	$\{4,6\}$	69	**141.6x**
1000	11346	$\{4,6\}$	71	**159.8x**

Table 8.1: An abbreviated version of Table 8.5. See Section 8.6 for further explanation.

of Kumar [206], who gives explicit parametrisations of the moduli space of genus-2 curves whose Jacobians are split by an (N, N)-isogeny. When we step to a new node in the Richelot isogeny graph, these parametrisations allow us to efficiently test whether *any* of the (N, N)-isogenous neighbours are isomorphic to a product of supersingular elliptic curves without computing any expensive (N, N)-isogenies. For example, over a field whose characteristic is a 100-bit prime, an optimised Richelot isogeny (see Section 8.2) requires 1176 $\mathbb{F}_p$-multiplications. This is the cost of taking a single step in the Richelot isogeny graph, which reveals only one neighbour and is thus the per-node cost of running the Costello–Smith attack. However, using the new algorithm we describe in Section 8.4 with $N = 3$, we are able to test whether any of the $(3, 3)$-isogenous neighbours are split with a total of 767 $\mathbb{F}_p$-multiplications. Since there are 40 such neighbours, the per-node cost of simultaneously searching these neighbours is less than 20 $\mathbb{F}_p$-multiplications each. The upshot is that when attacking an instance of the superspecial isogeny problem, we can sift through a larger proportion of superspecial Jacobians per unit time, thus reaching an elliptic curve product with fewer $\mathbb{F}_p$-multiplications.

In Section 8.6 we report on a number of experiments conducted over both small primes (where instances of the superspecial isogeny problem can be solved) and large primes of cryptographic size. Applying our accelerated algorithm to find paths to elliptic products when $p = 2^{31} - 1$, we solved 10 instances of the problem using an average of $2^{33.0}$ multiplications in $\mathbb{F}_p$ for an average wall time of $2^{16.3}$ seconds. Our optimised version of the original algorithm due to Costello and Smith [107] required an average of $2^{36.8}$ multiplications in $\mathbb{F}_p$ for an average runtime of $2^{20.5}$ seconds to solve the same 10 instances. In Table 8.1 we give a snapshot of the improvements that were observed in our implementation for a number of large primes of varying bitlength. We see that the relative speed-up improves as the prime p grows in size (see Section 8.6 for more details).

For primes of at least 150 bits, we argue in Section 8.5.3 that (heuristically) Algorithm 8.4

requires an expected number of

$$\left(\frac{14\log(p) + 34490}{5 \cdot 664}\right) p + O(\log(p))$$

$\mathbb{F}_p$-multiplications before encountering a product of elliptic curves. Under the same heuristics, our optimised version of the algorithm in [107] would require a larger expected $\left(\frac{12\log(p)+129}{5}\right) p + O(\log(p))$ $\mathbb{F}_p$-multiplications.

We note that our algorithm for detecting (N,N)-splittings may be of interest outside our target application of the dimension-2 superspecial isogeny problem. For example, it answers a question posed by Castryck and Decru for $d \leq 11$ [67, §11.3]:

> *Let $H/\mathbb{F}_{p^2}$ be a genus 2 curve with superspecial Jacobian $\mathcal{J}$, and $d > 1$ an integer. Is there a (d,d)-isogeny $\Phi : \mathcal{J} \to A$ such that A is a product of elliptic curves?*

Applications to cryptanalysis. In the security analysis of their hash function, Castryck, Decru, and Smith [72] correctly argue that, since the steps taken by their hash function correspond entirely to "good extensions" (see Section 8.2.2), the path returned by the Costello–Smith algorithm (which does not only consist of good extensions) is not a valid preimage [72, Footnote 11]. However, more recent work by Florit and Smith [155, §6.2 - 6.4] shows that collisions in the Castryck–Decru–Smith (CDS) hash function can be constructed once a walk to an elliptic product is known. So long as we assume our walks approximate the random distribution on $\mathcal{X}_2(\bar{\mathbb{F}}_p, 2)$ (more on this in Remark 8.2.1), then we consider it prudent to use the complexity of the product-finding algorithms to classify the security of a given instance of the CDS hash function, even if preimage resistance is the governing security property.

As will become apparent in Section 8.5, our acceleration of the Costello–Smith algorithm will return a $(2^n N, 2^n N)$-isogeny (for some $n \in \mathbb{N}$ and positive integer $N \leq 11$). However, for many cryptographic protocols in isogeny-based cryptography, the secret isogeny will be of a specified degree, usually a prime power ℓ^k. Though an algorithm that transforms a $(2^n N, 2^n N)$-isogeny to a (ℓ^k, ℓ^k)-isogeny has yet to be developed, for example by generalising the KLPT algorithm (see Section 5.1.1) to dimension 2, we find it prudent to conjecture such an algorithm exists, rather than betting the security of primitives on the converse.

Availability of Software

All of the source code accompanying this chapter is written in `MAGMA` [49] and can be found at

`https://github.com/mariascrs/SplitSearcher`.

Related work

At a high level, our improvements to the dimension-2 superspecial isogeny attack can be viewed as an analogue of those recently given by the SuperSolver algorithm presented in Chapter 7. Indeed, as described in the introduction of Part II, both attacks use random walks to find special nodes in the graph to reduce the (remainder of the) algorithm to a comparatively easier isogeny problem:

the special nodes in the Delfs–Galbraith algorithm are the isomorphism classes of elliptic curves defined over $\mathbb{F}_p$, while the special nodes in the Costello–Smith algorithm are the isomorphism classes of products of elliptic curves. The key to the improvements in Chapter 7 was an efficient method for determining whether modular polynomials have subfield roots without computing any such roots explicitly. This allows many nodes to be simultaneously searched over without being visited by means of expensive isogeny computations. The key to the improvements in this chapter stem from Kumar's parametrisations of the moduli space of genus 2 curves whose Jacobians are split by an (N, N)-isogeny [206]. In a similar vein to Chapter 7, we show that these can be used to simultaneously search over many neighbours without visiting the corresponding nodes in the isogeny walks.

It is worth noting that, relatively speaking, the improvements found in this work are significantly larger than the improvements reported in Chapter 7 in the dimension-1 case. At a first glance of Section 8.4, it seems our batch (N, N)-split searching requires a lot more computation than the analogous batch N-isogenous subfield curve searching done in SuperSolver. However, in dimension 2 we are processing $O(N^3)$ neighbours simultaneously (see Equation (2.8)), while the subfield search in dimension 1 is batch testing $O(N)$ neighbours each time (see Equation (2.7)). For primes of size 50 to 800 bits, Table 7.6 in Chapter 7 reports speed-ups ranging from 2.8x to 17.6x, while the speedups we found for primes of these same sizes range from 16.5x to 116.3x, as given in Table 8.5.

Outline

After detailing our optimised version of the original walk in $\mathcal{X}_2(\bar{\mathbb{F}}_p, 2)$ from [107] in Section 8.2, in Section 8.3 we recall standard results concerning moduli spaces for genus-2 curves with split Jacobians and Kumar's formulae [206]. In Section 8.4 we present the main contribution of this work: an efficient algorithm to detect (N, N)-splittings. We give the full algorithm and discuss our implementation in Section 8.5. Finally, we present the experimental results in Section 8.6 before we conclude by mentioning some possible avenues for improving the algorithm.

8.1 Preliminaries

For this chapter, a reader should be familiar with principally polarised abelian surfaces, namely products of elliptic curves and Jacobians of genus-2 hyperelliptic curves, as defined in Section 2.5. In particular, we work with superspecial abelian surfaces defined over finite fields and isogenies between them, detailed in Sections 2.8.2, 2.9 and 4.1. By studying the structure of the superspecial dimension-2 isogeny graph (see Section 4.2.2), we analyse and improve upon the concrete complexity of the best classical attack against the dimension-2 general isogeny problem detailed in Section 4.3.2.

8.2 Optimised product finding in $\mathcal{X}_2(\bar{\mathbb{F}}_p, 2)$

The best known algorithm for solving the superspecial isogeny problem in dimension 2 exploits the subset $X_2 = \mathcal{E}_2(\bar{\mathbb{F}}_p) \subseteq \mathcal{S}_2(\bar{\mathbb{F}}_p)$. It is depicted in Algorithm 8.1, and is due to Costello and Smith [107]. Recall from Section 4.3.2 that the overall cost of the algorithm is $\widetilde{O}(p)$ bit operations, with the bottleneck being Steps 1 and 2: finding paths $\varphi\colon A \to E_1 \times E_2$ and $\varphi'\colon A' \to E_1' \times E_2'$, where both $E_1 \times E_2 \in \mathcal{E}_2(\bar{\mathbb{F}}_p)$ and $E_1' \times E_2' \in \mathcal{E}_2(\bar{\mathbb{F}}_p)$. From this point onwards, we focus on improving the efficiency of this bottleneck step, as this will impact the concrete complexity of Algorithm 8.1.

Algorithm 8.1 Computing isogeny paths in $\mathcal{X}_2(\bar{\mathbb{F}}_p, N)$ [107]

```
Input: $A$ and $A'$ in $\mathcal{S}_2(\bar{\mathbb{F}}_p)$
Output: A path $\phi\colon A \to A'$ in $\mathcal{X}_2(\bar{\mathbb{F}}_p, N)$

 1: Find a path $\varphi$ from $A$ to some $E_1 \times E_2$ in $\mathcal{E}_2(\bar{\mathbb{F}}_p)$
 2: Find a path $\varphi'$ from $A'$ to some $E_1' \times E_2'$ in $\mathcal{E}_2(\bar{\mathbb{F}}_p)$
 3: Find a path $\psi_1\colon E_1 \to E_1'$ using (elliptic curve) path finding
 4: Find a path $\psi_2\colon E_2 \to E_2'$ using (elliptic curve) path finding
 5: if length($\psi_1$) $\not\equiv$ length($\psi_2$) $\pmod 2$ then
 6:     return $\perp$
 7: else
 8:     Construct a path $\pi\colon E_1 \times E_2 \to E_1' \times E_2'$ using $\psi_1, \psi_2$ as in [107, Lemma 3]
 9:     return the path $\phi := \widehat{\varphi}' \circ \pi \circ \varphi$ from $A$ to $A'$
10: end if
```

The aim of this section is to describe an optimised instantiation of the product finding algorithm given in Algorithm 8.1. Our instantiation uses pseudo-random walks in the superspecial subgraph of the Richelot isogeny graph [154, Definition 1] and exploits a streamlined version of Takashima and Yoshida's Richelot isogeny algorithm [304] to take efficient steps therein.

8.2.1 Taking a step in $\mathcal{X}_2(\bar{\mathbb{F}}_p, 2)$

We start by deriving a streamlined version of Takashima and Yoshida's Richelot isogeny algorithm [304, Algorithm 2] that will be used as the basis for pseudo-random walks in the superspecial subgraph of $\mathcal{X}_2(\bar{\mathbb{F}}_p, 2)$.

On input of the six-tuple $\boldsymbol{a} = (a_0, \dots, a_5) \in (\mathbb{F}_{p^2})^6$ defining[2] the genus-2 curve

$$C/\mathbb{F}_{p^2}\colon y^2 = (x - a_0)\cdots(x - a_5),$$

the algorithm outputs the six-tuple $\boldsymbol{a'} = (a_0', \dots, a_5') \in (\mathbb{F}_{p^2})^6$ that defines

$$C'/\mathbb{F}_{p^2}\colon y^2 = (x - a_0')\cdots(x - a_5'),$$

where $\phi\colon \mathcal{J}_C \to \mathcal{J}_{C'}$ is the Richelot isogeny whose non-trivial kernel is precisely the three points with Mumford coordinates $((x - a_i)(x - a_{i+1}), 0) \in \mathcal{J}_C[2]$ with $i \in \{0, 2, 4\}$. Using terminology introduced by Smith [294], we say that this Richelot isogeny corresponds to the quadratic splitting $\{(x - a_0)(x - a_1), (x - a_2)(x - a_3), (x - a_4)(x - a_5)\}$.

The main modifications we have made to their algorithm are:

- We assume that both the equations for C and C' are indeed given by the sextic polynomials whose six roots are rational elements of $\mathbb{F}_{p^2}$. This avoids the case distinctions made by

[2]For odd primes p, superspecial abelian surfaces always has full $\mathbb{F}_{p^2}$-rational 2-torsion (see Section 4.1), which in particular implies that the a_i are defined over $\mathbb{F}_{p^2}$.

Takashima and Yoshida that allow for quintic inputs and outputs (i.e., one of the a_i and/or a'_j being at infinity), which are unnecessary for our purposes. Indeed, they occur with negligible probability, and after a change of coordinates we may assume that C and C' are defined by sextics.

- We do not keep track of the leading coefficient of the sextic, since this merely determines which quadratic twist we are on, which is irrelevant for our application because twists correspond to the same node in $\mathcal{X}_2(\bar{\mathbb{F}}_p, 2)$. This means we avoid the final inversion in Line 33 of [304, Algorithm 2].
- Each of the three iterations of their main loop involves separate inversion and square root computations. In each case, we use $\mathsf{InvSqrt}$, which merges the inversion and square root into one combined inverse-and-square-root computation (see Line 7 of Algorithm 8.2) using the trick described by Scott [281].

On top of a small, fixed, number of field multiplications, Algorithm 8.2 computes a Richelot isogeny using 3 calls to $\mathsf{InvSqrt}$, which is essentially the same cost as a square root in $\mathbb{F}_{p^2}$ (see Section 7.2.1). This means our streamlined version saves all of the four additional inversions in $\mathbb{F}_{p^2}$ reported by Takashima and Yoshida [304, §5.5]. Otherwise, the notation and description of the algorithm is essentially unchanged: the indices in Line 3 of Algorithm 8.2 are taken modulo 6, and the indices in Line 5 are taken modulo 3.

Algorithm 8.2 RIsog: A Richelot isogeny in the general case

Input: $\boldsymbol{a} = (a_0, \ldots, a_5) \in (\mathbb{F}_{p^2})^6$ defining $C/\mathbb{F}_{p^2}\colon y^2 = (x - a_0)\cdots(x - a_5)$.
Output: $\boldsymbol{a'} = (a'_0, \ldots, a'_5) \in (\mathbb{F}_{p^2})^6$ defining $C'/\mathbb{F}_{p^2}\colon y^2 = (x - a'_0)\cdots(x - a'_5)$, where $\phi\colon \mathcal{J}_C \to \mathcal{J}_{C'}$ is a Richelot isogeny whose kernel contains the three points $((x - a_i)(x - a_{i+1}), 0)$ for $i = 0, 2, 4$; and `split`, a boolean that is true if the image of ϕ is in $\mathcal{E}_2(\bar{\mathbb{F}}_p)$.

```
1: Initialise $\boldsymbol{\lambda} \leftarrow [\boldsymbol{a}[0]\cdot\boldsymbol{a}[1], \boldsymbol{a}[2]\cdot\boldsymbol{a}[3], \boldsymbol{a}[4]\cdot\boldsymbol{a}[5]]$, $\boldsymbol{\theta} \leftarrow []$, $\boldsymbol{a'} \leftarrow []$
2: for $j = 0$ to $2$ do
3:     $\boldsymbol{\rho} \leftarrow [\boldsymbol{a}[2j+2] - \boldsymbol{a}[2j+4], \boldsymbol{a}[2j+3] - \boldsymbol{a}[2j+5], \boldsymbol{a}[2j+2] - \boldsymbol{a}[2j+5], \boldsymbol{a}[2j+3] - \boldsymbol{a}[2j+4]]$
4:     $\boldsymbol{\theta}[j] \leftarrow \boldsymbol{\rho}[0] + \boldsymbol{\rho}[1]$
5:     $\nu \leftarrow \boldsymbol{\lambda}[j+1] - \boldsymbol{\lambda}[j+2]$
6:     $\delta \leftarrow \boldsymbol{\rho}[0]\cdot\boldsymbol{\rho}[1]\cdot\boldsymbol{\rho}[2]\cdot\boldsymbol{\rho}[3]$.
7:     $(\mu, \kappa) \leftarrow \mathsf{InvSqrt}(\boldsymbol{\theta}_j, \delta)$
8:     $(\boldsymbol{a'}[2j], \boldsymbol{a'}[2j+1]) \leftarrow ((\nu + \kappa)\cdot\mu, (\nu - \kappa)\cdot\mu)$
9: end for
10: split $\leftarrow (\boldsymbol{\lambda}[0]\cdot\boldsymbol{\theta}[0] + \boldsymbol{\lambda}[1]\cdot\boldsymbol{\theta}[1] + \boldsymbol{\lambda}[2]\cdot\boldsymbol{\theta}[2]) = 0$
11: return $(\boldsymbol{a'}, \texttt{split})$
```

Alternatives for computing $(2^n, 2^n)$-isogenies. There are numerous ways to compute chains of $(2,2)$-isogenies that would be fit for our purposes, but we are yet to find one that can appreciably outperform repeated calls to Algorithm 8.2. Recall that each such call computes a $(2,2)$-isogeny using a fixed number of $\mathbb{F}_p$-multiplications on top of three calls to the merged inversion-and-square root computation (i.e., $\mathsf{InvSqrt}$). Castryck and Decru's multiradical variant of a Richelot isogeny also requires at least three square root computations in $\mathbb{F}_{p^2}$ [69, §4.2], so the most we could expect to gain using their formulae is in the constant number of additional $\mathbb{F}_p$-operations (assuming any field inversions required in their case can also be absorbed into the square root calls). Kunzweiler's efficient $(2^n, 2^n)$-isogeny algorithm [207] could also be used in our scenario, but in testing this algorithm against ours we observed that, on average, ours performs between 3x and 5x faster for the two primes considered by Kunzweiler. Note, Kunzweiler's formulae were derived with a different target application (i.e., G2SIDH) in mind, meaning computing a chain of $(2,2)$-isogenies of fixed length n is most efficient when $2^n \mid p+1$. In our algorithm, we compute chains of length much larger than any such n and, as a result, this comparison is unfair to [207]. Our comparison is to ensure that we are not sacrificing efficiency in our context. An alternative is the algorithm developed by Dartois, Maino, Pope, and Robert [117] for computing a chain of $(2,2)$-isogenies using theta coordinates. However, as with Kunzweiler's formulæ, their algorithm for computing a chain of length n is most efficient when our starting Jacobian has full $\mathbb{F}_{p^2}$-rational 2^n-torsion, and so is unlikely to be competitive with RIsog for a general prime p.

8.2.2 Walking in the superspecial subgraph of $\mathcal{X}_2(\bar{\mathbb{F}}_p, 2)$

We now turn to describing walks in the superspecial subgraph of $\mathcal{X}_2(\bar{\mathbb{F}}_p, 2)$ that take steps using the RIsog algorithm developed above. To ensure that these walks are non-backtracking and avoid short cycles, the output of RIsog must first be permuted so that the quadratic splitting implicit to its ordering (see Section 8.2.1) corresponds to a *good extension* of the previous $(2,2)$-isogeny, i.e., a $(2,2)$-isogeny whose kernel intersects trivially with the kernel of the dual of the previous $(2,2)$-isogeny.

Kernel permutations corresponding to good extensions. Following Castryck, Decru, and Smith [72], there are 8 non-equivalent permutations of our a_i which correspond to good extensions of the previous $(2,2)$-isogeny. Our walks are deterministically defined by pseudorandom bitstrings. Each step uses three bits to choose which of the 8 good extensions defines our next $(2,2)$-isogeny. Using [72, Proposition 3], we define the function $\mathbf{a} \leftarrow \mathsf{PermuteKernel}(\mathbf{a}, \texttt{bits})$ by

$$\mathbf{a} \leftarrow \begin{cases} (\mathbf{a}[0], \mathbf{a}[2], \mathbf{a}[1], \mathbf{a}[4], \mathbf{a}[3], \mathbf{a}[5]), & \texttt{bits} = 0|0|0; \\ (\mathbf{a}[0], \mathbf{a}[2], \mathbf{a}[1], \mathbf{a}[5], \mathbf{a}[3], \mathbf{a}[4]), & \texttt{bits} = 0|0|1; \\ (\mathbf{a}[0], \mathbf{a}[3], \mathbf{a}[1], \mathbf{a}[4], \mathbf{a}[2], \mathbf{a}[5]), & \texttt{bits} = 0|1|0; \\ (\mathbf{a}[0], \mathbf{a}[3], \mathbf{a}[1], \mathbf{a}[5], \mathbf{a}[2], \mathbf{a}[4]), & \texttt{bits} = 0|1|1; \\ (\mathbf{a}[0], \mathbf{a}[4], \mathbf{a}[1], \mathbf{a}[2], \mathbf{a}[3], \mathbf{a}[5]), & \texttt{bits} = 1|0|0; \\ (\mathbf{a}[0], \mathbf{a}[4], \mathbf{a}[1], \mathbf{a}[3], \mathbf{a}[2], \mathbf{a}[5]), & \texttt{bits} = 1|0|1; \\ (\mathbf{a}[0], \mathbf{a}[5], \mathbf{a}[1], \mathbf{a}[3], \mathbf{a}[2], \mathbf{a}[4]), & \texttt{bits} = 1|1|0; \\ (\mathbf{a}[0], \mathbf{a}[5], \mathbf{a}[1], \mathbf{a}[2], \mathbf{a}[3], \mathbf{a}[4]), & \texttt{bits} = 1|1|1. \end{cases}$$

Remark 8.2.1. Recall that under a mild conjecture on the associated eigenvalues stated in Conjecture 4.2.9, Florit and Smith [155] show that, despite Richelot isogeny graphs not having optimal expansion, walks of length $O(\log p)$ still approximate the stationary distribution on $\mathcal{X}_2(\bar{\mathbb{F}}_p, 2)$. This statement is implicitly assuming that walks are unrestricted, i.e., that each step can take any one of the 15 outgoing Richelot isogenies. In choosing to restrict each step in $\mathcal{X}_2(\bar{\mathbb{F}}_p, 2)$ to the 8 good edges with the aim of avoiding fruitless cycles, we are under the implicit assumption that these walks also rapidly approximate the stationary distribution. All of our experiments over small primes produced results that support this assumption (see Section 8.6), and Florit and Smith [155, §6.4] also comment in its favour. Nevertheless, if future research provides evidence to the contrary, modifying our walks to include the 6 other extensions is straightforward. In this case we could either aim to prohibit certain sequences of isogenies that cycle back to prior nodes, or (since we abandon walks after a small number of steps – see below) simply tolerate the possibility of revisiting prior nodes. Even if a walk did cycle back and hit a prior node, in general we would have a 14^{-n} chance of continuing along the same path for n steps thereafter.

Pseudorandom walks in the superspecial subgraph of $\mathcal{X}_2(\bar{\mathbb{F}}_p, 2)$. A given step of our pseudorandom walk can now be defined as $\mathbf{a} \leftarrow \mathsf{Step}(\mathbf{a}, \mathtt{bits})$, where the function Step is simply given by

$$\mathsf{Step}(\mathbf{a}, \mathtt{bits}) = \mathsf{RIsog}(\mathsf{PermuteKernel}(\mathbf{a}, \mathtt{bits})).$$

Recall from Lines 1 and 2 of Algorithm 8.1 that our goal is to find a path φ from $A \in \mathcal{S}_2(\bar{\mathbb{F}}_p)$ to some $E_1 \times E_2 \in \mathcal{E}_2(\bar{\mathbb{F}}_p)$. In principle, one could continue walking deterministically from the input node $A \in \mathcal{S}_2(\bar{\mathbb{F}}_p)$ for as long as it takes to find the splitting $E_1 \times E_2 \in \mathcal{E}_2(\bar{\mathbb{F}}_p)$, but the length of this path would be $O(p)$. To ensure a compact description of the solution, we instead take a relatively small number of steps from $A \in \mathcal{S}_2(\bar{\mathbb{F}}_p)$ before abandoning a walk, returning to $A \in \mathcal{S}_2(\bar{\mathbb{F}}_p)$, and starting again.

Our implementation uses `MAGMA`'s inbuilt function $\mathsf{SHA1}\colon \{0,1\}^* \to \{0,1\}^{160}$ to generate pseudorandom walks consisting of 160 Richelot isogeny steps as follows. We start by setting $H_0 := \mathsf{StartingSeed}(\mathbf{a})$, where $\mathbf{a} \in (\mathbb{F}_{p^2})^6$ defines the input node $A \in \mathcal{S}_2(\bar{\mathbb{F}}_p)$, and where $\mathsf{StartingSeed}$ merely concatenates and parses the 12 $\mathbb{F}_p$ components of $\mathbf{a}$ in order to be fed as input into $\mathsf{SHA1}$. We then define the function $\mathsf{Hash}\colon \{0,1\}^* \to \{0,1\}^{480}$ as $\mathsf{Hash}\colon s \mapsto \mathsf{SHA1}(s)||\mathsf{SHA1}^2(s)||\mathsf{SHA1}^3(s)$, where $\mathsf{SHA1}^2(s)$ denotes $\mathsf{SHA1}(\mathsf{SHA1}(s))$, etc. Our first walk in $\mathcal{X}_2(\bar{\mathbb{F}}_p, 2)$ is defined by $H_1 = \mathsf{Hash}(H_0)$; these 480 bits are used (three bits at a time) to give 160 steps away from $A \in \mathcal{S}_2(\bar{\mathbb{F}}_p)$, at which point we return back to $A \in \mathcal{S}_2(\bar{\mathbb{F}}_p)$ and repeat the process by using $H_{i+1} = \mathsf{Hash}(H_i)$ for $i = 1, 2, \ldots$, until one of our calls to RIsog returns $\mathtt{split} = \mathtt{true}$, at which point our walks have hit a node in the special subset $\mathcal{E}_2(\bar{\mathbb{F}}_p)$. To proceed to the elliptic curve path finding in Steps 3 and 4 of Algorithm 8.1, the j-invariants of the elliptic curves in the product of the final $(2,2)$-isogeny are determined using [72, §6.2]. This concludes the description of our implementation of the generic product finding algorithm from [107] that works entirely in $\mathcal{X}_2(\bar{\mathbb{F}}_p, 2)$.

Choice of Optimisations. In our search for product curves we use optimised walks in $\mathcal{X}_2(\bar{\mathbb{F}}_p, 2)$, rather than adopting Castryck and Decru's multiradical isogenies [69] to walk in $\Gamma_2(3; p)$. Indeed, their hash function built from multiradical $(3,3)$-isogenies between superspecial genus-2 Jacobians outperforms its $(2,2)$-counterpart by a factor of around 9. We first note that the bulk of the Castryck–Decru speedup comes from their hash function processing 3 trits of entropy per $(3,3)$-isogeny, rather than 3 bits of entropy processed by a $(2,2)$-isogeny. In our application, however, entropy is irrelevant, and we are only interested in the raw cost of taking one step in the graph. Nevertheless, Castryck and Decru still report approximately a 2.7x speed-up for a multiradical

$(3,3)$-isogeny (which is dominated by 3 cube roots over $\mathbb{F}_{p^2}$) compared to a multiradical $(2,2)$-isogeny (which is dominated by 3 square roots over $\mathbb{F}_{p^2}$), with this factor coming directly from the relative performance of cube roots and square roots in $\mathbb{F}_{p^2}$ in `MAGMA`. In our implementation, we optimise the computation of the square roots in $\mathbb{F}_{p^2}$ in terms of exponentiations and multiplications in $\mathbb{F}_p$ using the tricks due to Scott [281, §5.3] (see Section 7.2.1 for more details), and we are unaware of analogous (or any) tricks in the cube root case that could outperform the square root computation.

Furthermore, we use walks in $\mathcal{X}_2(\overline{\mathbb{F}}_p, 2)$ that do not store or recycle any information from previous steps. Indeed, we could not see an obvious way to (re)use any of the three square roots in Line 7 of Algorithm 8.2 to compute the other 7 good extensions. We remark that this is a feature of our choice to walk using only good extensions, and we could in fact recycle these square roots to compute some of the *bad* extensions. If it turns out that there *is* a way to compute all 8 of the image tuples $\mathbf{a}$ in appreciably fewer operations than calling the RIsog algorithm on all 8 kernels individually, then one could define an octonary tree in an analogous fashion to the binary tree used in SuperSolver, as described in Section 7.2.3.

Remark 8.2.2. Since the work in this chapter was completed, a new paper on computing radical $(2,2)$-isogenies using theta coordinates has been made public [208]. Similarly to RIsog, their formulas allow us to efficiently navigate $\mathcal{X}_2(\overline{\mathbb{F}}_p, 2)$ taking only good extensions. Due to the level-2 structure imposed by the theta model, radical 2-isogenies in the theta model may give a promising speed up. It wouldx therefore be interesting to further investigate if they can outperform RIsog.

8.3 Explicit moduli spaces for genus 2 curves with split Jacobians

We give a brief review of some well known facts about genus-2 curves with split Jacobians and their moduli. The reader wishing for a more in-depth discussion is encouraged to consult, for example, [57, §2], [163], [205], or [206].

8.3.1 The Igusa–Clebsch invariants of a genus-2 curve

Let $\Bbbk$ be a field of characteristic not equal to 2. Let $\mathcal{M}_2$ denote the variety whose points $[C] \in \mathcal{M}_2(\overline{\Bbbk})$ correspond to the $\overline{\Bbbk}$-isomorphism classes of genus-2 curves C defined over $\overline{\Bbbk}$.

As explicitly defined in Section 2.5.2, we may associate to any genus-2 curve $C/\Bbbk$ its Igusa–Clebsch invariants $I_2(C)$, $I_4(C)$, $I_6(C)$, and $I_{10}(C)$, where the subscript denotes the weight of the invariant. Moreover, the isomorphism class of $C/\overline{\Bbbk}$ is uniquely determined by its Igusa–Clebsch invariants. This induces a birational $\Bbbk$-morphism $\mathcal{M}_2 \hookrightarrow \mathbb{P}(1,2,3,5)$ given by associating to a class $[C]$ its Igusa–Clebsch invariants $[I_2(C) : I_4(C) : I_6(C) : I_{10}(C)]$.

8.3.2 Optimal splittings of Jacobians of a genus-2 curves

Let C be a curve of genus 2 defined over a field $\Bbbk$. Recall that we say the Jacobian $\mathcal{J}_C$ of C is *split* (over $\Bbbk$) if there exists a (polarised) separable $\Bbbk$-isogeny $\phi\colon \mathcal{J}_C \to E_1 \times E_2$ where $E_1/\Bbbk$ and $E_2/\Bbbk$

are elliptic curves.[3]

To work explicitly with subvarieties of $\mathcal{M}_2$ which parametrise genus-2 curves with split Jacobians, we will restrict our focus to Jacobians which are split by an (N, N)-isogeny. However, without imposing further conditions on the isogeny, our subvarieties will not be irreducible. Following Bruin–Doerksen [57, §2], we make the following definition:

Definition 8.3.1. Let $\mathbb{k}$ be a field, $C/\mathbb{k}$ be a curve of genus 2, and $E/\mathbb{k}$ be an elliptic curve. We say that a cover (i.e., a surjective morphism) $\psi\colon C \to E$ of degree N is *optimal* if N is coprime to the characteristic of $\mathbb{k}$ and ψ does not factor through a non-trivial unramified covering.

We say that a polarised separable isogeny $\varphi\colon \mathcal{J}_C \to E_1 \times E_2$ is an *optimal (polarised)* (N, N)-*splitting* if φ is an (N, N)-isogeny and the covering $C \to E_1$ induced by φ and the Abel–Jacobi map[4] is optimal. In this case $\mathcal{J}_C$ is said to be *optimally* (N, N)*-split.*

In our application N will be an integer ≤ 11 and $\mathbb{k}$ will be the finite field $\mathbb{F}_{p^2}$ for some prime number $p \gg 11$, so the assumption that φ is separable will be automatically satisfied. In fact, every splitting factors through an optimal (N, N)-splitting. More precisely, we have the following proposition.

Proposition 8.3.2. *If $\mathcal{J}_C$ is split (over $\mathbb{k}$) then there exists an integer $N \geq 2$ such that $\mathcal{J}_C$ is optimally (N, N)-split (over $\mathbb{k}$).*

Proof. We closely follow [57, Proposition 2.8]. Since $\mathcal{J}_C$ is split, there exists a separable $\mathbb{k}$-isogeny $\varphi\colon \mathcal{J}_C \to E_1 \times E_2$ where $E_1/\mathbb{k}$ and $E_2/\mathbb{k}$ are elliptic curves. Since φ is separable, there exists an elliptic curve $D_1/\mathbb{k}$ such that the morphism $C \to E_1$ induced by the Abel–Jacobi map and φ factors through an optimal cover $\psi\colon C \to D_1$. By [57, Lemma 2.6], ψ gives rise to an optimal (N, N)-splitting $\mathcal{J}_C \to D_1 \times D_2$, where D_2 is an elliptic curve and N is the degree of ψ. □

8.3.3 The surfaces $\widetilde{\mathcal{L}}_N$ and $\mathcal{L}_N$

We write $\widetilde{\mathcal{L}}_N$ for the surface whose $\mathbb{k}$-points parametrise ($\overline{\mathbb{k}}$-isomorphism classes of) pairs (C, φ) where C is a curve of genus 2 and $\varphi\colon \mathcal{J}_C \to E_1 \times E_2$ is an optimal (N, N)-splitting.

Replacing φ with its composition with the natural isomorphism $E_1 \times E_2 \to E_2 \times E_1$ gives an involution[5] on $\widetilde{\mathcal{L}}_N$. We write $\mathcal{L}_N$ for the quotient of $\widetilde{\mathcal{L}}_N$ by this involution. The natural map $\widetilde{\mathcal{L}}_N \longrightarrow \mathcal{M}_2$, given by $(C, \varphi) \longmapsto [C]$, factors via $\widetilde{\mathcal{L}}_N \to \mathcal{L}_N$.

Kumar [206] gave explicit models of the surface $\widetilde{\mathcal{L}}_N$ for each integer $N \leq 11$. In this range the surfaces $\mathcal{L}_N$ are rational (i.e., birational to $\mathbb{A}^2$), and they give an explicit model for the surface $\widetilde{\mathcal{L}}_N$ as a double cover (i.e., a degree 2 cover) of $\mathcal{L}_N$ together with the moduli interpretation of $\mathcal{L}_N$. More specifically, they compute rational functions $\mathcal{I}_2(r, s)$, $\mathcal{I}_4(r, s)$, $\mathcal{I}_6(r, s)$, $\mathcal{I}_{10}(r, s)$ which (after an appropriate projective rescaling) may be taken to lie in $\mathbb{Z}[r, s]$ and for which the following diagram commutes

$$\begin{tikzcd}
\mathbb{A}^2 \arrow[r, dashed, "\varphi_N"] \arrow[d, dashed] & \mathbb{P}(1,2,3,5) \\
\mathcal{L}_N \arrow[r] & \mathcal{M}_2 \arrow[u, hook]
\end{tikzcd}$$

[3]The convention that ϕ is separable contrasts with, e.g., [57, Definition 2.1].
[4]The Abel–Jacobi map $C \to \mathcal{J}_C$ gives an embedding of a curve C with genus $g \geq 1$ into its Jacobian $\mathcal{J}_C$.
[5]An involution is a morphism that is its own inverse.

Here the maps on the left and right are birational and the rational map φ_N is given by $(r,s) \mapsto [\mathcal{I}_2(r,s) : \mathcal{I}_4(r,s) : \mathcal{I}_6(r,s) : \mathcal{I}_{10}(r,s)]$.

We will employ the following lemma to detect whether a Jacobian $\mathcal{J}_C$ is optimally (N,N)-split over $\overline{\Bbbk}$.

Lemma 8.3.3. *The Jacobian of a genus-2 curve $C/\Bbbk$ is split over $\overline{\Bbbk}$ if and only if there exists an integer $N \geq 2$ such that the point $[C] \in \mathcal{M}_2(\overline{\Bbbk})$ lies in the image of $\mathcal{L}_N \to \mathcal{M}_2$.*

Proof. If $\mathcal{J}_C$ is split, then it is optimally (N,N)-split for some integer $N \geq 2$ by Proposition 8.3.2. In this case, the corresponding point on $\widetilde{\mathcal{L}}_N$ maps to $[C]$ on $\mathcal{M}_2$. Conversely, suppose $[C]$ lies in the image of $\mathcal{L}_N \to \mathcal{M}_2$. Since the morphism $\widetilde{\mathcal{L}}_N \to \mathcal{L}_N$ is a surjection on $\overline{\Bbbk}$-points, there exists an optimal (N,N)-splitting $\varphi \colon \mathcal{J}_C \to E_1 \times E_2$ such that the preimage of $[C]$ under this morphism is $(C,\varphi) \in \widetilde{\mathcal{L}}_N(\overline{\Bbbk})$. Hence, $\mathcal{J}_C$ is split. $\square$

Remark 8.3.4. Genus-2 curves with split Jacobians, and their moduli, have appeared many times elsewhere in the literature. Indeed, when $N = 2$, 3 and 4, generic families of genus-2 curves with optimally (N,N)-split Jacobians were known classically from work of Legendre, Jacobi, Hermite, Grousat, Burkhardt, Brioschi, and Bolza (see the introduction of [206] for a more in-depth discussion).

More recently, the surfaces $\widetilde{\mathcal{L}}_N$ for $2 \leq N \leq 5$ have been computed by exploiting the fact that if $\mathcal{J}_C$ is optimally (N,N)-split then there exist degree N morphisms $C \to E_1$ and $C \to E_2$. Kuhn [205] revisited this problem when $N = 3$ and Shaska [283] gave a method for general N for computing the surface $\widetilde{\mathcal{L}}_N$ together with a curve $C/\Bbbk(\widetilde{\mathcal{L}}_N)$ such that $\mathcal{J}_C$ is (N,N)-isogenous to a product $E_1 \times E_2$. This was extended to explicit computations when $N = 3, 5$ in [284] with further partial results when $N = 7$. When $2 \leq N \leq 5$, similar results also appear in various joint works of Shaska together with Magaard, Volklein, Wijesiri, Wolf, and Woodland [226, 285, 286, 287] and the work Gaudry–Schost [176] of Bröker–Lauter–Howe–Stevenhagen [53] and Djukanović [137, 138] when $N = 3$.

If a product of elliptic curves $E_1 \times E_2$ is (N,N)-isogenous over $\Bbbk$ to the Jacobian of a genus-2 curve then there exists a Galois equivariant isomorphism between their N-torsion subgroups which is anti-symplectic with respect to the Weil pairing (see e.g., [57, Proposition 2.8]). This description was employed by Bruin–Doerksen [57] to compute the surface $\widetilde{\mathcal{L}}_4$. Indeed, this implies that the surface $\widetilde{\mathcal{L}}_N$ is birational to the modular diagonal quotient surface $Z(N,-1)$ constructed by Kani and Schanz [197]. The surfaces $Z(N,-1)$ have been computed for several values of $N > 11$. In particular Fisher [152, 153] computed $Z(13,-1)$ and $Z(17,-1)$ and Frengley [162] computed $Z(12,-1)$. However, while these models recover the image $E_1 \times E_2$ of the splitting, they do not immediately give the genus-2 curve C. It would be interesting to give the degree 2 map $Z(N,-1) \to \mathcal{M}_2$ which recovers the moduli description of $\widetilde{\mathcal{L}}_N$.

8.3.4 The image of the morphism $\mathcal{L}_N \to \mathcal{M}_2$

Recall that we have a map $\mathcal{L}_N \to \mathcal{M}_2 \to \mathbb{P}(1,2,3,5)$ given by the Igusa–Clebsch invariants. The (Zariski closure of) the image of this map is a projective surface given by the vanishing of a polynomial $F_N \in \mathbb{Z}[I_2, I_4, I_6, I_{10}]$ which is homogeneous with respect to the weights.

If $\Bbbk$ is a field of characteristic coprime to $2N$, the Jacobian of a genus-2 curve $C/\Bbbk$ is optimally (N,N)-split over $\overline{\Bbbk}$ if and only if

$$F_N(I_2(C), I_4(C), I_6(C), I_{10}(C)) = 0.$$

For $2 \leq N \leq 5$ the polynomial F_N was computed by Bruin–Doerksen [56, 57, Theorem 1.2] and Shaska, Magaard, Volklein, Wijesiri, Wolf, and Woodland [226, 285, 287].

Such equations may be computed from Kumar's formulae [206]. For each $N \leq 5$ we interpolate the image of φ_N modulo a small number of primes $\approx 2^{128}$. Lifting these equations to characteristic zero with the LLL algorithm gives a candidate for F_N.

Since F_N is an irreducible polynomial and the image of φ_N is an irreducible variety, we verify the result in `MAGMA` by checking that F_N vanishes at the equations defining φ_N. These polynomials are available in the code accompanying this chapter, and their properties are summarised in Table 8.2.

Remark 8.3.5. As pointed out to us by Benjamin Smith, for a generic genus-2 curve $C : y^2 = (x - a_0) \dots (x - a_5)$ the polynomial $F_2(I_2(C), I_4(C), I_6(C), I_{10}(C))$ is (up to a scaling factor) equal to the square of the product of the determinants of the 15 Richelot kernels. This gives a connection to the classical work of Bolza [37, p. 51] where this is the invariant which Bolza calls R^2.

N	Weighted degree of F_N	Number of monomials in F_N	Average bitlength of the coefficients of F_N
2	30	34	~ 16.6
3	80	318	~ 64.3
4	180	2699	~ 197
5	480	43410	~ 617

Table 8.2: The number of monomials in the defining equation F_N for the image of $\mathcal{L}_N$ in $\mathbb{P}(1,2,3,5)$ and the total number of bytes required to (naively) store the coefficients of each F_N.

8.4 Efficient detection of (N,N)-splittings

In this section we present an algorithm to efficiently detect whether, at each step, the p.p. abelian surface $\mathcal{J}_C$ is (N,N)-isogenous over $\bar{\mathbb{F}}_p$ to a product of elliptic curves, without ever computing an (N,N)-isogeny. We are able to use resultants and gcd computations, rather than inefficient computations of (N,N)-isogenies, therefore avoiding all N^{th}-root calculations and the need to work in extension fields when the N-torsion is not fully $\mathbb{F}_{p^2}$-rational.

A natural starting point to perform this detection is to exploit the equations F_N for the image of the morphism $\mathcal{L}_N \to \mathbb{P}(1,2,3,5)$ (see Section 8.3.4). Indeed, if a genus-2 curve $C/\mathbb{F}_{p^2}$ is (N,N)-split, then $F_N(I_2(C), I_4(C), I_6(C), I_{10}(C)) = 0$. However, as demonstrated in Table 8.2, both the number of monomials in F_N and the bitlength of its coefficients grow rapidly with N. As a result, computing and storing F_N for $N > 5$ is challenging. Instead, we will use techniques in elimination theory to determine whether $[C]$ lies on the (Zariski closure of) the image of φ_N. Indeed, even for $N \leq 5$, evaluating the image at the Igusa–Clebsch invariants of C will not outperform this method.

Lemma 8.4.1. *Let $N \geq 2$ be an integer and $C/\Bbbk$ be a genus-2 curve defined over a field $\Bbbk$ of characteristic not dividing $2N$. Suppose that the Igusa–Clebsch invariants $I_2(C)$, $I_4(C)$, $I_6(C)$, and $I_{10}(C)$ are non-zero. Write $\alpha_1(C) = \frac{I_4(C)}{I_2(C)^2}$, $\alpha_2(C) = \frac{I_2(C)I_4(C)}{I_6(C)}$, and $\alpha_3(C) = \frac{I_4(C)I_6(C)}{I_{10}(C)}$. If there exist $r_0 \in \overline{\Bbbk} \cup \{\infty\}$ and $s_0 \in \overline{\Bbbk}$ satisfying*

$$\begin{cases} \alpha_1(C) = \frac{\mathcal{I}_4(r_0,s_0)}{\mathcal{I}_2(r_0,s_0)^2}, \\ \alpha_2(C) = \frac{\mathcal{I}_2(r_0,s_0)\mathcal{I}_4(r_0,s_0)}{\mathcal{I}_6(r_0,s_0)}, \\ \alpha_3(C) = \frac{\mathcal{I}_4(r_0,s_0)\mathcal{I}_6(r_0,s_0)}{\mathcal{I}_{10}(r_0,s_0)} \end{cases}$$

then $\mathcal{J}_C$ is optimally (N, N)-split over $\overline{\Bbbk}$. Here $\mathcal{I}_w(r, s)$ are as in Section 8.3.3.

Proof. The rational map $\psi\colon \mathbb{P}(1, 2, 3, 5) \dashrightarrow \mathbb{A}^3$ given by $[I_2 : I_4 : I_6 : I_{10}] \mapsto \left(\frac{I_4}{I_2^2}, \frac{I_2 I_4}{I_6}, \frac{I_4 I_6}{I_{10}}\right)$ is birational with inverse $(\alpha_1, \alpha_2, \alpha_3) \mapsto \left[1 : \alpha_1 : \frac{\alpha_1}{\alpha_2} : \frac{\alpha_1^2}{\alpha_2\alpha_3}\right]$. Moreover on the open subvariety (with respect to the Zariski topology) of $\mathbb{P}(1, 2, 3, 5)$ where I_2, I_4, I_6, and I_{10} are nonzero the map ψ restricts to an isomorphism onto its image. The claim follows from the discussion preceding the lemma. $\square$

Remark 8.4.2. It is common in the literature (e.g., [57, 190]) to choose the affine patch with coordinates the *absolute invariants* $\frac{6(I_2^2-16I_4)}{I_2^2}$, $\frac{-12(5I_2^3-176I_2I_4+384I_6)}{I_2^3}$, and $\frac{3888I_{10}}{I_2^5}$. Our choice is *ad hoc* and made to optimise the algorithms in Section 8.4.2. In particular, the choice in Lemma 8.4.1 yields polynomials $P_{i,j}$ in Lemma 8.4.5 of smaller degree. Choosing an affine patch of $\mathbb{P}(1, 2, 3, 5)$ so that the analogous polynomials to $P_{i,j}$ in Lemma 8.4.5 have minimal degree would likely lead to improved performance of our algorithm.

Remark 8.4.3. In the code accompanying this chapter we provide a function InvariantsFromWeierstrassPoints that, on input of the 6-tuple $\mathbf{a} = (a_0, \dots, a_5) \in (\mathbb{F}_{p^2})^6$ of Weierstrass points, computes the 3-tuple $\boldsymbol{\alpha}(C) = (\alpha_1(C), \alpha_2(C), \alpha_3(C)) \in (\mathbb{F}_{p^2})^3$ using a total of 291 multiplications and one (merged) inversion in $\mathbb{F}_p$. This is the first step of Algorithm 8.4.

Define polynomials $f_k(r, s) \in \mathbb{Z}[\alpha_1, \alpha_2, \alpha_3][r, s]$ by

$$\begin{aligned} f_1(r, s) &= \mathcal{I}_4(r, s) - \alpha_1\mathcal{I}_2(r, s)^2, \\ f_2(r, s) &= \mathcal{I}_2(r, s)\mathcal{I}_4(r, s) - \alpha_2\mathcal{I}_6(r, s), \\ f_3(r, s) &= \mathcal{I}_4(r, s)\mathcal{I}_6(r, s) - \alpha_3\mathcal{I}_{10}(r, s). \end{aligned}$$

The following proposition follows immediately from Lemma 8.4.1.

Proposition 8.4.4. *Suppose that $C/\Bbbk$ is a genus-2 curve with non-zero Igusa–Clebsch invariants. If there exist $r_0 \in \overline{\Bbbk} \cup \{\infty\}$ and $s_0 \in \overline{\Bbbk}$ such that for each $w \in \{2, 4, 6, 10\}$ we have $\mathcal{I}_w(r_0, s_0) \neq 0$ and $f_k(r_0, s_0) = 0$, then $\mathcal{J}_C$ is optimally (N, N)-split over $\overline{\Bbbk}$.*

In Section 8.4.2 we describe a method for determining whether, given a genus 2 curve $C/\overline{\mathbb{F}}_p$ with superspecial Jacobian, there exists a point $P \in \mathbb{A}^2(\overline{\mathbb{F}}_p)$ such that the polynomials $f_k(r, s)$

vanish at P. Moreover, we determine lower bounds on their costs in terms of $\mathbb{F}_p$-multiplications for each $N \in \{2, 3, \ldots, 11\}$.

8.4.1 The resultants of f_j and f_k

Fix an integer $2 \leq N \leq 11$. For each distinct pair $i, j \in \{1, 2, 3\}$, define polynomials[6]

$$R_{i,j}(s) := \mathrm{res}_r(f_i(r, s), f_j(r, s)) \in \mathbb{Z}[\alpha_1, \alpha_2, \alpha_3][s].$$

If $C/\Bbbk$ is a genus-2 curve then, since resultants are invariant under ring homomorphisms, by the elimination property of the resultant (see e.g., [110, §3.6 Lemma 1]) the specialisations $(R_{i,j})_{[C]}(s) \in \Bbbk[s]$, given by evaluating the coefficients of $R_{i,j}(s)$ at $\alpha_1(C)$, $\alpha_2(C)$, and $\alpha_3(C)$, vanish at the s-coordinate of any common solution to the specialised polynomials $(f_j)_{[C]}(r, s)$.

However, these resultants (generically) have factors which correspond to unwanted solutions (i.e., where one of the polynomials $\mathcal{I}_w$ vanishes). We make this more precise in the following lemma, which follows from a direct calculation in `MAGMA`.

Lemma 8.4.5. *Let $L = \mathbb{Q}(\alpha_1, \alpha_2, \alpha_3)$. When $i \neq j$, there exist polynomials $Q_{i,j} \in \mathbb{Z}[\alpha_1, \alpha_2, \alpha_3][s]$ dividing $R_{i,j}$ with the following property: for each pair $r_0, s_0 \in \bar{L}$ such that $f_k(r_0, s_0) = 0$ for $k = 1, 2, 3$ and $Q_{i,j}(s_0) = 0$, then $\mathcal{I}_w(r_0, s_0) = 0$ for some $w \in \{2, 4, 6, 10\}$.*

Moreover, the polynomials $P_{i,j} = \frac{R_{i,j}}{Q_{i,j}} \in \mathbb{Z}[\alpha_1, \alpha_2, \alpha_3][s]$ are coprime.

Applying [110, §3.6, Corollary 7] we have:

Proposition 8.4.6. *Let $C/\Bbbk$ be a genus-2 curve such that $I_w(C) \neq 0$ for each $w \in \{2, 4, 6, 10\}$. If there exist $r_0, s_0 \in \bar{\Bbbk}$ such that $(f_i)_{[C]}(r_0, s_0) = 0$ for each $i = 1, 2, 3$ then the degree of $\gcd((P_{1,2})_{[C]}, (P_{2,3})_{[C]})$ is at least 1.*

Conversely, if $s_0 \in \bar{\Bbbk}$ is a root of $\gcd((P_{1,2})_{[C]}, (P_{2,3})_{[C]})$ then there exist $r_0, r_0' \in \bar{\Bbbk} \cup \{\infty\}$ such that $(f_1)_{[C]}(r_0, s_0) = (f_2)_{[C]}(r_0, s_0) = 0$ and $(f_2)_{[C]}(r_0', s_0) = (f_3)_{[C]}(r_0', s_0) = 0$.

In the electronic data attached to this chapter we give the polynomials $P_{i,j} \in \mathbb{Z}[\alpha_1, \alpha_2, \alpha_3][s]$ for each pair $i \neq j$.

8.4.2 An algorithm to detect (N, N)-split Jacobians

We now present our algorithm to efficiently detect whether the Jacobian of a genus-2 curve $C/\mathbb{F}_{p^2}$ is (N, N)-split for some integer $2 \leq N \leq 11$. In Proposition 8.4.8, we then give an upper bound on the number of $\mathbb{F}_p$-multiplications required by the algorithm.

Precomputation step. We reduce the coefficients of the polynomials $P_{1,2}, P_{2,3} \in \mathbb{Z}[\alpha_1, \alpha_2, \alpha_3][s]$ from Lemma 8.4.5 modulo p to obtain polynomials $\widetilde{P}_{1,2}, \widetilde{P}_{2,3} \in \mathbb{F}_p[\alpha_1, \alpha_2, \alpha_3][s]$, which are stored.

Evaluation and gcd **step.** Our approach is summarised in Algorithm 8.3. To test a given genus-2 curve $C/\mathbb{F}_{p^2}$ with superspecial Jacobian, we specialise the coefficients of $\widetilde{P}_{1,2}, \widetilde{P}_{2,3}$ at $\boldsymbol{\alpha}(C) = (\alpha_1(C), \alpha_2(C), \alpha_3(C))$, by running the algorithm EvalCoeffs, to obtain the polynomials

[6]If necessary, we swap the roles of Kumar's r and s so that the polynomials $R_{i,j}$ are of lowest degree (as noted in the accompanying code). It would be interesting to find a birational transformation of $\mathbb{A}^2$ which minimises $\deg R_{i,j}$.

$(\widetilde{P}_{1,2})_{[C]}, (\widetilde{P}_{2,3})_{[C]} \in \mathbb{F}_{p^2}[s]$. The EvalCoeffs algorithm takes as input $\widetilde{P}_{i,j}$ and the invariants $\boldsymbol{\alpha}(C)$, and evaluates the coefficients of the polynomial at these invariants (see the proof of Proposition 8.4.8 for more details).

We then compute the gcd of $(\widetilde{P}_{1,2})_{[C]}$ and $(\widetilde{P}_{2,3})_{[C]}$ using the "inversion-free gcd" algorithm InvFreeGCD from Chapter 7, modified to output the gcd explicitly, rather than a boolean.

If this gcd has degree ≥ 1 then it has a root $s_0 \in \bar{\mathbb{F}}_p$ and (by Proposition 8.4.6) there exist $r_0, r_0' \in \bar{\mathbb{F}}_p \cup \{\infty\}$ such that $(f_1)_{[C]}(r_0, s_0) = (f_2)_{[C]}(r_0, s_0) = 0$ and $(f_2)_{[C]}(r_0', s_0) = (f_3)_{[C]}(r_0', s_0) = 0$. By Proposition 8.4.4 to verify that $\mathcal{J}_C$ is (N,N)-split it suffices to show that we may take $r_0 = r_0'$ such that $\mathcal{I}_w(r_0, s_0) \neq 0$ for each $w \in \{2,4,6,10\}$. We verify the first condition by computing the gcd of $(f_1)_{[C]}(r, s_0), (f_2)_{[C]}(r, s_0), (f_3)_{[C]}(r, s_0)$, and if it has degree ≥ 1 computing a root $r_0 \in \bar{\mathbb{F}}_p$. We verify the second condition by checking that $\mathcal{I}_w(r_0, s_0) \neq 0$ for each $w \in \{2,4,6,10\}$ – we abbreviate this to the function IgNonzero.

Remark 8.4.7. If $\mathcal{J}_C$ is optimally (N,N)-split then Algorithm 8.3 will return `true` with high probability. In this case $[C]$ is a $\mathbb{F}_{p^2}$-point on $\mathcal{L}_N$. Since $\varphi_N \colon \mathbb{A}^2 \dashrightarrow \mathcal{L}_N$ is birational (over $\mathbb{F}_p$) it is an isomorphism outside a closed $\mathbb{F}_p$-subvariety $X \subseteq \mathcal{L}_N$ of dimension 1. But from the Weil conjectures $\#\mathcal{L}_N(\mathbb{F}_{p^2}) = O(p^4)$ and $\#X(\mathbb{F}_{p^2}) = O(p^2)$. In particular, except in $O(1/p^2)$ of cases, there exist $r_0, s_0 \in \bar{\mathbb{F}}_p$ satisfying the conditions of Proposition 8.4.6.

Algorithm 8.3 IsSplit: detects whether a Jacobian is split

Input: A tuple $\boldsymbol{\alpha}(C) = (\alpha_1(C), \alpha_2(C), \alpha_3(C))$, the polynomials $\widetilde{P}_{1,2}, \widetilde{P}_{2,3} \in \mathbb{F}_p[\alpha_1, \alpha_2, \alpha_3][r]$, and an integer $2 \leq N \leq 11$.
Output: A boolean `bool` that is true if $\mathcal{J}_C$ is optimally (N,N)-split, and false otherwise.

```
1: $(\widetilde{P}_{1,2})_{[C]} \leftarrow$ EvalCoeffs$(\widetilde{P}_{1,2}, \boldsymbol{\alpha}(C))$
2: $(\widetilde{P}_{2,3})_{[C]} \leftarrow$ EvalCoeffs$(\widetilde{P}_{2,3}, \boldsymbol{\alpha}(C))$
3: $g \leftarrow$ InvFreeGCD$((\widetilde{P}_{1,2})_{[C]}, (\widetilde{P}_{2,3})_{[C]})$
4: if $\deg g \geq 1$ then
5:     $s_0 \leftarrow$ ComputeRoot$(g)$
6:     $(\widetilde{f}_1)_{[C]} \leftarrow$ EvalCoeffs$(\widetilde{f}_1, \boldsymbol{\alpha}(C))$
7:     $(\widetilde{f}_2)_{[C]} \leftarrow$ EvalCoeffs$(\widetilde{f}_2, \boldsymbol{\alpha}(C))$
8:     $(\widetilde{f}_3)_{[C]} \leftarrow$ EvalCoeffs$(\widetilde{f}_3, \boldsymbol{\alpha}(C))$
9:     $h \leftarrow$ InvFreeGCD(InvFreeGCD$(\widetilde{f}_1)_{[C]}(r, s_0), (\widetilde{f}_2)_{[C]}(r, s_0)), (\widetilde{f}_3)_{[C]}(r, s_0))$
10:    if $\deg h \geq 1$ then
11:        $r_0 \leftarrow$ ComputeRoot$(h)$
12:        bool $\leftarrow$ IgNonzero$(r_0, s_0)$
13:        if bool == true then
14:            return true
15:        end if
16:    end if
17: end if
18: return false
```

The cost of Algorithm 8.3. We now determine an upper bound for the number of $\mathbb{F}_p$-multiplications required for the online part of this method (i.e., ignoring the cost of precomputation). For ease of analysis, we assume that Karatsuba multiplication is used in $\mathbb{F}_{p^2}$, and hence we cost one $\mathbb{F}_{p^2}$-multiplication as three $\mathbb{F}_p$-multiplications. We remark that employing faster $\mathbb{F}_{p^2}$ arithmetic will only decrease the concrete cost of the algorithms we present.

Proposition 8.4.8. *Let $N \in \{2, \dots, 11\}$ be an integer, and let $\mathsf{mons}(N)$ be the set of monomials in $\alpha_1, \alpha_2, \alpha_3$ appearing in the coefficients of $\widetilde{P}_{1,2}$ and $\widetilde{P}_{2,3}$ (which lie in $\mathbb{F}_p[\alpha_1, \alpha_2, \alpha_3]$). For each $i = 1, 2, 3$, let*

$$d_i(N) = \max(\{\text{degree of } \alpha_i \text{ in } m \mid m \in \mathsf{mons}(N)\}).$$

The cost of steps 1–3 in Algorithm 8.3 (with input N) is at most

$$3(d_1(N) + d_2(N) + d_3(N)) + 6m(N) + 2M(N) + \frac{3}{2}(d_P(N) + 2)(d_P(N) + 3) - 27$$

$\mathbb{F}_p$-multiplications, where $d_P(N) = \deg \widetilde{P}_{1,2} + \deg \widetilde{P}_{2,3}$, $m(N) = \#\mathsf{mons}(N)$, and $M(N)$ is the number of monomials in $\alpha_1, \alpha_2, \alpha_3$ appearing in the coefficients of $\widetilde{P}_{1,2}$ and $\widetilde{P}_{2,3}$ counting repetitions.

Proof. We first evaluate the coefficients of $\widetilde{P}_{1,2}, \widetilde{P}_{2,3} \in \mathbb{F}_{p^2}[\alpha_1, \alpha_2, \alpha_3][s]$ at the normalised invariants $\alpha_1(C), \alpha_2(C), \alpha_3(C) \in \mathbb{F}_{p^2}$ using our evaluation algorithm $\mathsf{EvalCoeffs}$ on each polynomial. This runs as follows. We first compute powers $\alpha_1(C)^2, \dots, \alpha_1(C)^{d_1(N)}$ where $d_1(N)$ is the maximum degree of α_1 appearing in $\mathsf{mons}(N)$ (as defined in the statement of the proposition). Similarly we compute powers of $\alpha_2(C)$ and $\alpha_3(C)$ up to $d_2(N)$ and $d_3(N)$ respectively. This step is performed using $d_1(N) + d_2(N) + d_3(N) - 3$ multiplications in $\mathbb{F}_{p^2}$.

From these powers, we obtain the monomials appearing in the coefficients of $(\widetilde{P}_{1,2})_{[C]}(s)$ and $(\widetilde{P}_{2,3})_{[C]}(s)$ in at most $2m(N)$ $\mathbb{F}_{p^2}$-multiplications, where $m(N) = \#\mathsf{mons}(N)$. We then require $2M(N)$ $\mathbb{F}_p$-multiplications (and $2M(N)$ additions) to construct the coefficients of $(\widetilde{P}_{1,2})_{[C]}$ and $(\widetilde{P}_{2,3})_{[C]}$.

The final step computes the gcd of $(\widetilde{P}_{1,2})_{[C]}$ and $(\widetilde{P}_{2,3})_{[C]}$ using $\mathsf{InvFreeGCD}$. This requires $\frac{3}{2}(d_P(N) + 2)(d_P(N) + 3) - 18$ $\mathbb{F}_p$-multiplications by Proposition 7.3.4. □

The cost from Proposition 8.4.8 depends only on N. Therefore, for each $2 \le N \le 11$, we can determine the total number of $\mathbb{F}_p$-multiplications required for the detection per node revealed in $\mathcal{S}_2(\overline{\mathbb{F}}_p)$ for any prime p. We give these costs in Table 8.3.

Noting that, when $N \neq N'$ we may have non-empty intersection $\mathsf{mons}(N) \cap \mathsf{mons}(N')$, our implementation of Algorithm 8.4 stores all evaluated monomials to avoid repeated computations. In particular, the upper bound in Proposition 8.4.8 is often not sharp.

Remark 8.4.9. We note that, in practice, when our algorithm enters the if loop on Line 4 in Algorithm 8.3, we have yet to encounter a case where Steps 5–14 fail to return `true`. In these cases the bound in Proposition 8.4.8 yields a bound on the cost of Algorithm 8.3. It is however possible to construct examples of polynomials for which they would be necessary – e.g., $f_1(r, s) = r - 1$, $f_2(r, s) = s - r(r + 1)(r - 1)$, and $f_3(r, s) = r + 1$. It would be interesting to put this observation on rigorous footing by showing that with overwhelming probability the roots r_0 and r_0' guaranteed by Proposition 8.4.6 are equal.

Remark 8.4.10. We note that Remark 7.4.3 also applies in this context: replacing $\mathsf{InvFreeGCD}$ with an adaptation of the Stehlé–Zimmermann algorithm could lead to improved concrete com-

N	$d_1(N)$	$d_2(N)$	$d_3(N)$	$m(N)$	$M(N)$	$d_P(N)$	Total $\#\mathbb{F}_p$ mults.	Total $\#\mathbb{F}_p$ mults. per node revealed
2	1	2	1	6	23	6	175	12.5
3	2	3	2	11	97	16	767	19.2
4	6	8	6	78	1136	35	4882	46.9
5	6	10	6	64	2500	92	18818	120.6
6	7	11	7	91	4118	114	29188	52.1
7	10	14	10	190	24779	294	182641	456.6
8	16	24	16	433	73454	340	325606	395.2
9	12	16	12	271	69648	540	582474	539.3
10	24	32	24	1005	260178	606	1082007	495.4
11	28	38	28	1345	669432	1120	3237198	2211.2

Table 8.3: Values of $d_1(N), d_2(N), d_3(N)$, $m(N)$, $M(N)$, and $d_P(N)$ for $N \in \{2, \dots, 11\}$. The final columns respectively list the number of $\mathbb{F}_p$-multiplications in Proposition 8.4.8 and the ratio of multiplications to the number of (N, N)-isogenous p.p. abelian surfaces.

plexity of IsSplit. In fact, as the degrees of the polynomials we input into InvFreeGCD are larger (compared with those in NeighbourInFp from Chapter 7), the efficiency gain could be even greater.

Alternative approach for $N = 10$ and 11. When $N = 10, 11$, several megabytes are required to store the coefficients of the polynomials $\widetilde{P}_{i,j}$. Rather than computing the resultants $R_{1,2}$ and $R_{2,3}$ and dividing out by the generic factors described in Lemma 8.4.5 to obtain $P_{1,2}$, $P_{2,3}$ as a precomputation, the approach we pursued was to instead perform these two steps during the online phase. Even still, our experiments (which were reinforced by the cost analysis above) revealed that performing the detection for $N = 10, 11$ is suboptimal in our application to SplitSearcher (shown in Algorithm 8.4) and slows the overall search down, even when the characteristic of the field is very large. Thus, we leave the further optimisation of these computations as future work.

8.5 The full algorithm

In Section 8.2 we discussed our optimised implementation of the product-finding attack [107] that works entirely in the Richelot isogeny graph $\mathcal{X}_2(\bar{\mathbb{F}}_p, 2)$. In this section, we present SplitSearcher, which leverages our efficient detection of nodes that are (N, N)-isogenous to nodes in the special subset $X_2 = \mathcal{E}_2(\bar{\mathbb{F}}_p)$ by detecting (N, N)-splittings using Section 8.4.2. This improves on the concrete complexity of product-finding when solving the dimension 2 isogeny problem.

8.5.1 SplitSearcher

Each time we take a step using a Richelot isogeny, we will use the methods from the previous section to detect whether the current node is (N, N)-isogenous to a node in X_2 (i.e., a product of elliptic curves), for some subset of integers in $2 \leq N \leq 11$. Using the algorithm from Section 8.4.2

makes this check much more efficient than, say, walking in $\mathcal{X}_2(\bar{\mathbb{F}}_p, N)$; each node we step to would require computing an (N, N)-isogeny which, at minimum, requires three N-th roots in $\mathbb{F}_{p^2}$ [69].

Each time we take a step and arrive at a new abelian surface, A, we are in one of two cases: either $A \in X_2$, i.e., is isomorphic to a product of elliptic curves, in which case the algorithm terminates, or A is isomorphic to the Jacobian of a genus-2 curve $C/\mathbb{F}_{p^2}$. In the latter case, SplitSearcher calls Algorithm 8.3 to detect whether A is (N, N)-split for certain $2 \leq N \leq 11$. The set of N's for which this detection is performed is chosen to minimise the number of $\mathbb{F}_p$-multiplications per node revealed (either by stepping on them in $\mathcal{X}_2(\bar{\mathbb{F}}_p, 2)$ or inspecting them via our splitting detection) in $\mathcal{S}_2(\bar{\mathbb{F}}_p)$. Since it only depends on the prime p, determining this optimal list of N's is performed during precomputation.

If Algorithm 8.3 determines that A is (N, N)-split, the elliptic curves E_1 and E_2 can be recovered by applying [224, Algorithm 4] or [94] to compute all (N, N)-isogenies from A. Alternatively, E_1 and E_2 may be recovered from Kumar's equations [206] by solving for r_0 and s_0 in Proposition 8.4.4. As both of these costs are negligible and do not affect the cost of finding such a splitting, we may view this as a post-computation step and exclude it from our multiplication counts.

A precise formulation of the full algorithm for finding paths to the special subset X_2 is given by Algorithm 8.4. Along with the target abelian surface $A \in \mathcal{S}_2(\bar{\mathbb{F}}_p)$, the auxiliary inputs into the algorithm are the polynomials $\widetilde{P}_{1,2}, \widetilde{P}_{2,3} \in \mathbb{F}_p[\alpha_1, \alpha_2, \alpha_3][r]$ (see Lemma 8.4.5), and the optimal set $\mathcal{N} \subseteq \{2, \ldots, 11\}$ (see Section 8.5.2). The hash function on Line 4 is assumed to be of the form $\mathsf{Hash}\colon \{0,1\}^* \to \{0,1\}^{3\ell}$, where ℓ is a positive integer, since we use three bits of entropy each time we call the Richelot isogeny (i.e., Step algorithm) in Line 16. We choose ℓ to be large enough that we can expect to find an elliptic product in walks of ℓ steps, but not *too* large, since storing walks of up to ℓ steps requires more storage on average. Once the 3ℓ bits of entropy have been consumed, the hash function is called again and the walk is restarted from $\boldsymbol{a}_{\text{start}}$ (more on this in Remark 8.5.1). The output returned by Algorithm 8.4 is of the form $(\texttt{path}, N)$, where path is a sequence of $3k$ bits (with $k \leq \ell$) and N is an integer: the $3k$ bits define a sequence of k Richelot isogenies and the integer N specifies the final (N, N)-isogeny whose image is in $\mathcal{E}_2(\bar{\mathbb{F}}_p)$.

Remark 8.5.1. In a real-world attack, we would expect to return to Line 4 of Algorithm 8.4 an exponential number of times before the algorithm terminates. Thus, there are a number of ways one could recycle information computed in the early stages of each walk to avoid recomputing them over and over again. One solution that is easy to implement in view of Algorithm 8.4 would be to store a hash table whose entries each correspond to the (hash of the) Igusa–Clebsch invariants of any node that is visited and checked for (N, N)-splittings. Upon returning to a given node and finding a collision in the hash table, the walk could simply avoid the tests for (N, N)-splittings between Lines 8 and 11. Another approach would be to build a table of the six-tuples $\boldsymbol{a}$ that are computed after the first t Richelot steps have been taken, alongside the label of the $3t$-bit string that took us there. Each time we return back to Line 4 and iterate the hash function, we simply check to see if the first $3t$ bits are already in the table and, if so, we can skip straight to $\boldsymbol{a}$.

Finally, as is mentioned by Costello and Smith [107], parallelising the search for product curves is trivial. For P processors, we would simply compute P unique short walks from our target surface $A \in \mathcal{S}_2(\bar{\mathbb{F}}_p)$ and send each of the corresponding image surfaces $A_1, \ldots, A_P$ to a unique processor as its assigned input surface.

Algorithm 8.4 SplitSearcher: finding paths to elliptic curve products

Input: $\boldsymbol{a}_{\text{start}} = (a_0, \ldots, a_5) \in (\mathbb{F}_{p^2})^6$ defining a genus-2 curve $C/\mathbb{F}_{p^2}$ with superspecial Jacobian, and a set $\mathcal{N} \subseteq \{2, 3, \ldots, 11\}$.
Output: A pair (`path`, N) where `path` is a path $\varphi\colon \mathcal{J}_C \to \mathcal{J}_{C'}$ in $\mathcal{X}_2(\overline{\mathbb{F}}_p, 2)$ and N is an integer such that $\mathcal{J}_{C'}$ is optimally (N,N)-split.

```
 1: split ← false
 2: $H \leftarrow \mathsf{StartingSeed}(\boldsymbol{a}_{\text{start}})$                                    Section 8.2.1
 3: while not split do
 4:   $(H, i, \texttt{path}, \boldsymbol{a}) \leftarrow (\mathsf{Hash}(H), 0, \{\emptyset\}, \boldsymbol{a}_{\text{start}})$
 5:   while $i < \ell$ and not split do
 6:     if $\mathcal{N} \neq \emptyset$ then
 7:       $\boldsymbol{\alpha}(C) \leftarrow \mathsf{InvariantsFromWeierstrassPoints}(\boldsymbol{a})$          Remark 8.4.3
 8:       for $N \in \mathcal{N}$ do
 9:         split ← $\mathsf{IsSplit}(\boldsymbol{\alpha}(C), \widetilde{P}_{1,2}, \widetilde{P}_{2,3}, N)$                   Algorithm 8.3
10:         if split then
11:           return (path, $N$)
12:         end if
13:       end for
14:     end if
15:     bits ← $H[3i] \,\|\, H[3i+1] \,\|\, H[3i+2]$
16:     $\boldsymbol{a}$, split ← $\mathsf{Step}(\boldsymbol{a}, \texttt{bits})$                                     Section 8.2.2
17:     path ← path || bits
18:     $i \leftarrow i + 1$
19:   end while
20: end while
21: return (path, 2)
```

8.5.2 Determining the optimal set $\mathcal{N}$

Recall that, when we step to a new p.p. abelian surface $A \in \mathcal{S}_2(\overline{\mathbb{F}}_p)$, we want to determine if it is (N,N)-split for a set $\mathcal{N} \subseteq \{2, \ldots, 11\}$ of N. We wish to determine the optimal subset $\mathcal{N} \subseteq \{2, \ldots, 11\}$, i.e., the subset which minimises the number of $\mathbb{F}_p$-multiplications per node revealed in the graph. The first step towards determining this 'multiplications-per-node' ratio is to count the number of nodes in $\mathcal{S}_2(\overline{\mathbb{F}}_p)$ that are inspected inside the `for` loop of Algorithm 8.4 with a finite set of integers $\mathcal{N} \subseteq \mathbb{Z}_{\geq 2}$. A first attempt would be to simply count the number of neighbours a node $A \in \mathcal{S}_2(\overline{\mathbb{F}}_p)$ has in $\mathcal{X}_2(\overline{\mathbb{F}}_p, N)$, i.e., D_N given by Equation (2.8) in Section 2.8.2. However, this is an overcount as we now detail.

Suppose we take a non-backtracking walk

$$A_0 \xrightarrow{\phi_0} A_1 \xrightarrow{\phi_1} \cdots \xrightarrow{\phi_{n-1}} A_n \xrightarrow{\phi_n} \cdots \tag{8.1}$$

in $\mathcal{X}_2(\overline{\mathbb{F}}_p, 2)$, and we inspect (N,N)-splittings for $N \in \mathcal{N}$. If $0 \leq m \leq n$ are integers, let $\phi_{m,n}$ denote the $(2^{n-m}, 2^{n-m})$-isogeny $\phi_{m-1} \circ \cdots \circ \phi_n$ and let $\phi_{n,m}$ denote $\widehat{\phi_{m,n}}$.

Firstly, if both N and $2^k N$ are contained in $\mathcal{N}$ (for $k \geq 1$), then any abelian surfaces (N,N)-isogenous to A_n are automatically $(2^k N, 2^k N)$-isogenous to A_{n+k}. Therefore, we restrict to only considering subsets $\mathcal{N}$ which do not contain pairs of integers $M \neq N$ with $N = 2^k M$.

This restriction is not sufficient to stop double-counting nodes. Indeed, suppose $N \in \mathcal{N}$ with $N = 2M$. Then any abelian surface (N, N)-isogenous to A_n will be (M, M)-isogenous to A_{n+1}. In particular, such an abelian surface will also be (N, N)-isogenous to A_{n+2}. To rule out such scenarios, we introduce the following restriction on our paths.

Definition 8.5.2. Let $\mathcal{N} \subseteq \mathbb{Z}_{\geq 2}$ be a finite set of integers and let $\mathcal{P}$ be a walk of $(2, 2)$-isogenies in $\mathcal{X}_2(\overline{\mathbb{F}}_p, 2)$ as in Equation (8.1).

Let $M, N \in \mathcal{N}$ and suppose that there exist integers $m, n \geq 0$ and (M, M)- and (N, N)-isogenies $\psi_M \colon A_m \to B$ and $\psi_N \colon A_n \to B$. We say that $\mathcal{P}$ *resists collisions for* M, N if there exists an integer $i \geq 0$ and an isogeny $\Psi \colon A_i \to B$ such that $\psi_M = \Psi \circ \phi_{m,i}$ and $\psi_N = \Psi \circ \phi_{n,i}$.

We say that $\mathcal{P}$ *resists collisions for* $\mathcal{N}$ if it resists collisions for every pair $M, N \in \mathcal{N}$.

We are now able to state precisely the number of nodes checked between Lines 8–11 of Algorithm 8.4, assuming our paths resist collisions for the set $\mathcal{N}$.

Lemma 8.5.3. *Let $\mathcal{N} \subseteq \mathbb{Z}_{\geq 2}$ be a finite set of integers such that if $\mathcal{N}$ is non-empty, then there do not exist distinct $M, N \in \mathcal{N}$ with $N = 2^k M$ for any $k \geq 1$.*

Let $\mathcal{P}$ be a path in $\mathcal{X}_2(\overline{\mathbb{F}}_p, 2)$ which resists collisions for $\mathcal{N}$. The number of nodes inspected per step by running Algorithm 8.4 in $\mathcal{P}$ is at least

$$\mathrm{nodes}_{\mathcal{N}} := \begin{cases} \sum_{N \in \mathcal{N}} D'_{N,2} & \text{if } \mathcal{N} \text{ contains a power of } 2, \\ \sum_{N \in \mathcal{N}} D'_{N,2} + 1 & \text{otherwise} \end{cases}$$

where

$$D'_{N,2} = D_{N,2} - \sum_{\substack{1 \leq k \\ 2^k \mid N}} D_{N/2^k,2}$$

and $D_{N,2}$ is the number of neighbours of a node in $\mathcal{X}_2(\overline{\mathbb{F}}_p, N)$, given in Equation (2.8). Equality holds for steps taken after $\max_{N \in \mathcal{N}}(2\log(N))$ steps.

Remark 8.5.4. It is important to note that the assumption that $\mathcal{P}$ resists collisions for $\mathcal{N}$ is mild in practice. Indeed, when $\mathcal{N}$ contains only odd integers the assumption simplifies to requiring that, in a walk in the $(2, 2)$-isogeny graph, any abelian surface (N, N)-isogenous to A_n is not (M, M)-isogenous to A_m for some m. The set $\mathcal{N}$ will consist only of integers ≤ 11 and our walks have length $O(\log(p))$. A collision of this sort therefore implies that A_n has an endomorphism of degree $O(\log(p))$. Heuristically there should be very few such abelian surfaces. Indeed in the dimension 1 case, by Proposition B.3 in the unpublished appendix to [222], the proportion of supersingular elliptic curves with an endomorphism of degree at most $O(\log(p))$ is $O(\log(p)^{3/2}/p)$.

Proof. Suppose we have taken the following walk in $\mathcal{X}_2(\overline{\mathbb{F}}_p, 2)$

$$A_0 \to A_1 \to \cdots \to A_n \to A_{n+1} \to \cdots,$$

applying Algorithm 8.4.

First note that if $\mathcal{N}$ contains a power of 2, then each successive p.p. abelian surface A_i is known not to be a product of elliptic curves. By hypothesis, there do not exist distinct $M, N \in \mathcal{N}$ with

$N = 2^k M$ for any $k \geq 1$. Therefore, since $\mathcal{P}$ resists collisions for $\mathcal{N}$, for each *distinct* $M, N \in \mathcal{N}$ the p.p. abelian surfaces (M, M)-isogenous to A_m are not (N, N)-isogenous to A_n for all $m, n \geq 0$. In particular, it suffices to show that the number of p.p. abelian surfaces (N, N)-isogenous to A_i, but not (N, N)-isogenous to A_j for each $j < i$, is equal to $D'_{N,2}$.

The claim follows immediately when N is odd, since the walk takes place in $\mathcal{X}_2(\overline{\mathbb{F}}_p, 2)$. If N is even, write $N = 2^\ell M$ where $\ell \geq 1$ and M is odd. In this case, for each $1 \leq k \leq \ell$, any p.p. abelian surface $(2^{\ell-k}M, 2^{\ell-k}M)$-isogenous to A_{n-k} is (N, N)-isogenous to both A_{n-2k} *and* A_n. Therefore, $D_{N/2^k,2}$ surfaces (N, N)-isogenous to A_n are (N, N)-isogenous to A_{n-2k}.

The claim follows by summing over $1 \leq k \leq \ell$. Note that equality holds if $n - 2k \geq 0$ for each $1 \leq k \leq \ell$, i.e., we have taken at least 2ℓ steps. □

We use the lemma above to determine, for each prime p, an *optimal* set $\mathcal{N}$ for which we perform the detection of (N, N)-splittings during Algorithm 8.4.

Let c_{step} be the number of $\mathbb{F}_p$-multiplications required to take a step in $\mathcal{X}_2(\overline{\mathbb{F}}_p, 2)$ using Algorithm 8.2, and let c_{ig} be the number of $\mathbb{F}_p$-multiplications required to compute $\boldsymbol{\alpha}(C)$ using InvariantsFromWeierstrassPoints (see Remark 8.4.3). Finally, letting $c_{\text{split}}(N)$ be the total number of $\mathbb{F}_p$-multiplications required by Algorithm 8.3 (see Proposition 8.4.8 and Remark 8.4.9), we obtain the following lemma.

Lemma 8.5.5. *For a subset $\mathcal{N} \subseteq \{2, 3, \dots, 11\}$, the number of $\mathbb{F}_p$-multiplications required to run Steps 7-18 of Algorithm 8.4 is at most*

$$\text{cost}_{\mathcal{N}} := \begin{cases} c_{\text{step}} + c_{\text{ig}} + \sum_{N \in \mathcal{N}} c_{\text{split}}(N) & \text{if } \mathcal{N} \neq \emptyset, \\ c_{\text{step}} & \text{otherwise.} \end{cases}$$

Proof. Given input defining a genus-2 curve $C/\mathbb{F}_{p^2}$ if $\mathcal{N} = \emptyset$ then Steps 7-18 of Algorithm 8.4 require a single call to Step($\boldsymbol{a}$, bits), taking c_{step} $\mathbb{F}_p$-multiplications.

Otherwise, Step 7 calls InvariantsFromWeierstrassPoints taking c_{ig} multiplications in $\mathbb{F}_p$. For each $N \in \mathcal{N}$, the contents of the for-loop (i.e., Steps 8-11) require $c_{\text{split}}(N)$ multiplications in $\mathbb{F}_p$. Finally, Steps 15-18 call Step($\boldsymbol{a}$, bits), again requiring c_{step} $\mathbb{F}_p$-multiplications. □

We consider subsets of $\{2, \dots, 11\}$ satisfying the hypotheses of Lemma 8.5.3. As a precomputation, amongst these subsets we determine the optimal set $\mathcal{N}$ for Algorithm 8.4 by choosing $\mathcal{N}$ to minimise the number of $\mathbb{F}_p$-multiplications per node revealed (either visited by the Richelot walk or revealed by IsSplit). That is, we choose the $\mathcal{N}$ that minimises the ratio $\frac{\text{cost}_{\mathcal{N}}}{\text{nodes}_{\mathcal{N}}}$.

8.5.3 A bound on the cost of the SplitSearcher algorithm

We now discuss a heuristic upper bound for the concrete cost of finding a splitting of a genus-2 Jacobian using the SplitSearcher algorithm combined with an optimised walk in $\mathcal{X}_2(\overline{\mathbb{F}}_p, 2)$.

First recall that our function InvariantsFromWeierstrassPoints terminates with 291 $\mathbb{F}_p$-multiplications and 1 $\mathbb{F}_p$ inversion. Bounding this inversion by $2\log(p)$ $\mathbb{F}_p$-multiplications (i.e., by the worst case where the binary expansion of the exponent consists only of 1's), we have $c_{\text{ig}} \leq 291 + 2\log(p)$.

We now assume that the cost of IsSplit is bounded by the cost of its first 3 steps (see Proposition 8.4.8 and Table 8.3 bounds depending only on N, and Remark 8.4.9 for a justification).

Finally, RIsog requires 63 $\mathbb{F}_p$-multiplications and 3 calls to InvSqrt which costs at most $22+4\log(p)$ $\mathbb{F}_p$-multiplications (with the $\log(p)$ terms arising from 2 exponentiations). In particular, RIsog costs at most $129 + 12\log(p)$ $\mathbb{F}_p$-multiplications.

For primes of at least 150 bits, the set $\mathcal{N} = \{4, 6\}$ is the optimal set discussed in Section 8.5.2, and we obtain an upper bound of

$$\frac{14\log(p) + 34490}{664} \tag{8.2}$$

$\mathbb{F}_p$-multiplications per node revealed (assuming the heuristics from Remark 8.4.7 and Remark 8.5.4). If we assume that the proportion of product nodes (amongst nodes inspected by Algorithm 8.4) is equal to $5/p$[7] we would expect that Algorithm 8.4 requires

$$\left(\frac{14\log(p) + 34490}{5 \cdot 664}\right) p + O(\log(p)) \tag{8.3}$$

$\mathbb{F}_p$-multiplications before encountering a product node.

8.6 Experimental results

We conducted experiments over both small and large primes, and the results are reported in Table 8.4 and Table 8.5, respectively.

The small prime experiments were conducted so that we could run multiple instances of the full $\widetilde{O}(p)$ search for product curves to completion. The four Mersenne primes of the form $p = 2^m - 1$ with $m \in \{13, 17, 19, 31\}$ were chosen as the field characteristics, and instances of the product-finding problem were generated by taking a chain of 40 randomised Richelot isogenies away from the superspecial abelian surface corresponding to $C/\mathbb{F}_p \colon y^2 = x^5 + x$. For the three smaller primes, 256 instances were generated, while for $p = 2^{31} - 1$, we generated 10 such instances; each instance is specified by a 6-tuple of Weierstrass points (see Section 8.2.1). We remark that the shapes of the primes chosen in both tables is of little consequence: we merely made consistent choices of the prime shape so that the same form of superspecial starting surface could be used throughout the experiments.

All the instances were solved once using the original walk in $\mathcal{X}_2(\bar{\mathbb{F}}_p, 2)$ described in Section 8.2, and again using our improved SplitSearcher algorithm described in Section 8.5. For all four of these primes, the set $\mathcal{N} = \{2, 3\}$ was optimal for use in SplitSearcher. In Table 8.4 we report the average number of steps taken in $\mathcal{X}_2(\bar{\mathbb{F}}_p, 2)$ for both algorithms, as well as the average number of $\mathbb{F}_p$-multiplications required to solve the problem. In the case of SplitSearcher, we additionally report the average number of nodes searched. This includes both the nodes that were walked on and those that were inspected using our (N, N)-splitting detection. As we might expect, this is always relatively close to the number of steps taken in the Richelot-only walk.

Remark 8.6.1. Throughout this section, we assume that the number of nodes revealed by SplitSearcher after s steps is equal to $s \cdot \text{nodes}_{\mathcal{N}}$. Indeed, as discussed in Remark 8.4.7 and Re-

[7]This is the expected proportion of product nodes in a random walk in $\mathcal{X}_2(\bar{\mathbb{F}}_p, 2)$, see [155, §6.2]. However, preliminary experiments (see Table 8.4) indicate that in our walk (taking only *good extensions*) the proportion may be closer to $1/p$.

mark 8.5.4 an overcount should occur with very low probability. In particular, after $O(p)$ steps we would expect to overcount at most $o(p)$ nodes. This heuristic is also supported by the experiments reported in Table 8.4.

		Walks in $\mathcal{X}_2(\bar{\mathbb{F}}_p, 2)$ without additional searching [107] (optimised in Section 8.2)		Walks in $\mathcal{X}_2(\bar{\mathbb{F}}_p, 2)$ w. SplitSearcher in $\mathcal{X}_2(\bar{\mathbb{F}}_p, N)$ **This work**			
Prime p	No. inst. solved	Av. steps taken	Av. $\mathbb{F}_p$ mults.	Av. steps taken	Av. nodes covered	Av. $\mathbb{F}_p$ mults.	**Improv. factor**
$2^{13}-1$	256	6531	1839209	122	6536	188015	**9.8x**
$2^{17}-1$	256	101812	33538079	2154	116305	3474579	**9.7x**
$2^{19}-1$	256	475300	168095438	8593	464008	14104408	**11.9x**
$2^{31}-1$	10	238694656	118336348672	4856252	262237639	8787389743	**13.4x**

Table 8.4: Solving the product-finding problem using Richelot isogeny walks in $\mathcal{X}_2(\bar{\mathbb{F}}_p, 2)$ only (left) vs. using Richelot isogeny walks in $\mathcal{X}_2(\bar{\mathbb{F}}_p, 2)$ together with SplitSearcher in $\mathcal{X}_2(\bar{\mathbb{F}}_p, N)$ (right).

For cryptographically sized primes, we are unable to solve the product-finding problem, which is why Table 8.5 instead reports the number of nodes that were searched when the number of $\mathbb{F}_p$-multiplications was bounded at 10^8. The main trend to highlight (in both tables) is that the speed-up is increasing steadily as the prime grows in size: the number of $\mathbb{F}_p$-multiplications required for a single Richelot isogeny is proportional to the bitlength of p (due to the square root computations), while the number of $\mathbb{F}_p$-multiplications required to inspect the (N, N)-isogenous neighbours (after computing the Igusa–Clebsch invariants) remains fixed as p grows. This is also predicted by Equation (8.2), where the coefficient of the dominating $\log(p)$ term is $14/664$ versus 12.

Interestingly, as shown in Table 8.5 the set $\mathcal{N} = \{2, 3\}$ is optimal for the 50- and 100-bit primes, the set $\mathcal{N} = \{3, 4\}$ is optimal for the 150-bit prime, while the set $\mathcal{N} = \{4, 6\}$ takes over and reigns supreme for all other reported bitlengths. Our implementation can be used to obtain the same data for any other prime of interest, and the number of $\mathbb{F}_p$-multiplications used per node can be combined with the (average) number of nodes one expects to search through in order to get a very precise estimate on the concrete classical security of the superspecial isogeny problem.

8.6.1 Possible improvements

There have been a number of choices made throughout this chapter which open up possible avenues for improvement. We conclude by giving a non-exhaustive list of such improvements.

1. The parametrisation of $\mathcal{L}_N$ given by Kumar [206] may be altered through composition with a birational transformation of $\mathbb{A}^2$. There may be better choices of parametrisations for our purposes, i.e., ones which minimise the degree of $P_{i,j}$. Furthermore, as detailed in Remark 8.4.2, there are many ways to normalise the Igusa–Clebsch invariants, though it is unclear to us which normalisations minimise the degrees that arise in the resultant computations.

		Walks in $\mathcal{X}_2(\bar{\mathbb{F}}_p, 2)$ without additional searching [107] (optimised in Section 8.2)		Walks in $\mathcal{X}_2(\bar{\mathbb{F}}_p, 2)$ with SplitSearcher in $\mathcal{X}_2(\bar{\mathbb{F}}_p, N)$ **This work**			
Prime p	$\log(p)$	nodes per 10^8 mults.	$\mathbb{F}_p$-mults. per node	Set $N \in \{\cdots\}$	nodes per 10^8 mults.	$\mathbb{F}_p$-mults. per node	**Improv. factor**
$2^{11} \cdot 3^{24} - 1$	50	172712	579	$\{2, 3\}$	2830951	35	**16.5x**
$2^{44} \cdot 3^{35} - 1$	100	85034	1176		2076517	48	**24.5x**
$2^{27} \cdot 3^{77} - 1$	150	63492	1575	$\{3, 4\}$	1858912	54	**29.2x**
$2^{144} \cdot 3^{35} - 1$	200	42088	2376		1802816	55	**43.2x**
$2^{181} \cdot 3^{43} - 1$	250	34083	2934		1771608	56	**52.4x**
$5 \cdot 2^{193} \cdot 3^{66} - 1$	300	29317	3411		1745712	57	**59.8x**
$2^{201} \cdot 3^{94} - 1$	350	25581	3909		1719152	58	**67.4x**
$2^{231} \cdot 3^{106} - 1$	400	22753	4395		1694584	59	**74.5x**
$2^{204} \cdot 3^{155} - 1$	450	20729	4824		1672672	60	**80.4x**
$2^{113} \cdot 3^{244} - 1$	500	20239	4941		1667360	60	**82.4x**
$2^{293} \cdot 3^{162} - 1$	550	16835	5940		1619552	62	**95.8x**
$5 \cdot 2^{299} \cdot 3^{188} - 1$	600	15679	6378	$\{4, 6\}$	1599632	63	**101.2x**
$2^{404} \cdot 3^{155} - 1$	650	13848	7221		1562448	64	**112.8x**
$2^{83} \cdot 3^{389} - 1$	700	14530	6882		1580376	63	**109.2x**
$2^{477} \cdot 3^{172} - 1$	750	12046	8301		1517960	66	**125.7x**
$2^{107} \cdot 3^{437} - 1$	800	13228	7560		1548504	65	**116.3x**
$2^{166} \cdot 3^{431} - 1$	850	11968	8355		1515304	66	**126.6x**
$2^{172} \cdot 3^{459} - 1$	900	11427	8751		1500032	67	**130.6x**
$2^{536} \cdot 3^{261} - 1$	950	10233	9772		1443592	69	**141.6x**
$2^{721} \cdot 3^{176} - 1$	1000	8814	11346		1403752	71	**159.8x**

Table 8.5: The approximate number of multiplications required to search a single node using Richelot isogeny walks in $\mathcal{X}_2(\bar{\mathbb{F}}_p, 2)$ only (left) vs. using Richelot isogeny walks in $\mathcal{X}_2(\bar{\mathbb{F}}_p, 2)$ together with SplitSearcher in $\mathcal{X}_2(\bar{\mathbb{F}}_p, N)$ (right).

2. Since the Weierstrass points of genus-2 curve with superspecial Jacobian are all $\mathbb{F}_{p^2}$-rational, it may be desirable to work with the Rosenhain invariants which may be computed more efficiently. To use our methods one would need to compute a birational model for the surface $\mathcal{L}_N(2)$ whose points parametrise optimally (N, N)-split Jacobians with full level 2 structure. One approach is described in [179].

3. It may be possible to improve the complexity of the evaluations performed by EvalCoeffs (see Section 8.4.2) by taking longer walks in the $(2, 2)$-graph and then batching the evaluations using multi-point evaluation.

4. Knowledge of explicit equations for the surface $\mathcal{L}_N$ for larger N would allow us to perform efficient detection of (N, N)-splittings beyond $N = 11$. It may be possible to derive these from the pre-existing equations for the surfaces $Z(N, -1)$ (which parametrise pairs of elliptic curves (N, N)-isogenous to a genus-2 Jacobian) in [152, Theorem 2.4], [153, Theorem 1.2], and [162, Theorem 1.1], or by extending Kumar's computations.

5. As was pointed out to us by Thomas Decru, it is possible to detect $(2N, 2N)$-splittings more efficiently by taking *partial* steps in the $(2, 2)$-isogeny graph. Let $C/\mathbb{F}_{p^2}$ be a genus-2 curve given by a Weierstrass equation $y^2 = (x - a_0) \cdots (x - a_5)$. While we cannot take a full step in $\mathcal{X}_2(\bar{\mathbb{F}}_p, 2)$ (recovering the factorisation of the Weierstrass sextic for each of the $(2, 2)$-isogenous curves) without computing square roots, we can compute the Igusa–Clebsch invariants of *all*

the neighbours of $\mathcal{J}_C$ using only a small number of $\mathbb{F}_p$-multiplications and a single batched inversion. In this case we may detect $(2N, 2N)$-splittings of $\mathcal{J}_C$ by applying IsSplit to each of the $(2, 2)$-neighbours of $\mathcal{J}_C$. In the code attached to this chapter, this optimisation can be enabled by setting `split_after_22_flag = true`. In our implementation of this idea, and for the primes ranging between 50 and 1000 bits reported in Table 8.5, we observed additional improvement factors ranging between 1.3-1.6.

Part III

Two-dimensional isogenies

Chapter 9

$(3,3)$-isogenies on fast Kummer Surfaces

Robert's formulation of the SIDH attacks [271] gave us a new powerful way to represent one-dimensional isogenies of any degree, namely by embedding it into an isogeny of smooth degree between higher dimensional abelian varieties. This new representation is most efficient in dimension 2, i.e., when we restrict to (N,N)-isogenies between p.p. abelian surfaces. As such, two-dimensional isogenies have shown to be crucial tool in the construction of new isogeny-based protocols. In this chapter, we present work on finding efficient formulæ for the computation of (N,N)-isogenies between p.p. abelian surfaces. Though the method we exhibit is general and works for all odd N, we focus on the case $N=3$ giving highly optimised and explicit formulæ. Furthermore, we present a three-dimensional differential addition chain, and use it to construct (N^k,N^k)-isogeny kernels correctly, efficiently, and securely. Combining these two, we build a cryptographic hash function from $(3,3)$-isogenies, which allows us to benchmark our algorithms against the state-of-the-art. The benchmarks show that our algorithm for computing chains of $(3,3)$-isogenies outperforms those in the literature by at least a factor of 8. The methods we use lead to a constant time algorithm for computing chains of $(3,3)$-isogenies, and are therefore suited well for cryptographic applications. This chapter is based on the joint work with Craig Costello and Benjamin Smith called

Efficient $(3,3)$*-isogenies on fast Kummer surfaces*

which was accepted at the Sixteenth Algorithmic Number Theory Symposium (ANTS XVI), and will appear in the Springer journal 'Research in Number Theory' [91]. Apart from small editorial changes and merging the appendix with the main body, we present the paper as published.

Introduction

Isogenies of elliptic curves are well-understood, at least from an algorithmic point of view, in theory and in practice. Given the Weierstrass equation of an elliptic curve E, and a generator P of a finite subgroup of E, Vélu's formulæ [314] allow us to write down polynomials defining a normalized quotient isogeny $\Phi : E \to E/\langle P\rangle$ (with variants for alternative curve models [242], or for rational subgroups with irrational generators [204]). Building on these formulæ, there also exist highly efficient algorithms for evaluating an isogeny at points of E *without* deriving a polynomial representation for the isogeny itself (including [27], [102], and [266], for example). Interest in these formulæ and algorithms has recently intensified with the development of *isogeny-based cryptography* as a source of cryptosystems conjectured to be resistant against quantum attacks.

As a generalization of an elliptic curve, we consider principally polarized abelian varieties, and the first non-elliptic examples are Jacobians of genus-2 curves. Genus-2 curves are hyperelliptic curves with affine model[1]

$$\mathcal{C} : y^2 = f(x) \quad \text{where } f(x) \text{ is squarefree of degree 5 or 6}\,,$$

in characteristic not dividing 30, and $\mathcal{J}$ is the Jacobian of $\mathcal{C}$, a 2-dimensional principally polarised abelian variety as defined in Section 2.3.

Mumford [245], Cantor [64], Grant [178], and Flynn [156, 158] laid the ground for explicit geometric and number-theoretic computations with Jacobians of genus-2 curves. Cassels and Flynn's text [66] presents a unified view of genus-2 arithmetic. Later, Gaudry [175] proposed Kummer surfaces of genus-2 Jacobians as a setting for efficient discrete-logarithm-based cryptosystems, building on a variant [83] of Lentra's ECM factoring algorithm [215]. Recall from Section 2.5.1 that the Kummer surface $\mathcal{K}$ of a Jacobian $\mathcal{J}$ is the image of the quotient morphism $\pi : \mathcal{J} \to \mathcal{K} = \mathcal{J}/\langle \pm 1 \rangle$; as such, it is the genus-2 analogue of the x-coordinate of elliptic curves. Geometrically, Kummer surfaces have convenient models as quartic surfaces in $\mathbb{P}^3$ with 16 point singularities.

Cosset put Chudnovsky and Chudnovsky's Kummer ECM into practice in [97], while high-speed, high-security Kummer-based implementations of Diffie–Hellman key exchange [26, 48, 267] and signature schemes [267, 268] can give significant practical improvements over elliptic curves in many contexts.

However, while the basic arithmetic of genus-2 Jacobians and Kummer surfaces has matured, and while cryptographic applications have driven great improvements in the efficiency of the resulting formulæ and algorithms, the corresponding explicit theory of isogenies lags behind. First, recall that just as elliptic isogenies factor naturally into compositions of scalar multiplications and isogenies with prime cyclic kernel (i.e., isomorphic to $\mathbb{Z}/N\mathbb{Z}$ with N prime), isogenies of abelian surfaces (including Jacobians of genus-2 curves) decompose into compositions of scalar multiplications and (N, N)-*isogenies* (with kernel isomorphic to $(\mathbb{Z}/N\mathbb{Z})^2$).[2] The fundamental task, then, is to compute and evaluate (N, N)-isogenies where N is prime. We can do this on the level of the Jacobian (using e.g. correspondences on genus-2 curves [294]), or we can use the fact that isogenies commute with -1 to move down to the more tractable Kummer surfaces. Indeed, as Cassels and Flynn note, "we lose nothing by going down to the Kummers, because [the map] lifts automatically to a map of abelian varieties." [66, §9.3].

The case $N = 2$ is classical: explicit methods and formulæ go back to Richelot [269], and were re-developed in modern terms by Bost and Mestre [50] and Cassels and Flynn [66, §3]. Going further, we find some first efforts at explicit curve-based formulæ for the case $N = 3$ by Smith [293] (building on an ineffective general method due to Dolgachev and Lehavi [139]), and more general results due to Couveignes and Ezome [109]. Moving to general Kummer surfaces, Bruin, Flynn, and Testa [58], Nicholls [248], and Flynn [157] gave more powerful formulæ for $N = 3$, 4, and 5, respectively, in a number-theoretic context; Nicholls even gives a method for general N. Flynn and Ti [159] revisited the formulæ for $N = 3$ in a cryptographic context, and Decru and Kunzweiler [130] further optimised these formulæ, drastically improving their efficiency. However, none of these formulæ make use of the special symmetries of the most efficient Kummer surfaces that have

[1]In this chapter, we denote our algebraic curves by $\mathcal{C}$ rather than C to avoid a notational clash with the dual fundamental theta constants.

[2]In some special cases, depending on the endomorphism ring of the Jacobian, we can also have isogenies with cyclic kernel [141]. These isogenies are beyond the scope of this chapter.

been used in cryptographic implementations.

Bisson, Cosset, Lubicz, and Robert have advanced an ambitious program [35, 96, 223, 272, 273] based on the theory of theta functions [245] to provide asymptotically efficient algorithms for arbitrary odd N (and beyond genus 2 to arbitrarily high dimension). The `AVIsogenies` software package based on their results is publicly available [36]. These algorithms are certainly compatible with fast Kummer surfaces, but they target isogeny evaluation for general abelian varieties, rather than the construction of compact explicit formulæ in genus 2 that can be studied, analysed, and optimised in their own right. Nevertheless, these techniques were recently revisited by Dartois, Maino, Pope, and Robert [117] in the context of cryptography to efficiently compute chains of $(2, 2)$-isogenies between products of elliptic curves in the theta model.

Contributions

In this chapter, we give a general method for deriving explicit formulæ for isogenies of fast Kummer surfaces, optimizing the approach of Bruin, Flynn, and Testa by exploiting the high symmetry of these "fast" surfaces, which are the most relevant for applications over finite fields. Our methods are elementary in the sense that they avoid explicitly using the heavy machinery of theta functions required in [98, 117, 224] (though of course theta functions implicitly play a fundamental role in our techniques). We apply these methods to give explicit examples for $N = 3$ and 5. For example, for $N = 3$ we obtain a map $\phi : \mathcal{K} \to \mathcal{K}'$ defined by

$$\phi((X_1 : X_2 : X_3 : X_4)) = (\phi_1(X_1, X_2, X_3, X_4) : \cdots : \phi_4(X_1, X_2, X_3, X_4)) \;,$$

where

$$\begin{aligned}
\phi_1(X_1, X_2, X_3, X_4) &= X_1 \left(c_1 X_1^2 + c_2 X_2^2 + c_3 X_3^2 + c_4 X_4^2\right) + c_5 X_2 X_3 X_4 \,,\\
\phi_2(X_1, X_2, X_3, X_4) &= X_2 \left(c_2 X_1^2 + c_1 X_2^2 + c_4 X_3^2 + c_3 X_4^2\right) + c_5 X_1 X_3 X_4 \,,\\
\phi_3(X_1, X_2, X_3, X_4) &= X_3 \left(c_3 X_1^2 + c_4 X_2^2 + c_1 X_3^2 + c_2 X_4^2\right) + c_5 X_1 X_2 X_4 \,,\\
\phi_4(X_1, X_2, X_3, X_4) &= X_4 \left(c_4 X_1^2 + c_3 X_2^2 + c_2 X_3^2 + c_1 X_4^2\right) + c_5 X_1 X_2 X_3 \,,
\end{aligned}$$

and c_i are rational functions in the theta-null constants a, b, c, d defining $\mathcal{K}$ and the coordinates of the generators of the kernel (see Section 9.3.1 for the explicit expressions). This map can be evaluated with at most 88 multiplications and 12 squarings in the field containing the theta constants and generator coordinates.

To illustrate the potential benefits of our formulæ in practical applications, we give experimental results on cryptographic hash functions based on chains of $(3, 3)$-isogenies, as in [70] and [130]. In Section 9.4 we present 3DAC: a three-dimensional *differential addition chain*, and use it to construct (N^k, N^k)-isogeny kernels correctly, efficiently, and securely. Combined with our $(3, 3)$-isogeny formulæ, this allows us to efficiently compute $(3^k, 3^k)$-isogenies on fast Kummer surfaces. The hash function we define in Section 9.5 uses these isogenies, exploiting the efficient arithmetic of fast Kummer surfaces for the first time, to gain speed-ups of between 8x and 9x over the Castryk–Decru hash function [70] and between 32x and 34x over the Decru–Kunzweiler hash function [130].

Availability of Software

The source code accompanying this paper is written in `MAGMA` [49], Python and SageMath [311] and is publicly available under the MIT license. It is available at

https://github.com/mariascrs/KummerIsogenies.

9.1 Fast Kummer surfaces and their arithmetic

Let $\Bbbk$ be a perfect field—typically, a finite field or a number field—of characteristic $p \neq 2, 3$, or 5, and fix an algebraic closure $\overline{\Bbbk}$. If $\Bbbk = \mathbb{F}_q$, then we measure the time complexity of our algorithms in terms of elementary operations in $\mathbb{F}_q$. We let **M**, **S**, and **a** denote the cost of a single multiplication, squaring, and addition (or subtraction) in $\mathbb{F}_q$, respectively.

In this chapter, we focus work in dimension 2, and study Kummer surfaces corresponding to Jacobians of genus-2 curves defined over $\Bbbk$. Therefore, we assume a reader is familiar with the material introduced in Sections 2.3 and 2.5. In particular, we work with genus-2 hyperelliptic curves $\mathcal{C}$ in Rosenhain form

$$\mathcal{C} \cong \mathcal{C}_{\lambda,\mu,\nu}/\Bbbk : y^2 = x(x-1)(x-\lambda)(x-\mu)(x-\nu) \quad \text{with } \lambda, \mu, \nu \in \Bbbk\,;$$

the values λ, μ, and ν are called Rosenhain invariants of $\mathcal{C}_{\lambda,\mu,\nu}$. For more details we refer to Definition 2.3.4.

Fix a prime N not divisible by $\operatorname{char} \Bbbk$. Recalling Section 2.6 and Section 2.8.2, if G is a maximal isotropic subgroup of $\mathcal{J}[N]$, then the quotient isogeny of abelian varieties

$$\Phi : \mathcal{J} \to A' := \mathcal{J}/G$$

is an isogeny of p.p. abelian varieties with kernel G. Such an isogeny Φ is an (N,N)-isogeny, and the kernel is called as (N,N)-subgroup.

Being a p.p. abelian surface, A' is (as a p.p. abelian variety) the Jacobian of a genus-2 curve, say $\mathcal{J}'$, or a product of elliptic curves $E_1' \times E_2'$. The case $A' = \mathcal{J}'$ is the general case, and the primary focus of this paper.

In Section 9.5 we restrict superspecial Jacobians $\mathcal{J}$, as defined in Section 4.1, which is the case of most interest to cryptography.

9.1.1 Isogenies and Kummer surfaces

We recall from Section 2.5.1 that the Kummer surface $\mathcal{K}$ of a Jacobian $\mathcal{J}$ is defined to be the image of the quotient map $\pi : \mathcal{J} \to \mathcal{K} = \mathcal{J}/\{\pm 1\}$, and has sixteen nodes. Any (N,N)-isogeny $\Phi : \mathcal{J} \to \mathcal{J}'$ descends to a morphism of Kummer surfaces $\varphi : \mathcal{K} \to \mathcal{K}'$, such that the following diagram commutes:

$$\begin{tikzcd}
\mathcal{J} \arrow[r, "\Phi"'] \arrow[d, "\pi"] & \mathcal{J}' \arrow[d, "\pi'"'] \\
\mathcal{K} \arrow[r, "\varphi"] & \mathcal{K}'
\end{tikzcd}$$

Abusing terminology, we say a morphism φ of Kummer surfaces is an (N, N)-isogeny if it is induced by an (N, N)-isogeny Φ between the corresponding Jacobians.

9.1.2 Fast Kummer surfaces

Following Gaudry [175], fast Kummer surfaces are defined by four *fundamental theta constants*, which can be computed from the Rosenhain invariants of a genus-2 curve $\mathcal{C}/\Bbbk$. Given a hyperelliptic curve $\mathcal{C}/\Bbbk$ with Rosenhain invariants $\lambda, \mu, \nu \in \Bbbk$, we define *fundamental theta constants* $a, b, c, d \in \overline{\Bbbk}$ and *dual theta constants* as $A, B, C, D \in \overline{\Bbbk}$ such that

$$\begin{aligned}
A^2 &= a^2 + b^2 + c^2 + d^2, \quad B^2 = a^2 + b^2 - c^2 - d^2, \\
C^2 &= a^2 - b^2 + c^2 - d^2, \quad D^2 = a^2 - b^2 - c^2 + d^2.
\end{aligned}$$

The theta constants are related to Rosenhain invariants through the relations

$$\lambda = \frac{a^2c^2}{b^2d^2}, \qquad \mu = \frac{c^2e^2}{d^2f^2}, \qquad \nu = \frac{a^2e^2}{b^2f^2},$$

where $e, f \in \overline{\Bbbk}$ satisfy $e^2/f^2 = (AB + CD)/(AB - CD)$.

We define the *fast* Kummer model $\mathcal{K}$ corresponding to $\mathcal{C}$ as

$$\begin{aligned}
\mathcal{K} : X_1^4 + X_2^4 + X_3^4 + X_4^4 - 2E \cdot X_1X_2X_3X_4 - F \cdot (X_1^2X_4^2 + X_2^2X_3^2) \\
- G \cdot (X_1^2X_3^2 + X_2^2X_4^2) - H \cdot (X_1^2X_2^2 + X_3^2X_4^2) = 0,
\end{aligned} \tag{9.1}$$

where X_1, X_2, X_3, X_4 are coordinates on $\mathbb{P}^3$ and the coefficients E, F, G, H are rational functions in a, b, c, d, namely

$$\begin{aligned}
E &:= 256abcdA^2B^2C^2D^2/(a^2d^2 - b^2c^2)(a^2c^2 - b^2d^2)(a^2b^2 - c^2d^2), \\
F &:= (a^4 - b^4 - c^4 + d^4)/(a^2d^2 - b^2c^2), \\
G &:= (a^4 - b^4 + c^4 - d^4)/(a^2c^2 - b^2d^2), \\
H &:= (a^4 + b^4 - c^4 - d^4)/(a^2b^2 - c^2d^2).
\end{aligned} \tag{9.2}$$

This model of $\mathcal{K}$ is often referred to as the *canonical* parameterisation [268]. Note that A^2, B^2, C^2, and D^2 are linear combinations of a^2, b^2, c^2, and d^2, so the equation of $\mathcal{K}$ is determined entirely by a, b, c, d; in fact, $\mathcal{K}$ is determined by the projective point $(a : b : c : d) \in \mathbb{P}^3$. The identity element on $\mathcal{K}$ is $0_{\mathcal{K}} = (a\colon b\colon c\colon d)$.

9.1.3 Nodes of the Kummer surface

The *nodes* of $\mathcal{K}$ are the sixteen points

$$\begin{array}{rrrr}
0_{\mathcal{K}} = (a:b:c:d), & T_1 = (a:b:-c:-d), & T_2 = (a:-b:c:-d), & T_3 = (a:-b:-c:d), \\
T_4 = (b:a:d:c), & T_5 = (b:a:-d:-c), & T_6 = (b:-a:d:-c), & T_7 = (b:-a:-d:c), \\
T_8 = (c:d:a:b), & T_9 = (c:d:-a:-b), & T_{10} = (c:-d:a:-b), & T_{11} = (c:-d:-a:b), \\
T_{12} = (d:c:b:a), & T_{13} = (d:c:-b:-a), & T_{14} = (d:-c:b:-a), & T_{15} = (d:-c:-b:a).
\end{array}$$

Each T_i is the image in $\mathcal{K}$ of a two-torsion point $\widetilde{T}_i$ in $\mathcal{J}[2]$. Since $\widetilde{T}_i = -\widetilde{T}_i$, the translation-by-$\widetilde{T}_i$ map on $\mathcal{J}$ induces a morphism $\sigma_i : \mathcal{K} \to \mathcal{K}$. In fact, σ_i lifts to a linear map on $\mathbb{A}^4$: that is, it acts like a matrix on the coordinates (X_1, X_2, X_3, X_4) on $\mathbb{P}^3$. Further, σ_i and σ_j commute, respectively anticommute, if $e_2(\widetilde{T}_i, \widetilde{T}_j) = 1$, respectively -1.

In particular, if we define

$$U_1 := \operatorname{diag}(1,1,-1,-1), \qquad U_2 := \operatorname{diag}(1,-1,1,-1),$$

and

$$V_1 := \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}, \qquad V_2 := \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}.$$

Then

$$U_1^2 = U_2^2 = I_4, \text{ and } U_1U_2 = U_2U_1\,, \qquad V_1^2 = V_2^2 = I_4, \text{ and } V_1V_2 = V_2V_1\,, \tag{9.3}$$

and

$$U_1V_2 = -V_2U_1\,, \qquad U_2V_1 = -V_1U_2\,, \qquad U_1V_1 = V_1U_1\,, \qquad U_2V_2 = V_2U_2\,. \tag{9.4}$$

Taking the labelling of the nodes above, with $T_0 = (a:b:c:d)$ as the image of the identity $\widetilde{T}_0 = 0_{\mathcal{J}}$, the corresponding translations are such that

$$T_i = \sigma_i((a:b:c:d)) \qquad \text{for } 0 \le i \le 15\,;$$

that is,

$$\begin{array}{llll}
\sigma_0 = I_4 & \sigma_1 = U_1 & \sigma_2 = U_2 & \sigma_3 = U_1U_2 \\
\sigma_4 = V_1 & \sigma_5 = V_1U_1 & \sigma_6 = V_1U_2 & \sigma_7 = V_1U_1U_2 \\
\sigma_8 = V_1V_2 & \sigma_9 = V_1V_2U_1 & \sigma_{10} = V_1V_2U_2 & \sigma_{11} = V_1V_2U_1U_2 \\
\sigma_{12} = V_2 & \sigma_{13} = V_2U_1 & \sigma_{14} = V_2U_2 & \sigma_{15} = V_2U_1U_2
\end{array}$$

Now Equation (9.3) and Equation (9.4) show that $(\widetilde{T}_1, \widetilde{T}_2, \widetilde{T}_{12}, \widetilde{T}_4)$ is a symplectic basis (with respect to the 2-Weil pairing) of $\mathcal{J}[2]$, where we define a symplectic basis as follows.

Definition 9.1.1. Let $\mathcal{J}$ be the Jacobian of a genus-2 curve $\mathcal{C}$. A basis $\{Q_1, Q_2, Q_3, Q_4\}$ for $\mathcal{J}[D]$ is *symplectic* with respect to the D-Weil pairing if

$$e_D(Q_1, Q_3) = e_D(Q_2, Q_4) = \zeta$$

where ζ is a primitive D-th root of unity, and $e_D(Q_i, Q_j) = 1$ otherwise.

9.1.4 Operations on the Kummer surface

Let $\pi : \mathcal{J} \to \mathcal{K}$ be the quotient by -1. The multiplication-by-m maps $[m]$ on $\mathcal{J}$ induce *pseudo-multiplications* $\pi(P) \mapsto [m]_*(\pi(P)) = \pi([m]P)$. We can express the pseudo-doubling map $[2]_*$ on $\mathcal{K}$ as a composition of four basic building blocks, each a morphism from $\mathbb{P}^3$ to $\mathbb{P}^3$:

1. the *Hadamard involution* $\mathtt{H} : \mathbb{P}^3 \to \mathbb{P}^3$, which is induced by the linear map on $\mathbb{A}^4$ defined by the matrix
$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix};$$

2. the *squaring* map
$$\mathtt{S} : (X_1 : X_2 : X_3 : X_4) \longmapsto (X_1^2 : X_2^2 : X_3^2 : X_4^2)\,;$$

3. the *scaling* maps
$$\mathtt{C}_{(\alpha:\beta:\gamma:\delta)} : (X_1 : X_2 : X_3 : X_4) \longmapsto (\alpha X_1 : \beta X_2 : \gamma X_3 : \delta X_4)$$
for each $(\alpha : \beta : \gamma : \delta) \in \mathbb{P}^3(\Bbbk)$; and

4. the *inversion* map
$$\begin{aligned} \mathtt{Inv} : (X_1 : X_2 : X_3 : X_4) &\longmapsto (X_2X_3X_4 : X_1X_3X_4 : X_1X_2X_4 : X_1X_2X_3) \\ &= (1/X_1 : 1/X_2 : 1/X_3 : 1/X_4)\,, \end{aligned}$$
well-defined when all $X_i \neq 0$.

We can readily see that $\mathtt{H}$ costs 8 $\Bbbk$-additions, $\mathtt{S}$ costs 4 $\Bbbk$-squarings, $\mathtt{C}$ costs 4 $\Bbbk$-multiplications, and $\mathtt{Inv}$ costs 6 $\Bbbk$-multiplications. Now, if $\mathcal{K}$ is a Kummer surface with fundamental theta constants $(a : b : c : d)$, then pseudo-doubling is given by

$$[2]_* = \mathtt{C}_{\mathtt{Inv}(0_{\mathcal{K}})} \circ \mathtt{H} \circ \mathtt{S} \circ \mathtt{C}_{\mathtt{Inv}((A:B:C:D))} \circ \mathtt{H} \circ \mathtt{S}\,.$$

Recall from Section 2.5.1 that while $\mathcal{K}$ inherits scalar multiplication from $\mathcal{J}_{\mathcal{C}}$, it loses the group law: for divisors $D_P, D_Q, D_{P+Q} \in \mathcal{J}_{\mathcal{C}}$, the points $P = \pi(\pm D_P)$ and $Q = \pi(\pm D_Q)$ on $\mathcal{K}$ do not uniquely determine $P + Q = \pi(\pm D_{P+Q})$, unless at least one of P and Q is the image of a point in $\mathcal{J}_{\mathcal{C}}[2]$.

However, the operation $\{P, Q\} \mapsto \{P+Q, P-Q\}$ is well-defined, so we have a *pseudo-addition* operation $(P, Q, P-Q) \mapsto P+Q$.

By abuse of notation, we let $R = (r_1\colon r_2\colon r_3\colon r_4)$ and $S = (s_1\colon s_2\colon s_3\colon s_4)$ be points on $\mathcal{K}$, and let $T^+ = (t_1^+ : t_2^+ : t_3^+ : t_4^+)$ and $T^- = (t_1^- : t_2^- : t_3^- : t_4^-)$ denote the sum $R+S$ and difference $R-S$, respectively. There exist biquadratic forms B_{ij} [66, Theorem 3.9.1] for $\mathcal{K}$ such that for $1 \leq i, j \leq 4$ we have

$$t_i^+ t_j^- + t_i^- t_j^+ = \lambda B_{ij}(r_1, r_2, r_3, r_4; s_1, s_2, s_3, s_4) = \lambda B_{ij}(R; S),$$

where $\lambda \in \overline{\Bbbk}$ is a common projective factor depending only on the affine representations chosen for R, S, T^+, T^-. The biquadratic forms $B_{i,j}$ for fast Kummer surfaces are given explicitly by Renes and Smith [268, §5.2].

These biquadratic forms are the basis of explicit pseudo-addition and doubling laws on $\mathcal{K}$. For example, if the difference T^- is known, then the B_{ij} can be used to compute the coordinates of T^+. As we will see in Section 9.2, the B_{ij} will also be crucial in determining equations for our (N, N)-isogenies.

9.2 (N, N)-isogenies on fast Kummer surfaces

Throughout this section, the morphism $\Phi : \mathcal{J} \to \mathcal{J}'$ is an (N, N)-isogeny with kernel $G \subset \mathcal{J}[N]$, a maximal N-Weil isotropic subgroup of $\mathcal{J}[N]$, where N is a prime number not equal to the characteristic of the base field $\Bbbk$. Our goal is to compute an explicit and efficiently-computable collection of polynomials defining the induced map $\varphi : \mathcal{K} \to \mathcal{K}'$ when $\mathcal{K}$ and $\mathcal{K}'$ admit *fast* models.

9.2.1 A warm-up with $N = 2$

We first dispose of the case $N = 2$. Let $\widetilde{T}_0, \ldots, \widetilde{T}_{15}$ be the 16 points in $\mathcal{J}[2]$, and let $T_0, \ldots, T_{15}$ be their images in $\mathcal{K}$. Recall that $T_i = \sigma_i(T_0)$, where $\sigma_i : \mathcal{K} \to \mathcal{K}$ is a morphism defining the translation-by-T_i map. There are precisely fifteen (images of) $(2,2)$-subgroups in $\mathcal{K}$, and they are the images of

$$\widetilde{G}_{ij} := \{\widetilde{T}_0,\ \widetilde{T}_i,\ \widetilde{T}_j,\ \widetilde{T}_i + \widetilde{T}_j\} \subset \mathcal{J}[2]$$

in $\mathcal{K}$, where $1 \leq i \neq j \leq 15$ such that the linear maps corresponding to σ_i and σ_j commute. Explicitly, the $(2,2)$-subgroups on $\mathcal{K}$ with $0_{\mathcal{K}} = (a : b : c : d)$ are

$$\begin{aligned}
G_{1,2} &:= \{(a\colon b\colon c\colon d),\ (a\colon b\colon -c\colon -d),\ (a\colon -b\colon c\colon -d),\ (a\colon -b\colon -c\colon d)\},\\
G_{1,4} &:= \{(a\colon b\colon c\colon d),\ (a\colon b\colon -c\colon -d),\ (b\colon a\colon d\colon c),\ (b\colon a\colon -d\colon -c)\},\\
G_{1,6} &:= \{(a\colon b\colon c\colon d),\ (a\colon b\colon -c\colon -d),\ (b\colon -a\colon d\colon -c),\ (b\colon -a\colon -d\colon c)\},\\
G_{2,8} &:= \{(a\colon b\colon c\colon d),\ (a\colon -b\colon c\colon -d),\ (c\colon d\colon a\colon b),\ (c\colon -d\colon a\colon -b)\},\\
G_{2,9} &:= \{(a\colon b\colon c\colon d),\ (a\colon -b\colon c\colon -d),\ (c\colon d\colon -a\colon -b),\ (c\colon -d\colon -a\colon b)\},\\
G_{3,12} &:= \{(a\colon b\colon c\colon d),\ (a\colon -b\colon -c\colon d),\ (d\colon c\colon b\colon a),\ (d\colon -c\colon -b\colon a)\},\\
G_{3,14} &:= \{(a\colon b\colon c\colon d),\ (a\colon -b\colon -c\colon d),\ (d\colon c\colon -b\colon -a),\ (d\colon -c\colon b\colon -a)\},\\
G_{4,8} &:= \{(a\colon b\colon c\colon d), (b\colon a\colon d\colon c), (c\colon d\colon a\colon b), (d\colon c\colon b\colon a)\},
\end{aligned}$$

$$\begin{aligned}
G_{4,9} &:= \{(a\colon b\colon c\colon d), (b\colon a\colon d\colon c), (c\colon d\colon -a\colon -b), (d\colon c\colon -b\colon -a)\},\\
G_{5,10} &:= \{(a\colon b\colon c\colon d), (b\colon a\colon -d\colon -c), (c\colon -d\colon a\colon -b), (d\colon -c\colon -b\colon a)\},\\
G_{5,11} &:= \{(a\colon b\colon c\colon d), (b\colon a\colon -d\colon -c), (c\colon -d\colon -a\colon b), (d\colon -c\colon b\colon -a)\},\\
G_{6,8} &:= \{(a\colon b\colon c\colon d), (b\colon -a\colon d\colon -c), (c\colon d\colon a\colon b), (d\colon -c\colon b\colon -a)\},\\
G_{6,9} &:= \{(a\colon b\colon c\colon d), (b\colon -a\colon d\colon -c), (c\colon d\colon -a\colon -b), (d\colon -c\colon -b\colon a)\},\\
G_{7,10} &:= \{(a\colon b\colon c\colon d), (b\colon -a\colon -d\colon c), (c\colon -d\colon a\colon -b), (d\colon c\colon -b\colon -a)\},\\
G_{7,11} &:= \{(a\colon b\colon c\colon d), (b\colon -a\colon -d\colon c), (c\colon -d\colon -a\colon b), (d\colon c\colon b\colon a)\}.
\end{aligned}$$

This gives 15 corresponding $(2,2)$-isogenies given by $\varphi : \mathcal{K} \to \mathcal{K}' = \mathcal{K}/G_{i,j}$. For each unique $(2,2)$-subgroup, we can associate a morphism $\alpha : \mathcal{K} \to \mathcal{K}$ induced by a linear map on $\mathbb{A}^4$ such that the corresponding $(2,2)$-isogeny $\mathcal{K} \to \widetilde{\mathcal{K}}$ is given by

$$\psi := \mathtt{H} \circ \mathtt{S} \circ \alpha.$$

The non-zero entries of the 4×4 matrix $\mathbf{A}$ defining α are all fourth roots of unity. Let i be a primitive fourth root of unity in $\overline{\Bbbk}$. We give matrices $\mathbf{A}$ specifying the linear map α for each of the $(2,2)$-subgroups below.

$$G_{1,2} : \mathbf{A} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \qquad G_{1,4} : \mathbf{A} = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{pmatrix},$$

$$G_{1,6} : \mathbf{A} = \begin{pmatrix} 1 & i & 0 & 0 \\ 1 & -i & 0 & 0 \\ 0 & 0 & 1 & i \\ 0 & 0 & 1 & -i \end{pmatrix}, \qquad G_{2,8} : \mathbf{A} = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & -1 \end{pmatrix},$$

$$G_{2,9} : \mathbf{A} = \begin{pmatrix} 1 & 0 & i & 0 \\ 1 & 0 & -i & 0 \\ 0 & 1 & 0 & i \\ 0 & 1 & 0 & -i \end{pmatrix}, \qquad G_{3,12} : \mathbf{A} = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & -1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & -1 & 0 \end{pmatrix},$$

$$G_{3,14} : \mathbf{A} = \begin{pmatrix} 1 & 0 & 0 & i \\ 1 & 0 & 0 & -i \\ 0 & 1 & i & 0 \\ 0 & 1 & -i & 0 \end{pmatrix}, \qquad G_{4,8} : \mathbf{A} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix},$$

$$G_{4,9} : \mathbf{A} = \begin{pmatrix} 1 & 1 & i & i \\ 1 & 1 & -i & -i \\ 1 & -1 & i & -i \\ 1 & -1 & -i & i \end{pmatrix}, \qquad G_{5,10} : \mathbf{A} = \begin{pmatrix} -1 & 1 & 1 & 1 \\ 1 & -1 & 1 & 1 \\ 1 & 1 & -1 & 1 \\ 1 & 1 & 1 & -1 \end{pmatrix},$$

$$G_{5,11} : \mathbf{A} = \begin{pmatrix} 1 & -1 & -i & -i \\ 1 & -1 & i & i \\ 1 & 1 & -i & i \\ 1 & 1 & i & -i \end{pmatrix}, \qquad G_{6,8} : \mathbf{A} = \begin{pmatrix} 1 & i & 1 & i \\ 1 & i & -1 & -i \\ 1 & -i & 1 & -i \\ 1 & -i & -1 & i \end{pmatrix},$$

$$G_{6,9} : \mathbf{A} = \begin{pmatrix} 1 & -i & -i & -1 \\ 1 & -i & i & 1 \\ 1 & i & -i & 1 \\ 1 & i & i & -1 \end{pmatrix}, \qquad G_{7,10} : \mathbf{A} = \begin{pmatrix} 1 & -i & -1 & -i \\ 1 & -i & 1 & i \\ 1 & i & -1 & i \\ 1 & i & 1 & -i \end{pmatrix},$$

$$G_{7,11} : \mathbf{A} = \begin{pmatrix} 1 & i & i & 1 \\ 1 & i & -i & -1 \\ 1 & -i & i & -1 \\ 1 & -i & -i & 1 \end{pmatrix}.$$

We note, however, that $\widetilde{\mathcal{K}}$ will not be in the fast Kummer correct form. We must therefore apply a final scaling $\mathtt{C}_U$ where $U = (\mathtt{S}^{-1} \circ \mathtt{Inv} \circ \psi)(0_{\mathcal{K}})$. From this we obtain a $(2,2)$-isogeny $\varphi := \mathtt{C}_U \circ \psi : \mathcal{K} \to \mathcal{K}' = \mathcal{K}/G_{i,j}$, where $\mathcal{K}$ and $\mathcal{K}'$ are fast Kummer surfaces.

Example 9.2.1. Consider the $(2,2)$-subgroup

$$G_{1,2} = \{(a\colon b\colon c\colon d), (a\colon b\colon -c\colon -d), (a\colon -b\colon c\colon -d), (a\colon -b\colon -c\colon d)\}.$$

Here α is the identity map and the $(2,2)$-isogeny is given by

$$(X_1\colon X_2\colon X_3\colon X_4) \mapsto \left(\frac{X_1^2+X_2^2+X_3^2+X_4^2}{A} : \frac{X_1^2+X_2^2-X_3^2-X_4^2}{B} : \right. \\ \left. : \frac{X_1^2-X_2^2+X_3^2-X_4^2}{C} : \frac{X_1^2-X_2^2-X_3^2+X_4^2}{D}\right)$$

We call this the *distinguished kernel*: the kernel of the first half of doubling. Indeed, $U = (A : B : C : D)$ and we recover the first three steps $\mathtt{C}_{\mathtt{Inv}(A:B:C:D)} \circ \mathtt{H} \circ \mathtt{S}$ of the doubling map given by Gaudry [175, §3.2] on fast Kummer surfaces.

Remark 9.2.2. Though the formulæ for $(2,2)$-isogenies on fast Kummer surfaces are extremely compact, we remark that the final scaling requires the computation of square roots in $\overline{\Bbbk}$. If R, S are the 2-torsion points generating the isogeny, let P, $Q \in \mathcal{K}$ be such that $R = [2]_*P$, $S = [2]_*Q$. The coordinates of P, Q contain the square roots needed for the final scaling (up to a projective factor). We therefore suspect that these square roots may be inferred directly from coordinates of the 4-torsion, though we have not been able to derive such formulæ.[3]

[3]Since this work was made public, it has become clear to this author that these square roots can instead be extracted from the 8-torsion points lying above the kernel generators; see [117].

9.2.2 The general case: odd N

From this point forward, we suppose N is an odd prime. Since N is odd, the (N, N)-isogeny Φ restricts to an isomorphism of 2-torsion subgroups $\mathcal{J}[2] \to \mathcal{J}'[2]$. Furthermore, since Φ is an isogeny of p.p. abelian varieties it is compatible with the N-Weil pairing by definition, and so it maps the symplectic structure on $\mathcal{J}[2]$ associated with the fast Kummer $\mathcal{K}$ onto a symplectic 2-torsion structure on $\mathcal{J}'[2]$, which is associated with a fast Kummer $\mathcal{K}'$. The isogeny Φ therefore descends to a morphism $\varphi : \mathcal{K} \to \mathcal{K}'$ of fast Kummers. Our goal is to construct explicit equations for φ.

To do so, we follow the strategy taken by Cassels and Flynn [66, §9], Bruin, Flynn, and Testa [58], Nicholls [248, §5], and Flynn [157], adapting it to the case of *fast* Kummer surfaces. We will observe that the special forms of the affine translation maps σ_i are very helpful in this setting, and lead to nice results.

In practice, we are given a fast Kummer $\mathcal{K}$ and the image $\pi(G)$ of an (N, N)-subgroup G of $\mathcal{J}$ in $\mathcal{K}$. There exists a fast Kummer $\mathcal{K}' \cong (\mathcal{J}'/G)/\langle \pm 1\rangle$, and our goal is to find $\mathcal{K}'$ and the map $\varphi : \mathcal{K} \to \mathcal{K}'$ induced by the quotient (N, N)-isogeny Φ with kernel G. Crucially, φ "commutes" with the action by 2-torsion points, in the sense of the following definition.

Definition 9.2.3. An *isogeny of fast Kummer surfaces* is a morphism $\varphi : \mathcal{K} \to \mathcal{K}'$ induced by an isogeny $\Phi : \mathcal{J} \to \mathcal{J}'$ such that when lifted to a map on the ambient space, we have

$$\varphi \circ U_i^{\mathcal{K}} = U_i^{\mathcal{K}'} \circ \varphi \quad \text{and} \quad \varphi \circ V_i^{\mathcal{K}} = V_i^{\mathcal{K}'} \circ \varphi \quad \text{for } i = 1, 2\,,$$

where U_i and V_i are as defined in Section 9.1.3.

We want to compute $\varphi : \mathcal{K} \to \mathcal{K}'$, but $\mathcal{K}'$ is unknown. However, as $\mathcal{K}$ and $\mathcal{K}'$ are both embedded in $\mathbb{P}^3$, φ must be defined by forms of degree N, and it must commute with the actions of the U_i and V_i. This imposes heavy constraints on the shape of the forms defining φ, and we can hope to interpolate them using linear algebra given the action of G, and therefore to interpolate the image Kummer $\mathcal{K}'$ by pushing the theta constants $(a\colon b\colon c\colon d)$ through the isogeny.

Let $\mathcal{K}[N]$ be the image of $\mathcal{J}[N]$ in $\mathcal{K}$ and fix $R, S \in \mathcal{K}[N]$. From this point forward, we write $\langle R, S\rangle \subset \mathcal{K}[N]$ for the image of the subgroup G of $\mathcal{J}[N]$ generated by the preimages of R, S. By abuse of notation, we say that R, S are N-torsion points on $\mathcal{K}$.

The first step is to compute two sets of homogeneous forms of degree N in the coordinates of $\mathcal{K}$ that are invariant under translation by R and by S. The following lemma, due to Nicholls [248, §5.8.4], describes how we can use the biquadratic forms associated to the Kummer surface introduced in Section 9.1.2 to construct these homogeneous forms.

Lemma 9.2.4. *Fix Kummer surface $\mathcal{K}$ with coordinates X_1, X_2, X_3, X_4 and associated biquadratic forms $B_{i,j}$ for $1 \le i, j \le 4$. Let N be an odd prime number, and fix a point $R \in \mathcal{K}[N]$ of order N.*

We denote by $I \in \{1, 2, 3, 4\}^N$ a list of indices $I = (i_1, \ldots, i_N)$. Letting τ be a permutation of $\{1, \ldots, N\}$, we write

$$\tau(I) = \tau((i_1, \ldots, i_N)) := (i_{\tau(1)}, \ldots, i_{\tau(N)}).$$

Then, for each $I \in \{1,2,3,4\}^N$ we define

$$F_I := \sum_{\tau \in C_N} X_{i_{\tau(1)}} \cdot \prod_{k=1}^{(N-1)/2} B_{i_{\tau(2k)}, i_{\tau(2k+1)}}(X_1, X_2, X_3, X_4;\ kR)\,,$$

where C_N is the cyclic group of order N. Then, the set $\mathcal{F}_R := \{F_I\}$ contains homogeneous forms of degree N invariant under translation by R.

Applying the lemma above to N-torsion points R and S, we obtain the two sets $\mathcal{F}_R$ and $\mathcal{F}_S$. The homogeneous forms of degree N in each set will generate a space of dimension $m_N \geq 4$. Our experiments suggest $m_N = 2N + 2$ for $N \leq 19$, and possibly beyond, though we have not proven this.[4]

The next step is to compute a basis for these two spaces, say $F_1^R, \ldots, F_{m_N}^R$ is a basis for the space generated by the homogeneous forms in $\mathcal{F}_R$, and $F_1^S, \ldots, F_{m_N}^S$ a basis for the space generated by $\mathcal{F}_S$.

The intersection of these spaces contains homogeneous forms of degree N that are invariant under translation by any point in the kernel G of our (N,N)-isogeny. The intersection will be of dimension 4, and a basis for this intersection gives an (N,N)-isogeny $\psi : \mathcal{K} \to \widetilde{\mathcal{K}}$. Explicitly, the third step is to compute a basis of this intersection, say f_1, f_2, f_3, f_4. Then, our (N,N)-isogeny is given by $\psi = (f_1 : f_2 : f_3 : f_4)$.

We note that $\widetilde{\mathcal{K}}$ may not be in the correct form given by Equation (9.1). When computing chains of isogenies, however, it is important to ensure that our (N,N)-isogenies have domain and image in the same form. Therefore, the last step is to apply a linear transformation $\mathbf{M} : \widetilde{\mathcal{K}} \to \mathcal{K}'$, where $\mathcal{K}'$ is a fast Kummer surface. Post-composing the map ψ with this linear transformation gives a $\bar{\mathbb{k}}$-rational (N,N)-isogeny $\varphi : \mathcal{K} \to \mathcal{K}'$ between fast Kummer surfaces generated by the kernel $G = \langle R, S \rangle$.

Remark 9.2.5. The compactness and efficiency of our isogeny formulae is determined by the choice of basis we make for the spaces generated by the forms in $\mathcal{F}_R$ and $\mathcal{F}_S$. An open question that arises from this work, therefore, is finding a solution to the following problem: let $f_1, \ldots, f_n \in \mathbb{Q}(a_1, \ldots, a_k)[x_1, \ldots, x_m]$ be a basis of polynomials defined over a function field. Find a "nice" basis $g_1, \ldots, g_n$ where $g_1, \ldots, g_n$ are $\mathbb{Q}(a_1, \ldots, a_k)$-linear combinations of the f_i.

9.3 Explicit $(3,3)$-isogenies on fast Kummers

We now specialise the discussion in Section 9.2 to $N = 3$ to construct $(3,3)$-isogenies between fast Kummer surfaces.

Let $\mathcal{J}$ be the Jacobian of a genus-2 curve $\mathcal{C}$ defined over $\bar{\mathbb{k}}$. Suppose we have a $(3,3)$-subgroup of $\mathcal{J}[3]$, which induces an isogeny φ on the corresponding fast Kummer surface $\mathcal{K} = \mathcal{J}/\{\pm 1\}$ with kernel $G = \langle R, S \rangle$ for some $R, S \in \mathcal{K}[3]$ (i.e., G is the image of the $(3,3)$-subgroup in $\mathcal{K}$).

Exploiting the fact that φ is an isogeny of fast Kummer surfaces, we obtain the following lemma, demonstrating that it is determined by five $\bar{\mathbb{k}}$-rational functions in the coordinates of $0_{\mathcal{K}} = (a\colon b\colon c\colon d)$, R and S.

[4] This has since been proved in follow-up joint work with Flynn [94].

Lemma 9.3.1. *Let R and S be distinct 3-torsion points on $\mathcal{K}$ generating a $(3,3)$-subgroup $G \subset \mathcal{K}[3]$, and set $0_{\mathcal{K}} = (a\colon b\colon c\colon d)$. The $(3,3)$-isogeny of fast Kummer surfaces $\varphi\colon \mathcal{K} \to \mathcal{K}'$ generated by kernel G is in the form*

$$(X_1 : X_2 : X_3 : X_4) \longmapsto (\varphi_1(X_1, X_2, X_3, X_4) : \cdots : \varphi_4(X_1, X_2, X_3, X_4)) \ ,$$

where

$$\begin{aligned}
\varphi_1(X_1, X_2, X_3, X_4) &= X_1 \left(c_1 X_1^2 + c_2 X_2^2 + c_3 X_3^2 + c_4 X_4^2\right) + c_5 X_2 X_3 X_4 \,,\\
\varphi_2(X_1, X_2, X_3, X_4) &= X_2 \left(c_2 X_1^2 + c_1 X_2^2 + c_4 X_3^2 + c_3 X_4^2\right) + c_5 X_1 X_3 X_4 \,,\\
\varphi_3(X_1, X_2, X_3, X_4) &= X_3 \left(c_3 X_1^2 + c_4 X_2^2 + c_1 X_3^2 + c_2 X_4^2\right) + c_5 X_1 X_2 X_4 \,,\\
\varphi_4(X_1, X_2, X_3, X_4) &= X_4 \left(c_4 X_1^2 + c_3 X_2^2 + c_2 X_3^2 + c_1 X_4^2\right) + c_5 X_1 X_2 X_3 \,,
\end{aligned}$$

with $c_i \in \overline{\Bbbk}[a, b, c, d, r_1, r_2, r_3, r_4, s_1, s_2, s_3, s_4]$.

Proof. By Lemma 9.2.4, the isogeny φ is given by cubic forms. That is, φ is defined by polynomials

$$\begin{aligned}
\varphi_i = {} & c_{i,1} X_1^3 + c_{i,2} X_1 X_2^2 + c_{i,3} X_1 X_3^2 + c_{i,4} X_1 X_4^2 + c_{i,5} X_2 X_3 X_4 \\
& + c_{i,6} X_2 X_1^2 + c_{i,7} X_2^3 + c_{i,8} X_2 X_3^2 + c_{i,9} X_2 X_4^2 + c_{i,10} X_1 X_3 X_4 \\
& + c_{i,11} X_3 X_1^2 + c_{i,12} X_3 X_2^2 + c_{i,13} X_3^3 + c_{i,14} X_3 X_4^2 + c_{i,15} X_1 X_2 X_4 \\
& + c_{i,16} X_4 X_1^2 + c_{i,17} X_4 X_2^2 + c_{i,18} X_4 X_3^2 + c_{i,19} X_4^3 + c_{i,20} X_1 X_2 X_3 \,,
\end{aligned}$$

where $c_{i,j}$ are $\overline{\Bbbk}$-rational functions in the coordinates of $0_{\mathcal{K}}$, R, and S for $1 \le i \le 4$ and $1 \le j \le 20$. We are looking for an isogeny of fast Kummer surfaces in the sense of Definition 9.2.3, and compatibility with the translation-by-2-torsion maps forces

$$\sigma_i'((\varphi_1\colon \varphi_2\colon \varphi_3\colon \varphi_4)) = \varphi(\sigma_i(X_1\colon X_2\colon X_3\colon X_4)), \tag{9.5}$$

for all $1 \le i \le 15$. Here, σ_i is the action of $T_i \in \mathcal{K}[2]$, and similarly σ_i' is the action of $T_i' \in \mathcal{K}'[2]$ (as defined in Section 9.1.3). Equation (9.5) gives rise to relations between the coefficients of the cubic monomials, from which we deduce that φ is of the form as in the statement of the lemma. See `section4/lemma-4_1.m` in the code accompanying this paper. Clearing denominators (as our Kummer surfaces lie in $\mathbb{P}^3$), we obtain the $c_i \in \overline{\Bbbk}[a, b, c, d, r_1, r_2, r_3, r_4, s_1, s_2, s_3, s_4]$. $\square$

By Lemma 9.3.1, to determine explicit formulae for the $(3,3)$-isogeny φ generated by kernel $G = \langle R, S\rangle \subset \mathcal{K}$, it suffices to determine the coefficients $c_1, \ldots, c_5$. We follow the method given in Section 9.2 and compute the G-invariant cubic forms. Define

$$\begin{aligned}
B_{ij}^R(X_1, X_2, X_3, X_4) &:= B_{i,j}(X_1, X_2, X_3, X_4; R),\\
B_{ij}^S(X_1, X_2, X_3, X_4) &:= B_{i,j}(X_1, X_2, X_3, X_4; S).
\end{aligned}$$

By Lemma 9.2.4, the cubic forms invariant under translation by R and S are given by

$$F^R_{ijk} := X_i B^R_{jk} + X_j B^R_{ki} + X_k B^R_{ij},$$
$$F^S_{ijk} := X_i B^S_{jk} + X_j B^S_{ki} + X_k B^S_{ij},$$

respectively, where $1 \leq i, j, k \leq 4$. Let $\mathcal{F}_R = \{F^R_{ijk}\}_{1\leq i,j,k\leq 4}$ and similarly define $\mathcal{F}_S$. The cubic forms in $\mathcal{F}_R$ and $\mathcal{F}_S$ each generate a space of dimension 8, for which we choose a basis

$$\{F^R_{111},\ F^R_{234},\ F^R_{222},\ F^R_{134},\ F^R_{333},\ F^R_{124},\ F^R_{444},\ F^R_{123}\},$$

and similarly for $\mathcal{F}_S$. These spaces will intersect in a space of dimension 4, which will give a description of the $(3, 3)$-isogeny. We compute a basis

$$f_1 := z_1 F^R_{111} + z_2 F^R_{234}\,, \qquad f_2 := z_3 F^R_{222} + z_4 F^R_{134}\,,$$
$$f_3 := z_5 F^R_{333} + z_6 F^R_{124}\,, \qquad f_4 := z_7 F^R_{444} + z_8 F^R_{123}$$

for the intersection, with $z_1, \dots, z_8 \in \overline{\Bbbk}$ such that there exist $w_1, \dots, w_8 \in \overline{\Bbbk}$ with

$$f_1 = w_1 F^S_{111} + w_2 F^S_{234}\,, \qquad f_2 = w_3 F^S_{222} + w_4 F^S_{134}\,,$$
$$f_3 = w_5 F^S_{333} + w_6 F^S_{124}\,, \qquad f_4 = w_7 F^S_{444} + w_8 F^S_{123}\,.$$

From this, we obtain a $(3, 3)$-isogeny $\psi = (f_1 \colon f_2 \colon f_3 \colon f_4) : \mathcal{K} \to \widetilde{\mathcal{K}}$. To move $\widetilde{\mathcal{K}}$ to the correct form, we first define

$$D_1 := (ab - cd)(ab + cd),\ D_2 := (ac - bd)(ac + bd),\ D_3 := (ad - bc)(ad + bc).$$

Let $D_{ij} := D_i \cdot D_j$. For a point $P = (x_1 \colon x_2 \colon x_3 \colon x_4)$, we define

$$\gamma(P) := (D_{23}(x_1x_2ab - x_3x_4cd) + D_{13}(x_1x_3ac - x_2x_4bd) + D_{12}(x_1x_4ad - x_2x_3bc)),$$

and $h_i(P)$ as the coordinates of $(\mathtt{H} \circ \mathtt{S})(P)$, for $i = 1, 2, 3, 4$. Applying a linear transformation $\mathbf{A}$ to $\widetilde{\mathcal{K}}$, where $\mathbf{A}$ is defined as

$$\mathbf{A} := \begin{pmatrix} 1/\alpha_1 & 0 & 0 & 0 \\ 0 & 2/\alpha_2 & 0 & 0 \\ 0 & 0 & 2/\alpha_1 & 0 \\ 0 & 0 & 0 & 2/(3\alpha_2) \end{pmatrix}$$

and where

$$\alpha_1 := D_3(\gamma(R)(s_4s_3ab - s_1s_2cd) - \gamma(S)(r_4r_3ab - r_1r_2cd)),$$
$$\alpha_2 := D_1(\gamma(R)(s_2s_3ad - s_1s_4bc) - \gamma(S)(r_2r_3ad - r_1r_4bc)),$$

we get a simple and efficiently computable expression for our $(3, 3)$-isogeny $\varphi := \mathbf{A}(f_1, f_2, f_3, f_4)^T$, whose image is in the desired form. The formulæ for the intersection and linear transformation can be found and verified in the file `section4/linear-transform.m` in the accompanying code.

9.3.1 Explicit formulae for $(3,3)$-isogenies

We now give the explicit formulae for the isogeny $\varphi : \mathcal{K} \to \mathcal{K}'$ generated by kernel $G = \langle R, S\rangle$. Recall from Lemma 9.3.1 that it suffices to give the explicit formulæ for the coefficients $c_i \in \overline{\Bbbk}[a, b, c, d, r_1, r_2, r_3, r_4, s_1, s_2, s_3, s_4]$ for $i \in \{1, 2, 3, 4, 5\}$. We set

$$\begin{aligned}
\beta_1 &:= D_{23}\big(\gamma(R) \cdot (s_3 s_4 ab - s_1 s_2 cd) - \gamma(S) \cdot (r_3 r_4 ab - r_1 r_2 cd)\big),\\
\beta_2 &:= h_1(R) \cdot h_2(S) - h_2(R) \cdot h_1(S).
\end{aligned}$$

Then, maintaining the notation above, we find

$$\begin{aligned}
c_1 &= 2\beta_1 h_1(R)h_1(S),\\
c_2 &= \beta_1\big(h_1(R)h_2(S) + h_2(R)h_1(S)\big)\\
&\quad + \beta_2\big(\gamma(R)(s_3 s_4 ab - s_1 s_2 cd) + \gamma(S)(r_3 r_4 ab - r_1 r_2 cd)\big)D_{23},\\
c_3 &= \beta_1\big(h_1(R)h_3(S) + h_3(R)h_1(S)\big)\\
&\quad + \beta_2\big(\gamma(R)(s_2 s_4 ac - s_1 s_3 bd) + \gamma(S)(r_2 r_4 ac - r_1 r_3 bd)\big)D_{13},\\
c_4 &= \beta_1\big(h_1(R)h_4(S) + h_4(R)h_1(S)\big)\\
&\quad + \beta_2\big(\gamma(R)(s_2 s_3 ad - s_1 s_4 bc) + \gamma(S)(r_2 r_3 ad - r_1 r_4 bc)\big)D_{12},\\
c_5 &= 2\beta_2\gamma(S)\gamma(R).
\end{aligned}$$

Note that the $c_1, \dots, c_5$ are symmetric in R and S, as one would expect. Indeed, our formulæ should not depend on whether we evaluate the coefficients at the coordinates of $0_{\mathcal{K}}, R, S$ or $0_{\mathcal{K}}, S, R$.

Remark 9.3.2. The $(3,3)$-isogeny is defined over the field of definition of the fundamental constants a, b, c, d of $\mathcal{K}$ and the kernel generators R, S (rather than the subgroup $\langle R, S\rangle$).

9.3.2 Evaluating points under the $(3,3)$-isogeny

Consider the $(3,3)$-isogeny $\varphi : \mathcal{K} \to \mathcal{K}'$ and assume the coefficients $c_1, \dots, c_5$ have been computed. Given a point $P = (x_1 : x_2 : x_3 : x_4) \in \mathcal{K}$, the image $\varphi(P) = (x'_1 : x'_2 : x'_3 : x'_4)$ is given by

$$\begin{aligned}
x'_1 &:= x_1(c_1 x_1^2 + c_2 x_2^2 + c_3 x_3^2 + c_4 x_4^2) + c_5 x_2 x_3 x_4,\\
x'_2 &:= x_2(c_2 x_1^2 + c_1 x_2^2 + c_4 x_3^2 + c_3 x_4^2) + c_5 x_1 x_3 x_4,\\
x'_3 &:= x_3(c_3 x_1^2 + c_4 x_2^2 + c_1 x_3^2 + c_2 x_4^2) + c_5 x_1 x_2 x_4,\\
x'_4 &:= x_4(c_4 x_1^2 + c_3 x_2^2 + c_2 x_3^2 + c_1 x_4^2) + c_5 x_1 x_2 x_3.
\end{aligned}$$

The fundamental theta constants of the image surface $\mathcal{K}'$ can be computed in the same way, i.e., as $\varphi((a : b : c : d))$. Via Equation (9.2), we can then compute the constants E', F', G', H' defining the equation of the surface $\mathcal{K}'$.

9.3.3 Implementation

We implemented $(3,3)$-isogeny evaluation using the formulæ above. We give explicit operation counts for $\Bbbk = \mathbb{F}_q$, which will be necessary for our cryptographic application in Section 9.5. To optimise the computation, we implement the following algorithms:

1. TriplingConstantsFromThetas: given fundamental theta constants $(a\colon b\colon c\colon d)$, compute *tripling constants* consisting of:
   - their inverses $(1/a\colon 1/b\colon 1/c\colon 1/d)$;
   - their squares $(a^2\colon b^2\colon c^2\colon d^2)$;
   - squared dual theta constants $(A^2\colon B^2\colon C^2\colon D^2)$; and
   - their inverses $(1/A^2\colon 1/B^2\colon 1/C^2\colon 1/D^2)$.

   For $\Bbbk = \mathbb{F}_q$, this requires $12\mathbf{M}$, $4\mathbf{S}$, and $6\mathbf{a}$.

2. Compute33Coefficients: given coordinates of R, S and the tripling constants, compute the coefficients $c_1, \ldots, c_5$ defining the $(3,3)$-isogeny. When $\Bbbk = \mathbb{F}_q$, this requires $76\mathbf{M}$, $8\mathbf{S}$, and $97\mathbf{a}$.

3. Isogeny33Evaluate: given the coefficients $c_1, \ldots, c_5$, computes the image of a point $P \in \mathcal{K}$ under the corresponding $(3,3)$-isogeny (as explained in Section 9.3.2). For $\Bbbk = \mathbb{F}_q$, this requires $26\mathbf{M}$, $4\mathbf{S}$ and $16\mathbf{a}$.

4. ComputeImageThetas: given coefficients $c_1, \ldots, c_5$ and the tripling constants, compute the fundamental theta constants defining the image curve. For $\Bbbk = \mathbb{F}_q$, this requires $26\mathbf{M}$ and $16\mathbf{a}$.

Details of the implementation can be found in the accompanying code.

9.3.4 A note on $(5,5)$-isogenies on fast Kummers

Suppose we now have a $(5,5)$-subgroup of $\mathcal{J}[5]$, which induces a $(5,5)$-isogeny φ on the corresponding fast Kummer surface $\mathcal{K} = \mathcal{J}/\{\pm 1\}$ with kernel $G = \langle R, S\rangle$ for some $R, S \in \mathcal{K}[5]$, i.e., G is the image of the $(5,5)$-subgroup in $\mathcal{K}$.

Following the method in Section 9.2, we first compute the G-invariant quintic forms. Let $B^R_{i,j}$ and $B^S_{i,j}$ be as before (where now R, S are the 5-torsion points), and define

$$B^{2R}_{ij}(X_1, X_2, X_3, X_4) := B_{i,j}(X_1, X_2, X_3, X_4; 2R),$$
$$B^{2S}_{ij}(X_1, X_2, X_3, X_4) := B_{i,j}(X_1, X_2, X_3, X_4; 2S).$$

By Lemma 9.2.4, and following Flynn [157], the quintic forms invariant under translation by R are given by

$$\begin{aligned}F^R_{ijklm} := {} & X_iB^R_{jk}B^{2R}_{lm} + X_iB^R_{jl}B^{2R}_{km} + X_iB^R_{jm}B^{2R}_{kl} + \\ & X_iB^R_{kl}B^{2R}_{jm} + X_iB^R_{km}B^{2R}_{jl} + X_iB^R_{lm}B^{2R}_{jk} + \\ & X_jB^R_{ik}B^{2R}_{lm} + X_jB^R_{il}B^{2R}_{km} + X_jB^R_{im}B^{2R}_{kl} + \\ & X_jB^R_{kl}B^{2R}_{im} + X_jB^R_{km}B^{2R}_{il} + X_jB^R_{lm}B^{2R}_{ik} + \\ & X_kB^R_{ji}B^{2R}_{lm} + X_kB^R_{jl}B^{2R}_{im} + X_kB^R_{jm}B^{2R}_{il} + \\ & X_kB^R_{il}B^{2R}_{jm} + X_kB^R_{im}B^{2R}_{jl} + X_kB^R_{lm}B^{2R}_{ji} + \\ & X_lB^R_{jk}B^{2R}_{im} + X_lB^R_{ji}B^{2R}_{km} + X_lB^R_{jm}B^{2R}_{ki} + \\ & X_lB^R_{ki}B^{2R}_{jm} + X_lB^R_{km}B^{2R}_{ji} + X_lB^R_{im}B^{2R}_{jk} + \\ & X_mB^R_{jk}B^{2R}_{li} + X_mB^R_{jl}B^{2R}_{ki} + X_mB^R_{ji}B^{2R}_{kl} + \\ & X_mB^R_{kl}B^{2R}_{ji} + X_mB^R_{ki}B^{2R}_{jl} + X_mB^R_{li}B^{2R}_{jk},\end{aligned}$$

where $1 \leq i, j, k, l, m \leq 4$. We similarly define F^S_{ijklm}, the quintic forms invariant under translation-by-S.

Let $\mathcal{F}_R = \{F^R_{ijklm}\}$ and $\mathcal{F}_R = \{F^S_{ijklm}\}$. Working modulo the equation defining the Kummer surface $\mathcal{K}$, the quintic forms in $\mathcal{F}_R$ and $\mathcal{F}_S$ each generate a space of dimension 12, for which we choose a basis

$$\Big\{ F^R_{14444},\ F^R_{23333},\ F^R_{23334},\ F^R_{23344},\ F^R_{23444},\ F^R_{24444},\\ F^R_{33333},\ F^R_{33334}, F^R_{33344},\ F^R_{33444},\ F^R_{34444},\ F^R_{44444} \Big\},$$

and similarly for $\mathcal{F}_S$. These spaces intersect in a space of dimension 4, which gives a description of the $(5,5)$-isogeny. Finding the final scaling map to put the image into the correct form is left as future work.[5]

9.4 Generating (N^k, N^k)-subgroups

In the remaining two sections, we turn to building a hash function based on the $(3,3)$-isogenies derived in the previous section. The hash function will compute $(3^k, 3^k)$-isogenies as chains of $(3,3)$-isogenies, and will start each such chain by computing a $(3^k, 3^k)$-subgroup on a fast Kummer surface. This section describes how such (N^k, N^k)-subgroups can be computed (for prime number N and integer $k \geq 1$) in a way that is amenable to efficient and secure cryptographic implementations.

We do this in two steps: first, in Section 9.4.1 we show how to compute a *symplectic* basis for the N^k-torsion on the Jacobian $\mathcal{J}$, which we then push down to the corresponding fast Kummer surface $\mathcal{K} = \mathcal{J}/\{\pm 1\}$; and second, we use this basis to compute the generators R, S of the (N^k, N^k)-subgroup G using the *three-dimensional differential addition chain* introduced in Section 9.4.2.

[5]This has been resolved in follow-up joint work with Flynn [94].

9.4.1 Generating a symplectic basis on $\mathcal{K}$

We first compute a symplectic basis for the N^k-torsion on the Jacobian $\mathcal{J}$ of the genus-2 curve $\mathcal{C}_{\lambda,\mu,\nu}$ corresponding to $\mathcal{K}$. We compute this basis by generating N^k-torsion points on $\mathcal{J}$ until the N^k-Weil pairing condition given in Definition 9.1.1 is satisfied.[6] These points are then pushed down to $\mathcal{K}$ via

$$\begin{aligned}\pi\colon \mathcal{J} &\to \mathcal{K},\\ P &\mapsto \left(\mathtt{C}_{\mathtt{Inv}(0_{\mathcal{K}})} \circ \mathtt{H} \circ \mathtt{C}_{(\mathtt{Inv}\circ\mathtt{H}\circ\mathtt{S})(0_{\mathcal{K}})}\mathtt{S} \circ \mathtt{H} \circ \kappa\right)(P).\end{aligned}$$

Here, $\mathtt{C}$, $\mathtt{H}$, $\mathtt{S}$ and $\mathtt{Inv}$ are the standard Kummer operations defined in Section 9.1.4 and $\kappa : \mathcal{J} \to \mathcal{K}^{\mathrm{Sqr}}$ maps points in Mumford coordinates on $\mathcal{J}$ to the *squared* Kummer surface $\mathcal{K}^{\mathrm{Sqr}}$ as follows. For generic points $P = (x^2 + u_1x + u_0, v_1x + v_0) \in \mathcal{J}$, we have

$$\kappa : (x^2 + u_1x + u_0, v_1x + v_0) \mapsto (X_1 \colon X_2 \colon X_3 \colon X_4)$$

with

$$\begin{aligned}X_1 &= a^2(u_0(\mu - u_0)(\lambda + u_1 + \nu) - v_0^2), & X_2 &= b^2(u_0(\nu\lambda - u_0)(1 + u_1 + \mu) - v_0^2),\\ X_3 &= c^2(u_0(\nu - u_0)(\lambda + u_1 + \mu) - v_0^2), & X_4 &= d^2(u_0(\mu\lambda - u_0)(1 + u_1 + \nu) - v_0^2).\end{aligned}$$

For special points $P = (x - u_0, v_0)$, the map κ is defined by first adding a point of order 2 on $\mathcal{J}$. Indeed, by adding $(x - u_0', 0) \in \mathcal{J}[2]$ (with $u_0' \neq u_0$) we get the point

$$\left(x^2 - (u_0 + u_0')x + u_0u_0', \frac{v_0}{u_0 - u_0'}x - \frac{v_0u_0'}{u_0 - u_0'}\right),$$

to which we can apply κ. The translation is then undone by applying the action of the corresponding 2-torsion point on $\mathcal{K}^{\mathrm{Sqr}}$. Finally, $\kappa(0_{\mathcal{J}}) := (a^2 \colon b^2 \colon c^2 \colon d^2)$.

The map κ is due to Bisson, Cosset and Robert [36], while the subsequent operations that map from $\mathcal{K}^{\mathrm{Sqr}}$ to $\mathcal{K}$ appear in Renes and Smith [268, Section 4.3]. Note that the map π corresponds to a $(2, 2)$-isogeny, which will not affect the order of the basis points if N is coprime to 2. If, however, $2 \mid N$ then one must additionally check the order of the image points on $\mathcal{K}$.

Remark 9.4.1. For applications where N, k, and the domain Kummer $\mathcal{K}$ are fixed, the symplectic basis for $\mathcal{J}[N^k]$ can be computed as part of the set-up once and for all, and the image of these basis points (under π) can be hardcoded as system parameters. For instance, this will be the case for our cryptographic application in Section 9.5. In these scenarios, optimising the efficiency of the operations in this subsection is not a priority; the important goal is to optimise the efficiency of the *online* part of the (N^k, N^k)-subgroup generation procedure, which amounts to optimising the three-dimensional differential addition chain in the following subsection.

[6] In our implementation, we compute the N^k-Weil pairing of points on $\mathcal{J}[N^k]$ using `MAGMA`'s in-built functionality.

9.4.2 Three-dimensional differential addition chains

Let Q_1, Q_2, Q_3, $Q_4 \in \mathcal{K}$ be the images of a symplectic basis for $\mathcal{J}[N^k]$ under the map π described above. In this subsection we show how to use this basis to compute the two generators R and S of our (N^k, N^k)-subgroup.

As a first simplification, we restrict to (N^k, N^k)-subgroups with generators of the form

$$\begin{aligned} R &= Q_1 + [\alpha]Q_3 + [\beta]Q_4, \\ S &= Q_2 + [\beta]Q_3 + [\gamma]Q_4, \end{aligned} \tag{9.6}$$

where $\alpha, \beta, \gamma \in \mathbb{Z}/N^k\mathbb{Z}$. There are N^{3k} such (N^k, N^k)-subgroups (for example, see [209, Table 1]), and there are $O(N^{3k-1})$ subgroups that we lose by imposing this restriction [209, Def. 3]. In other words, at least half of the (N^k, N^k)-subgroups can be obtained with kernel generators of the form in Equation (9.6), so in a cryptographic context we lose at most one bit of security by simplifying in this manner.

The remainder of this subsection presents the 3DAC algorithm that allows us to compute the kernel generators R and S via Equation (9.6). Our task is to define an algorithm that computes $P_1 + [\beta]P_2 + [\gamma]P_3$ for given scalars $\beta, \gamma \in \mathbb{Z}/N^k\mathbb{Z}$ and for the points P_1, P_2 and P_3 on $\mathcal{K}$. The analogous computation on $\mathcal{J}$ could utilise a straightforward 3-way multi-exponentation algorithm, but on the Kummer surface we need a three-dimensional *differential addition chain.* Such an addition chain is only allowed to include the computation of the sum $Q + R$ if the difference $\pm(Q - R)$ has already been computed at a previous stage.

Three-dimensional differential addition chains have been studied previously by Rao [263] and more generally by Hutchinson and Karabina [185]. However, both of those works study the general scenario whereby three scalars are in play. Viewing Equation (9.6), we see that there is no scalar multiplication of the point P_1 in our case, which allows for some convenient simplifications. Moreover, the chain due to Rao [263] is non-uniform and the chain due to Hutchinson and Karabina [185] is not fixed length unless the input scalars are. Our algorithm 3DAC satisfies both of these properties regardless of the input scalars, making it secure for use in cryptographic applications, such as the hash function we present in Section 9.5. We remark that these properties would not be necessary for a hash function involving only public data, but for other cryptographic applications where inputs to the hash function (or, more broadly, the scalars β and γ) are secret, such as key derivation functions, these properties are an imperative first step towards protecting the secret data from side-channel attacks.

We derived the 3DAC chain by extending the two-dimensional differential addition chain due to Bernstein [24], the fastest known two-dimensional differential addition chain that is fixed length and uniform. Beyond the points P_1, P_2, $P_3 \in \mathcal{K}$, 3DAC also needs seven additional combinations of sums and/or differences of these three points. We specify the full ten-tuple of inputs as

$$\mathcal{D} := \big(P_1, P_2, P_3, P_2 + P_3, P_2 - P_3, P_1 - P_2, P_1 - P_3, [2](P_2 + P_3), P_1 + P_2 + P_3, P_1 - P_2 - P_3\big),$$

for points P_1, P_2, $P_3 \in \mathcal{K}$. In line with Remark 9.4.1, these additional sums and differences can be (pre)computed on $\mathcal{J}$ and their image in $\mathcal{K}$ specified as part of the system parameters.

The three main subroutines used in 3DAC are DBLTHRICEADD, an encoding algorithm ENCODE and an indexing algorithm IND. The algorithm DBLTHRICEADD is defined to compute the mapping

$$\big(P, Q, P-Q, R, S, R-S, T, U, T-U\big) \mapsto \big([2]P, P+Q, R+S, T+U\big)\,,$$

for points $P, Q, R, S, T, U \in \mathcal{K}$, using one pseudo-doubling and three pseudo-additions on the Kummer surface $\mathcal{K}$.

The encoding algorithm ENCODE takes as input two ℓ-bit scalars $\beta, \gamma \in \mathbb{Z}/3^k\mathbb{Z}$ and outputs a single bit b and four $(\ell-1)$-bit scalars b_i for $i = 0, 1, 2, 3$. The bit b determines one of two possible input combinations that is fed into a differential addition that kickstarts the chain (see Step 3 of Algorithm 9.3), after which the j-th bits (for $j = 1, \ldots, \ell$) of each of the four b_i determine one of 16 input permutations that is fed into the DBLTHRICEADD routine (see Step 10 of Algorithm 9.3).

Algorithm 9.1 ENCODE(β, γ):

Input: two ℓ-bit scalars $\beta = (\beta[\ell-1], \ldots, \beta[0])$ and $\gamma = (\gamma[\ell-1], \ldots, \gamma[0])$
Output: a bit $\mathsf{b} \in \{0, 1\}$, and four $(\ell-1)$-bit scalars (b_0, b_1, b_2, b_3)

```
1: $\mathsf{b} \leftarrow \beta[1]$
2: for $i = 1$ to $\ell - 1$ do
3:     $b_1[i] \leftarrow \beta[i] \oplus \beta[i+1]$
4:     $b_0[i] \leftarrow b_1[i] \oplus \gamma[i] \oplus \gamma[i+1]$
5:     $b_2[i] \leftarrow \beta[i+1] \oplus \gamma[i+1]$
6:     $b_3[i] \leftarrow \mathsf{b}$
7:     $\mathsf{b} \leftarrow b_1[i] \oplus (b_0[i] \oplus 1) \otimes \mathsf{b}$
8: end for
9: return $\mathsf{b}$, $(b_0, b_1, b_2, b_3)$
```

The indexing algorithm IND is used to choose one of four points as the last input to the DBLTHRICEADD algorithm.

Algorithm 9.2 IND(I):

Input: a 6-tuple of integers $I = (I_1, \ldots, I_6)$
Output: an integer index $\mathsf{ind} \in \{1, 2, 3, 4\}$

```
1: switch $([I_3 - I_1, I_4 - I_2])$
2: case $[-1, -1]$:  $\mathsf{ind} \leftarrow 1$
3: case $[\ \ 1, \ \ 1]$:  $\mathsf{ind} \leftarrow 2$
4: case $[\ \ 1, -1]$:  $\mathsf{ind} \leftarrow 3$
5: case $[-1, \ \ 1]$:  $\mathsf{ind} \leftarrow 4$
6: end switch
7: return ind
```

The full algorithm 3DAC is specified in Algorithm 9.3. Given the tuple $\mathcal{D}$ above, the two scalars $\beta, \gamma \in \mathbb{Z}/N^k\mathbb{Z}$, the length ℓ of the chain (i.e., the bitlength of N^k), the fundamental theta constants $0_{\mathcal{K}}$ of the fast Kummer surface $\mathcal{K}$ that we are working on, our 3DAC algorithm computes the point $P_1 + [\beta]P_2 + [\gamma]P_3$ on $\mathcal{K}$ using $3\ell - 2$ pseudo-additions and $\ell - 1$ pseudo-doublings on $\mathcal{K}$.

Algorithm 9.3 $\mathsf{3DAC}(\mathcal{D}, \beta, \gamma, \ell, 0_{\mathcal{K}})$:

Input: Scalars $\beta, \gamma \in \mathbb{Z}/3^k\mathbb{Z}$, the length of chain ℓ, fundamental theta constants $0_{\mathcal{K}}$, and a tuple $\mathcal{D}$ of points on $\mathcal{K}$. Let $\mathcal{D}_i$ be the i-th element in $\mathcal{D}$.
Output: $P_1 + [\beta]P_2 + [\gamma]P_3$ where P_1, P_2, P_3 are $\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3$, respectively.

```
1: initialise $P \leftarrow (\mathcal{D}_4, \mathcal{D}_8, \mathcal{D}_4, \mathcal{D}_9)$, $D \leftarrow (\mathcal{D}_2, \mathcal{D}_3, \mathcal{D}_4, \mathcal{D}_5)$, $\Delta \leftarrow (\mathcal{D}_1, \mathcal{D}_{10}, \mathcal{D}_6, \mathcal{D}_7)$
2: initialise $I \leftarrow (1, 1, 2, 2, 1, 1)$
3: $\mathsf{b}, (b_0, b_1, b_2, b_3) \leftarrow \mathsf{ENCODE}(\beta, \gamma)$
4: if $\mathsf{b} = 1$ then
5:     $(P_3, I_6) \leftarrow ((P_3, D_2, D_1), I_6 + 1)$
6: else
7:     $(P_3, I_5) \leftarrow ((P_3, D_1, D_2), I_5 + 1)$
8: end if
9: for $i = 1$ to $\ell - 1$ do
10:     switch $(b_0[i], b_1[i], b_2[i], b_3[i])$
11:     case $(0,0,0,0)$: $I \leftarrow (I_1 + I_3, I_2 + I_4, 2I_3, 2I_4, I_3 + I_5, I_4 + I_6)$
        $(P_2, P_1, P_3, P_4) \leftarrow \mathsf{DBLTHRICEADD}(P_2, P_1, D_3, P_3, P_2, D_2, P_2, P_4, \Delta_{\mathsf{IND}(I)})$
12:     case $(0,0,0,1)$: $I \leftarrow (I_1 + I_3, I_2 + I_4, 2I_3, 2I_4, I_3 + I_5, I_4 + I_6)$
        $(P_2, P_1, P_3, P_4) \leftarrow \mathsf{DBLTHRICEADD}(P_2, P_1, D_3, P_3, P_2, D_1, P_2, P_4, \Delta_{\mathsf{IND}(I)})$
13:     case $(0,0,1,0)$: $I \leftarrow (I_1 + I_3, I_2 + I_4, 2I_3, 2I_4, I_3 + I_5, I_4 + I_6)$
        $(P_2, P_1, P_3, P_4) \leftarrow \mathsf{DBLTHRICEADD}(P_2, P_1, D_4, P_3, P_2, D_2, P_2, P_4, \Delta_{\mathsf{IND}(I)})$
14:     case $(0,0,1,1)$: $I \leftarrow (I_1 + I_3, I_2 + I_4, 2I_3, 2I_4, I_3 + I_5, I_4 + I_6)$
        $(P_2, P_1, P_3, P_4) \leftarrow \mathsf{DBLTHRICEADD}(P_2, P_1, D_4, P_3, P_2, D_1, P_2, P_4, \Delta_{\mathsf{IND}(I)})$
15:     case $(0,1,0,0)$: $I \leftarrow (I_1 + I_3, I_2 + I_4, 2I_1, 2I_2, I_1 + I_5, I_2 + I_6)$
        $(P_2, P_1, P_3, P_4) \leftarrow \mathsf{DBLTHRICEADD}(P_1, P_2, D_3, P_3, P_1, D_2, P_1, P_4, \Delta_{\mathsf{IND}(I)})$
16:     case $(0,1,0,1)$: $I \leftarrow (I_1 + I_3, I_2 + I_4, 2I_1, 2I_2, I_1 + I_5, I_2 + I_6)$
        $(P_2, P_1, P_3, P_4) \leftarrow \mathsf{DBLTHRICEADD}(P_1, P_2, D_3, P_3, P_1, D_1, P_1, P_4, \Delta_{\mathsf{IND}(I)})$
17:     case $(0,1,1,0)$: $I \leftarrow (I_1 + I_3, I_2 + I_4, 2I_1, 2I_2, I_1 + I_5, I_2 + I_6)$
        $(P_2, P_1, P_3, P_4) \leftarrow \mathsf{DBLTHRICEADD}(P_1, P_2, D_4, P_3, P_1, D_2, P_1, P_4, \Delta_{\mathsf{IND}(I)})$
18:     case $(0,1,1,1)$: $I \leftarrow (I_1 + I_3, I_2 + I_4, 2I_1, 2I_2, I_1 + I_5, I_2 + I_6)$
        $(P_2, P_1, P_3, P_4) \leftarrow \mathsf{DBLTHRICEADD}(P_1, P_2, D_4, P_3, P_1, D_1, P_1, P_4, \Delta_{\mathsf{IND}(I)})$
19:     case $(1,0,0,0)$: $I \leftarrow (I_1 + I_3, I_2 + I_4, 2I_5, 2I_6, I_3 + I_5, I_4 + I_6)$
        $(P_2, P_3, P_1, P_4) \leftarrow \mathsf{DBLTHRICEADD}(P_3, P_2, D_2, P_1, P_2, D_3, P_3, P_4, \Delta_{\mathsf{IND}(I)})$
20:     case $(1,0,0,1)$: $I \leftarrow (I_1 + I_3, I_2 + I_4, 2I_5, 2I_6, I_1 + I_5, I_2 + I_6)$
        $(P_2, P_3, P_1, P_4) \leftarrow \mathsf{DBLTHRICEADD}(P_3, P_1, D_1, P_1, P_2, D_3, P_3, P_4, \Delta_{\mathsf{IND}(I)})$
21:     case $(1,0,1,0)$: $I \leftarrow (I_1 + I_3, I_2 + I_4, 2I_5, 2I_6, I_3 + I_5, I_4 + I_6)$
        $(P_2, P_3, P_1, P_4) \leftarrow \mathsf{DBLTHRICEADD}(P_3, P_2, D_2, P_1, P_2, D_4, P_3, P_4, \Delta_{\mathsf{IND}(I)})$
22:     case $(1,0,1,1)$: $I \leftarrow (I_1 + I_3, I_2 + I_4, 2I_5, 2I_6, I_1 + I_5, I_2 + I_6)$
        $(P_2, P_3, P_1, P_4) \leftarrow \mathsf{DBLTHRICEADD}(P_3, P_1, D_1, P_1, P_2, D_4, P_3, P_4, \Delta_{\mathsf{IND}(I)})$
23:     case $(1,1,0,0)$: $I \leftarrow (I_1 + I_3, I_2 + I_4, 2I_5, 2I_6, I_1 + I_5, I_2 + I_6)$
        $(P_2, P_3, P_1, P_4) \leftarrow \mathsf{DBLTHRICEADD}(P_3, P_1, D_2, P_1, P_2, D_3, P_3, P_4, \Delta_{\mathsf{IND}(I)})$
24:     case $(1,1,0,1)$: $I \leftarrow (I_1 + I_3, I_2 + I_4, 2I_5, 2I_6, I_3 + I_5, I_4 + I_6)$
        $(P_2, P_3, P_1, P_4) \leftarrow \mathsf{DBLTHRICEADD}(P_3, P_2, D_1, P_1, P_2, D_3, P_3, P_4, \Delta_{\mathsf{IND}(I)})$
25:     case $(1,1,1,0)$: $I \leftarrow (I_1 + I_3, I_2 + I_4, 2I_5, 2I_6, I_1 + I_5, I_2 + I_6)$
        $(P_2, P_3, P_1, P_4) \leftarrow \mathsf{DBLTHRICEADD}(P_3, P_1, D_2, P_1, P_2, D_4, P_3, P_4, \Delta_{\mathsf{IND}(I)})$
26:     case $(1,1,1,1)$: $I \leftarrow (I_1 + I_3, I_2 + I_4, 2I_5, 2I_6, I_3 + I_5, I_4 + I_6)$
        $(P_2, P_3, P_1, P_4) \leftarrow \mathsf{DBLTHRICEADD}(P_3, P_2, D_1, P_1, P_2, D_4, P_3, P_4, \Delta_{\mathsf{IND}(I)})$
27:     end switch
28: end for
29: return $P_4$
```

9.5 A hash function from $(3, 3)$-isogenies

Isogenies between *superspecial* Jacobians of genus-2 curves have been proposed for use in post-quantum isogeny-based cryptography (e.g., [72, 159]). We follow suit and henceforth restrict our attention superspecial Jacobians defined over $\bar{\Bbbk} = \overline{\mathbb{F}}_p$. Recall from Section 4.1 that every superspecial $\mathcal{J}/\overline{\mathbb{F}}_p$ is $\overline{\mathbb{F}}_p$-isomorphic to a Jacobian defined over $\mathbb{F}_{p^2}$. Similarly, the corresponding superspecial Kummer surface $\mathcal{K}/\overline{\mathbb{F}}_p$ is $\overline{\mathbb{F}}_p$-isomorphic to a Kummer surface with model defined over $\mathbb{F}_{p^2}$.

As an application to exhibit our algorithms, we construct a fundamental cryptographic primitive: a hash function. The first isogeny-based hash function was introduced by Charles, Goren and Lauter who use isogenies between supersingular elliptic curves [76]. The use of higher dimensional isogenies between superspecial Jacobians of genus-2 curves to construct a variant of the Charles–Goren–Lauter (CGL) hash function was previously explored by Castryck, Decru and Smith [72] using $(2, 2)$-isogenies. They argue that although the computation of higher dimensional isogenies is more expensive, breaking the security of the hash function requires $\widetilde{O}(p^{3/2})$ time, rather than $\widetilde{O}(p^{1/2})$ time as in the CGL hash function. Therefore, smaller parameters can be used to obtain the same security. Following this, Castryck and Decru [70] use multiradical formulae for $(3, 3)$-isogenies to construct such a hash function and obtain an asymptotic speed-up of around a factor of 9. Later work by Decru and Kunzweiler [130] construct a hash function using $(3, 3)$-isogenies between general Kummer surfaces by improving on the formulae given by Bruin, Flynn and Testa [58].

In this section, we describe a variant of the CGL hash function [76], called KuHash, that uses the formulae introduced in Section 9.3 to compute chains of $(3, 3)$-isogenies between fast Kummer surfaces. We obtain a speed-up of around $8 - 9$x, and around $32 - 34$x compared to the Castryck–Decru and Decru–Kunzweiler hash functions, respectively, for security levels $\lambda = 128, 192$ and 256.

9.5.1 Chains of $(3, 3)$-isogenies

We first present the Isogeny33Chain routine for computing chains of $(3, 3)$-isogenies. Though we use it as a building block for a hash function, we note that it can also be used in a variety of cryptographic applications. In particular, we have been careful to ensure that each component of the algorithm is amenable to constant-time cryptographic software.

Tripling algorithm. We start by presenting TPL, the algorithm to compute $[3]P$ from a point $P \in \mathcal{K}$ and the associated tripling constants (as defined in Section 9.3.3). This algorithm requires 26**M**, 12**S** and 32**a**.

Naïve strategies. Given a $(3^k, 3^k)$-subgroup $G = \langle R, S\rangle \subset \mathcal{K}[3^k]$, we use TPL and the algorithms from Section 9.3.3 to compute an isogeny with kernel $\langle R, S\rangle$ as a chain of $(3, 3)$-isogenies of length k. A naïve way of doing so is the following. Set $P_0 := R$, $Q_0 := S$ and $\mathcal{K}_0 := \mathcal{K}$, and then execute the following four steps for $i = 1$ to k:

1. Compute the tripling constants on $\mathcal{K}_{i-1}$ using TriplingConstantsFromThetas

Algorithm 9.4 $\mathsf{TPL}(P, \mathtt{TC})$:

Input: Point $P \in \mathcal{K}$ and tripling constants $\mathtt{TC}$ (with i-th entry denoted $\mathtt{TC}_i$)
Output: Point $Q \in \mathcal{K}$ where $Q = [3]P$.

```
1: $R \leftarrow \mathcal{S}(P)$
2: $R \leftarrow \mathcal{H}(R)$
3: $Q \leftarrow \mathcal{S}(R)$
4: $Q \leftarrow \mathcal{C}_{\mathtt{TC}_5}(Q)$
5: $Q \leftarrow \mathcal{H}(Q)$
6: $Q \leftarrow \mathcal{C}_{\mathtt{TC}_4}(Q)$
7: $Q \leftarrow \mathcal{S}(Q)$
8: $Q \leftarrow \mathcal{H}(Q)$
9: $S \leftarrow \mathcal{C}_{\mathtt{TC}_5}(R)$
10: $Q \leftarrow \mathcal{C}_S(Q)$
11: $Q \leftarrow \mathcal{H}(Q)$
12: $Q \leftarrow \mathcal{C}_{\mathcal{I}(P)}(Q)$
13: return $Q$
```

2. Compute 3-torsion points $(P_i, Q_i) := (3^{k-i}R, 3^{k-i}S)$ using $k - i$ repeated applications of TPL on R and S.

3. Compute the $(3,3)$-isogeny $\varphi_i : \mathcal{K}_{i-1} \to \mathcal{K}_i$ with kernel $\langle P_i, Q_i \rangle$, and the images of P_{i-1}, Q_{i-1} under this isogeny using Compute33Coefficients and Isogeny33Evaluate.

4. Compute the theta constants of the image $\mathcal{K}_i$ of φ_i using ComputeImageThetas.

The $(3^k, 3^k)$-isogeny with kernel G will be given by $\varphi_k \circ \cdots \circ \varphi_1 : \mathcal{K}_0 \to \mathcal{K}_k$.

Optimal strategies. A more efficient way to compute isogeny chains is to use *optimal strategies* [123]. These allow us to reduce the number of executions of TPL needed to compute the kernel at each step in the chain by storing intermediate points obtained during the triplings and pushing them through each isogeny. In our case, the cost of tripling is around 1.1x the cost of computing the image of a point under the isogeny, and so we shift the cost in this way to obtain the strategies (see [123] for further details). We give this algorithm in detail in Algorithm 9.5, and note that invoking the optimal strategies results in a 3.7–6.6x reduction of the cost to compute a chain of $(3,3)$-isogenies for the set of parameters we specify in Section 9.5.3.

9.5.2 A cryptographic hash function

We are now ready to present the hash function KuHash that uses chains of $(3,3)$-isogenies between fast Kummer surfaces. For a fixed security parameter λ and working over characteristic $p \approx 2^\lambda$, the hash function parses the message into three scalars $\alpha, \beta, \gamma \in \mathbb{Z}/3^k\mathbb{Z}$ which are fed into the 3DAC algorithm to compute a $(3^k, 3^k)$-subgroup $G = \langle R, S \rangle$. This is then used to compute the corresponding $(3^k, 3^k)$-isogeny $\varphi : \mathcal{K} \to \mathcal{K}'$, and the output of the hash function is the fundamental theta constants of the image surface $\mathcal{K}'$.

To optimise our hash function, we want to ensure that the $(3^k, 3^k)$-isogeny is $\mathbb{F}_{p^2}$-rational. Given a security parameter λ, we choose a suitably sized prime $p = 16f \cdot 3^k - 1 \approx 2^\lambda$ (the fact that

Algorithm 9.5 Isogeny33Chain($k, 0_\mathcal{K}, R, S,$ `strategy`):

Input: Fundamental theta constants $0_\mathcal{K} = (a\colon b\colon c\colon d)$ defining Kummer surface $\mathcal{K}$, generators $R, S \in \mathcal{K}$ of $(3^k, 3^k)$-subgroup for $k \geq 1$, and optimal strategy `strategy`.
Output: Fundamental theta constants defining image Kummer surface $\mathcal{K}'$ of $(3^k, 3^k)$-isogeny $\varphi : \mathcal{K} \to \mathcal{K}'$ with $\ker \varphi = \langle R, S \rangle$.

```
1:  for $e = k - 1$ to 1 do
2:      $P, Q = R, S$
3:      pts = [ ]
4:      inds = [ ]
5:      $i \leftarrow 0$
6:      TC ← TriplingConstantsFromThetas($0_\mathcal{K}$)
7:      while $i < k - e$ do
8:          Append $[R, S]$ to pts
9:          Append $i$ to inds
10:         $m \leftarrow$ strategy$[k - i - e + 1]$
11:         for $j = 1$ to $m$ do
12:             $R \leftarrow$ TPL($R$, TC)
13:             $S \leftarrow$ TPL($S$, TC)
14:         end for
15:         $i \leftarrow i + m$
16:     end while
17:     cs ← Compute33Coefficients($P, Q$, TC)
18:     $0_\mathcal{K} \leftarrow$ ComputingImageThetas(cs)
19:     for $[P_1, P_2]$ in pts do
20:         $P_1 \leftarrow$ Isogeny33Evaluate($P_1$, cs)
21:         $P_2 \leftarrow$ Isogeny33Evaluate($P_2$, cs)
22:     end for
23:     if pts not empty then
24:         $[R, S] \leftarrow$ pts$[-1]$
25:         $i \leftarrow$ inds$[-1]$
26:         Remove last element from pts and inds
27:     end if
28: end for
29: TC ← TriplingConstantsFromThetas($0_\mathcal{K}$)
30: cs ← Compute33Coefficients($R, S$, TC)
31: $0_\mathcal{K} \leftarrow$ ComputingImageThetas(cs)
32: return $0_\mathcal{K}$
```

$16 \mid p+1$ ensures rational 2-torsion), where f is a small cofactor, and take a small pseudo-random walk from the superspecial Jacobian of the curve $\mathcal{C}_0 : y^2 = x^6 + 1$ to arrive at our starting Jacobian $\mathcal{J} = \mathrm{Jac}(\mathcal{C})$. In our implementation, we used `MAGMA`'s inbuilt Richelot isogeny routine to take 20 $(2,2)$-isogenies away from $\mathcal{C}_0$. Since $\mathcal{J}(\mathbb{F}_{p^2}) \cong (\mathbb{Z}/(p+1)\mathbb{Z})^4$, the Rosenhain invariants of $\mathcal{C}$ are all rational in $\mathbb{F}_{p^2}$, and we use these to compute $0_{\mathcal{K}} = (a\colon b\colon c\colon d)$, i.e., the fundamental theta constants of the starting Kummer surface $\mathcal{K} = \mathcal{J}/\{\pm 1\}$, where $a, b, c, d \in \mathbb{F}_{p^2}$. Using the methods described in Section 9.4.1, we compute a symplectic basis for $\mathcal{J}[3^k]$, which is pushed down to our starting fast Kummer surface $\mathcal{K}$ via $\pi : \mathcal{J} \to \mathcal{K}$ together with the auxiliary sums and differences defined in Section 9.4.2 to obtain the two tuples of points $\mathcal{D}_R$ and $\mathcal{D}_S$. The setup routine outputs $\texttt{gens} = \{\mathcal{D}_R,\ \mathcal{D}_S\}$ and $\texttt{data} = \{k,\ \lceil \lambda/\log(3) \rceil,\ 0_{\mathcal{K}}\}$.

The hash function takes as input `data`, `gens` and a message $\texttt{msg} = (\alpha, \beta, \gamma)$, where $\alpha, \beta, \gamma \in \mathbb{Z}/3^k\mathbb{Z}$. In practice, the input message `msg` to the hash function would be a bit string, which would then be parsed into the scalars α, β, γ. The output of the hash function is a tuple of elements $\mathbb{F}_{p^2}$, namely the fundamental theta constants of the image Kummer surface $\mathcal{K}'$ under the $(3^k, 3^k)$-isogeny defined by scalars (α, β, γ). The output is of size $8\log(p)$ without normalising the theta constants (a', b', c', d'), and of size $6\log(p)$ with normalisation $(a'/d', b'/d', c'/d')$, which comes at a cost of one inversion and $3\mathbf{M}$. We fully specify KuHash in Algorithm 9.6.

We remark that, as the message must be parsed into three scalars lying in $\mathbb{Z}/3^k\mathbb{Z}$, our hash function requires an input message of length $3k\log(3)$ bits. To allow for arbitrary length messages, after each $(3^k, 3^k)$-isogeny is computed, we could, for example, sample a new symplectic basis on the image Kummer surface following Kunzweiler, Ti, and Weitkämper [210, Algorithm 1]. However, we emphasize that our main objective in presenting the hash function KuHash is to benchmark our algorithm Isogeny33Chain for computing chains of $(3,3)$-isogenies against others in the literature. We therefore restrict our implementation and experiments to messages of length $3k\log(3)$.

Algorithm 9.6 KuHash(`msg`, `data`, `gens`)

Input: A message `msg`, auxiliary data `data` and generators `gens` of $\mathcal{K}[3^k]$.
Output: Fundamental theta constants (a', b', c', d') of image Kummer surface $\mathcal{K}'$

```
1: Parse msg as $\alpha, \beta, \gamma$.
2: Parse data as $k, \ell, \mathcal{O}_{\mathcal{K}}$.
3: Parse gens as two sets $\mathcal{D}_R, \mathcal{D}_S$ (see Section 9.4.2).
4: $R \leftarrow \mathsf{3DAC}(\mathcal{D}_R, \alpha, \beta, \ell, 0_{\mathcal{K}})$
5: $S \leftarrow \mathsf{3DAC}(\mathcal{D}_S, \beta, \gamma, \ell, 0_{\mathcal{K}})$
6: $(a'\colon b\colon c'\colon d') \leftarrow \mathsf{Isogeny33Chain}(k, 0_{\mathcal{K}}, R, S)$
7: return $(a', b', c', d')$
```

9.5.3 Implementation

We implement KuHash and give parameters for security levels $\lambda = 128, 192$, and 256.

Security. Let λ be the security parameter and $p \approx 2^\lambda$. We follow the discussion in [72, §7.4] to determine the security of the hash function KuHash. In particular, the security of our hash function is not affected by taking $N = 3$ rather than $N = 2$, and as a result the security of our hash function relies on similar problems, namely Problem 9.5.1 and Problem 9.5.2 below.

Problem 9.5.1. Given two superspecial genus-2 curves $\mathcal{C}_1, \mathcal{C}_2$ defined over $\mathbb{F}_{p^2}$ find a $(3^k, 3^k)$-isogeny between $\mathrm{Jac}(\mathcal{C}_1)$ and $\mathrm{Jac}(\mathcal{C}_2)$.

Problem 9.5.2. Given a superspecial genus-2 curve $\mathcal{C}_1$ defined over $\mathbb{F}_{p^2}$, find

- a curve $\mathcal{C}_2$ and a $(3^k, 3^k)$-isogeny $\mathrm{Jac}(\mathcal{C}_1) \to \mathrm{Jac}(\mathcal{C}_2)$,
- a curve $\mathcal{C}_2'$ and a $(3^{k'}, 3^{k'})$-isogeny $\mathrm{Jac}(\mathcal{C}_1) \to \mathrm{Jac}(\mathcal{C}_2')$,

such that $\mathrm{Jac}(\mathcal{C}_2)$ and $\mathrm{Jac}(\mathcal{C}_2')$ are $\overline{\mathbb{F}}_p$-isomorphic. Here, we can have $k = k'$, but the kernels of the corresponding isogenies must be different.

Previous works take the general Pollard-ρ attack to be the best classical attack against these problems, which runs in $\widetilde{O}(p^{3/2})$. We take a more conservative approach and consider the Costello–Smith algorithm [107] to be the best classical attack, which runs in $\widetilde{O}(p)$. This attack is described in detail in Section 4.3.2 of Part I of this thesis, and its concrete security is the focus of Chapter 8. We further recall that the best quantum attack is also due to Costello and Smith, and is based on Grover's claw-finding algorithm running in $\widetilde{O}(p^{1/2})$. Considering these attacks, we obtain the following parameters:

- $\lambda = 128 \;:\; p = 5 \cdot 2^4 \cdot 3^k - 1$ with $k = 75$;
- $\lambda = 192 \;:\; p = 37 \cdot 2^4 \cdot 3^k - 1$ with $k = 115$;
- $\lambda = 256 \;:\; p = 11 \cdot 2^4 \cdot 3^k - 1$ with $k = 154$.

Cost Metric. To benchmark KuHash, we count $\mathbb{F}_p$-operations. Indeed, our implementation in Python/SageMath will call underlying $\mathbb{F}_p$-operations to compute $\mathbb{F}_{p^2}$-operations. For simplicity, our cost metric will take $\mathbf{M} = \mathbf{S}$ and ignore $\mathbf{a}$, as additions have only a very minor impact on performance. Note that it is relatively straightforward to convert this cost into a more fine-grained metric (e.g., bit operations, cycle counts, etc.).

Results. We ran KuHash in SageMath version 10.1 using Python 3.11.1 and record the cost, as per the cost metric above, averaging over 100 random inputs for each prime size. We present the results in Table 9.1. Taking the $\mathbb{F}_{p^2}$-operation count from [130, §3.2] and using our cost metric, the cost of computing the coefficients of Decru and Kunzweiler's $(3,3)$-isogeny is 6702 (assuming 1 $\mathbb{F}_{p^2}$-multiplication is equivalent to 3 $\mathbb{F}_p$-multiplications). We use this to obtain a lower bound on the cost of the Decru–Kunzweiler hash function. Though this is a lower bound, we see in Table 9.1 that the cost already exceeds the total cost of KuHash.

The codebase for the other $(3,3)$-isogeny CGL hash variant by Castryck and Decru [70] uses Gröbner basis calculations at each step of the $(3,3)$-isogeny chain, which is not realistic to convert to a fixed number of $\mathbb{F}_p$-operations. Therefore, for fair comparison, we ran all three hash functions in `MAGMA` V2.25-6 on Intel(R) Core™ i7-1065G7 CPU @ 1.30GHx $\times$ 8 with 15.4 GiB memory, and record the time taken to run the hash functions for the different λ in Table 9.1. We again average over 100 random inputs for each prime size.

	λ	Message Length	Cost	Time (s)	Time per input bit (ms)
KuHash	128	$225 \log(3)$	177956	0.18	0.50
	192	$345 \log(3)$	286636	0.29	0.53
	256	$462 \log(3)$	396942	0.53	0.72
[130]	128	$240 \log(3)$	> 536160	5.99	15.75
	171	$315 \log(3)$	> 703710	10.16	20.35
	256	$477 \log(3)$	> 1065618	18.29	24.19
[70]	128	357	-	1.51	4.23
	171	547	-	2.82	5.16
	256	732	-	4.81	6.57

Table 9.1: Comparison of cost using cost metric and time taken to run KuHash and hash functions in [70] and [130]. All results are averaged over 100 runs with random inputs. We remark that the cost of KuHash is the same for all runs because it is uniform.

Comparing the time taken per input bit for $\lambda = 128$ and 256, we observe a speed-up of around 8 – 9x compared to the Castryck–Decru hash function, and around 32 – 34x compared to the Decru–Kunzweiler hash function. For a precise comparison between implementations, however, exact $\mathbb{F}_p$-operation counts of the Castryck–Decru and Decru–Kunzweiler hash functions are required. We note that an advantage of the algorithms developed with our approach is that we do not rely on in-built functionality and all our algorithms are uniform. Therefore, we are able to give precise $\mathbb{F}_p$-operation counts for KuHash.

Remark 9.5.3. Since the work in this chapter was finalised, Kunzweiler, Maino, Moriya, Petit, Pope, Robert, Stopar, and Ti [208] give explicit descriptions for radical 2-isogenies in dimensions one, two, and three using theta coordinates. As an application, these formulæ are used to construct different versions of the CGL hash function [76], with an accompanying Rust implementation. In light of this new research, it would be interesting to understand how KuHash compares to the various hash functions presented in [208].

Part IV

Accelerating SQIsign

Overview

In recent years, another type of post-quantum cryptography based on lattices has developed practical digital signature schemes that will be standardized by NIST [140, 161]. Lattice-based signatures provide fast signing and verification, but have larger key and signature sizes than were previously acceptable in pre-quantum signatures. For applications where the communication cost must remain as low as possible, lattice-based schemes may not be a viable option. Due to this, NIST is looking to standardise other signature schemes with desirable properties such as smaller combined public key and signature sizes for a smooth transition to a post-quantum world [310].

In the final part of this thesis, we focus on a potential solution to this: SQIsign [125]. SQIsign is the only isogeny-based candidate submitted to NIST's new call for signatures, and boasts the smallest combined size of the signature and the public key, comparable to those in pre-quantum elliptic curve signatures [193, 195]. Its main disadvantage, however, is its relative inefficiency compared to lattice-based schemes. We focus on accelerating the signing and verification routines in SQIsign. In Chapter 10, we present a novel method for finding SQIsign-friendly parameters, suited for obtaining faster signing without a large degradation in verification time. In this chapter, we assume that elliptic curve operations take place over $\mathbb{F}_{p^2}$. By adapting the signing procedure to allow for arithmetic over an extension field of $\mathbb{F}_{p^2}$, in Chapter 11 we present a new variant of SQIsign called AprèsSQI, which aims to maximise the performance of verification. Throughout this part of the thesis, we focus on instantiations of SQIsign that provide NIST Level I, III and IV security, as defined in Table 1 in the introduction.

Chapter 10

Cryptographic Smooth Neighbours

SQIsign parameters must satisfy many requirements in order for it to be secure and practical. In this chapter, we present and optimise a new method for finding SQIsign-friendly primes. The primes p we present are such that $p^2 - 1$ is *smooth*, however only divisible by moderately sized powers of 2 and 3. As such, they are more suited towards fast signing, where the smoothness bound has the largest impact. This chapter is based on the paper

Cryptographic Smooth Neighbors

published at ASIACRYPT 2023 [60], which is joint work with Giacomo Bruno, Craig Costello, Jonathan Komada Eriksen, Michael Meyer, Michael Naehrig, and Bruno Sterner. Apart from minor editorial edits and merging the appendix into the main body, the chapter presents the paper as published.

In light of recent work. We briefly remark on the contributions of this work in light of other work that has been made public since its publication. New high-dimensional variants of SQIsign, namely SQIsignHD [116] and SQIsign2D [17, 142, 247], have removed the restrictions on the primes needed to instantiate the signature scheme. Indeed, they use primes of the form $p = 2^e f - 1$, where $e, f \in \mathbb{N}$. However, even with these new higher-dimensional techniques available, the recent public key encryption (PKE) scheme constructed with the POKE framework [16] requires very special primes. More specifically, it requires primes $p \approx 2^{2\lambda}$ such that $2^e T \mid p^2 - 1$ for $e \in \mathbb{N}$, T is some smooth odd integer, and $p - 1$ has a prime divisor $x \approx 2^{\lambda/2}$. Recent joint work with Eriksen, Meyer, and Rodríguez-Henríquez [277] shows that similar prime-finding techniques can find POKE-friendly primes. This highlights that these new variants do not make finding special primes obsolete. Indeed, they were first explored in the context of B-SIDH, before being applied to SQIsign. Further to this, the picture in isogeny-based cryptography changes quickly with new developments. As such, we may see improvements that lead us back to one-dimensional variants, or even two-dimensional variants that require similar prime shapes.

Introduction

In recent years the tantalising problem of finding two large, consecutive, smooth integers has emerged in the context of instantiating efficient isogeny-based public key cryptosystems. Though the problem was initially motivated in the context of key exchange [99], a wave of polynomial time attacks [67, 227, 271] has broken the isogeny-based key exchange scheme SIDH [123], leaving post-quantum signatures as the most compelling cryptographic application of isogenies at present. In terms of practical potential, the leading isogeny-based signature scheme is SQIsign [125]; it boasts the smallest public keys and signatures of all post-quantum signature schemes, at the price of a signing algorithm that is orders of magnitude slower than its post-quantum counterparts. Finding secure parameters for SQIsign is related to the twin smooth problem mentioned above. Indeed, SQIsign is instantiated over large primes p such that $p^2 - 1$ is divisible by a large, B-smooth factor. If, for example, we find B-smooth twins r and $r + 1$ whose sum is a prime $p = 2r + 1$, then $p^2 - 1$ is immediately B-smooth. A large contributing factor to the overall efficiency of the protocol is the smoothness bound, B, of the rational torsion used in isogeny computations. This bound corresponds to the degree of the largest prime-degree isogeny computed in the protocol, for which the fastest algorithm runs in $\widetilde{O}(\sqrt{B})$ field operations [27]. Part of the reason for SQIsign's performance drawback is that the problem of finding parameters with small B is difficult: the fastest implementation to date targets security comparable to NIST Level I [309, §4.A] and has $B = 3923$ [126]. Additionally, methods for finding efficient SQIsign parameters have to date not been able to obtain suitable primes reaching NIST Level III and V security.

The CHM algorithm. In this work we introduce new ways of finding large twin smooth instances based on the Conrey–Holmstrom–McLaughlin (CHM) "Smooth neighbors" algorithm [86]. For a fixed smoothness bound B, the CHM algorithm starts with the set of integers $S = \{1, 2, \ldots, B-1\}$ representing the smooth neighbours $(1, 2), (2, 3), \ldots, (B - 1, B)$, and recursively grows this set by constructing new twin smooth integers from unordered pairs in $S \times S$ until a full pass over all such pairs finds no new twins, at which point the algorithm terminates. Although the CHM algorithm is not guaranteed to find the set of all B-smooth twins, for moderate values of B it converges with the set S containing *almost all* such twins. The crucial advantage is that, unlike the algorithm of Lehmer [214] that exhaustively solves $2^{\pi(B)}$ Pell equations to guarantee the full set of B-smooth twins, the CHM algorithm terminates much more rapidly. For example, in 2011 Luca and Najman [225] used Lehmer's approach with $B = 100$ to compute the full set of 13,374 twin smooths in 15 days (on a quad-core 2.66 GHz processor) by solving $2^{\pi(B)} = 2^{25}$ Pell equations, the solutions of which can have as many as 10^{10^6} decimal digits. The largest pair of 100-smooth twins they found were the 58-bit integers

$$\begin{aligned}
166055401586083680 &= 2^5 \cdot 3^3 \cdot 5 \cdot 11^3 \cdot 23 \cdot 43 \cdot 59 \cdot 67 \cdot 83 \cdot 89, \text{ and} \\
166055401586083681 &= 7^2 \cdot 17^{10} \cdot 41^2.
\end{aligned}$$

In 2012, Conrey, Holmstrom and McLaughlin ran their algorithm on a similar machine to find 13,333 (i.e. all but 41) of these twins in 20 minutes [86]. Subsequently, they set $B = 200$ and found a list of 346,192 twin smooths in about 2 weeks, the largest of which were the 79-bit integers

$$\begin{aligned}
589864439608716991201560 &= 2^3 \cdot 3^3 \cdot 5 \cdot 7^2 \cdot 11^2 \cdot 17 \cdot 31 \cdot 59^2 \cdot 83 \cdot 139^2 \\
&\quad \cdot 173 \cdot 181, \text{ and} \\
589864439608716991201561 &= 13^2 \cdot 113^2 \cdot 127^2 \cdot 137^2 \cdot 151^2 \cdot 199^2.
\end{aligned}$$

Exhausting the full set of 200-smooth twins would have required solving $2^{\pi(200)} = 2^{46}$ Pell equations, which is pushing the limit of what is currently computationally feasible. The largest run of Lehmer's algorithm reported in the literature used $B = 113$ [99, §5.3], which required solving 2^{30}

Pell equations and a significant parallelised computation that ran over several weeks. The largest set of 113-smooth twins found during that computation were the 75-bit integers

$$\begin{aligned}
19316158377073923834000 &= 2^4 \cdot 3^6 \cdot 5^3 \cdot 7 \cdot 23^2 \cdot 29 \cdot 47 \cdot 59 \cdot 61 \cdot 73 \cdot 97 \cdot 103, \\
19316158377073923834001 &= 13^2 \cdot 31^2 \cdot 37^2 \cdot 43^4 \cdot 71^4.
\end{aligned}$$

Remark 10.0.1. The above examples illustrate some important phenomena that are worth pointing out before we move forward. Observe that, in the first and third examples, the largest prime not exceeding B is not found in the factors of the largest twins. The largest 89-smooth twins are the same as the largest 97-smooth twins, and the largest 103-smooth twins are the same as the largest 113-smooth twins. In other words, increasing B to include more primes necessarily increases the size of the set of B-smooth twins, but it does not mean we will find any new, larger twins. This trend highlights part of the difficulty we face in trying to find optimally smooth parameters of cryptographic size: increasing the smoothness bound B makes the size of the set of twins grow rapidly, but the growth of the largest twins we find is typically painstakingly slow. The set of 100-smooth twins has cardinality 13,374, with the largest pair being 58 bits; increasing B to 200 gives a set of cardinality (at least) 345,192, but the largest pair has only grown to be 79 bits. In fact, most of this jump in the bitlength of the largest twins occurs when increasing $B = 97$ (58 bits) to include two more primes with $B = 103$ (76 bits). Including the 19 additional primes up to 199 only increases the bitlength of largest twins with $B = 199$ by 3 (79 bits), and this is indicative of what we observe when B is increased even further.

Contributions

We give an optimised implementation of CHM that allows us to run the algorithm for much larger values of B in order to find larger sized twins. For example, the original CHM paper reported that the full algorithm with $B = 200$ terminated in approximately 2 weeks; our implementation did the same computation in around 943 seconds on a laptop. Increasing the smoothness bound to $B = 547$, our implementation converged with a set of 82,026,426 pairs of B-smooth twins, the largest of which are the 122-bit pair $(r, r+1)$ with

$$\begin{aligned}
r &= 5^4 \cdot 7 \cdot 13^2 \cdot 17^2 \cdot 19 \cdot 29 \cdot 41 \cdot 109 \cdot 163 \cdot 173 \cdot 239 \cdot 241^2 \cdot 271 \cdot 283 \\
&\quad \cdot 499 \cdot 509, \qquad \text{and} \\
r+1 &= 2^8 \cdot 3^2 \cdot 31^2 \cdot 43^2 \cdot 47^2 \cdot 83^2 \cdot 103^2 \cdot 311^2 \cdot 479^2 \cdot 523^2.
\end{aligned} \tag{10.1}$$

Although it remains infeasible to increase B to the point where the twins found through CHM are large enough to be used out-of-the-box in isogeny-based schemes (i.e. close to 2^{256}), we are able to combine the larger twins found through CHM with techniques from the literature in order to find much smoother sets of SQIsign parameters. In this case we are aided by the requirements for SQIsign, which permit us to relax the size of the smooth factor that divides $p^2 - 1$. The current state-of-the-art instantiation [126] uses primes p such that

$$\ell^f \cdot T \mid (p^2 - 1),$$

where ℓ is a small prime (typically $\ell = 2$), where f is as large as possible, and where $T \approx p^{5/4}$ is both coprime to ℓ and B-smooth. For example, the original SQIsign implementation [125] used a 256-bit prime p such that

$$p^2 - 1 = 2^{34} \cdot T_{1879} \cdot R,$$

where T_{1879} is an odd 334-bit integer[1] whose largest prime factor is $B = 1879$, and R is the *rough* factor; a 144-bit integer containing no prime factors less than or equal to B. As another example, De Feo, Leroux, Longa, and Wesolowski [126, §5] instead use a 254-bit prime p with

$$p^2 - 1 = 2^{66} \cdot T_{3923} \cdot R,$$

where T_{3923} is an odd 334-bit integer whose largest prime factor is $B = 3923$, and where all of R's prime factors again exceed B.

During the search mentioned above that found the record 547-smooth twins in Equation (10.1), over 82 million other pairs of smaller sized twins were found. One such pair was the 63-bit twins $(r - 1, r)$ with $r = 8077251317941145600$. Taking $p = 2r^4 - 1$ gives a 253-bit prime p such that

$$p^2 - 1 = 2^{49} \cdot T_{479} \cdot R,$$

where T_{479} is an odd 328-bit integer that is 479-smooth. This represents a significant improvement in smoothness over the T values obtained in [125] and [126]. Although the smoothness of T is not the only factor governing the efficiency of the scheme, our analysis in Section 10.5 suggests that the parameters found in this paper are interesting alternatives to those currently found in SQIsign implementations, giving instantiations with a significantly lower expected signing cost, but with a modest increase in verification cost.

Just as we transformed a pair of 85-bit twins into a 255-bit prime by taking $p = 2r^3 - 1$, we combine the use of twins found with CHM and primes of the form $p = 2r^n - 1$ with $n \geq 3$ to obtain several SQIsign-friendly primes that target higher security levels. For example, with some 64-bit twins $(r, r + 1)$ found through CHM, we give a 382-bit prime $p = 2r^6 - 1$ such that $p^2 - 1 = 2^{80} \cdot T_{10243} \cdot R$, where T is an odd 495-bit integer that is 10243-smooth; this prime would be suitable for SQIsign signatures geared towards NIST Level III security. As another example, with some 85-bit twins $(r, r+1)$, we give a 508-bit prime $p = 2r^6 - 1$ such that $p^2 - 1 = 2^{86} \cdot T_{150151} \cdot R$, where T is a 639-bit integer that is 150151-smooth; this prime would be suitable for SQIsign signatures targeting NIST Level V security.

Remark 10.0.2. A recent paper by Eriksen, Panny, Sotáková, and Veroni [150] showed that computing the constructive Deuring correspondence, needed to construct SQIsign signatures, is feasible without choosing a specific characteristic p beforehand. However, the paper further confirms (comparing [150, Figure 3] with [150, Table 2]) that the efficiency of this computation depends heavily on the factorisation of $p^2 - 1$ (or more generally $p^k - 1$ for small k). In a setting that allows us to freely choose a fixed characteristic p, for instance in the SQIsign setting, it is clear that one should choose p carefully for optimal performance.

[1]The initial SQIsign requirements [125] had $T \approx p^{3/2}$, but T_{1879} corresponds to the new requirements.

Remark 10.0.3. Another recent work introduces SQIsignHD [116], a variant of SQIsign in higher dimensions. Although the signature generation could be significantly faster in SQIsignHD, the verification algorithm requires computing 4-dimensional isogenies. Since the research of implementing practical 4-dimensional isogenies has mainly only begun since the SIDH attacks, there is no implementation of SQIsignHD verification yet.[2] While breakthroughs in this area of research could change the picture of the field, it remains unclear whether the verification algorithm can be implemented efficiently enough to consider SQIsignHD for practical applications, or to reach similar performance as SQIsign verification.

Availability of software

Our implementation of the CHM algorithm is written in C/C++ and is found at

https://github.com/GiacomoBruno/TwinsmoothSearcher.

Outline

Section 10.1 reviews prior methods for generating large instances of twin smooths. In Section 10.2, we recall the CHM algorithm and give a generalisation of it that may be of independent interest. Section 10.3 details our implementation of the CHM algorithm and presents a number of optimisations that allowed us to run it for much larger values of B. In Section 10.4, we discuss the combination of CHM with primes of the form $p = 2x^n - 1$ to give estimates on the probabilities of finding SQIsign parameters at various security levels. Section 10.5 presents our results, giving record-sized twin smooth instances and dozens of SQIsign-friendly primes that target NIST's security levels I, III, and V.

10.1 Preliminaries and prior methods

We start by fixing some definitions and terminology.

Definition 10.1.1. A positive integer n is called *B-smooth* for some real number $B > 0$ if all prime divisors of n are at most B. An integer n *generates a B-smooth value* of a polynomial $f(X)$ if $f(n)$ is B-smooth. In this case we call n a B-smooth value of $f(X)$. We call two consecutive integers *B-smooth twins* if their product is B-smooth. An integer n is called *B-rough* if all of its prime factors exceed B.

We now review prior methods of searching for twin smooth integers by following the descriptions of the three algorithms reviewed in [105, §2] and including the method introduced in [105] itself.

Solving Pell equations. Fix B, let $\{2, 3, \ldots, q\}$ be the set of primes up to B with cardinality $\pi(B)$,[3] and consider the B-smooth twins $(r, r + 1)$. Let $x = 2r + 1$, so that $x - 1$ and $x + 1$ are

[2] Since the publication of this work, Dartois [115] has implemented four-dimensional $(2, 2, 2, 2)$-isogenies in the theta model in Python/SageMath. We see that computing a chain of $(2, 2, 2, 2)$-isogenies is $16 - 18$x slower than computing a chain of 2-isogenies of the same length [115, Table 2 and 3].

[3] In a slight clash of notation with previous chapters, here π denotes the prime-counting function: $\pi(x)$ counts the number of prime numbers less than or equal to $x \in \mathbb{N}$.

also B-smooth, and let D be the squarefree part of their product $(x-1)(x+1)$, i.e. $x^2-1=Dy^2$ for some $y \in \mathbb{Z}$. It follows that Dy^2 is B-smooth, which means that

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

with $\alpha_i \in \{0,1\}$ for $i = 2,3,\dots,q$. For each of the $2^{\pi(B)}$ squarefree possibilities for D, Størmer [300] 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 twins.

The largest pair of 2-smooth integers is $(1,2)$, the largest pair of 3-smooth integers is $(8,9)$, and the largest pair of 5-smooth integers is $(80,81)$. Unfortunately, solving $2^{\pi(B)}$ Pell equations becomes infeasible before the size of the twins we find is large enough (i.e. exceeds 2^{200}) for our purposes. As we saw in the introduction, Costello [99] reports that with $B = 113$ the largest twins $(r, r+1)$ found upon solving all 2^{30} Pell equations have $r = 19316158377073923834000 \approx 2^{75}$.

The extended Euclidean algorithm. The most naïve way of searching for twin smooth integers is to compute B-smooth numbers r until either $r-1$ or $r+1$ also turns out to be B-smooth. A much better method [99, 125] is to instead choose two coprime B-smooth numbers α and β that are both of size roughly the square root of the target size of r and $r+1$. On input of α and β, 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 reason this performs much better than the naïve method above is that $s \cdot t$ with $s \approx t$ is much more likely to be B-smooth than a random integer of similar size.

Searching with $r = x^n - 1$. A number of works [99, 125, 126] have found performant parameters by searching for twins of the form $(r, r+1) = (x^n - 1, x^n)$, for relatively small $n \in \mathbb{Z}$. For example, suppose we are searching for b-bit twins $(r, r+1)$, and we take $n = 4$ so that $r = (x^2+1)(x-1)(x+1)$. Instead of searching for two b-bit numbers that are smooth, we are now searching for three smooth $(b/4)$-bit numbers (i.e. $x-1$, x, and $x+1$) and one smooth $(b/2)$-bit number, which increases the probability of success (see [105]).

Searching with PTE solutions. The approach taken by Costello, Meyer, and Naehrig [105] can be viewed as an extension of the method above, where the important 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 (like the irreducible quadratic x^2+1 above) significantly hampers the probability that values of $x^n - 1 \in \mathbb{Z}$ are smooth. Instead, the algorithm in [105] takes $(r, r+1) = (f(x), g(x))$, where $f(x)$ and $g(x)$ are both of degree n and are composed entirely of linear factors. This boosts the success probability again, but one of the difficulties facing this method is that polynomials $f(x)$ and $g(x)$ that differ by a constant and are completely split are difficult to construct for $n \geq 4$. Fortunately, instances of these polynomials existed in the literature since they can be trivially constructed using solutions to the Prouhet-Tarry-Escott (PTE) problem (see [105]).

10.2 The CHM algorithm

We first recall the Conrey, Holmstrom, and McLaughlin (CHM) algorithm [86], a remarkably simple algorithm that generates twin smooth integers (or *smooth neighbours* as they are called in [86]), i.e., smooth values of the polynomial $X(X+1)$. We then present a generalisation of this algorithm, which generates smooth values of any monic quadratic polynomial. The algorithm generalises the CHM algorithm, as well as another algorithm in the literature by Conrey and Holmstrom [87], which generates smooth values of the polynomial X^2+1. In the end, we are primarily interested in the CHM algorithm, but present the generalised algorithm here, as it may be of independent interest.

10.2.1 Finding Smooth Twins with the CHM Algorithm

Conrey, Holmstrom, and McLaughlin [86] present the following algorithm for producing many B-smooth values of $X(X+1)$. It starts with the initial set

$$S^{(0)} = \{1, 2, \ldots, B-1\}$$

of all integers less than B, representing the B-smooth twins $(1,2)$, $(2,3)$, …, $(B-1,B)$. Next, it iteratively passes through all pairs of distinct $r, s \in S^{(0)}, r < s$ and computes

$$\frac{t}{t'} = \frac{r}{r+1} \cdot \frac{s+1}{s},$$

writing $\frac{t}{t'}$ in lowest terms. If $t' = t+1$, then clearly t also represents a twin smooth pair. The next set $S^{(1)}$ is formed as the union of $S^{(0)}$ and the set of all solutions t such that $t' = t+1$. Now the algorithm iterates through all pairs of distinct $r, s \in S^{(1)}$ to form $S^{(2)}$ and so on. We call the process of obtaining $S^{(d)}$ from $S^{(d-1)}$ the d-th CHM iteration. Once $S^{(d)} = S^{(d-1)}$, the algorithm terminates.

Example 10.2.1. We illustrate the algorithm for $B = 5$, i.e. with the goal to generate 5-smooth twin integers. The starting set is

$$S^{(0)} = \{1, 2, 3, 4\}.$$

Going through all pairs $(r, s) \in S^{(0)}$ with $r < s$, we see that the only ones that yield a new twin smooth pair $(t, t+1)$ via Equation (10.2) with t not already in $S^{(0)}$ are $(2,3)$, $(2,4)$ and $(3,4)$, namely,

$$\frac{2}{2+1} \cdot \frac{3+1}{3} = \frac{8}{9}, \quad \frac{2}{2+1} \cdot \frac{4+1}{4} = \frac{5}{6}, \quad \text{and} \quad \frac{3}{3+1} \cdot \frac{4+1}{4} = \frac{15}{16}.$$

Hence, we add 5, 8 and 15 to get the next set as

$$S^{(1)} = \{1, 2, 3, 4, 5, 8, 15\}.$$

The second and third CHM iterations give

$$S^{(2)} = \{1, 2, 3, 4, 5, 8, 9, 15, 24\} \text{ and } S^{(3)} = \{1, 2, 3, 4, 5, 8, 9, 15, 24, 80\}.$$

The fourth iteration does not produce any new numbers, i.e. we have $S^{(4)} = S^{(3)}$, the algorithm terminates here and returns $S^{(3)}$. This is indeed the full set of twin 5-smooth integers as shown in [300], see also [214, Table 1A].

Remark 10.2.2. The CHM check that determines whether a pair (r, s) yields an integer solution t to the equation

$$\frac{t}{t+1} = \frac{r}{r+1} \cdot \frac{s+1}{s} \tag{10.2}$$

can be rephrased by solving this equation for t, which yields

$$t = \frac{r(s+1)}{s-r}. \tag{10.3}$$

This shows that in order for (r, s) to yield a new pair, $s - r$ must divide $r(s + 1)$ and in particular, must be B-smooth as well.

10.2.2 Generalising the CHM Algorithm

We now present a generalisation of the CHM algorithm, which finds smooth values of any monic quadratic polynomial $f(X) = X^2 + aX + b \in \mathbb{Z}[X] \subseteq \mathbb{Q}[X]$. The algorithm works with elements in the $\mathbb{Q}$-algebra $R = \mathbb{Q}[X]/\langle f(X)\rangle$. Let $\overline{X}$ denote the residue class of X in R. The generalisation closely follows the idea of the CHM algorithm and is based on the observation that for any $r \in \mathbb{Q}$, we have that

$$N_{R/\mathbb{Q}}(r - \overline{X}) = f(r),$$

where $N_{R/\mathbb{Q}}(\alpha)$ denotes the algebraic norm of $\alpha \in R$ over $\mathbb{Q}$. The algorithm now starts with an initial set

$$S^{(0)} = \{r_1 - \overline{X}, \ldots, r_d - \overline{X}\},$$

where r_i are smooth integer values of $f(X)$, which means that the element $r_i - \overline{X}$ has smooth non-zero norm. Next, in the d-th iteration of the algorithm, given any two $\alpha, \beta \in S^{(d-1)}$, compute

$$\alpha \cdot \beta^{-1} \cdot N_{R/\mathbb{Q}}(\beta) = r - s\overline{X}$$

for integers r, s (notice that β is invertible, since it has non-zero norm). Now, if s divides r, we obtain an integer $t = \frac{r}{s}$. It follows that

$$\begin{aligned} f(t) &= N_{R/\mathbb{Q}}\left(\frac{r}{s} - \overline{X}\right) \\ &= N_{R/\mathbb{Q}}(r - s\overline{X})s^{-2} \\ &= N_{R/\mathbb{Q}}(\alpha \cdot \beta^{-1} \cdot N_{R/\mathbb{Q}}(\beta))s^{-2} \\ &= N_{R/\mathbb{Q}}(\alpha)N_{R/\mathbb{Q}}(\beta)s^{-2}. \end{aligned}$$

Since both $N_{R/\mathbb{Q}}(\alpha)$ and $N_{R/\mathbb{Q}}(\beta)$ are B-smooth and s is an integer, it follows that t is a B-smooth value of $f(X)$. The set $S^{(d)}$ is then formed as the union of $S^{(d-1)}$ and the set of all such integral solutions. Finally, we terminate when $S^{(d)} = S^{(d-1)}$.

10.2.3 Equivalence with Previous Algorithms

We now show that the CHM algorithm, as well as another algorithm by Conrey and Holmstrom [87], are special cases of the generalised algorithm, for the polynomials $f(x) = X^2 + X$, and $f(X) = X^2 + 1$ respectively.

Smooth values of X^2+X. To see that the CHM algorithm (see Section 10.2.1) is indeed a special case of the generalised algorithm above, we show how it works for $f(X) = X(X+1) = X^2 + X$. Consider the algebra $R = \mathbb{Q}[X]/\langle X^2 + X\rangle$. This embeds into the matrix algebra $M_{2\times 2}(\mathbb{Q})$ via

$$\psi : r + s\overline{X} \to \begin{pmatrix} r & 0 \\ s & r-s \end{pmatrix}.$$

Instead of working with elements in R, we will work with elements in $\psi(R) \subseteq M_{2\times 2}(\mathbb{Q})$ since this simplifies the argument. In this case, for $\alpha \in R$, we have

$$N_{R/\mathbb{Q}}(\alpha) = \det(\psi(\alpha)).$$

The set corresponding to the initial set in the CHM algorithm is

$$S^{(0)} = \{\left(\begin{smallmatrix} 1 & 0 \\ -1 & 2 \end{smallmatrix}\right), \left(\begin{smallmatrix} 2 & 0 \\ -1 & 3 \end{smallmatrix}\right), \ldots, \left(\begin{smallmatrix} B-1 & 0 \\ -1 & B \end{smallmatrix}\right)\}.$$

All these elements clearly have B-smooth norm. The d-th CHM iteration proceeds as follows: For all $\left(\begin{smallmatrix} r & 0 \\ -1 & r+1 \end{smallmatrix}\right), \left(\begin{smallmatrix} s & 0 \\ -1 & s+1 \end{smallmatrix}\right)$ in $S^{(d-1)}$, we try

$$\begin{aligned}\begin{pmatrix} r & 0 \\ -1 & r+1 \end{pmatrix} \begin{pmatrix} s & 0 \\ -1 & s+1 \end{pmatrix}^{-1} s(s+1) &= \begin{pmatrix} r & 0 \\ -1 & r+1 \end{pmatrix} \left(\begin{pmatrix} s+1 & 0 \\ 1 & s \end{pmatrix} \frac{1}{s(s+1)} \right) s(s+1) \\ &= \begin{pmatrix} r(s+1) & 0 \\ -(s-r) & (r+1)s \end{pmatrix}.\end{aligned}$$

Finally, we transform this matrix into the right form, i.e. into a matrix corresponding to an element of the form $\tau = t - \overline{X}$, which means that $\psi(\tau)$ has a -1 in the lower left corner. So, we divide by $s - r$ and end up with the matrix

$$\begin{pmatrix} \frac{r(s+1)}{s-r} & 0 \\ -1 & \frac{(r+1)s}{s-r} \end{pmatrix} = \begin{pmatrix} \frac{r(s+1)}{s-r} & 0 \\ -1 & \frac{r(s+1)}{s-r} + 1 \end{pmatrix}.$$

Now if $\frac{r(s+1)}{s-r}$ is an integer, we add this matrix to the next set $S^{(d+1)}$.

As we have seen in Remark 10.2.2, this integer indeed corresponds to the solution given in Equation (10.3) of Equation (10.2) and therefore, the generalised algorithm in the case $f(X) = X^2 + X$ is equivalent to the original CHM algorithm.

Smooth values of X^2+1. Conrey and Holmstrom later presented a method to generate smooth values of X^2+1 [87]. Similar to the CHM algorithm, it starts with an initial set $S^{(0)}$ of positive smooth values of X^2+1. Again, for $d>0$ and given $r,s \in S^{(d-1)}, r<s$, they compute

$$\frac{rs-1}{s+r}.$$

The next set $S^{(d)}$ is then again formed as the union of $S^{(d-1)}$ and the set of all such values that are integers.

It is equally straightforward to verify that this algorithm is also a special case of the generalised CHM algorithm described above in Section 10.2.2. We could again work with matrices in $M_{2\times 2}(\mathbb{Q})$, but here, we are actually working in $K=\mathbb{Q}[X]/\langle X^2+1\rangle$, which is isomorphic to the number field $\mathbb{Q}(i)$, where $i^2=-1$. The product of the elements $\alpha=r-i$ and $\beta=s-i$ is given as

$$\alpha\beta=(r-i)(s-i)=(rs-1)-(r+s)i.$$

Conrey and Holmstrom's method then simply tries all such products $\alpha\beta$. However, a possibly better choice could be to use

$$\alpha\beta^{-1}N_{K/\mathbb{Q}}(\beta)=\alpha\overline{\beta}=(r-i)(s+i)=(rs+1)-(s-r)i$$

as described in our generalisation. This is due to the fact that the new denominator, $s-r$, is smaller and hence

$$\frac{rs+1}{s-r}$$

is more likely to be an integer (assuming that the numerator follows a random, uniform distribution). As a result, we can expect the algorithm to converge faster. Another alternative is to include both positive and negative values in the initial set $S^{(0)}$. Observe that in this case, it does not matter whether one uses $(rs+1)/(s-r)$ or $(rs-1)/(s+r)$, as $(rs+1)/(s-r)=-(s(-r)+1)/(s+(-r)))$.

Whichever option is chosen, one tries to divide by $r+s$ resp. $s-r$, and if the result is an element in $\mathbb{Z}[i]$, it is added to the next set $S^{(d)}$ of smooth values of X^2+1. Conrey and Holmstrom's method is therefore another special case of the generalised algorithm.

Remark 10.2.3. We note that neither the generalised CHM algorithm, nor any of the previous special cases give any guarantees to what proportion of B-smooth values of $f(X)$ it finds. However, for the previous special case algorithms, certain conjectural results have been stated, based on numerical evidence, which suggests that the algorithm returns all but a small fraction of all smooth values of the respective quadratic polynomials. We make no similar claims for the general case algorithm.

10.3 Searching for large twin smooth instances: CHM in practice

Ideally, the CHM algorithm could be run as described in the original paper [86] with a large enough smoothness bound B to find twin smooths of cryptographic sizes. However, experiments suggest that this is not feasible in practice. We report on data obtained from an implementation of the pure CHM algorithm in Section 10.3.1, present several optimisations in Section 10.3.2 and detail our optimised implementation in Section 10.3.3.

10.3.1 Running CHM in practice

To collect data and assess the feasibility of finding large enough twin smooths, we implemented a somewhat optimised version of the pure CHM algorithm. In particular, this implementation is parallelised, and avoids multiple checks of the same pairs of twin smooths (r, s). Furthermore, we iterate through smoothness bounds: We start with a small bound B_1 and the initial set $S_1^{(0)} = \{1, \ldots, B_1 - 1\}$, and use the CHM algorithm to iteratively compute sets $S_1^{(i)}$ until we reach some d_1 such that $S_1^{(d_1)} = S_1^{(d_1-1)}$. In the next iteration, we increase the smoothness bound to $B_2 > B_1$ and define the initial set $S_2^{(0)} = S_1^{(d_1)} \cup \{B_1, \ldots, B_2 - 1\}$. Again we compute CHM iterations until we find d_2 such that $S_2^{(d_2)} = S_2^{(d_2-1)}$, where we avoid checking pairs (r, s) that have been processed in earlier iterations. Ideally, we could repeat this procedure until we reach a smoothness bound B_i for which the CHM algorithm produces large enough twin smooths for cryptographic purposes. However, our data suggests that this is infeasible in practice due to both runtime and memory limitations.

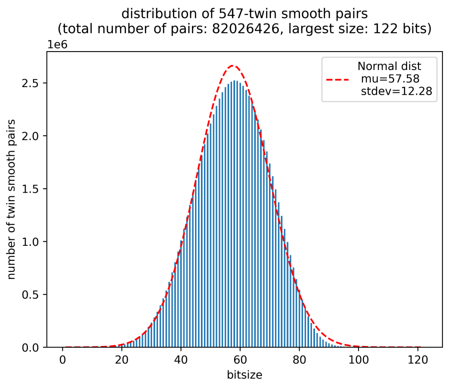
In particular, we ran this approach up to the smoothness bound $B = 547$, and extrapolating the results gives us rough estimations of the largest possible pair and number of twin smooths per smoothness bound.

After the $B = 547$ iteration, the set of twin smooths contains 82,026,426 pairs, whose bitlength distribution roughly resembles a normal distribution centered around bitlength 58. The largest pair has a bitlength of 122 bits. An evaluation of the obtained set is shown in Figure 10.1. Figure 10.1a shows the distribution of bitsizes in the full set, while Figure 10.1b shows that of the subset of all 199-smooth twins obtained in this run. Figure 10.1c shows the bitsize of the largest q-smooth twin pairs for each prime q between 3 and 547. And Figure 10.1d and Figure 10.1e show the number of q-smooth twins for each such q.

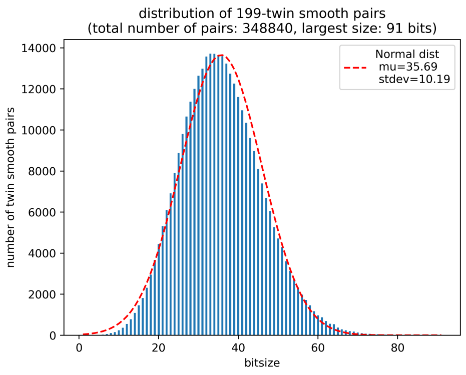
Using the data of these experiments, we can attempt to estimate at which smoothness bound B this approach can be expected to reach twin smooths of cryptographic sizes, and how much memory is required to run iterations to reach this B. The data visualised in Figure 10.1c indicates that the bound necessary for the largest twin smooth pair obtained by running CHM with this bound to reach a bitlength of 256 lies in the thousands, possibly larger than 5,000. Similarly, the data displayed in Figure 10.1d and Figure 10.1e shows how quickly the number of B-smooth twins increases with B. Given that the effort for CHM iterations grows quadratically with the set size, these estimates indicate that it is not feasible to reach cryptographically sized smooth twins with the original CHM algorithm.

10.3.2 Optimisations

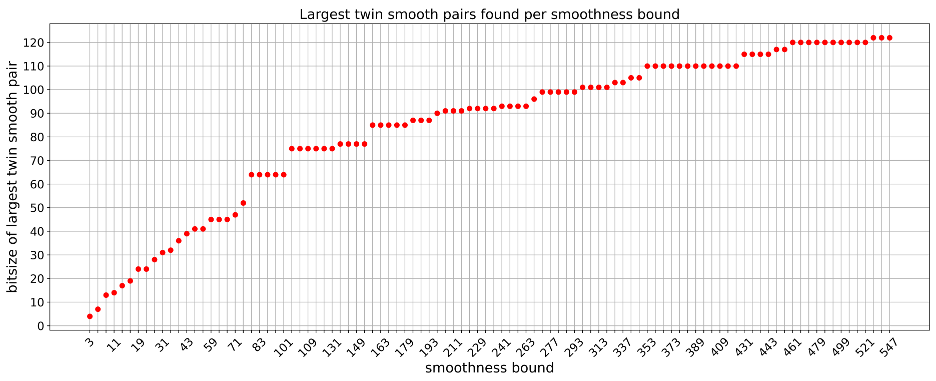
One major issue with running the plain CHM algorithm for increasing smoothness bound is the sheer size of data that needs to be dealt with. The sets $S_i^{(d_i)}$ grow very rapidly and the quadratic complexity of checking all possible pairs (r, s) leads to a large runtime. The natural question that



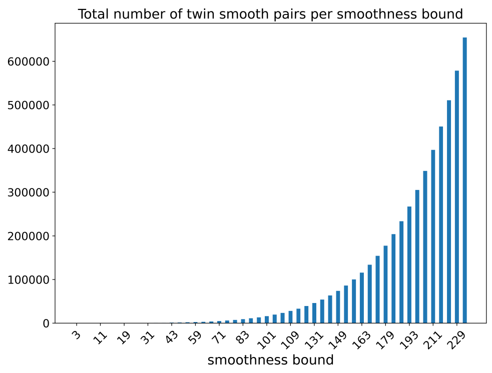
(A) Distribution of bitsizes for the full set of 547-twin smooth pairs.



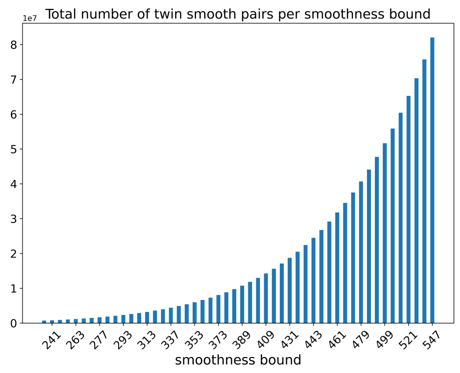
(B) Distribution of bitsizes for the subset of 199-twin smooth pairs.



(C) Bitsizes of the largest q-smooth twins for all primes q between 3 and 547.



(D) Number of q-smooth twins for all primes q between 3 and 233.



(E) Number of q-smooth twins for all primes q between 239 and 547.

FIGURE 10.1: Evaluation of the set of 547-smooth twins obtained by running the original CHM algorithm with smoothness bound $B = 547$. The bitsize of a pair $(r, r + 1)$ is $\lfloor \log r \rfloor + 1$. Data for the number of q-smooth twins for all primes q up to 547 has been split into two histograms of different scale.

arises is whether CHM can be restricted to checking only a certain subset of such pairs without losing any or too many of the new smooth neighbours. Furthermore, if the purpose of running the CHM algorithm is not to enumerate all twin smooth pairs for a given smoothness bound but instead, to produce a certain number of pairs of a given size or to obtain some of the largest pairs, it might even be permissible to omit a fraction of pairs.

To find a sensible way to restrict to a smaller set, we next discuss which pairs (r, s), $r < s$ result in a given twin smooth pair $(t, t + 1)$ via

$$\frac{r}{r+1} \cdot \frac{s+1}{s} = \frac{t}{t+1}. \tag{10.4}$$

This is discussed in [86, §3], but we elaborate on it in a slightly different way here. Let $t > 0$, let u be any divisor of t and v any divisor of $t + 1$. Let $h, x \in \mathbb{Z}$ be given by $t = uh$ and $t + 1 = vx$ (where $u, v, h, x > 0$). Therefore, $v/u = h/x + 1/(ux)$. If $u < v$ then $h > x$ and if $u > v$ then $h < x$. We therefore fix $u < v$ (otherwise switch the roles of u, v and h, x). Since $u < v$, the pair

$$(r, s) = (t - \frac{u}{v}(t + 1),\ \frac{v}{u}t - (t + 1) = \frac{v}{u}r) \tag{10.5}$$

satisfies Equation (10.4), and it follows that

$$r = u(h - x),\ r + 1 = x(v - u),\ s = v(h - x),\ s + 1 = h(v - u). \tag{10.6}$$

Therefore, $s/r = v/u$ and $(s + 1)/(r + 1) = h/x$, $u < v$, $h > x$ and $0 < r < s$. This also means that $s = r + (v - u)(h - x)$, $t = r + ux$ and that $\gcd(r(s + 1), s(r + 1)) = s - r = (v - u)(h - x)$ (note that $\gcd(uh, vx) = \gcd(t, t + 1) = 1$).

Conversely, given (r, s) with $r > 0$ that satisfy Equation (10.4), define $u = r/\gcd(r, s)$ and $v = s/\gcd(r, s)$, then $s > r$, $u < v$ and $u \mid t$, $v \mid (t + 1)$. Hence, we have the correspondence between the set of pairs (r, s) with $r < s$ that yield a new twin pair $(t, t + 1)$ via Equation (10.4) and the set of pairs of divisors of t and $t + 1$ described in [86, §3] as follows:

$$\begin{aligned} &\{(r, s) \mid r < s \text{ and } r(s + 1)(t + 1) = s(r + 1)t\} \\ &\longleftrightarrow \{(u, v) \mid u < v \text{ and } u \mid t,\ v \mid (t + 1)\}. \end{aligned} \tag{10.7}$$

However, this correspondence does not identify the pairs (r, s) corresponding to twin smooths, i.e. given (u, v) there is no guarantee that any of $r, r + 1, s, s + 1$ are B-smooth. This is not discussed in [86, §3]. The next lemma fills this gap by stating an explicit condition on the divisors u, v, h, x.

Lemma 10.3.1. *Let $t \in \mathbb{Z}$ such that $t(t + 1)$ is B-smooth. Let (u, v) be a pair of divisors such that $t = uh$, $t + 1 = vx$ and let (r, s) be defined as in Equation (10.5).*

Then $r(r + 1)s(s + 1)$ is B-smooth if and only if $(v - u)(h - x) = s - r$ is B-smooth.

Proof. As divisors of t and $t + 1$, u and v as well as h and x are all B-smooth. The statement follows from the Equation (10.6). □

Using similar sized pairs. We next consider the following condition to restrict the visited pairs (r, s) in CHM as a mechanism to reduce the set size and runtime. Let $k > 1$ be a constant parameter. We then only check pairs (r, s) if they satisfy

$$0 < r < s < kr. \tag{10.8}$$

Assume that (r, s) results in a pair $(t, t+1)$ through satisfying Equation (10.4). As seen above, $\frac{s}{r} = \frac{v}{u}$ for $u \mid t$, $v \mid (t+1)$, so we can use (u, v) to determine which values k are useful. Since $\frac{v}{u} < k$, it follows $s = \frac{v}{u}t - (t+1) < (k-1)t$. If we are only interested in obtaining a new t from a pair (r, s) such that $s < t$, we can take $k \le 2$, overall resulting in $1 < k \le 2$.

This constant k seems to be a good quantity to study as we can relate it to the factors of $v - u$. Indeed, $v - u = u(\frac{v}{u} - 1) = u(\frac{s}{r} - 1)$ and we have $s < kr$.

Definition 10.3.2. Let $(r, r+1)$ and $(s, s+1)$ be twin smooths with $r < s$ and $k \in \mathbb{R}$ with $1 < k \le 2$. We call the pair (r, s) *k-balanced* if $r < s < kr$.

We want to find a k such that a k-balanced pair (u, v) subject to the above conditions will yield a balanced r, s such that $r, r+1, s, s+1$ are B-smooth, or equivalently that $v - u$ and $h - x$ are.

Running the CHM algorithm only with 2-balanced pairs (r, s) then guarantees that any t produced by Equation (10.4) will be larger than the inputs r and s. Although we sacrifice completeness of the set of twin B-smooths with this approach, we can significantly reduce the runtime.

We can even push this approach further. Recall that we require $\gcd(r(s+1), (r+1)s) = s - r$ in order to generate a new pair of twin smooths $(t, t+1)$. By Lemma 10.3.1, this can only hold if $\Delta := s - r$ is B-smooth. Hence, only checking pairs (r, s) for which Δ is likely to be smooth increases the probability for a successful CHM step. Heuristically, the smaller Δ is, the better the chances for Δ to be smooth. Furthermore, if Δ contains small and only few prime factors, the probability for the condition $\Delta = \gcd(r(s+1), (r+1)s)$ is relatively high. We can summarise this in the following heuristic.

Heuristic 10.3.3. Let $k_1, k_2 \in \mathbb{R}$ with $1 < k_1 < k_2 \le 2$, and (r_1, s_1) resp. (r_2, s_2) a k_1- resp. k_2-balanced pair of twin smooths. Then the probability for (r_1, s_1) to generate new twin smooths via the CHM equation is larger than that for (r_2, s_2).

To save additional runtime, we can thus pick k closer to 1, and only check the pairs (r, s) that are most likely to generate new twin smooths. Therefore, we can still expect to find a significant portion of all twin B-smooths for a given smoothness bound B. We expand on the choice of k and different ways of implementing this approach in Section 10.3.3.

Thinning out between iterations. Another approach to reduce both runtime and memory requirement is to thin out the set of twin smooths between iterations. In particular, once we finished all CHM steps for a certain smoothness bound B_i, we can remove twins from the set $S_i^{(d_i)}$ based on their likeliness to produce new twin smooths before moving to the next iteration for B_{i+1}.

One possible condition for removing twins is to look at their smoothness bounds. Let $(r, r+1)$ be B_1-smooth, $(s, s+1)$ be B_2-smooth (but not B-smooth for any $B < B_2$), and $B_1 \ll B_2$. Since $(s, s+1)$ contains (multiple) prime factors larger than B_1, they cannot be contained in $(r, r+1)$, which makes the requirement $\gcd(r(s+1), (r+1)s) = s - r$ heuristically less likely to be satisfied. However, in practice it turns out that the differences between the smoothness bounds we are concerned with are not large enough for this heuristic to become effective.

In our experiments, it turned out to be more successful to keep track of how many new twin smooths each r produces. We can then fix some bound m, and discard twins that produced less than m twins after a certain number of iterations. Our experiments suggest that using this approach with carefully chosen parameters yields a noticeable speed-up, but fails completely at reducing the memory requirements, as we still need to keep track of the twins we already found. Furthermore, in practice the approach of only using k-balanced twins turned out to be superior, and hence we focus on this optimisation in the following.

10.3.3 Implementation

We implemented the CHM algorithm with several of the aforementioned optimisations in C++, exploiting the fact that it parallelises perfectly. Note that some of our approaches require the set of twin smooths to be sorted with respect to their size. Hence, an ordered data structure is used for storing the twins set. We used the following techniques and optimisations.

CHM step. For each pair (r, s) considered by the implementation, we have to check if Equation (10.4) holds. As mentioned in Section 10.3.2, this requires that $\gcd(r(s+1), (r+1)s) = s - r$ is satisfied. However, we can completely avoid the GCD calculation by observing that we require $r \cdot (s+1) \equiv 0 \mod (s-r)$ (see Equation (10.3)). Only if this is the case we perform a division to compute t, which represents the new pair of twin smooths $(t, t+1)$. Therefore, we only perform one modular reduction per considered pair (r, s), followed by one division if the CHM step is successful. This is significantly cheaper than a naïve implementation of Equation (10.4) or a GCD computation.

Data structure. Initially the set of twins was organised in a standard C array, that each time an iteration completed was reallocated to increase its size, and reordered. To avoid the overall inefficiency of this method we moved to use the C++ standard library `std::set`. This data structure is implemented with a Red Black tree, guarantees $O(\log N)$ insertion and search, while keeping the elements always ordered.

We then moved to use B+Trees [34], that have the same guarantees for insertion, search, and ordering, but are more efficient in the memory usage. Because the elements of a B+Tree are stored close to each other in memory it becomes much faster to iterate through the set, an operation that is necessary for creating the pairs used in each computation.

Implemented optimisations. As discussed in Section 10.3.2, we focus on the case of k-balanced pairs (r, s), which satisfy $r < s < kr$. Compared to the full CHM algorithm, this leads to a smaller set of twin smooths, but allows for much faster running times. We implemented the k-balanced approach in various different flavours.

Global-k. In the simplest version - the `global-`k approach - we initially pick some k with $1 < k \leq 2$, and restrict the CHM algorithm to only check k-balanced pairs (r, s). The choice of k is a subtle matter. Picking k too close to 1 may lead to too many missed twin smooths, such that we cannot produce any meaningful results. On the other hand, picking k close to 2 may result in a relatively small speed-up, which does not allow for running CHM for large enough smoothness bounds B. Unfortunately, there seems to be no theoretical handle on the optimal choice of k, which means that it has to be determined experimentally. We note that when picking an aggressive bound factor $k \approx 1$, small numbers r in the set of twins S may not have any suitable $s \in S$ they can be checked with. Thus, we pick a different bound, e.g. $k = 2$, for numbers below a certain bound, e.g. for $r \leq 2^{20}$.

Iterative-k. Instead of iterating through smoothness bounds B_i as described in Section 10.3.1 and using the `global-`k approach, we can switch the roles of B and k if we are interested in running CHM for a fixed smoothness bound B. We define some initial value k_0, a target value $k_{\max}$, and a step size $k_{\text{step}} > 0$. In the first iteration, we run CHM as in the `global-`k approach, using k_0. The next iteration then increases to $k_1 = k_0 + k_{\text{step}}$, and we add the condition to not check pairs (r, s) if they were already checked in previous iterations. We repeat this iteration step several times until we reach $k_{\max}$. Compared to the `global-`k approach, this allows us to generate larger B-smooth twins faster, since we restrict to the pairs (r, s) first that are most likely to generate new twins. However, the additional checks if previous pairs have been processed in earlier iterations add a significant runtime overhead. Thus, this method is more suitable for finding well-suited choices of k, while actual CHM searches benefit from switching to the `global-`k approach.

Constant-range. In both the `global-`k and `iterative-`k approach, the checks if a pair (r, s) is k-balanced, or has been processed in earlier iterations, consumes a significant part of the overall runtime. Therefore, we can use constant ranges to completely avoid these checks. Since we always keep the set of twins S sorted by size, the numbers s closest to r (with $s > r$) are its neighbours in S. Thus, we can sacrifice the exactness of the k-balanced approaches above, and instead fix a range R and for each r check (r, s) with the R successors s of r in S. As shown below, this method significantly outperforms the `global-`k approach due to the elimination of all checks for k-balance. This is true even when R is large enough to check more pairs than are considered in the `global-`k approach for a given k.

Variable-range. Similar to the `constant-range` approach, we can adapt the range R depending on the size of r. For instance, choosing r at the peak of the size distribution will lead to many possible choices of s such that (r,s) are balanced. Hence, we can choose a larger range R whenever more potential pairs exist, while decreasing R otherwise. In practice, the performance of this method ranks between `global-`k and `constant-range` by creating roughly the same pairs that `global-`k creates without any of the overhead of the balance checks. If R is chosen large enough such that the `constant-range` approach ends up generating more pairs than `global-`k, then `variable-range` performs better. Realistically, the size of the range R increases by (very) roughly 3% for each prime number smaller than the smoothness bound B, and slows down the algorithm drastically at higher smoothness, similarly to the k-based approaches.

Remark 10.3.4. Similar to the `variable-range` approach, we experimented with a variant of the `global-`k approach, which adjusts k according to the size of r to find suitable s for the CHM step. However, the `constant-range` and `variable-range` approaches turned out to be superior in terms of performance, and therefore we discarded this `variable-`k variant.

Variant	Parameter	Runtime	Speed-up	#twins	#twins from largest 100
Full CHM	-	4705s	1	2300724	100
`global-`k	$k = 2.0$	364s	13	2289000	86
	$k = 1.5$	226s	21	2282741	82
	$k = 1.05$	27s	174	2206656	65
`constant-range`	$R = 10000$	82s	57	2273197	93
	$R = 5000$	35s	134	2247121	87
	$R = 1000$	16s	294	2074530	75

Table 10.1: Performance results for different variants of our CHM implementation for smoothness bound $B = 300$. Speedup factors refer to the full CHM variant.

Performance comparison. To compare the implications of the optimisations in practice, we ran different variants of the CHM implementation for the fixed smoothness bound $B = 300$. All experiments ran on a machine configured with 4 x Xeon E7-4870v2 15C 2.3 GHz, 3072 GB of RAM. The total amount of parallel threads available was 120. As described above, the `global-`k and `constant-range` approach significantly outperform their respective variants, hence we focus on different configurations of these two methods.

The results are summarised in Table 10.1. For both the `global-`k and the `constant-range` approach we measured the results for conservative and more aggressive instantiations, where smaller values of k and R are considered more aggressive. It is evident that already for the conservative instantiations, we gain significant performance speed-up, while still finding almost the full set of twin smooths, and most of the 100 largest 300-smooth twins. For the more aggressive instantiations, we miss more twins, yet still find a significant amount of large twins.

As discussed above, the `constant-range` approach outperforms the `global-`k approach in terms of runtime, due to the elimination of all checks for k-balance of twins. Interestingly, while very aggressive instantiations of `constant-range` miss more twin smooths, they find a larger share of the largest 100 twins than their `global-`k counterpart. Therefore, we conclude that for larger smoothness bounds B, for which we cannot hope to complete the full CHM algorithm, `constant-range` is the most promising approach for obtaining larger twin smooths within feasible runtimes.

Remark 10.3.5. While all optimisations lose a small proportion of the largest twin smooths, they are not necessarily lost permanently. In practice, when iterating to larger smoothness bounds B_i, we often also find some B_j-smooth twins for bounds $B_j < B_i$. Thus, the size of the set of 300-smooth twins usually increases in the optimised variants when moving to larger B.

Remark 10.3.6. In the following sections, we will require twin smooths of a certain (relatively small) bitlength. This can easily be incorporated into all implemented variants by removing all twins above this bound after each iteration. This means that we cut off the algorithm at this size, and do not attempt to obtain larger twins, which significantly improves the runtime and memory requirements.

10.4 Cryptographic primes of the form $p = 2r^n - 1$

This section focuses on finding primes suitable for isogeny-based cryptographic applications. As discussed in the previous sections, the pure CHM method does not allow us to directly compute twins of at least 256 bits as required for this aim. However, some cryptographic applications, for example the isogeny-based signature scheme SQIsign, do not need twins $(r, r+1)$ that are fully smooth. Indeed, the current[4] incarnation of SQIsign requires a prime p that satisfies $2^f T \mid p^2 - 1$, where f is as large as possible, and $T \approx p^{5/4}$ is smooth and odd [126]. This flexibility allows us to move away from solely using CHM and, instead, use CHM results as inputs to known methods for finding such primes. At a high level, we will find fully smooth twins of a smaller bit-size via CHM and boost them up using the polynomials $p_n(x) = 2x^n - 1$ for carefully chosen n. If $r, r+1$ are fully smooth integers and n is not too large, we can guarantee a large proportion of $p_n(r)^2 - 1$ to be smooth.

Notation. For a variable x, we will denote $2x^n - 1$ by $p_n(x)$, and the evaluated polynomial $p_n(r)$ by p, emphasising that it is an integer.

General method. In this section, we will give a more in-depth description of the approach to obtaining cryptographic sized primes p, such that $p^2 - 1$ has $\log(T')$ bits of B-smoothness, where $T' = 2^f T$. We recall that for our SQIsign application, we have $\log(p) \in \{256, 384, 512\}$ for NIST Level I, III and V (respectively), $T \approx p^{5/4}$ and f as large as possible. In the current implementation of SQIsign, we have $f \approx \lfloor \log(p^{1/4}) \rfloor$ (i.e., $T' \approx p^{3/2}$), and so we aim for this when finding primes.

Fix a smoothness bound B and let $p_n(x) = 2x^n - 1$. We have $p_n(x)^2 - 1 = 4x^n(x-1)f(x)$ for some polynomial $f(x)$ given in Table 10.2.

n	$p_n(x)^2 - 1$
2	$4x^2(x-1)(x+1)$
3	$4x^3(x-1)(x^2+x+1)$
4	$4x^4(x-1)(x+1)(x^2+1)$
5	$4x^5(x-1)(x^4+x^3+x^2+x+1)$
6	$4x^6(x-1)(x+1)(x^2-x+1)(x^2+x+1)$

TABLE 10.2: Factorisation of $p_n(x)^2 - 1$ for $n = 2, 3, 4, 5, 6$, where $p_n(x) = 2x^n - 1$

We observe that for even n, both $x+1$ and $x-1$ appear in the factorisation of $p_n(x)^2 - 1$. In this case, for twin smooths $(r, r \pm 1)$, evaluating $p_n(x)$ at r guarantees that we have a smooth factor $4r^n(r \pm 1)$ in $p^2 - 1$. For n odd, we will only have that $x - 1$ appears in the factorisation, and therefore only consider twins $(r, r-1)$ to guarantee we have B-smooth factor $4r^n(r-1)$.

The first step is to use our implementation of the CHM algorithm, described in Section 10.2 and Section 10.3, to obtain B-smooth twins $(r, r \pm 1)$ of bitsize approximately $(\log p - 1)/n$. We then obtain primes of suitable sizes by computing $p = p_n(r)$ for all candidate r. By construction, $p^2 - 1$ has guaranteed $\frac{n+1}{n}(\log(p) - 1) + 2$ bits of smoothness. We then require that the remaining

[4]At the time of writing, [126] was the most promising iteration of SQIsign.

factors have at least

$$\max\left(0,\ \frac{3}{2}\log p - \left(\frac{n+1}{n}(\log p - 1) + 2\right)\right)$$

bits of B-smoothness. In Section 10.4.2, we will discuss the probability obtaining this smoothness from the remaining factors.

10.4.1 Choosing n

For small n, we require CHM to find twin smooths of large bit size. For certain bit sizes, running full CHM may be computationally out of reach, and therefore we use a variant that may not find all twins. In this case, however, we have more guaranteed smoothness in $p^2 - 1$ and so it is more likely that the remaining factors will have the required smoothness. For large n, we can obtain more twin smooths from CHM (in some cases, we can even exhaustively search for all twin smooths), however we have less guaranteed smoothness in $p^2 - 1$. Finding values of n that balance these two will be the focus of this section.

$n = 2$. Let $(r, r \pm 1)$ be twin smooth integers and let $p = 2r^2 - 1$. In this case, $2r^2(r \pm 1) \mid T'$, meaning that $\log T' \geq \frac{3}{2}\log p$, and we have all the required smoothness. Write $T' = 2^f T = 2r^2(r \pm 1)$ where T is odd. If $f \approx \lfloor \frac{1}{4}\log p \rfloor$, we have $T \approx p^{5/4}$, and we do not have to rely on a large power of 2 dividing $r \mp 1$. Otherwise, we turn to Section 10.4.2 to estimate the probability of $r \mp 1$ having enough small factors to make up for this difference.

Suppose we target primes with λ bits of classical security, i.e., we need a prime of order $p \approx 2^{2\lambda}$. For $n = 2$, this corresponds to finding twin smooths of size $\approx 2^{\lambda - \frac{1}{2}}$, and so is only suitable for finding NIST Level I parameters due to the limitations of the CHM method (see Section 10.3). One could instead use other techniques for finding large enough twins for $n = 2$, such as the PTE sieve [105], at the cost of significantly larger smoothness bounds. Alternatively, we can move to higher n, which comes at the cost of loosing guaranteed smoothness. Another challenge here is that, given the relatively large size of the twins, it appears difficult to find enough twins to obtain primes with a large power of two.

$n = 3$. Let $(r, r - 1)$ be twin smooth integers and let $p = 2r^3 - 1$. Here, we can guarantee that the smooth factor T' of $p^2 - 1$ is at least of size $\approx p^{4/3}$. If $f < \lfloor \frac{1}{12}\log p \rfloor$, we have $T > p^{5/4}$. Otherwise, we require that there are enough smooth factors in $r^2 + r + 1$ to reach this requirement.

Here, for λ bits of classical security, we need to target twin smooth integers of size $\approx 2^{\frac{2\lambda - 1}{3}}$. In this case, the CHM method will (heuristically) allow us to reach both NIST Level I and III parameters.

$n = 4$. Let $(r, r \pm 1)$ be twin smooth integers and $p = 2r^4 - 1$. Here we can only guarantee a factor of size $\approx p^{5/4}$ of $p^2 - 1$ to be smooth. When accounting for the power of two, we must hope for other smooth factors. As $p_n(x) - 1$ splits into (relatively) small degree factors, namely $p_n(x) - 1 = 2(x - 1)(x + 1)(x^2 + 1)$, the probability of having enough B-smooth factors is greater than if there was, for example, a cubic factor.

In contrast to the previous cases, this setting should be suitable for targeting all necessary security parameters. However, for the NIST Level I setting, the work by De Feo, Leroux, Longa, and Wesolowski [126, §5.2] showed that the best one could hope for here while maximising the power of two gives SQIsign parameters with a smoothness bound of around 1800. While this is a better smoothness bound than the NIST Level I prime with the best performance for SQIsign, it does not perform as well in practice. Indeed, most of the odd primes less than 1800 that appear in p^2-1 are relatively large, making isogeny computation relatively slow. In the best performing prime, however, a large power of 3 divides p^2-1, and most of its other odd prime divisors are fairly small. We note that the authors of [126] only searched for parameters that maximise the power of two, and hence there could be some scope to find parameters that have slightly smaller powers of two.

Other n. For larger n, the amount of guaranteed smoothness decreases, and thus the probability that the remaining factors have the required smoothness is small. Indeed, we find that only $n=6$ has the correct balance of requiring small twin smooths while still having a reasonable probability of success. This is primarily due to the factorisation of $p_6^2(x)-1 = 2(x-1)(x+1)(x^2-x+1)(x^2+x+1)$, having factors of degree at most 2, which improves the probability that we have enough smooth factors. In contrast, $n=5$ results in more guaranteed smoothness than $n=6$, but requires the quartic factor in $p_5^2(x)-1$ to provide the necessary smoothness, which is relatively unlikely.

While one could use $n=6$ to find NIST Level I parameters, this larger n shines in its ability to give us both NIST Level III and V parameters.

10.4.2 Probability of sufficient smoothness

We determine the probability of obtaining cryptographic primes with sufficient smoothness using the methods outlined above. Following Banks and Shparlinski [15], we determine the probability of p^2-1 being sufficiently smooth for some prime p. More precisely, given that the factor $r(r\pm1) \mid p^2-1$ is already fully smooth, we want to calculate the probability of p^2-1 having $\log(T')$-bits of B-smoothness.

First, we find the probability that the factor $r(r\pm1) \mid p^2-1$ is fully smooth, i.e., the probability of finding fully B-smooth twins $(r, r\pm1)$. To do so, we use the following counting function:

$$\Psi(X,B) = \#\{N \le X : N \text{ is } B\text{-smooth}\}.$$

For a large range of X and B, it is known that

$$\Psi(X,B) \sim \rho(u)X,$$

where $u = (\log X)/(\log B)$ and ρ is the Dickman function [55, 134]. The Dickman function is implemented in most computational algebra packages, including SageMath [311], which allows us to evaluate $\Psi(X,B)$ for various X and B. In practice, we find B-smooth twins $(r, r\pm1)$ using our implementation of the CHM algorithm as described in Section 10.3.

Next, we calculate the probability of p^2-1 having $\log(T')$-bits of B-smoothness. As p^2-1 may only be partially smooth, we will use the following counting function

$$\Theta(X,B,D) = \#\{N \le X : D < \text{largest } B\text{-smooth divisor of } N\}.$$

The value $\Theta(X, B, D)$ will give the number of positive integers $N \leq X$ for which there exists a divisor $d \mid N$ with $d > D$ and such that d is B-smooth. This function has been previously studied in the literature, for example see [307, 308]. For X, B, D varying over a wide domain, Banks and Shparlinski [15, Theorem 1] derive the first two terms of the asymptotic expansion of $\Theta(X, B, D)$. By implementing this expansion, we are able to estimate the value of Θ at various X, B, D in the correct range.

As discussed in the section above, we restrict to $n = 2, 3, 4, 6$. Recall that $p_n(x)^2 - 1 = 4x^n(x-1)f(x)$, as given in Table 10.2 for each $2 \leq n \leq 6$. Write $f(x) = f_1(x) \cdots f_k(x)$, where each f_i is irreducible of degree $d_i = \deg(f_i)$ and $d = \deg(f)$. To calculate the probabilities, we assume that the probability of $f(x)$ having at least $\log(D)$-bits of B-smoothness is the product of the probabilities of each of its factors f_i having at least $\log(D_i)$-bits of B-smoothness where $\log(D) = \sum_{i=1}^{k} \log(D_i)$. We can view this as an extension of Heuristic 1 in [105]. Note that the only constraint on how the smoothness is distributed between the factors $f_i(x)$ is that the total bit size of B-smooth factors must equal $\log(D)$. We could, for example, sum over all the possible distributions of smoothness using the inclusion-exclusion principle. However, in distributions where one of the factors has a very small amount of smoothness, we fall out of the ranges allowed as input into the function Θ, as determined by [15, Theorem 1]. Therefore, for simplicity, we assume that smoothness is distributed evenly between the remaining factors (weighted by the degree), i.e., $\log(D_i) = (d_i \log(D))/d$. In reality, this only gives us a lower bound for the probability, but this suffices for our purposes. Obtaining a more theoretical and accurate grasp on these probabilities is left as an avenue for future research.

In Table 10.3, we give an overview of the relevant probabilities for NIST Level I, III, and V parameters, calculated as described above. We observe that as n gets larger, the probability of finding B-smooth integers of the appropriate bitsize increases. In contrast, for bigger n we are guaranteed less smoothness in $p^2 - 1$. As a result, given B-smooth twins, the probability of finding a SQIsign prime p decreases as n increases. For each NIST level, we predict that the n that balance these two contrasting probabilities have a higher chance of finding a p satisfying our requirements. As discussed in the next section, this trend is reflected in practice.

10.5 Results and comparisons

In this section we give the concrete results that were obtained from our experiments with the CHM algorithm, and analyse the various twins in relation to SQIsign in accordance with the relevant bitsizes mentioned in Table 10.3.

10.5.1 Record twin smooth computations

We ran the optimised full CHM algorithm with $B = 547$ and found a total of 82,026,426 pairs of B-smooth twins. Among these pairs, we found 2,649 additional 200-smooth twins that were not found by the original authors of the algorithm [86]. This showcases the validity of Remark 10.2.3 that the algorithm does not guarantee us to find all B-smooth twins. Furthermore, there is no guarantee that running CHM with $B = 547$ will produce all 200-smooth twins. As mentioned in

	n	$\log(r)$	**Probability of B-smooth** $(r, r \pm 1)$	**Probability of p^2-1 $\log T'$-bits B-smooth given $(r, r \pm 1)$ twin smooth**	**Extra Smoothness Needed**
NIST-I	2	≈ 127.5	$2^{-58.5}$	1	0
$B = 2^9$	3	≈ 85.0	$2^{-32.1}$	$2^{-8.4}$	42
$\log p = 256$	4	≈ 63.8	$2^{-20.5}$	$\approx 2^{-12.7}$	63.3
$\log T' = 384$	6	≈ 42.5	$2^{-10.4}$	$\approx 2^{-16.8}$	84.5
NIST-III	2	≈ 191.5	$2^{-55.7}$	1	0
$B = 2^{14}$	3	≈ 127.7	$2^{-30.5}$	$2^{-8.2}$	63.3
$\log p = 384$	4	≈ 95.8	$2^{-19.4}$	$\approx 2^{-12.4}$	95.3
$\log T' = 576$	6	≈ 63.8	$2^{-9.7}$	$\approx 2^{-16.2}$	127.2
NIST-V	2	≈ 255.5	$2^{-63.7}$	1	0
$B = 2^{17}$	3	≈ 170.3	$2^{-35.2}$	$2^{-9.6}$	84.7
$\log p = 512$	4	≈ 127.8	$2^{-22.6}$	$\approx 2^{-14.5}$	127.3
$\log T' = 768$	6	≈ 85.2	$2^{-11.5}$	$\approx 2^{-19.2}$	169.8

Table 10.3: Assuming that $(r, r \pm 1)$ are twin smooth integers and p has $\log p$ bits, calculates the probability of having a B-smooth divisor $T' \mid p^2 - 1$ of size $\approx p^{3/2}$. More details in text.

the introduction, the only way to see how far away we are from the exact number of 200-smooth twins is to solve all 2^{46} Pell equations.

For the application mentioned in the previous section, we only need twins of a certain bitsize. Within this set of twins, 9,218,648 pairs $(r, r+1)$ fall in the range $2^{60} < r < 2^{64}$; 1,064,249 pairs fall in the range $2^{81} < r < 2^{85}$; 31,994 pairs fall in the range $2^{92} < r < 2^{96}$; and, only 1 pair falls in the range $2^{120} < r < 2^{128}$. This pair in the final interval is the largest pair found in this run, with $r = 4012031241848866526424165796047749375$, and factorisations:

$$\begin{aligned} r &= 5^4 \cdot 7 \cdot 13^2 \cdot 17^2 \cdot 19 \cdot 29 \cdot 41 \cdot 109 \cdot 163 \cdot 173 \cdot 239 \cdot 241^2 \cdot 271 \cdot 283 \\ &\quad \cdot 499 \cdot 509, \text{ and} \\ r + 1 &= 2^8 \cdot 3^2 \cdot 31^2 \cdot 43^2 \cdot 47^2 \cdot 83^2 \cdot 103^2 \cdot 311^2 \cdot 479^2 \cdot 523^2. \end{aligned}$$

As we will see later, the number of 64-bit and 85-bit twins we found in this run is enough to find attractive parameters for SQIsign. The 96-bit twins will give us parameters with the required smoothness, however we do not have enough pairs to hope to find a prime p where $p^2 - 1$ is divisible by a large power of two.

Table 10.3 shows that finding many twins of around 128 bits in size is likely to be fruitful in the search for SQIsign-friendly parameters, so we ran the algorithm for $B = 1300$ using the `constant-range` optimisation with a range $R = 5000$, in order to specifically target twins $(r, r+1)$ with $r > 2^{115}$. In this run we found 1,091 such pairs - the largest of these pairs is the following 145-bit twin $(r, r+1)$ with $r = 36132012096025817587153962195378848686084640$, where

$$\begin{aligned} r &= 2^5 \cdot 5 \cdot 7 \cdot 11^2 \cdot 13 \cdot 23 \cdot 53 \cdot 71 \cdot 109 \cdot 127 \cdot 131 \cdot 193 \cdot 251 \cdot 283 \cdot 307 \\ &\quad \cdot 359 \cdot 367 \cdot 461 \cdot 613 \cdot 653 \cdot 1277, \text{ and} \\ r+1 &= 3^2 \cdot 29^2 \cdot 31^2 \cdot 43^2 \cdot 59^2 \cdot 61^2 \cdot 73^2 \cdot 79^2 \cdot 89^2 \cdot 167^2 \cdot 401^2 \cdot 419^2. \end{aligned}$$

Among the 1,091 twins CHM found, 184 pairs fall in the range $2^{120} < r < 2^{128}$, which was sufficient to find some SQIsign-friendly parameters (though not at all NIST security levels).

In addition, we also ran CHM with $B = 2^{11}$ to obtain a large number of twin smooth integers in the range $2^{55} < r < 2^{100}$ (see Remark 10.3.6 in the setting where we want to find twins in such an interval). This run was performed using the `constant-range` optimisation with a range $R = 2500$, and produced 608,233,761 pairs of twins lying in this range. Compared with the $B = 547$ run, the yield from this run gave ample twins with $2^{92} < r < 2^{96}$, which was sufficient to find SQIsign parameters with the desirable large power of two.

All of these searches were done using the machine specified in Section 10.3.3, taking between 1 and 2 days each to run.

10.5.2 Concrete parameters for SQISign

We give a list of SQIsign-friendly primes that target NIST Level I, III, and V, i.e, we find primes p with $2^f T \mid p^2 - 1$ and T odd. We need the exponent f to be as large as possible and the cofactor $T \approx p^{5/4}$ to be B-smooth, aiming to keep the ratio $\sqrt{B}/f$ as small as possible; this quantity is a rough cost metric for the performance of the signing algorithm in SQIsign [126, §5.1]. To complement this, the exponent f controls the performance of the verification of SQIsign; the larger this exponent is, the faster the verification is. We may run into circumstances where the signing cost metric is minimised, but the power of two is not large enough or vice-versa. We aim to balance these as much as possible, thus finding parameters that maximise the power of two while minimising the signing cost metric. We refer to Section 10.5.3 for more details on the practicability of our parameters.

Though we need $T \approx p^{5/4}$, if this cofactor is too close to $p^{5/4}$, then the underlying heuristics within the generalised KLPT algorithm might fail and one cannot guarantee a successful signature in SQIsign [126, §3.2]. Thus, in practice we need $T \approx p^{5/4+\epsilon}$ for some small ϵ (e.g., $0.02 < \epsilon < 0.1$).

We find parameters for NIST Level I, III and V by searching for 256, 384 and 512-bit primes, respectively. For those primes targeting the higher security levels, these are the first credible SQIsign-friendly primes. In what follows, we look at each security level and analyse the most noteworthy primes found in our searches. From here on out, when stating the factorisations of $p \pm 1$ for the mentioned primes, the factors in **bold** are the rough factors which are not needed for SQIsign, while the other factors are the smooth factors of T. A full collection of our best SQIsign-friendly primes that were found using the CHM machinery is showcased in Table 10.4.

Remark 10.5.1. We note that in all of the forthcoming searches, the post-processing of the CHM twins to find the SQIsign-friendly parameters can be made reasonably efficient with straightforward techniques. In particular, the runtime is negligible in comparison to running the CHM searches mentioned in Section 10.5.1 and can be done using naïve trial division.

NIST-I[5] parameters. We targeted 256-bit primes using $n = 2, 3$ and 4. Given that our CHM runs produced a lot more twins of smaller bit-size compared to the 128-bit level, we expect to find more primes using $n = 3, 4$, which was indeed the case. It is worth noting that some primes found with $n = 2$ gave rise to $p^2 - 1$ being divisible by a relatively large power of two. However, in these cases, most of the primes dividing $p^2 - 1$ are relatively large and would therefore give rise to slower isogeny computations during the SQIsign protocol [126].

Through the experimentation with the 85-bit twins produced from CHM with $B = 547$, we found the following 254-bit prime $p = 2r^3 - 1$ with $r = 20461449125500374748856320$. All the specific criteria that we need for a SQIsign parameter set are met, while obtaining an attractively small signing cost metric $\sqrt{B}/f$. For this prime, we have

$$\begin{aligned} p+1 &= 2^{46} \cdot 5^3 \cdot 13^3 \cdot 31^3 \cdot 73^3 \cdot 83^3 \cdot 103^3 \cdot 107^3 \cdot 137^3 \cdot 239^3 \cdot 271^3 \cdot 523^3, \text{ and} \\ p-1 &= 2 \cdot 3^3 \cdot 7 \cdot 11^2 \cdot 17^2 \cdot 19 \cdot 101 \cdot 127 \cdot 149 \cdot 157 \cdot 167 \cdot 173 \cdot 199 \cdot 229 \cdot 337 \\ &\quad \cdot 457 \cdot 479 \cdot \mathbf{141067} \cdot \mathbf{3428098456843} \cdot \mathbf{4840475945318614791658621}. \end{aligned}$$

While the associated cofactor T here exceeds $p^{5/4}$, it does not exceed it by much. As we mentioned earlier, it might therefore be prone to signing failures and hence might not currently be suitable for SQIsign. The next 255-bit prime worthy of mention, $p = 2r^3 - 1$ with $r = 26606682403634464748953600$, is very similar to the previous prime, however the cofactor T exceeds $p^{5/4}$ by a larger margin, so would be less prone to these failures. In this case we have

$$\begin{aligned} p+1 &= 2^{40} \cdot 5^6 \cdot 11^3 \cdot 47^3 \cdot 67^6 \cdot 101^3 \cdot 113^3 \cdot 137^3 \cdot 277^3 \cdot 307^3 \cdot 421^3, \text{ and} \\ p-1 &= 2 \cdot 3^2 \cdot 19^3 \cdot 37 \cdot 59 \cdot 61 \cdot 97 \cdot 181^2 \cdot 197 \cdot 223 \cdot 271 \cdot 281 \cdot 311 \cdot 397 \cdot 547 \\ &\quad \cdot \mathbf{1015234718965008560203} \cdot \mathbf{3143438922304814418457}. \end{aligned}$$

We additionally ran experiments with the 64-bit twins produced from CHM with $B = 547$ and found a 253-bit prime $p = 2r^4 - 1$ with $r = 8077251317941145600$, where we have

$$\begin{aligned} p+1 &= 2^{49} \cdot 5^8 \cdot 13^4 \cdot 41^4 \cdot 71^4 \cdot 113^4 \cdot 181^4 \cdot 223^4 \cdot 457^4, \text{ and} \\ p-1 &= 2 \cdot 3^2 \cdot 7^5 \cdot 17 \cdot 31 \cdot 53 \cdot 61 \cdot 73 \cdot 83 \cdot 127 \cdot 149 \cdot 233 \cdot 293 \cdot 313 \cdot 347 \cdot 397 \\ &\quad \cdot 467 \cdot 479 \cdot \mathbf{991} \cdot \mathbf{1667} \cdot \mathbf{19813} \cdot \mathbf{211229} \cdot \mathbf{107155419089} \\ &\quad \cdot \mathbf{295288804621}. \end{aligned}$$

Among all the primes that we found for NIST-I security, this appears to be the best. It has both a larger power of two compared to the primes mentioned above found with $n = 3$ and a smaller smoothness bound, thus making the signing cost metric attractively small. Additionally, the cofactor T is large enough to be practical for SQIsign without any failures. We note once again that this prime would have been out of scope for the authors of [126] to find since they constrained their search to only find primes for which the power of two is larger than the one found here.

NIST-III parameters. We targeted 384-bit primes using $n = 3, 4$ and 6. The challenge in all three of these scenarios is finding enough twins whose product is divisible by a large power of two.

[5]NIST Level I, III, and V parameters are also referred to as NIST-I, NIST-III, NIST-IV parameters.

With the limited yield of 128-bit twins, finding such primes is not straightforward; the example with $n = 3$ in Table 10.4 is the only such instance that we managed to find. The picture is somewhat similar with the 96-bit twins: while we have more of them, the success probabilities in Table 10.3 suggest that we need a lot more twins with a large power of two in order to produce any SQIsign-friendly instances. One exceptional prime that was found in this search was the following 375-bit prime, $p = 2r^4 - 1$ with $r = 12326212283367463507272925184$. Here, we have

$$\begin{aligned} p + 1 &= 2^{77} \cdot 11^4 \cdot 29^4 \cdot 59^4 \cdot 67^4 \cdot 149^4 \cdot 331^4 \cdot 443^4 \cdot 593^4 \cdot 1091^4 \cdot 1319^4, \text{ and} \\ p - 1 &= 2 \cdot 3 \cdot 5 \cdot 13 \cdot 17 \cdot 31 \cdot 37 \cdot 53 \cdot 83 \cdot 109 \cdot 131 \cdot 241 \cdot 269 \cdot 277 \cdot 283 \cdot 353 \cdot 419 \\ &\quad \cdot 499 \cdot 661 \cdot 877 \cdot 1877 \cdot 3709 \cdot 9613 \cdot 44017 \cdot 55967 \cdot \mathbf{522673} \cdot \mathbf{3881351} \\ &\quad \cdot \mathbf{4772069} \cdot \mathbf{13468517} \cdot \mathbf{689025829} \cdot \mathbf{300114179456737662 53}. \end{aligned}$$

Of the NIST Level III primes listed in Table 10.4, the prime that shows the most promise is the 382-bit prime $p = 2r^6 - 1$ with $r = 1189664338866 2145024$. Not only is the power of two particularly large, but also the smoothness bound of the cofactor T is quite small, reflected in its small signing cost metric (when compared to other p where $p^2 - 1$ is divisible by a large power of 2). We have the factorisations

$$\begin{aligned} p + 1 &= 2^{79} \cdot 3^6 \cdot 23^{12} \cdot 107^6 \cdot 127^6 \cdot 307^6 \cdot 401^6 \cdot 547^6, \text{ and} \\ p - 1 &= 2 \cdot 5^2 \cdot 7 \cdot 11 \cdot 17 \cdot 19 \cdot 47 \cdot 71 \cdot 79 \cdot 109 \cdot 149 \cdot 229 \cdot 269 \cdot 283 \cdot 349 \cdot 449 \\ &\quad \cdot 463 \cdot 1019 \cdot 1033 \cdot 1657 \cdot 2179 \cdot 2293 \cdot 4099 \cdot 5119 \cdot 10243 \cdot \mathbf{381343} \\ &\quad \cdot \mathbf{19115518067} \cdot \mathbf{7408818089724412 33} \cdot \mathbf{832321437914821351 63921}. \end{aligned}$$

NIST-V parameters. We targeted 512-bit primes using $n = 4$ and 6. Once again, combining our CHM runs with $n = 6$ proved to be the best option for finding SQIsign parameters at this level. None of the twins found at the 128-bit level combined with $n = 4$ to produce any SQIsign friendly primes. From the set of 85-bit twins found in the $B = 547$ CHM run, the 510-bit prime $p = 2r^6 - 1$ with $r = 31929740427944870006521856$ is particularly attractive. The power of two here is the largest found from this run. We have

$$\begin{aligned} p + 1 &= 2^{91} \cdot 19^6 \cdot 61^6 \cdot 89^6 \cdot 101^6 \cdot 139^6 \cdot 179^6 \cdot 223^6 \cdot 239^6 \cdot 251^6 \cdot 281^6, \text{ and} \\ p - 1 &= 2 \cdot 3^2 \cdot 5 \cdot 7 \cdot 13 \cdot 23 \cdot 29 \cdot 31 \cdot 41 \cdot 53 \cdot 109 \cdot 149 \cdot 157 \cdot 181 \cdot 269 \cdot 317 \cdot 331 \\ &\quad \cdot 463 \cdot 557 \cdot 727 \cdot 10639 \cdot 31123 \cdot 78583 \cdot 399739 \cdot 545371 \cdot 550657 \cdot \mathbf{4291141} \\ &\quad \cdot \mathbf{32208313} \cdot \mathbf{47148917} \cdot \mathbf{69050951} \cdot \mathbf{39618707467} \cdot \mathbf{220678058317} \\ &\quad \cdot \mathbf{107810984992771213} \cdot \mathbf{1779937809321608257}. \end{aligned}$$

The 85-bit twins found in the CHM run with $B = 2^{11}$ were used to try to find NIST-V parameters. The largest power of two that was found in this run which is suitable for SQIsign was $f = 109$. The prime with smallest signing cost metric while having a relatively large power of two is the following 508-bit prime, $p = 2r^6 - 1$ where $r = 26697973900446483680608256$. Here, we have

$$\begin{aligned} p+1 &= 2^{85} \cdot 17^{12} \cdot 37^6 \cdot 59^6 \cdot 97^6 \cdot 233^6 \cdot 311^{12} \cdot 911^6 \cdot 1297^6, \text{ and} \\ p-1 &= 2 \cdot 3^2 \cdot 5 \cdot 7 \cdot 11^2 \cdot 23^2 \cdot 29 \cdot 127 \cdot 163 \cdot 173 \cdot 191 \cdot 193 \cdot 211 \cdot 277 \cdot 347 \cdot 617 \\ &\quad \cdot 661 \cdot 761 \cdot 1039 \cdot 4637 \cdot 5821 \cdot 15649 \cdot 19139 \cdot 143443 \cdot 150151 \cdot \mathbf{3813769} \\ &\quad \cdot \mathbf{358244059} \cdot \mathbf{992456937347} \cdot \mathbf{3532404817819653698238975 07} \\ &\quad \cdot \mathbf{86010200695145744013716588914 03021}. \end{aligned}$$

10.5.3 Performance estimates

We would ideally implement our primes using the SQIsign code provided in [126] to determine how well these parameters perform in practice. However, the current implementation is specifically tailored towards the particular primes that are being used, and is limited to NIST-I parameter sizes. Including our NIST-I primes from Table 10.4 results in failures during key generation, which seem to stem from using parameters with different powers of two. Thus, implementing and benchmarking our parameters would require a major rework of the provided code, which is out of the scope of this work.

NIST-I. The state-of-the-art implementation of SQIsign uses a 254-bit prime that was found using the extended Euclidean algorithm (XGCD) [99, 125] (see Section 10.1). With this method, it is possible to, for example, force $p \pm 1$ and $p \mp 1$ to be divisible by a large power of 2 and 3 (respectively). Indeed, with this approach, a smooth factor of size $\approx \sqrt{p}$ comes for free in both $p \pm 1$. Concretely, the prime p_{3923} used in [126] is of bitsize 254 and has $f = 65$ and $B = 3923$:

$$\begin{aligned} p+1 &= 2^{65} \cdot 5^2 \cdot 7 \cdot 11 \cdot 19 \cdot 29^2 \cdot 37^2 \cdot 47 \cdot 197 \cdot 263 \cdot 281 \cdot 461 \cdot 521 \cdot 3923 \cdot \mathbf{62731} \\ &\quad \cdot \mathbf{96362257} \cdot \mathbf{3924006112952623}, \text{ and} \\ p-1 &= 2 \cdot 3^{65} \cdot 13 \cdot 17 \cdot 43 \cdot 79 \cdot 157 \cdot 239 \cdot 271 \cdot 283 \cdot 307 \cdot 563 \cdot 599 \cdot 607 \cdot 619 \\ &\quad \cdot 743 \cdot 827 \cdot 941 \cdot 2357 \cdot \mathbf{10069}. \end{aligned}$$

As this prime is not constructed by boosting twin-smooth integers $(r, r \pm 1)$, we do not display it in Table 10.4. For comparison with other primes, we provide details of p_{3923}: $\sqrt{B}/f = 0.96$ and $\log_p(T) = 1.32$.

The primes from Table 10.4 provide various alternatives for NIST-I parameters, and we can give simplified estimates for their performance in comparison to p_{3923}. As an example, we will consider p_{479}, the 253-bit prime from Table 10.4 having $B = 479$. With $f = 49$, it features a slightly smaller power of two compared to p_{3923} with $f = 65$. This means that we would have to verify the signature isogeny in 21 blocks of 2^{49}-isogenies, instead of 16 blocks of 2^{65}-isogenies for p_{3923}. Given that the computational bottleneck for this is the generation of the respective kernel points per block, and ignoring the savings of computing 2^{49}-isogenies instead of 2^{65}-isogenies and the relatively cheap recomputation of the challenge isogeny, this results in an estimated slowdown of roughly $21/16 \approx 1.31$. Thus, we expect a modest slowdown from a verification time of 6.7ms (see [126]) to roughly 8.8ms on a modern CPU.

Security level	n	r	$\lceil\log(p)\rceil$	f	B	$\sqrt{B}/f$	$\log_p(T)$
NIST-I	2	1211460311716772790566574529001291776	241	49	1091	0.67	1.28
	2	2091023014142971802357816084152713216	243	49	887	0.61	1.28
	3	3474272816789867297357824	246	43	547	0.54	1.29
	3	10227318375788227199589376	251	31	383	0.63	1.31
	3	21611736033260878876800000	254	31	421	0.66	1.28
	3	20461449125500374748856320	254	46	523	0.50	1.26
	3	26606682403634464748953600	255	40	547	0.58	1.28
	4	1466873880764125184	243	49	701	0.54	1.28
	4	8077251317941145600	253	49	479	0.45	1.30
	4	12105439990105079808	255	61	1877	0.71	1.31
	4	13470906659953016832	256	61	1487	0.63	1.30
NIST-III	3	1374002035005713149550405343373848576	362	37	1277	0.97	1.25
	4	5139734876262390964070873088	370	45	11789	2.41	1.26
	4	12326212283367463507272925184	375	77	55967	3.07	1.31
	4	18080754980295452456023326720	377	61	95569	5.07	1.26
	4	27464400309146790228660255744	379	41	13127	2.79	1.29
	6	2628583629218279424	369	73	13219	1.58	1.27
	6	5417690118774595584	375	79	58153	3.05	1.27
	6	11896643388662145024	382	79	10243	1.28	1.30
	12	5114946480 [128]	389	49	31327	3.61	1.30
NIST-V	6	946978778058060446433280	499	109	703981	7.70	1.25
	6	1223346860574068600780800	502	73	376963	8.41	1.28
	6	2669797390044648368060825	508	85	150151	4.56	1.26
	6	3192974042794487000652185	510	91	550657	8.15	1.25
	6	4134024820090081905679360	512	67	224911	7.08	1.28

TABLE 10.4: A table of SQIsign parameters $p = p_n(r)$ for twin-smooth integers $(r, r \pm 1)$ found using CHM at each security level. The f is the power of two dividing $(p^2-1)/2$ and B is the smoothness bound of the odd cofactor $T \approx p^{5/4}$. It also includes existing primes in the literature.

However, we expect a significant speed-up for signing. The computational bottleneck during the signature generation is the repeated computation of T-isogenies; one computes two T-isogenies per block of 2^f-isogenies in the verification. Since the T-isogeny computation is dominated by its largest prime factor B, and its cost can be estimated by $\sqrt{B}$, the ratio of the signing cost metrics $\sqrt{B}/f$ from Table 10.4 reflects the overall comparison. Given this metric, we expect a speed-up factor of roughly $0.45/0.96 \approx 0.47$. For the running time, this would mean an improvement from 424ms (see [126]) to roughly 199ms on a modern CPU.

We can also consider a different cost-estimate, given by summing the cost $\sqrt{\ell_i}$ for the five biggest (not necessarily distinct) prime factors $\ell_i \mid T$, before dividing by f. The advantage of considering more factors of T is that it constitutes a larger portion of the time it takes to compute a T-isogeny, while the disadvantage is that the cost $\sqrt{\ell}$ becomes increasingly inaccurate for smaller prime factors ℓ. In this metric, the speed-up is smaller, but is still significant. Specifically, we expect a speed-up factor of roughly $2.19/3.04 \approx 0.72$, which would result in an improvement from 424ms to roughly 305ms.

In a nutshell, even though we can only give rough estimates for running times, we expect our NIST-I parameters to achieve much better signing times due to the smaller smoothness bounds B, at the cost of a very modest slowdown for verification due to slightly smaller values of f. In the light of the relatively slow signing times in SQIsign, this option seems worthwhile for applications that require faster signing.

NIST-III and -V. As mentioned earlier, our work showcases the first credible primes for SQIsign at the NIST-III and NIST-V security level. A beneficial feature about most of the primes found in Table 10.4 is that the majority of the smooth factors are relatively small (e.g. $B < 2^{10}$). In comparison, we expect the XGCD method to scale worse for larger security levels, i.e., requiring much larger smoothness bounds. This is similar to the analysis in [105], which shows that while the XGCD approach has reasonable smoothness probabilities for NIST-I parameters, other methods become superior for larger sizes.

We note that there are other 384 and 512-bit primes in the literature for which $p^2 - 1$ is smooth [105, 128]. None of the primes from [105] have a large enough power of two for a suitable SQIsign application. Some primes were found in the context of the isogeny-based public-key encryption scheme Séta [128] that could be suitable for SQIsign. As part of their parameter setup, they required finding $\approx$ 384-bit primes, satisfying some mild conditions outside just requiring $p^2 - 1$ to be smooth. Of the 7 primes that they found, the 389-bit prime, $p = 2r^{12} - 1$ with $r = 5114946480$ appears to be somewhat SQIsign-friendly achieving NIST-III security (see Table 10.4). However, in addition to its worse signing metric, representations of $\mathbb{F}_p$-values require an additional register in this case compared to our primes of bitlengths slightly below 384. Thus, we can expect implementations of $\mathbb{F}_p$-arithmetic to perform significantly worse for this prime.

Remark 10.5.2. The requirement we impose on $p^2 - 1$ being divisible by $2^f \cdot T$ is to ensure that it fits within the current implementation of SQIsign. At present, the SQIsign implementation has a fine-grained optimisation of their ideal-to-isogeny algorithm to the setting with $\ell = 2$. In general, one could instead allow $p^2 - 1$ to be divisible by $L \cdot T$, for a smooth number L with $\gcd(L, T) = 1$. This could open new avenues to find SQIsign-friendly primes, but would require a reconfiguration of the SQIsign code. For example, using the prime found with $r = 20910230141429718023578160841527132\!16$ from Table 10.4, we could use $L = 2^{49} \cdot 3^4 \cdot 5 \mid p^2 - 1$ and still have a large enough smooth factor T to exceed $p^{5/4}$, thereby further minimising the expected slowdown for verification.

Remark 10.5.3. The focus of this work has been on finding parameters for SQIsign but there are other isogeny-based cryptosystems that could benefit from such quadratic twist-style primes. While traditional SIDH [123] is now broken, there have been proposed countermeasures [18, 19, 160] that aim to thwart the attacks from [67, 227, 271]. Currently, these countermeasures use SIDH-style primes, but could potentially benefit from quadratic twist-style primes like those explored in this work for SQIsign. However, these countermeasures require primes of larger sizes, so it is unclear if our CHM-based approach scales to these sizes, especially when aiming to balance the size of the smooth cofactors of $p + 1$ and $p - 1$. Nevertheless, our techniques might give a good starting point for future research in this direction.

10.5.4 Other techniques for finding SQISign parameters

As seen in Section 10.1, we can collect twin smooth integers via different methods, and use them as inputs to $p_n(x)$ to search for primes. Though these alternative methods are expected to have greater smoothness bound, they have certain advantages. Namely, we are able to force larger powers of 2 into $p^2 - 1$ and search for twin smooths of large bitsizes (targeting NIST-III and -V).

Although we expect most primes in this section to perform worse when instantiated in SQIsign compared to the primes from Section 10.5.2, their concrete performance can only be evaluated by integrating them in the SQIsign software from [125, 126]. In this section, we present the best primes found with each approach in the hopes that future implementations of SQIsign benefit from a larger pool of potential primes to choose from. We give a list of these primes in Table 10.5.

XGCD twin smooths. For generating smaller twins, the XGCD approach can be used to yield relatively high smoothness probabilities. Although this increases the smoothness bound compared to CHM, we can choose smooth factors of roughly n bits combined when searching for n-bit twin smooths. This allows us to force larger powers of 2.

As an example, the 261-bit prime $p = 2r^4 - 1$ with $r = 34848218231355211776$ was found using this approach. Here we have

$$\begin{aligned} p + 1 &= 2^{77} \cdot 3^{20} \cdot 23^4 \cdot 151^4 \cdot 157^4 \cdot 233^4 \cdot 2153^4, \text{ and} \\ p - 1 &= 2 \cdot 5^2 \cdot 17 \cdot 41 \cdot 61 \cdot 71 \cdot 97 \cdot 101^2 \cdot 113 \cdot 137 \cdot 257 \cdot 263 \cdot 313 \cdot 353 \cdot 547 \cdot 853 \\ &\quad \cdot 1549 \cdot 2017 \cdot 2081 \cdot 2311 \cdot \mathbf{3019} \cdot \mathbf{24989} \cdot \mathbf{58601} \cdot \mathbf{551134016677928131 3}. \end{aligned}$$

This prime is similar to the primes found in [126], giving a smaller smoothness bound and a larger power of 2 compared to the state-of-the-art. However, it exceeds the size of 256 bits, and thus we expect it to perform significantly worse due to the fact that representations of values in $\mathbb{F}_p$ require an additional register in this case. Additionally, a large majority of the factors in $p^2 - 1$ are relatively large, making isogeny computations rather slow. This is consistent with the primes in [126].

Remark 10.5.4. This approach is revisited in joint work with Eriksen, Meyer and Rodríguez-Henríquez [277] and was used to find parameters for the NIST submission of SQIsign.

PTE twin smooths. As the number of 128-bit twins that were found using CHM is relatively small, in some cases we were not able to find suitable SQIsign parameters. This mainly concerns the setting where we take $n = 4$ to find NIST-V parameters, for which data from the CHM run with $B < 1300$ did not yield any NIST-V SQIsign-friendly instances.

To find more large twins, we can use the PTE approach [105] (see Section 10.1) to find 2^{14}-smooth 128-bit twins, sacrificing the smaller smoothness bounds that were used during our CHM runs. In total, we found 3,648 such 128-bit twins that resulted in a prime of the form $p = 2r^4 - 1$. Of these, two primes show strong potential to be used in SQIsign and are thus also given in Table 10.4.

Larger values of n. We could also consider finding primes of the form $p = 2r^n - 1$ for larger values of n, where the only restriction is that r is a smooth number. Compared to the previous ideas this restriction decreases the amount of guaranteed smoothness, but if n is chosen carefully then we can obtain increased smoothness probabilities. The polynomial $p_n(x)^2 - 1$ is highly related to the cyclotomic polynomials Φ_d for $d \mid n$ as

$$p_n(x)^2 - 1 = 2x^n(2x^n - 2) = 4x^n(x^n - 1) = 4x^n \prod_{d \mid n} \Phi_d.$$

Recall that Φ_d is an irreducible polynomial of degree $\phi(d)$, where ϕ denotes Euler's totient function. Therefore, the largest irreducible factor of $p_n(x)^2 - 1$ is of degree $\phi(n)$. This in turn means that the largest factor that $p = 2r^n - 1$ can possibly have is around the size of $r^{\phi(n)} \approx p^{\phi(n)/n}$. Therefore, we would like to minimise the value $\phi(n)/n$.

As we allow n to increase, this value can get arbitrarily low. Indeed, setting $n = P_k$, where P_k denotes the k-th primorial,[6] we find that

$$\frac{\phi(P_k)}{P_k} = \prod_{i=1}^{k} \frac{p_i - 1}{p_i} = \prod_{i=1}^{k} \left(1 - \frac{1}{p_i}\right),$$

and as k tends towards infinity, we see that

$$\lim_{k \to \infty} \frac{\phi(P_k)}{P_k} = \prod_{i=1}^{\infty} \left(1 - \frac{1}{p_i}\right) = \frac{1}{\zeta(1)},$$

where $\zeta(s)$ denotes the Riemann-Zeta function, which has a pole at $s = 1$.

However, we cannot allow n to become too large as we still need a sufficiently large range of inputs, so that there exists a smooth r such that $2r^n - 1$ is prime. Therefore, consider a bound $n < B_n$ where B_n is chosen such that we still have a large enough search space. Based on the multiplicative property of the totient function, the fact that $\phi(q) = q - 1$ when q is prime, and that

$$\frac{\phi(n)}{n} = \frac{\phi(\mathrm{rad}(n))}{\mathrm{rad}(n)},$$

[6]The k-th primorial P_k is defined as the product of the first k primes.

where $\operatorname{rad}(n)$ is the product of all distinct primes dividing n, the optimal choices of n are in the set

$$\{2^{e_1}3^{e_2}\dots p_k^{e_k} < B_n \mid P_k < B_n < P_{k+1}, e_i \geq 1\}.$$

As an example, we look for NIST-V parameters $p \in [2^{500}, 2^{512}]$. If we want at least a range of size 2^{25} such that $2r^n - 1 \in [2^{500}, 2^{512}]$, we see that we have to have $n < B_n = 20$. Therefore, our set of optimal choices of n becomes

$$n \in \{2 \cdot 3, 2^2 \cdot 3, 2 \cdot 3^2\} = \{6, 12, 18\}.$$

Using $n = 6$, the range of suitable r-values becomes large enough that we cannot search through all of them. Thus, searches would require further restrictions on the suitable r-values, such as only considering twin-smooths.

For $n \in \{12, 18\}$, we can exhaust the full search space, and obtain several promising candidates. These are included in Table 10.4. Among all of these, the 510-bit prime $p = 2r^{12} - 1$ with $r = 5594556480768$ seems very suitable for NIST-V. It has a low cost factor and has a large power of three, which could be beneficial for SQIsign implementations. Here we have

$$\begin{aligned}
p + 1 &= 2^{97} \cdot 3^{60} \cdot 239^{12} \cdot 571^{12} \cdot 659^{12}, \text{ and} \\
p - 1 &= 2 \cdot 5^2 \cdot 7 \cdot 13^2 \cdot 17 \cdot 19 \cdot 43 \cdot 83 \cdot 103 \cdot 109 \cdot 139^2 \cdot 151 \cdot 223 \cdot 277 \cdot 317 \cdot 1249 \\
&\quad \cdot 1373 \cdot 2311 \cdot 3067 \cdot 4133 \cdot 28279 \cdot 28447 \cdot 40087 \cdot 60089 \cdot 69073 \cdot 88469 \\
&\quad \cdot \mathbf{2226517} \cdot \mathbf{5856073} \cdot \mathbf{6242671} \cdot \mathbf{14237127193} \cdot \mathbf{25752311173} \\
&\quad \cdot \mathbf{63101553683977} \cdot \mathbf{38380249844433998662503841}.
\end{aligned}$$

Security level	n	r	$\lceil \log(p) \rceil$	f	B	$\sqrt{B}/f$	$\log_p(T)$
NIST-I	4	348482182313552117776	261	77	2311	0.62	1.30
NIST-III	12	2446635904	376	85	9187	1.13	1.29
NIST-V	4	11421678154858170943951287580127979 1104	507	65	75941	4.24	1.26
	4	123794274387474298912742543819242587136	508	41	15263	3.01	1.29
	12	5594556480768	510	97	88469	3.07	1.29
	18	335835120	511	73	24229	2.13	1.29

Table 10.5: A table of SQIsign parameters $p = p_n(r)$ found using twin-smooth integers $(r, r \pm 1)$ at each security level. The twins used here were not found using CHM. The other quantities are just as in Table 10.4.

Remark 10.5.5. Unlike the CHM method and other similar methods, we cannot generate more values to input into this technique, as the amount is small enough to quickly exhaust the full search space. This is in stark contrast to CHM, which, given more computing power, could potentially generate more twin smooths of given sizes to give new suitable SQIsign parameters. Hence, we conclude that the CHM method with smaller values of n will ultimately give rise to new, better

SQIsign parameters than the ones found with the larger values of n.

Chapter 11

Faster verification for SQIsign

In this chapter, we aim to accelerate SQIsign verification as much as possible, and are willing to accept a decrease in the efficiency of signing. In this way, we focus on applications that require fast verification and small signatures. Unlike in the previous Chapter 10, where we fix signing to be performed over $\mathbb{F}_{p^2}$ for efficiency, in this chapter we instead maximise the efficiency of verification by allowing signing to occur over an extension field of $\mathbb{F}_{p^2}$. This chapter presents joint work with Jonathan Komada Eriksen, Michael Meyer, and Krijn Reijnders entitled

AprèsSQI: Extra Fast Verification for SQIsign Using Extension-Field Signing

as published at Eurocrypt 2024 [92], where it received the Best Early-Career Paper Award. The software associated to this code was published as an artifact at Eurocrypt 2024 [93]. The only differences between this chapter and the published paper arise from the correction of typographical errors, the removal of superfluous preliminaries and moving the appendix to the body of the paper.

In light of recent work. Since the publication of the paper discussed in this chapter, an impressive new variant of SQIsign has been developed: SQIsign2D [17, 142, 247]. It achieves similar signature sizes to SQIsignHD, slower signing than SQIsignHD though faster than SQIsign, and the faster verification than both SQIsign and SQIsignHD. In light of these advancements, it would be interesting to understand in future work whether an optimised implementation of AprèsSQI in a low-level programming language can still obtain the fastest verification to target applications where verification time is key. Additionally, a new ideal-to-isogeny algorithm was introduced by Onuki and Nakagawa [252] that uses two-dimensional isogenies. When applied to SQIsign, key generation and the signing procedures are around twice as fast as those in the NIST submission for Level I security, and there is no change to verification time. The advantage becomes greater for higher security levels. It would be interesting to see the effect of combining this new ideal-to-isogeny algorithm with the techniques developed in this chapter to accelerate SQIsign verification.

Introduction

SQIsign is most interesting in scenarios that require small signature sizes and fast verification, particularly in those applications where the performance of signing is not the main concern. A few common examples include long-term signatures, specifically public-key certificates, code updates for small devices, authenticated communication with embedded devices or other microcontrollers that solely run verification, and smart cards. For such use cases it is imperative to bring down the cost of verification as much as possible.

The bottleneck of verification in SQIsign is the computation of an isogeny of fixed degree 2^e with $e \approx \frac{15}{4}\log(p)$, where p denotes the prime one is working over, e.g. $\log(p) \approx 256$ for NIST Level I security. However, the rational 2-power torsion, from here on denoted as the $2^\bullet$-torsion, is limited, since we work with supersingular elliptic curves over $\mathbb{F}_{p^2}$ of order $(p+1)^2$ and $(p-1)^2$. This sets a theoretical limit of $2^{\log p}$ for the $2^\bullet$-torsion. Therefore, the verifier needs to perform several *blocks* of degree $2^\bullet$ to complete the full 2^e-isogeny, where each of these blocks involves costly steps such as computing a $2^\bullet$-torsion basis or isogeny kernel generator. Hence, in general, a smaller number of blocks improves the performance of verification.

On the other hand, the bottleneck in signing is the computation of several T-isogenies for odd smooth $T \approx p^{5/4}$. Current implementations of SQIsign therefore require $T \mid (p-1)(p+1)$, such that $\mathbb{F}_{p^2}$-rational points are available for efficient T-isogeny computations. The performance of this step is dominated by the smoothness of T, i.e., its largest prime factor.

While this additional divisibility requirement theoretically limits the maximal $2^\bullet$-torsion to roughly $p^{3/4}$, current techniques for finding SQIsign-friendly primes suggest that achieving this with acceptable smoothness of T is infeasible [78, 86, 99, 105, 125]. In particular, the NIST submission of SQIsign achieving Level I security uses a prime with rational 2^{75}-torsion and 1973 as largest factor of T. In this case $e = 975$, meaning that the verifier has to perform $\lceil e/75 \rceil = 13$ costly isogeny blocks. Increasing the $2^\bullet$-torsion further is difficult as it decreases the probability of finding a smooth and large enough T for current implementations of SQIsign.

Contributions

In this work, we deploy a range of techniques to increase the $2^\bullet$-torsion and push the SQIsign verification cost far below the state of the art at the time of publication. Alongside these technical contributions, we aim to give an accessible description of SQIsign, focusing primarily on verification, which solely uses elliptic curves and isogenies and does not require knowledge of quaternion algebras.

Even though we target faster verification, our main contribution is signing with field extensions. From this, we get a much weaker requirement on the prime p, which in turn enables us to increase the size of the $2^\bullet$-torsion.

Focusing on NIST Level I security, we study the range of possible $2^\bullet$-torsion to its theoretical maximum, and measure how its size correlates to verification time. We do this using an implementation which uses an equivalent to the number of field multiplications as a cost metric. Compared to the state of the art, increasing the $2^\bullet$-torsion alone makes verification almost 1.7 times faster. Further, we implement the new signing procedure as a proof-of-concept in SageMath and show that signing times when signing with field extensions are in the same order of magnitude as when signing only using operations in $\mathbb{F}_{p^2}$.

For verification, in addition to implementing some known general techniques for improvements compared to the reference implementation provided in the NIST submission of SQIsign, we show that increasing the $2^\bullet$-torsion also opens up a range of optimisations that were previously not possible. For instance, large $2^\bullet$-torsion allows for an improved challenge isogeny computation and improved basis and kernel generation. Furthermore, we show that size-speed trade-offs as first proposed by De Feo, Kohel, Leroux, Petit, and Wesolowski [125] become especially worthwhile for large $2^\bullet$-torsion. When pushing the $2^\bullet$-torsion to its theoretical maximum, this even allows for uncompressed signatures, leading to significant speed-ups at the cost of roughly doubling the signature sizes.

For two specific primes with varying values of $2^\bullet$-torsion, we combine all these speed-ups, and measure the performance of verification. Compared to the implementation of the SQIsign NIST submission [78], we reach a speed-up up to a factor 2.70 at NIST Level I when keeping the signature size of 177 bytes. When using our size-speed trade-offs, we reach a speed-up by a factor 3.11 for signatures of 187 bytes, or a factor 4.46 for uncompressed signatures of 322 bytes. Compared to the state of the art [221], these speed-ups are factors 2.07, 2.38 and 3.41, respectively.

Related work

De Feo, Kohel, Leroux, Petit, and Wesolowski [125] published the first SQIsign implementation, superseded by the work of De Feo, Leroux, Longa, and Wesolowski [126]. Subsequently, Lin, Wang, Xu, and Zhao [221] introduced several improvements for this implementation. The NIST submission of SQIsign [78] – SQIsign (NIST)– features a new implementation that does not rely on any external libraries. Since this is the latest and best documented implementation, we will use it as a baseline for performance comparison. Since the implementation by Lin, Wang, Xu, and Zhao [221] is not publicly available, we included their main improvement for verification in SQIsign (NIST), and refer to this as SQIsign (LWXZ).

Dartois, Leroux, Robert, and Wesolowski [116] recently introduced SQIsignHD, which massively improves the signing time in SQIsign, in addition to a number of other benefits, but at the cost of a slowdown in verification. This could make SQIsignHD an interesting candidate for applications that prioritise the combined cost of signing and verification over the sole cost of verification.

Recent work by Eriksen, Panny, Sotáková, and Veroni [150] explored the feasibility of computing the Deuring correspondence (see Section 3.2) for *general* primes p using higher extension fields. We apply the same techniques and tailor them to *specialised* primes for use in the signing procedure of SQIsign.

Outline

The rest of the chapter is organised as follows. Section 11.1 recalls the necessary background, including a high-level overview of SQIsign. Section 11.2 describes how using field extensions in signing affects the cost and relaxes requirements on the prime. Section 11.3 analyses how the size of the $2^\bullet$-torsion correlates to verification time. Section 11.4 presents optimisations enabled by the increased $2^\bullet$-torsion, while Section 11.5 gives further optimisations enabled by increased signature sizes. Finally, Section 11.6 gives some example parameters, and measures their performance compared to the state of the art.

Availability of software

We make our Python and SageMath software publicly available under the MIT licence at

https://github.com/TheSICQ/ApresSQI.

11.1 Preliminaries

For this chapter, we require the preliminaries on elliptic curves, given in Section 2.2, and isogenies between them from Section 2.8.1. As usual, we focus in particular on supersingular elliptic curves defined over $\mathbb{F}_{p^2}$, introduced in Sections 2.9 and 4.1. In this work, we accelerate verification in the signature scheme SQIsign. As such, a reader should be familiar with the tools used to construct SQIsign, most notably the Deuring correspondence in Section 3.2 and the generalised KLPT algorithm from Section 5.1.1, and details on the signature scheme itself given in Section 5.2. In SQIsign, we use supersingular elliptic curves in Montgomery form

$$E_A : y^2 = x^3 + Ax^2 + x = x(x - \alpha)(x - 1/\alpha)$$

for $A, \alpha \in \mathbb{F}_{p^2}$, to exploit fast x-only arithmetic.

The runtime of the best-known attacks against SQIsign, as given in Section 5.2.2.1, depends only on the size of p. Furthermore, with high probability, they do not recover the original secret isogeny, but rather a different isogeny between the same curves. Therefore, their complexity should be unaffected by the changes we introduce to the SQIsign protocol in Section 11.2. Indeed, for these attacks it does not matter whether the original secret isogeny had kernel points defined over a larger extension field. In short, the changes to SQIsign in this chapter do not affect its security.

11.1.1 SQIsign-friendly primes

Next, we briefly recall details on the parameter requirements in SQIsign. We refer to Section 5.2.2.2 for a detailed description of their origins. For a security level λ, the following parameters are needed:

- A prime p of bitsize $\log(p) \approx 2\lambda$ with $p \equiv 3 \mod 4$.
- The torsion group $E[2^f]$ as large as possible, for E a supersingular elliptic curve defined over $\mathbb{F}_{p^2}$, that is $2^f \mid p + 1$.
- A smooth odd factor $T \mid (p^2 - 1)$ of size roughly $p^{5/4}$.
- The degree of φ_{com}, $D_{\text{com}} \mid T$, of size roughly $2^{2\lambda} \approx p$.
- The degree of φ_{chall}, $D_{\text{chall}} \mid 2^f T$, of size roughly $2^\lambda \approx p^{1/2}$.
- Coprimality between D_{com} and D_{chall}.

To achieve NIST Level I, III, and V security, we set the security parameter as $\lambda = 128, 192, 256$, respectively. Concretely, this means that, for each of these security parameters, we have $\log(p) \approx 256, 384, 512$, and $\log(T) \approx 320, 480, 640$, with f as large as possible given the above restrictions. The smoothness of T directly impacts the signing time, and the problem of finding primes p with a large enough T that is reasonably smooth is difficult. We refer to Chapter 10 or other recent work on this problem [78, 99, 105, 125, 126, 277] for techniques to find suitable primes.

The crucial observation for this work is that T occupies space in $p^2 - 1$ that limits the size of f, hence current SQIsign primes balance the smoothness of T with the size of f.

Remark 11.1.1. SQIsign (NIST) further requires $3^g \mid p+1$ such that $D_{\text{chall}} = 2^f \cdot 3^g \geq p^{1/2}$ and $D_{\text{chall}} \mid p+1$. While this is not a strict requirement in the theoretical sense, it facilitates efficiency of computing φ_{chall}. From this point on, we ensure that this requirement is always fulfilled.

Remark 11.1.2. Since the curves E and their twists E^t satisfy

$$E(\mathbb{F}_{p^2}) \cong \mathbb{Z}/(p \pm 1)\mathbb{Z} \oplus \mathbb{Z}/(p \pm 1)\mathbb{Z} \text{ and } E^t(\mathbb{F}_{p^2}) \cong \mathbb{Z}/(p \mp 1)\mathbb{Z} \oplus \mathbb{Z}/(p \mp 1)\mathbb{Z},$$

and we work with both simultaneously, choosing T and f is often incorrectly described as choosing divisors of p^2-1. There is a subtle issue here: even if 2^f divides p^2-1, $E[2^f]$ may not exist as a subgroup of $\langle E(\mathbb{F}_{p^2}), \rho^{-1}(E^t(\mathbb{F}_{p^2}))\rangle \subseteq E(\mathbb{F}_{p^4})$, where $\rho : E \to E^t$ is the twisting isomorphism. While this does not usually matter in the case of SQIsign (we pick 2^f as a divisor of $p+1$, and T is odd), this becomes a problem when working over multiple extension fields. In Section 11.2.2, we make this precise and reconcile it using Theorem 11.2.3.

11.1.2 Computing rational isogenies from irrational generators

Finally, to facilitate signing with field extensions, we recall the techniques for computing $\mathbb{F}_{p^2}$-rational isogenies, i.e., isogenies defined over $\mathbb{F}_{p^2}$, generated by *irrational* kernel points, that is, not defined over $\mathbb{F}_{p^2}$. In the context of this chapter, we again stress that such isogenies will only be computed by the signer; the verifier will only work with points in $\mathbb{F}_{p^2}$.

The main computational task of most isogeny-based cryptosystems (including SQIsign) lies in evaluating isogenies given the generators of their kernels. Explicitly, given an elliptic curve $E/\mathbb{F}_q$, a point $K \in E(\mathbb{F}_{q^k})$ such that $\langle K\rangle$ is *defined over* $\mathbb{F}_q$,[1] and a list of points $(P_1, P_2, \ldots, P_n)$ in E, we wish to compute the list of points $(\varphi(P_1), \varphi(P_2), \ldots, \varphi(P_n))$, where φ is the separable isogeny with $\ker\varphi = \langle K\rangle$. Since we work with curves E whose p^2-Frobenius π is equal to the multiplication-by-$(-p)$ map, *every* subgroup of E is closed under the action of the Galois group $\mathrm{Gal}(\bar{\mathbb{F}}_p/\mathbb{F}_{p^2})$, hence every isogeny from E can be made $\mathbb{F}_{p^2}$-rational, by composing with the appropriate isomorphism.

Computing isogenies of smooth degree. Recall from Theorem 2.6.8 that the isogeny factors as a composition of small prime degree isogenies, which we compute using Vélu-style algorithms. For simplicity, for the rest of the section, we therefore assume that $\langle K\rangle$ is a subgroup of order $N > 2$, where N is a small prime.

At the heart of these Vélu-style isogeny formulas is evaluating the kernel polynomial. Pick any subset $S \subseteq \langle K\rangle$ such that $\langle K\rangle = S \sqcup -S \sqcup \{\infty\}$. Then the kernel polynomial can be written as

$$f_S(X) = \prod_{P\in S}(X - x(P)). \tag{11.1}$$

Here, the generator K can be either a rational point, i.e., lying in $E(\mathbb{F}_q)$, or an irrational point, i.e., lying in $E(\mathbb{F}_{q^k})$ for $k > 1$, but whose group $\langle K\rangle$ is defined over $\mathbb{F}_q$. Next, we discuss how to solve the problem efficiently in the latter case.

Irrational generators. For $K \notin E(\mathbb{F}_q)$ of order N, we can speed up the computation of the

[1] That is, the group $\langle K\rangle$ is closed under the action of the Galois group $\mathrm{Gal}(\bar{\mathbb{F}}_q/\mathbb{F}_q)$.

kernel polynomial using the action of Frobenius. This was used in two recent works [13, 150], though the general idea was used even earlier by Tsukazaki [312].

As $\langle K\rangle$ is defined over $\mathbb{F}_q$, we know that the q-power Frobenius π acts as an endomorphism on $\langle K\rangle \subseteq E(\mathbb{F}_{p^k})$ and so maps K to a multiple $[\gamma]K$ for some $\gamma \in \mathbb{Z}$. This fully determines the action on $\langle K\rangle$, i.e., $\pi|_{\langle K\rangle}$ acts as $P \mapsto [\gamma]P$ for all $P \in \langle K\rangle$. For the set S as chosen above, this action descends to an action on its x-coordinates $X_S = \{x(P) \in \mathbb{F}_{q^k} \mid P \in S\}$ and thus partitions X_S into orbits $\{x(P), x([\gamma]P), x([\gamma^2]P), \dots\}$ of size equal to the order of γ in $(\mathbb{Z}/N\mathbb{Z})^\times/\{1,-1\}$.

If we pick one representative $P \in S$ per orbit, and call this set of points S_0, we can compute the kernel polynomial in Equation (11.1) as a product of the minimal polynomials $\mu_{x(P)}$ of the $x(P) \in \mathbb{F}_{q^k}$ for these $P \in S_0$, with each $\mu_{x(P)}$ defined over $\mathbb{F}_q$, as

$$f_S(X) = \prod_{P\in S_0} \mu_{x(P)}(X), \tag{11.2}$$

where μ_β denotes the minimal polynomial of β over $\mathbb{F}_q$.

To compute $f_S(\alpha)$ for $\alpha \in \mathbb{F}_q$, we only require the smaller polynomial $f_{S_0}(X)$ and compute the norm

$$\mathrm{Norm}_{\mathbb{F}_{q^k}/\mathbb{F}_q}(f_{S_0}(\alpha)) = \prod_{\pi\in G} \pi(f_{S_0}(\alpha)) = \prod_{P\in S_0}\prod_{\pi\in G}(\alpha - \pi(x(P))) = \prod_{P\in S_0} \mu_{x(P)}(\alpha),$$

where $G = \mathrm{Gal}(\mathbb{F}_{q^k}/\mathbb{F}_q)$, as per Banegas, Gilchrist, Dévéhat, and Smith [13]. This allows us to compute the image under f_S of x-values of points in $E(\mathbb{F}_q)$, but only works for values in $\mathbb{F}_q$. To evaluate $f_S(\alpha)$ for general $\alpha \in \bar{\mathbb{F}}_p$, i.e., to compute the image of a point in $E(\bar{\mathbb{F}}_p)$, we instead compute the larger polynomial $f_S(X)$, where we use Shoup's algorithm [290] to compute each $\mu_{x(P)}$ given $x(P)$. Computing $f_S(X)$ requires a total of $O(\ell k) + \widetilde{O}(\ell)$ operations, with k such that each $x(P) \in \mathbb{F}_{q^k}$. Evaluating f_S at α takes $\widetilde{O}(\ell k')$ operations, with k' the smallest value such that $\alpha \in \mathbb{F}_{q^{k'}}$ [150, Section 4.3].

Remark 11.1.3. The biggest drawback to using this technique is that $\sqrt{\text{élu}}$ is no longer practical. If the kernel point P generating the isogeny is defined over $\mathbb{F}_{p^{2k_i}}$, and we want to evaluate the isogeny at a point Q defined over $\mathbb{F}_{p^{2k_j}}$, we need to work in the smallest field where they are both defined, namely the compositum $\mathbb{F}_{p^{2\,\mathrm{lcm}(k_i,k_j)}}$.

11.2 Signing with extension fields

By allowing torsion T from extension fields, we enable more flexibility in choosing SQIsign primes p, thus enabling a larger $2^\bullet$-torsion. Such torsion T requires us to compute rational isogenies with kernel points in extension fields $\mathbb{F}_{p^{2k}}$. This section describes how to adapt SQIsign's signing procedure to enable such isogenies, and the increased cost this incurs. In particular, we describe two approaches for T: allowing torsion T from a particular extension field $\mathbb{F}_{p^{2k}}$, or from all extension fields $\mathbb{F}_{p^{2n}}$ for $1 \le n \le k$. The first approach means that we can look for T dividing an integer of bit size $\Theta(k \log p)$, and the second approach allows for $\Theta(k^2 \log p)$. In Section 11.3, we explore how increased $2^\bullet$-torsion affects the performance of verification.

11.2.1 Changes in the signing procedure

Recall from Section 5.2 that the signing operation in SQIsign requires us to work with both elliptic curves and quaternion algebras, and to translate back and forth between these worlds. Note that the subroutines that work solely with objects in the quaternion algebra $\mathcal{B}_{p,\infty}$, including all operations in the KLPT variants used for key generation and signing, are indifferent to what extension fields the relevant torsion groups lie in. Hence, a large part of signing is unaffected by torsion from extension fields.

In fact, the only subroutines that are affected by moving to extension fields are those relying on $\mathsf{IdealToIsogeny}_D$ introduced in Section 3.2.1, which translates $\mathcal{O}_0$-ideals I of norm dividing D to their corresponding isogenies φ_I. Following the discussion in Sections 3.2.2 and 5.2, $\mathsf{IdealToIsogeny}_D$ is not used during verification, and is used only in the following parts of signing:

Commitment: The signer translates a random ideal of norm D_{com} to its corresponding isogeny, using one execution of $\mathsf{IdealToIsogeny}_{D_{\mathrm{com}}}$.

Response: The signer translates an ideal of norm 2^e to its corresponding isogeny, requiring $2\lceil e/f\rceil$ executions of $\mathsf{IdealToIsogeny}_T$.

Remark 11.2.1. We choose parameters such that $2^f \mid p+1$ and $D_{\mathrm{chall}} \mid p+1$, so that $E[2^f]$ and $E[D_{\mathrm{chall}}]$ are defined over $\mathbb{F}_{p^2}$. As a result, the verifier only works in $\mathbb{F}_{p^2}$ and the added complexity of extension fields applies only to the signer.

Adapting ideal-to-isogeny translations to field extensions. To facilitate signing with field extensions, we slightly adapt $\mathsf{IdealToIsogeny}_D$ so that it works with prime powers separately. Note that the additional cost of this is negligible compared to the cost of computing the isogeny from the generators because finding the action of the relevant endomorphisms consists of simple linear algebra. See Algorithm 11.1 for details.

Algorithm 11.1 $\mathsf{IdealToIsogeny}_D(I)$

Input: I, a left $\mathcal{O}_0$-ideal of norm dividing D.
Output: The corresponding isogeny φ_I.

```
 1: Compute $\alpha$ such that $I = \mathcal{O}_0\langle \alpha, \mathrm{nrd}\, I\rangle$
 2: Let $\mathbf{A} = [1, i, \frac{i+j}{2}, \frac{1+k}{2}]$ denote a basis of $\mathcal{O}_0$
 3: Compute $\mathbf{v}_{\bar\alpha} := [x_1, x_2, x_3, x_4]^T \in \mathbb{Z}^4$ such that $\mathbf{A}\mathbf{v}_{\bar\alpha} = \bar\alpha$
 4: for $\ell^e \mid\mid D$ do
 5:     $\bar\alpha|_{\langle P_{\ell^e}, Q_{\ell^e}\rangle} := x_1\mathbf{I} + x_2(i|_{\langle P_{\ell^e}, Q_{\ell^e}\rangle}) + x_3(\frac{i+j}{2}|_{\langle P_{\ell^e}, Q_{\ell^e}\rangle}) + x_4(\frac{1+k}{2}|_{\langle P_{\ell^e}, Q_{\ell^e}\rangle})$
 6:     Let $a, b, c, d$ be integers such that $\bar\alpha|_{\langle P_{\ell^e}, Q_{\ell^e}\rangle} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$
 7:     $K_{\ell^e} := [a]P_{\ell^e} + [c]Q_{\ell^e}$
 8:     if $\mathrm{ord}(K_{\ell^e}) < \ell^e$ then
 9:         $K_{\ell^e} = [b]P_{\ell^e} + [d]Q_{\ell^e}$
10:     end if
11: end for
12: Set $\varphi_I$ to be the isogeny generated by the points $K_{\ell^e}$.
13: return $\varphi_I$
```

In Line 5 of Algorithm 11.1, the notation $\beta|_{\langle P_{\ell^e}, Q_{\ell^e}\rangle}$ refers to the action of an endomorphism β on a fixed basis P_{ℓ^e}, Q_{ℓ^e} of $E[\ell^e]$. This action is described by a matrix in $M_2(\mathbb{Z}/\ell^e\mathbb{Z})$. These matrices can be precomputed, hence the only operations in which the field of definition of $E[\ell^e]$ matters are the point additions in Lines 7 and 9, and isogenies generated by each K_{ℓ^e} in Line 12.

11.2.2 Increased torsion availability from extension fields

We detail the two approaches to allow torsion groups from extension fields, which permits more flexibility in choosing the final prime p.

Working with a single field extension of $\mathbb{F}_{p^2}$. Although the choice of solely working in $\mathbb{F}_{p^2}$ occurs naturally, as it is the smallest field over which every isomorphism class of supersingular elliptic curves has a model, there is no reason *a priori* that this choice is optimal. Instead, we can choose to work in the field $\mathbb{F}_{p^{2k}}$. We emphasise that this does *not* affect signature sizes; the only drawback is that we now perform more expensive $\mathbb{F}_{p^{2k}}$-operations during signing in IdealToIsogeny. The upside, however, is a relaxed prime requirement: we are no longer bound to $E[T] \subseteq \langle E(\mathbb{F}_{p^2}), \rho^{-1}(E^t(\mathbb{F}_{p^2}))\rangle$ and can instead use

$$E[T] \subseteq \langle E(\mathbb{F}_{p^{2k}}), \rho^{-1}(E^t(\mathbb{F}_{p^{2k}}))\rangle.$$

By Equations (4.1) and (4.2), we have $E(\mathbb{F}_{p^{2k}}) \cong E[p^k \pm 1]$ and $E^t(\mathbb{F}_{p^{2k}}) \cong E[p^k \mp 1]$, thus we simply get

$$E[T] \subseteq E\left[\frac{p^{2k}-1}{2}\right],$$

since $\langle E[A], E[B]\rangle = E[\mathrm{lcm}(A, B)]$. Hence, by using torsion from $E(\mathbb{F}_{p^{2k}})$, we increase $T \mid (p^2-1)/2$ to $T \mid (p^{2k}-1)/2$. This implies there are $2(k-1)\log(p)$ more bits available to find T with adequate smoothness.

Working with multiple field extensions of $\mathbb{F}_{p^2}$. Instead of fixing a single higher extension field $\mathbb{F}_{p^{2k}}$, we can also choose to work with multiple field extensions, in particular all fields $\mathbb{F}_{p^{2n}}$, where $1 \leq n \leq k$. The torsion group we can access by this relaxed requirement is described by the following definition.

Definition 11.2.2. Let E be a supersingular elliptic curve over $\mathbb{F}_{p^2}$ and let E_n^t denote an arbitrary quadratic twist of E over $\mathbb{F}_{p^{2n}}$ with respect to the twisting isomorphism $\rho_n : E \to E_n^t$. We define the *k-available torsion* of E to be the group generated by $E(\mathbb{F}_{p^{2n}})$ and $\rho_n^{-1}(E_n^t(\mathbb{F}_{p^{2n}}))$ for $1 \leq n \leq k$.

Any point P in the k-available torsion can thus be written as a sum

$$P = \sum_{i=1}^{k}(P_i + \rho_i^{-1}(P_i^t))$$

of points $P_i \in E(\mathbb{F}_{p^{2i}})$ and $P_i^t \in E_i^t(\mathbb{F}_{p^{2i}})$. Since the twisting isomorphism keeps the x-coordinate fixed, the computation of this isomorphism ρ_i can be ignored when using x-only arithmetic, and we simply obtain a sum of points whose x-coordinates lie in $\mathbb{F}_{p^{2n}}$ for $1 \leq n \leq k$. This justifies the name k-available torsion, as we do not have to go beyond $\mathbb{F}_{p^{2k}}$ to do arithmetic with P by working with the summands separately.

The structure of the k-available torsion is completely captured by the following result.

Theorem 11.2.3. *Let $p > 2$ be a prime, and let $E/\mathbb{F}_{p^2}$ be a supersingular curve with $\operatorname{tr}\pi = \pm 2p$, where π is the Frobenius endomorphism. Then the k-available torsion is precisely the group $E[N]$ with*

$$N = \prod_{n=1}^{k} \Phi_n(p^2)/2,$$

where Φ_n denotes the n-th cyclotomic polynomial.

We first give a proof of the following lemma.

Lemma 11.2.4. *For any integer $m \geq 2$, we have the following identity*

$$\operatorname{lcm}\left(\{m^n - 1\}_{n=1}^{k}\right) = \prod_{n=1}^{k} \Phi_n(m)$$

where Φ_n denotes the n-th cyclotomic polynomial.

Proof. We denote the left-hand side and right-hand side of the equation in the statement of the lemma by LHS and RHS, respectively. We show that any prime power dividing one side, also divides the other.

For any prime ℓ and $e > 0$, if ℓ^e divides the LHS, then, by definition, it divides $m^i - 1 = \prod_{d|i} \Phi_d(m)$ for some $1 \leq i \leq k$. Hence, it also divides the RHS. Conversely, if ℓ^e divides the RHS, then ℓ^e also divides the LHS. To show this, we need to know when $\Phi_i(m)$ and $\Phi_j(m)$ are coprime. We note that

$$\gcd(\Phi_i(m), \Phi_j(m)) \mid R$$

where R is the resultant of $\Phi_i(X)$ and $\Phi_j(X)$, and a classic result by Apostol [4, Theorem 4], tells us that

$$\operatorname{Res}(\Phi_i(X), \Phi_j(X)) > 1 \Rightarrow i = jm$$

for $i > j$ and some integer m.

Using this, if ℓ^e divides the RHS, then it also divides the product

$$\prod_{n=1}^{\lfloor k/d \rfloor} \Phi_{dn}(m),$$

for some integer d, and this product divides the LHS, as it divides $m^{d\lfloor k/d \rfloor} - 1$. ☐

We can now conclude the proof of Theorem 11.2.3.

Proof. From the structure of $E(\mathbb{F}_{p^{2n}})$ (see Equation (4.1)), where E is as in the statement, the k-available torsion can be seen as the group generated by the full torsion groups

$$E[p^n \pm 1]$$

for $1 \leq n \leq k$. Using the fact that

$$\langle E[A], E[B] \rangle = E[\operatorname{lcm}(A, B)],$$

we see that the k-available torsion is $E[N]$ where

$$N = \operatorname{lcm}\left(\{p^n - 1\}_{n=1}^k \cup \{p^n + 1\}_{n=1}^k\right) = \operatorname{lcm}\left(\{p^{2n} - 1\}_{n=1}^k\right)/2,$$

where the last equality only holds for $p > 2$. Applying Lemma 11.2.4 with $m = p^2$, we obtain

$$N = \prod_{k=1}^{n} \Phi_k(p^2)/2.$$

$\square$

We find that using all extension fields $\mathbb{F}_{p^{2n}}$ for $1 \leq n \leq k$ increases $T \mid p^2 - 1$ to $T \mid N$, with N as given by Theorem 11.2.3. Given that

$$\log N = \sum_{n=1}^{k} \log(\Phi_n(p^2)/2) \approx 2\sum_{n=1}^{k} \phi(n)\log(p),$$

and the fact that $\sum_{n=1}^{k} \phi(n)$ is in the order of $\Theta(k^2)$, where ϕ denotes Euler's totient function, we find that $T \mid N$ gives roughly $k^2 \log p$ more bits to find T with adequate smoothness, compared to the $\log(p)$ bits in the classical case of working over $\mathbb{F}_{p^2}$, and $k\log(p)$ bits in the case of working over $\mathbb{F}_{p^{2k}}$. Due to this, we only consider working in multiple field extensions from this point on.

11.2.3 Cost of signing using extension fields

In SQIsign, operations over $\mathbb{F}_{p^2}$ make up the majority of the cost during signing [126, Section 5.1]. Hence, we can roughly estimate the cost of signing by ignoring purely quaternionic operations, in which case the bottleneck of the signing procedure becomes running $\mathsf{IdealToIsogeny}_T$ as many times as required by the IdealToIsogenyEichler algorithm from Section 3.2.2 in the response phase. In other words, we estimate the total signing cost from the following parameters:

- f, such that $2^f \mid p + 1$.
- T, the chosen torsion to work with.
- For each $\ell_i^{e_i} \mid T$, the smallest k_i such that $E[\ell_i^{e_i}]$ is defined over $\mathbb{F}_{p^{2k_i}}$.

Since Algorithm 11.1 works with prime powers separately, we can estimate the cost of a single execution by considering the cost per prime power.

Cost per prime power. For each $\ell_i^{e_i} \mid T$, let k_i denote the smallest integer so that $E[\ell_i^{e_i}] \subseteq E(\mathbb{F}_{p^{2k_i}})$, and let $M(k_i)$ denote the cost of operations in $\mathbb{F}_{p^{2k_i}}$ in terms of $\mathbb{F}_{p^2}$-operations. Computing the generator $K_{\ell_i^{e_i}}$ consists of a few point additions in $E[\ell_i^{e_i}]$, hence is $O(M(k) \cdot e\log\ell)$. The cost of computing the isogeny generated by $K_{\ell_i^{e_i}}$ comes from computing e isogenies of degree ℓ at a cost of $O(\ell k) + \widetilde{O}(\ell)$, using the techniques from Section 11.1.2.

To compute the whole isogeny, we need to push the remaining generators $K_{\ell_j^{e_j}}$, through this isogeny. By picking the greedy strategy of always computing the smaller ℓ first, we bound the cost of evaluating K_{ℓ^e} in *other* isogenies by $O(M(k) \cdot \ell)$.

Total cost of signing. Based on the analysis above, we let

$$\textsc{Cost}_p(\ell_i^{e_i}) = c_1 M(k_i) e \log \ell + c_2 e_i \ell_i k_i + c_3 e_i \ell_i \log(\ell_i) + c_4 M(k_i)\ell$$

where k_i, and $M(k)$ are as before, and c_i are constants corresponding to the differences in the asymptotic complexities. Since we can estimate the total cost of executing $\mathsf{IdealToIsogeny}_T$ by summing the cost of each maximal prime power divisor of T, and observing that signing consists of executing $\mathsf{IdealToIsogeny}_{D_{\mathsf{com}}}$ one time, and $\mathsf{IdealToIsogeny}_T$ a total of $2 \cdot \lceil e/f \rceil$ times, we get a rough cost estimate of signing as

$$\textsc{SigningCost}_p(T) = (2 \cdot \lceil e/f \rceil + 1) \cdot \sum_{\ell_i^{e_i} | T} \textsc{Cost}_p(\ell_i^{e_i}).$$

In Section 11.6, we use this function to pick p and T minimising this cost. While this cost metric is very rough, we show that our implementation roughly matches the times predicted by this function. Further, we show that this cost metric suggests that going to extension fields gives signing times within the same order of magnitude as staying over $\mathbb{F}_{p^2}$, even when considering the additional benefit of using $\sqrt{\text{élu}}$ to compute isogenies in the latter case. An explicit comparision of the cost of signing over $\mathbb{F}_{p^2}$ versus over extension fields is given in Table 11.3.

11.3 Effect of increased $2^\bullet$-torsion on verification

In Section 11.2, we showed that signing with extension fields gives us more flexibility in choosing the prime p, and, in particular, allows us to find primes with rational 2^f-torsion for larger f. In this section, we analyse how such an increase in $2^\bullet$-torsion affects the cost of SQIsign verification, e.g., computing φ_{resp} and $\widehat{\varphi_{\text{chall}}}$, in terms of $\mathbb{F}_p$-multiplications, taking the SQIsign (NIST) implementation, with no further optimisations, as the baseline for comparison. As standard, we denote $\mathbb{F}_p$-multiplications by $\mathbf{M}$, $\mathbb{F}_p$-squarings by $\mathbf{S}$, and $\mathbb{F}_p$-additions by $\mathbf{a}$.

11.3.1 Detailed description of verification

Before giving an in-depth analysis of verification performance, we give a detailed description of how verification is executed. Recall that a SQIsign signature σ for a message `msg` created by a signer with secret signing key $\varphi_{\text{sk}} : E_0 \to E_{\text{pk}}$ proves knowledge of an endomorphism on E_{pk}[2] by describing an isogeny $\varphi_{\text{resp}} : E_{\text{pk}} \to E_{\text{chall}}$ of degree 2^e. Given a message `msg`, it is hashed with E_{com} to a point K_{chall} of order D_{chall}, hence represents an isogeny $\varphi_{\text{chall}} : E_{\text{com}} \to E_{\text{chall}}$. A signature is valid if the composition of φ_{resp} with $\widehat{\varphi_{\text{chall}}}$ is cyclic of degree $2^e \cdot D_{\text{chall}}$.

To verify a signature σ, the verifier must **(a)** recompute φ_{resp}, **(b)** compute the dual of φ_{chall}, to confirm that both are well-formed, and finally **(c)** recompute the hash of the message `msg` to confirm the validity of the signature.

In SQIsign, the size of the sample space for φ_{chall} impacts soundness of the underlying Σ-protocol. In SQIsign (NIST), to obtain negligible soundness error (in the security parameter λ) the

[2]To emphasise that E_{pk} is a Montgomery curve with coefficient A, we often denote it as E_A in this chapter.

message is hashed to an isogeny of degree $D_{\text{chall}} = 2^f \cdot 3^g$ so that the size of cyclic isogenies of degree D_{chall} is larger than 2^λ. In contrast, when $f \geq \lambda$, we can simply set $D_{\text{chall}} = 2^\lambda$.

The signature σ consists of a compressed description of the isogenies φ_{resp} and $\widehat{\varphi_{\text{chall}}}$. For $f < \lambda$ and $D_{\text{chall}} = 2^f \cdot 3^g$ it is of the form

$$\sigma = (b, s^{(1)}, \ldots, s^{(n)}, r, b_2, s_2, b_3, s_3)$$

with $s^{(j)}, s_2 \in \mathbb{Z}/2^f\mathbb{Z}$, $s_3 \in \mathbb{Z}/3^g\mathbb{Z}$, $r \in \mathbb{Z}/2^f3^g\mathbb{Z}$, and $b, b_2, b_3 \in \{0, 1\}$. If $f \geq \lambda$, we set $D_{\text{chall}} = 2^f$ and have $s_2 \in \mathbb{Z}/2^\lambda\mathbb{Z}$ and $r \in \mathbb{Z}/2^f\mathbb{Z}$, while b_3, s_3 are omitted.

Algorithmically, the verification process mostly requires three subroutines.

FindBasis: Given a curve E, find a deterministic basis (P, Q) of $E[2^f]$.

FindKernel: Given a curve E with basis (P, Q) for $E[2^f]$ and $s \in \mathbb{Z}/2^f\mathbb{Z}$, compute the kernel generator $K = P + [s]Q$.

ComputeIsogeny: Given a curve E and a kernel generator K, compute the isogeny $\varphi : E \to E/\langle K\rangle$ and $\varphi(Q)$ for some $Q \in E$.

Below we detail each of the three verification steps **(a)-(c)**.

Step (a). Computing φ_{resp} is split up into $n - 1$ *blocks* $\varphi^{(j)} : E^{(j)} \to E^{(j+1)}$ of size 2^f, and a last block of size 2^{f_0}, where $f_0 = e - (n - 1) \cdot f$. For every $\varphi^{(j)}$, the kernel $\langle K^{(j)}\rangle$ is given by the generator $K^{(j)} = P^{(j)} + [s^{(j)}]Q^{(j)}$ for a deterministic basis $(P^{(j)}, Q^{(j)})$ of $E^{(j)}[2^f]$.

In the first block, after sampling $(P^{(1)}, Q^{(1)})$ via FindBasis, the bit b indicates whether $P^{(1)}$ and $Q^{(1)}$ have to be swapped before running FindKernel. For the following blocks, the verifier pushes $Q^{(j)}$ through the isogeny $\varphi^{(j)}$ to get a point $Q^{(j+1)} \leftarrow \varphi^{(j)}(Q^{(j)})$ on $E^{(j+1)}$ of order 2^f above $(0, 0)$.[3] Hence, for $j > 1$ FindBasis only needs to find a suitable point $P^{(j)}$ to complete the basis $(P^{(j)}, Q^{(j)})$. Furthermore, $K^{(j)}$ is never above $(0, 0)$ for $j > 1$, which ensures cyclicity when composing $\varphi^{(j)}$ with $\varphi^{(j-1)}$. In all cases we use $s^{(j)}$ from σ to compute the kernel generator $K^{(j)}$ via FindKernel and $\varphi^{(j)}$ via ComputeIsogeny.

The last block of degree 2^{f_0} uses $Q^{(n)} \leftarrow [2^{f-f_0}]\varphi^{(n-1)}(Q^{(n-1)})$ and samples another point $P^{(n)}$ as basis of $E^{(n)}[2^{f_0}]$. In the following, we will often assume $f_0 = f$ for the sake of simplicity.[4]

Step (b). Computing $\widehat{\varphi_{\text{chall}}}$ requires a single isogeny of smooth degree $D_{\text{chall}} \approx 2^\lambda$. For the primes given in SQIsign (NIST), we have $E_{\text{chall}}[D_{\text{chall}}] \subseteq E_{\text{chall}}(\mathbb{F}_{p^2})$. Thus, we compute φ_{chall} by deterministically computing a basis (P, Q) for $E_{\text{chall}}[D_{\text{chall}}]$ and finding the kernel $\langle K\rangle$ for $\widehat{\varphi_{\text{chall}}} : E_{\text{chall}} \to E_{\text{com}}$. For $f < \lambda$, we have $D_{\text{chall}} = 2^f \cdot 3^g$, and split this process into two parts.

Given the basis (P, Q) for $E_{\text{chall}}[D_{\text{chall}}]$, we compute $(P_2, Q_2) = ([3^g]P, [3^g]Q)$ as basis of $E_{\text{chall}}[2^f]$, and use $K_2 = P_2 + [s_2]Q_2$, where b_2 indicates whether P_2 and Q_2 have to be swapped prior to computing K_2. We compute $\varphi_2 : E_{\text{chall}} \to E'_{\text{chall}}$ with kernel $\langle K_2\rangle$, and $P_3 = [2^f]\varphi_2(P)$ and $Q_3 = [2^f]\varphi_3(Q)$ form a basis of $E'_{\text{chall}}[3^g]$. Then b_3 indicates a potential swap of P_3 and Q_3, while $K_3 = P_3 + [s_3]Q_3$ is the kernel generator of the isogeny $\varphi_3 : E'_{\text{chall}} \to E_{\text{com}}$. Thus, we have $\widehat{\varphi_{\text{chall}}} = \varphi_3 \circ \varphi_2$. If $f \geq \lambda$, we require only the first step.

[3]A point P is said to be *above* a point R if $[k]P = R$ for some $k \in \mathbb{N}$.

[4]In contrast to earlier versions, SQIsign (NIST) fixes $f_0 = f$. However, our analysis benefits from allowing $f_0 < f$.

We furthermore verify that the composition of φ_{resp} and $\widehat{\varphi_{\text{chall}}}$ is cyclic, by checking that the first 2-isogeny step of φ_2 does not revert the last 2-isogeny step of $\varphi^{(n)}$. This guarantees that $\widehat{\varphi_{\text{chall}}} \circ \varphi_{\text{resp}}$ is non-backtracking, hence cyclic.

Step (c). On E_{com}, the verifier uses the point $Q' \leftarrow \widehat{\varphi_{\text{chall}}}(Q')$, where Q' is some (deterministically generated) point, linearly independent from the generator of $\widehat{\varphi_{\text{chall}}}$, and r (a scalar given in the signature σ) to compute $[r]Q'$, and checks if $[r]Q'$ matches the hashed point $K_{\text{chall}} = \mathsf{H}(\mathtt{msg}, E_{\text{com}})$, where H is a hash function.

11.3.2 Impact of large f on verification

The techniques of Section 11.2 extend the possible range of f to any size below $\log(p)$. This gives two benefits to the cost of verification, especially when $f \geq \lambda$.

Number of blocks in φ_{resp}. The larger f is, the fewer blocks of size 2^f are performed in **Step (a)**. Per block, the dominating part of the cost are FindBasis and FindKernel as we first need to complete the torsion basis $(P^{(j)}, Q^{(j)})$ for $E^{(j)}[2^f]$ (given $Q^{(j)}$ if $j > 1$), followed by computing $K^{(j)} = P^{(j)} + [s^{(j)}]Q^{(j)}$. By minimizing the number of blocks n, we minimize the amount of times we perform FindBasis and FindKernel, and the cost of each individual FindKernel only mildly increases, as $s^{(j)}$ increases in size. The overall cost of ComputeIsogeny, that is, performing the n isogenies of degree 2^f given their kernels $K^{(j)}$, only moderately increases with growing f.

We further note that larger f requires fewer T-isogeny computations for the signer, hence signing performance also benefits from smaller n.

Challenge isogeny. When $f \geq \lambda$, we can simply set $D_{\text{chall}} = 2^\lambda$, which has two main benefits.

- The cost of FindBasis for this step is reduced as finding a basis for $E[2^\lambda]$ is much easier than a basis search for $E[2^f \cdot 3^g]$.
- The cost of ComputeIsogeny for φ_{chall} decreases as we only have to compute a chain of 2-isogenies instead of additional 3-isogenies.

11.3.3 Implementation and benchmark of cost in $\mathbb{F}_p$-multiplications

To measure the influence of the size of f on the performance, we implemented SQIsign verification for the NIST Level I security parameter set in Python, closely following SQIsign (NIST). As is standard in isogeny-based schemes, we use x-only arithmetic and represent points and curve coefficients projectively. The benchmark counts $\mathbb{F}_p$-operations and uses a cost metric that allows us to estimate the runtime of real-world implementations for 256-bit primes $p^{(f)}$, where $p^{(f)}$ denotes a prime such that 2^f divides $p^{(f)} + 1$. We benchmark primes $p^{(f)}$ for all values $50 \leq f \leq 250$. These results serve as a baseline to which we compare the optimisations that we introduce in Sections 11.4 and 11.5.

We briefly outline how SQIsign (NIST) implements the three subroutines FindBasis, FindKernel, and ComputeIsogeny.

FindBasis. We search for points of order 2^f by sampling x-coordinates in a specified order,[5] and check if the corresponding point P lies on E (and not on its twist E^t). We then compute $P \leftarrow [\frac{p+1}{2^f}]P$ and verify that $[2^{f-1}]P \neq 0_E$. Given two points $P, Q \in E$ of order 2^f, we verify linear independence by checking that $[2^{f-1}]P \neq [2^{f-1}]Q$, and discard and re-sample the second point otherwise. As discussed above in the description of **Step (a)**, after the first step is performed, the verifier pushes a 2^f-torsion point Q lying above $(0,0)$ through the previous isogeny step. As such, the verifier only needs to find a point P that completes a basis for $E[2^f]$ using $\mathsf{CompleteBasis}_{2^f}$.

FindKernel. Given a basis (P, Q), FindKernel computes $K = P + [s]Q$ via the ThreePointLadder algorithm as used in SIKE [10]. In addition to the x-coordinates x_P and x_Q of P and Q, it requires the x-coordinate x_{P-Q} of $P - Q$. Hence, after running FindBasis, we further compute x_{P-Q} as described in SQIsign (NIST) [78].

ComputeIsogeny. Given a kernel generator K of order 2^f, ComputeIsogeny follows the approach of SIKE [10], and computes the 2^f-isogeny $\varphi^{(j)}$ as a chain of 4-isogenies for efficiency reasons via FourIsogenyChain. If f is odd, we further compute a single 2-isogeny. Following SQIsign (NIST), ComputeIsogeny proceeds as follows:

1. Compute $R = [2^{f-2}]K$ and the corresponding 4-isogeny φ with kernel $\langle R \rangle$. Note that the point $(0,0)$ might be contained in $\langle R \rangle$ for the first block in φ_{resp}, which requires a special 4-isogeny formula. Thus, we check if this is the case and call the suitable 4-isogeny function. We set $K \leftarrow \varphi(K)$.

2. If f is odd, we compute $R = [2^{f-3}]K$, the 2-isogeny φ with kernel $\langle R \rangle$, and $K \leftarrow \varphi(K)$.

3. Compute the remaining isogeny of degree $2^{f'}$ with even f' as a chain of 4-isogenies, where $(0,0)$ is guaranteed not to lie in any of the kernels.

In the last step, SQIsign (NIST) uses *optimal strategies* as in SIKE [10] to compute a chain of 4-isogenies. Naive multiplicative strategies would compute $R = [2^{f'-2j}]K$, the 4-isogeny φ with kernel $\langle R \rangle$, and $K \leftarrow \varphi(K)$ for $j = 1, \dots, f'/2$. However, this strategy is dominated by costly doublings. Instead, we can save intermediate multiples of K during the computation of $R = [2^{f'-2j}]K$, and push them through isogenies to save multiplicative effort in following iterations. Optimal strategies that determine which multiples are pushed through isogenies and minimise the cost can be found efficiently [10, 123].

We note that for $f < \lambda$ the computation of $\widehat{\varphi_{\text{chall}}}$ requires small adaptations to these algorithms to allow for finding a basis of $E[D_{\text{chall}}]$ and computing 3-isogenies. Most notably, SQIsign (NIST) does *not* use optimised formulas or optimal strategies for 3-isogenies from SIKE [10], but uses a multiplicative strategy and general odd-degree isogeny formulas [102, 234]. We slightly deviate from SQIsign (NIST) by implementing optimised 3-isogeny formulas, but note that the performance difference is minor and in favour of SQIsign (NIST).

An algorithmic description of a single block of SQIsign (NIST) summarising the discussion above is given in Algorithm 11.2. Here, ProjectiveToAffine simply denotes an algorithm to move from projective to affine coordinates/coefficients.

[5]SQIsign (NIST) fixes the sequence $x_k = 1 + k \cdot i$ with $i \in \mathbb{F}_{p^2}$ such that $i^2 = -1$ and picks the smallest k for which we find a suitable point.

Algorithm 11.2 Single block in verification of SQIsign (NIST)

Input: Affine Montgomery coefficient $A \in \mathbb{F}_{p^2}$ of elliptic curve E_A, a basis x_P, x_Q, x_{P-Q} for $E_A[2^f]$ with Q above $(0,0)$ and $s \in \mathbb{Z}/2^f\mathbb{Z}$ defining a kernel.
Output: Affine coefficient $A' \in \mathbb{F}_{p^2}$ describing E'_A as the codomain of $E_A \to E_{A'}$ of degree 2^f, with a basis x_P, x_Q, x_{P-Q} for $E'_A[2^f]$ with Q above $(0,0)$.

```
1: $K \leftarrow \mathsf{ThreePointLadder}(x_P, x_Q, x_{P-Q}, s, A)$
2: $A_{\text{proj.}}, x_Q \leftarrow \mathsf{FourIsogenyChain}(K, x_Q, A)$
3: $A, x_Q \leftarrow \mathsf{ProjectiveToAffine}(A_{\text{proj.}}, x_Q)$
4: $x_P, x_{P-Q} \leftarrow \mathsf{CompleteBasis}_{2^f}(x_Q, A)$
5: return $A, x_P, x_Q, x_{P-Q}$
```

Cost metric. In implementations, $\mathbb{F}_{p^2}$-operations usually call underlying $\mathbb{F}_p$-operations. We follow this approach and use the total number of $\mathbb{F}_p$-operations in our benchmarks. As cost metric, we express these operations in terms of $\mathbb{F}_p$-multiplications, with $\mathbf{S} = 0.8 \cdot \mathbf{M}$, ignoring $\mathbb{F}_p$-additions and subtractions due to their small impact on performance. $\mathbb{F}_p$-inversions, $\mathbb{F}_p$-square roots, and Legendre symbols over $\mathbb{F}_p$ require exponentiations by an exponent in the range of p, hence we count their cost as $\log(p)$ $\mathbb{F}_p$-multiplications. In contrast to measuring clock cycles of an optimised implementation, our cost metric eliminates the dependence on the level of optimisation of finite field arithmetic and the specific device running SQIsign, hence, can be considered more general.

Benchmark results. Figure 11.1 shows the verification cost for the NIST Level I-sized primes $p^{(f)}$ for $50 \leq f \leq 250$, fixing $e = 975$, using our cost metric. For more efficient benchmarking, we sample random public key curves and signatures σ of the correct form instead of signatures generated by the SQIsign signing procedure.

The graph shows the improvement for $f \geq 128$. Furthermore, we can detect when the number of blocks n decreases solely from the graph (e.g. $f = 122, 140, 163, 195, 244$). The cost of sampling a 2^f-torsion basis is highly variable between different runs for the same prime, which is visible from the oscillations of the graph. The performance for odd f is worse in general due to the inefficient integration of the 2-isogeny, which explains the zigzag-shaped graph.

From the above observations, we conclude that $f \geq \lambda$ is significantly faster for verification, with local optima found at $f = 195$ and $f = 244$, due to those being (almost) exact divisors of the signing length $e = 975$.

Remark 11.3.1. The average cost of FindBasis differs significantly between primes p even if they share the same 2^f-torsion. This happens because SQIsign (NIST) finds basis points from a predetermined sequence $[x_1, x_2, x_3, \ldots]$ with $x_j \in \mathbb{F}_{p^2}$. As we will see in Section 11.4, these x_j values can not be considered random: some values x_j are certain to be above a point of order 2^f, while others are certain not to be, for any supersingular curve over p.

11.4 Optimisations for verification

In this section, we show how the improvements from Section 11.2 that increase f beyond λ together with the analysis in Section 11.3 allow several other optimisations that improve the verification

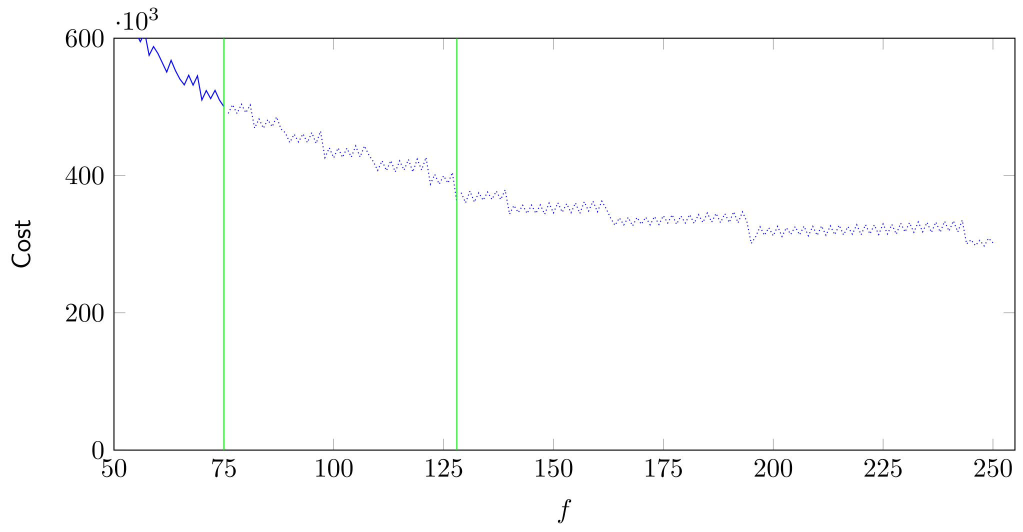


FIGURE 11.1: Cost in $\mathbb{F}_p$-multiplications for verification at NIST Level I security, for varying f and $p^{(f)}$, averaged over 1024 runs per prime. The green vertical lines mark $f = 75$ as used in SQIsign (NIST) for signing without extension fields, and $f = \lambda = 128$, beyond which we can set $D_{\text{chall}} = 2^\lambda$. The dotted graph beyond $f = 75$ is only accessible when signing with extension fields.

time of SQIsign in practice. Whereas the techniques in Section 11.2 allow us to decrease the *number* of blocks, in this section, we focus on the operations occurring *within blocks*. We optimise the cost of FindBasis, FindKernel and ComputeIsogeny.

We first analyse the properties of points that have full 2^f-torsion, and use them to improve FindBasis and FindKernel for general f. We then describe several techniques specifically for $f \geq \lambda$. Altogether, these optimisations significantly change the implementation of verification in comparison to SQIsign (NIST). We remark that the implementation of the signing procedure must be altered accordingly, as exhibited by our implementation.

Notation. As we mostly focus on the subroutines *within* a specific block $E^{(j)} \to E^{(j+1)}$, we will omit the superscripts in $E^{(j)}, K^{(j)}, P^{(j)}, \dots$ and write $E, K, P, \dots$ to simplify notation.

For reference throughout this section, the pseudocode for a single block in the verification procedure of our optimised variant is given by Algorithm 11.3.

11.4.1 Basis generation for full 2-power torsion

We first give a general result on points having full 2^f-torsion that we will use throughout this section. This theorem generalises previous results [103, 221] and will set the scene for easier and more efficient basis generation for $E[2^f]$.

Theorem 11.4.1. *Let $E : y^2 = (x - \lambda_1)(x - \lambda_2)(x - \lambda_3)$ be an elliptic curve over $\mathbb{F}_{p^2}$ with $E[2^f] \subseteq E(\mathbb{F}_{p^2})$ the full 2-power torsion. Let $L_i = (\lambda_i, 0)$ denote the points of order 2 and $[2]E$ denote the image of E under $[2] : P \mapsto P + P$ so that $E \setminus [2]E$ are the points with full 2^f-torsion. Then*

$$Q \in [2]E \text{ if and only if } x_Q - \lambda_i \text{ is square for } i = 1, 2, 3.$$

More specifically, for $Q \in E \setminus [2]E$, Q is above L_i if and only if $x_Q - \lambda_i$ is square and $x_Q - \lambda_j$ is non-square for $j \neq i$.

Proof. It is well-known that $Q = (x, y) \in [2]E$ if and only if $x - \lambda_1$, $x - \lambda_2$ and $x - \lambda_3$ are all three squares (see Theorem 4.1 in [184, Chapter 1]). Thus, for $Q \in E \setminus [2]E$, one of these three values must be a square, and the others non-squares (as their product must be y^2, hence square). We proceed similarly as the proof of Theorem 3 by Lin, Wang, Xu, and Zhao [221]. Namely, let P_1, P_2 and P_3 denote points of order 2^f above $L_1 = (\lambda_1, 0)$, $L_2 = (\lambda_2, 0)$ and $L_3 = (\lambda_3, 0)$, respectively, of order 2. A point $Q \in E \setminus [2]E$ must lie above one of the L_i. Therefore, the reduced Tate pairing of degree 2^f of P_i and Q gives a primitive 2^f-th root of unity if and only if Q is not above L_i. Let $\zeta_i = e_{2^f}(P_i, Q)$, then by [171, Theorem IX.9] we have

$$\zeta_i^{2^{f-1}} = e_2(L_i, Q).$$

We can compute $e_2(L_i, Q)$ by evaluating a Miller function f_{2,L_i} in Q, where $\operatorname{div} f_{2,L_i} = 2(L_i) - 2(0_E)$. The simplest option is the line that doubles L_i, that is, $f_{2,L_i}(x, y) = x - \lambda_i$, hence

$$e_2(L_i, Q) = (x_Q - \lambda_i)^{\frac{p^2-1}{2}}.$$

Applying Euler's criterion to this last term, we get that if $x_Q - \lambda_i$ is square, then ζ_i is not a primitive 2^f-th root and hence Q must be above L_i, whereas if $x_Q - \lambda_i$ is non-square, then ζ_i is a primitive 2^f-th root and hence Q is not above L_i. $\square$

Note that for Montgomery curves $y^2 = x^3 + Ax^2 + x = x(x - \alpha)(x - 1/\alpha)$, the theorem above tells us that non-squareness of x_Q for $Q \in E(\mathbb{F}_{p^2})$ is enough to imply Q has full 2^f-torsion and is not above $(0, 0)$ [221, Thm. 3].

Finding points with 2^f-torsion above $(0,0)$. We describe two methods to efficiently sample Q above $(0, 0)$, based on Theorem 11.4.1.

1. **Direct x sampling.** By deterministically sampling $x_Q \in \mathbb{F}_p$, we ensure that x_Q is square in $\mathbb{F}_{p^2}$. Hence, if Q lies on E and $x_Q - \alpha \in \mathbb{F}_{p^2}$ is non-square, where α is a root of $x^2 + Ax + 1$, then Theorem 11.4.1 ensures that $Q \in E \setminus [2]E$ and above $(0, 0)$.

2. **Smart x sampling.** We can improve this using the fact that α is always square [9, 100]. Hence, if we find $z \in \mathbb{F}_{p^2}$ such that z is square and $z - 1$ is non-square, we can choose $x_Q = z\alpha$ square and in turn $x_Q - \alpha = (z - 1)\alpha$ non-square. Again, by Theorem 11.4.1 if Q is on E, this ensures Q is above $(0, 0)$ and contains full 2^f-torsion. Hence, we prepare a list $[z_1, z_2, \ldots]$ of such values z for a given prime, and try $x_j = z_j\alpha$ until x_j is on E.

Both methods require computing α, dominated by one $\mathbb{F}_{p^2}$-square root. Direct sampling computes a Legendre symbol of $x^3 + Ax^2 + x$ per x to check if the corresponding point lies on E. If so, we check if $x - \alpha$ is non-square via the Legendre symbol. On average, this requires four samplings of x and six Legendre symbols to find a suitable x_Q with $Q \in E(\mathbb{F}_{p^2})$, and, given that we can choose x_Q to be small, we can use fast scalar multiplication on x_Q (see Section 11.4.2.1).

In addition to computing α, smart sampling requires the Legendre symbol computation of $x^3 + Ax^2 + x$ per x. On average, we require two samplings of an x to find a suitable x_Q, hence saving four Legendre symbols in comparison to direct sampling. However, we can no longer choose x_Q small, which means that improved scalar multiplication for small x_Q is not available.

Finding points with 2^f-torsion *not* above $(0,0)$. As shown by Lin, Wang, Xu, and Zhao [221], we find a point P with full 2^f-torsion *not* above $(0,0)$ by selecting a point on the curve with non-square x-coordinate. Non-squareness depends only on p, not on E, so a list of small non-square values can be precomputed. In this way, finding such a point P simply becomes finding the first value x_P in this list such that the point $(x_P, -)$ lies on $E(\mathbb{F}_{p^2})$, that is, $x_P^3 + Ax_P^2 + x_P$ is square. On average, this requires two samplings of x, hence two Legendre symbol computations.

11.4.2 General improvements to verification

In this section, we describe improvements to SQIsign verification and present new optimisations, decreasing the cost of the three main subroutines of verification.

11.4.2.1 Known techniques from literature

There are several state-of-the-art techniques in the literature on efficient implementations of elliptic curve or isogeny-based schemes that allow for general improvements to verification, but are not included in SQIsign (NIST).

We use $\mathsf{xDBL}(x_P)$ to denote x-only point doubling of a point P, and similarly we denote by $\mathsf{xADD}(x_P, x_Q, x_{P-Q})$ x-only differential addition of points P and Q. We use $\mathsf{xMUL}(x_P, m)$ to denote x-only scalar multiplication of a point P by the scalar m.

Faster scalar multiplications. We describe three improvements to the performance of xMUL, that can be applied in different situations during verification.

1. **Affine A.** Throughout verification and specifically in FindBasis and FindKernel, we work with the Montgomery coordinate A in projective form. However, some operations, such as computing the point difference x_{P-Q} given x_P and x_Q require A in affine form. Having an affine A allows an additional speed-up, as xDBL requires one $\mathbb{F}_{p^2}$-multiplication less in this case. Thus, xMUL with affine A is cheaper by $3\mathbf{M}$ per bit of the scalar.

2. **Affine points.** Using batched inversion, whenever we require A in affine form we can get x_P and x_Q in affine form for almost no extra cost. An xMUL with affine x_P or x_Q saves another $\mathbb{F}_{p^2}$-multiplication, hence again $3\mathbf{M}$, per bit of the scalar.

3. **Small x-coordinate.** For a point P with $x_P = a + bi$ with small a and b, we can replace a $\mathbb{F}_{p^2}$-multiplication by x_P with $a + b$ additions. This, in turn, saves almost $3\mathbf{M}$ per bit of the scalar in any xMUL of x_P.

As we can force P and Q to have $b \in \{0, 1\}$ and small a when sampling them in FindBasis, these points are affine and have small x-coordinates. Together with the affine A, this saves almost $9\mathbf{M}$ per bit for such scalar multiplications, saving roughly 27% per xMUL. We call such a xMUL a *fast* xMUL. Whenever xMUL uses 2 of these optimisations, we call it *semi-fast*.

Whenever possible, we use differential addition chains [25] to improve scalar multiplications by certain system parameters, such as $\frac{p+1}{2^f}$. In particular, we will only need to multiply by a few, predetermined scalars, and therefore we follow the method described by Cervantes-Vázquez, Chenu, Chi-Domínguez, De Feo, Rodríguez-Henríquez, and Smith [75, §4.2]. Our optimal differential addition chains were precomputed using the CTIDH software [12].

Faster square roots. We apply several techniques from the literature to further optimise low-level arithmetic in verification. The most significant of these is implementing faster square roots in $\mathbb{F}_{p^2}$ following Scott [281, §5.3], as described in Section 7.2.1 of Chapter 7. This decreases the cost of finding square roots to two $\mathbb{F}_p$-exponentiations[6] and a few multiplications.

Projective point difference. The implementation of $\mathsf{SQIsign}$ (NIST) switches between affine and projective representations for x_P, x_Q and A within each block (see, for example, Algorithm 11.2). It does so to be able to derive the point difference x_{P-Q} from x_P and x_Q in order to complete the basis P, Q in terms of x-coordinates. However, it is possible to compute the point difference entirely projectively using Proposition 3 from [268]. This allows us to stay projective during the $\mathsf{SQIsign}$ (NIST) verification until we reach E_{chall}, where we do normalization of the curve. This saves costly inversions during verification and has the additional benefit of improved elegance for $\mathsf{SQIsign}$ (NIST). However, in our variant of verification, we make no use of projective point difference, as the improvements of Section 11.4 seem to outperform this already.

11.4.2.2 Improving the subroutine $\mathsf{FindBasis}$

In $\mathsf{SQIsign}$ (NIST), to find a complete basis for $E[2^f]$ we are given a point $Q \in E[2^f]$ lying above $(0, 0)$ and need to find another point $P \in E(\mathbb{F}_{p^2})$ of order 2^f not lying above $(0, 0)$. We sample P directly using x_P non-square, as described above and demonstrated by Lin, Wang, Xu, and Zhao [221]. In particular, we can choose x_P small. We then compute $P \leftarrow [\frac{p+1}{2^f}]P$ via fast scalar multiplication to complete the torsion basis (P, Q).

11.4.2.3 Improved strategies for $\mathsf{ComputeIsogeny}$

Recall that $\mathsf{ComputeIsogeny}$ follows three steps in $\mathsf{SQIsign}$ (NIST): it first computes a 4-isogeny that may contain $(0, 0)$ in the kernel, and a 2-isogeny if f is odd, before entering an optimal strategy for computing the remaining chain of 4-isogenies. However, the first two steps include many costly doublings. We improve this by adding these first two steps in the optimal strategy. If f is even, this is straightforward, with a simple check for $(0, 0)$ in the kernel in the first step. For odd f, we add the additional 2-isogeny in this first step. For simplicity of the implementation, we determine optimal strategies as in SIKE [10], thus we assume that only 4-isogenies are used.

Note that techniques for strategies with variable isogeny degrees are available from the literature on CSIDH implementations [81]. However, the performance difference is very small, hence our simplified approach appears to be preferable.

In addition to optimising 4-isogeny chains, we implemented optimised 3-isogeny chains from SIKE [10] for the computation of $\widehat{\varphi_{\text{chall}}}$ when $f < 128$.

[6]Indeed, using notation as in Section 7.2.1, we have $2^f \mid p + 1$ as $p \equiv 3 \bmod 4$, and so the value $e = 1$.

11.4.3 To push, or not to push – that is, the Q

In SQIsign (NIST), the point Q is pushed through φ so that we easily get the basis point above $(0,0)$ on the image curve, and we can then use Theorem 11.4.1 to sample the second basis point P. Instead of pushing Q, one can also use Theorem 11.4.1 to efficiently sample this basis point Q above $(0,0)$. Although pushing Q seems very efficient, for larger f we are pushing Q through increasingly larger isogeny chains, whereas sampling becomes increasingly more efficient as multiplication cost by $\frac{p+1}{2^f}$ decreases. Furthermore, sampling *both* P and Q allows us to use those points as an *implicit basis* for $E[2^f]$, even if their orders are multiples of 2^f, as described in more detail below. We observe experimentally that this makes sampling Q, instead of pushing Q, more efficient for $f > 128$.

Using implicit bases. Using Theorem 11.4.1, it is possible to find points P and Q efficiently so that both have full 2^f-torsion. The pair (P, Q) is not an *explicit basis* for $E[2^f]$, as the orders of these points are likely to be multiples of 2^f. However, instead of multiplying both points by the cofactor to find an explicit basis, we can use these points implicitly, as if they were a basis for $E[2^f]$. This allows us to compute $K = P + [s]Q$ first, and only then multiply K by the cofactor. This saves a full scalar multiplication by the cofactor $\frac{p+1}{2^f}$. We refer to such a pair (P, Q) as an *implicit basis* of $E[2^f]$. Algorithmically, implicit bases combine FindBasis and FindKernel into a single routine FindBasisAndKernel.

11.4.4 Improved challenge for $f \geq \lambda$

Recall from Section 11.3.2 that when $f \geq \lambda$, we can simply set $D_{\text{chall}} = 2^\lambda$. This decreases the cost of FindBasis for the challenge computation considerably, as we can now use Theorem 11.4.1 to find a basis for $E[2^\lambda]$.

Improving FindBasis for the challenge isogeny when $f \geq \lambda$. We use Theorem 11.4.1 twice, first to find P not above $(0,0)$ having full 2^f-torsion and then to find Q above $(0,0)$ having full 2^f-torsion. We choose x_P and x_Q small such that faster scalar multiplication is available. We find the basis for $E[2^\lambda]$ by $P \leftarrow [\frac{p+1}{2^f}]P$ followed by $f - \lambda$ doublings, and $Q \leftarrow [\frac{p+1}{2^f}]Q$ followed by $f - \lambda$ doublings. Algorithmically, this is faster than a single scalar multiplication by $2^{f-\lambda} \cdot \frac{p+1}{2^f}$. Alternatively, if Q is pushed through isogenies, we can reuse $Q \leftarrow \varphi^{(n)}(Q^{(n)}) \in E[2^f]$ from the computation of the last step of φ_{resp}, so that we get a basis point for $E[2^\lambda]$ by $f - \lambda$ doublings of Q. Reusing this point Q also guarantees cyclicity of $\widehat{\varphi_{\text{chall}}} \circ \varphi_{\text{resp}}$.

Remark 11.4.2. For SQIsign without extension fields, obtaining $f \geq \lambda$ seems infeasible, hence the degree D of φ_{chall} is $2^f \cdot 3^g$. Nevertheless, some optimizations are possible in the computation of φ_{chall} in this case. FindBasis for $E[2^f \cdot 3^g]$ benefits from similar techniques as previously used in SIDH/SIKE, as we can apply known methods to improve generating a torsion basis for $E[3^g]$ coming from 3-descent [103, § 3.3]. Such methods are an analogue to generating a basis for $E[2^f]$ as described in Theorem 11.4.1 and [221, Theorem 3].

11.5 Size-speed trade-offs in SQIsign signatures

The increase in f also enables several size-speed trade-offs by adding further information in the signature or by using uncompressed signatures. Some trade-offs were already present in earlier versions of SQIsign [125], however, by using large f and the improvements from Section 11.4, they become especially worthwhile.

We take a slightly different stance from previous work on SQIsign as for many use cases the main road block to using SQIsign is the efficiency of verification in cycles. In contrast, in several applications the precise size of a signature is less important as long as it is below a certain threshold.[7] For example, many applications can handle the combined public key and signature size of RSA-2048 of 528 bytes, while SQIsign (NIST) features a combined size of only 241 bytes. In this section, we take the 528 bytes of RSA-2048 as a baseline, and explore size-speed trade-offs for SQIsign verification with data sizes up to this range.

We note that the larger signatures in this section encode the same information as standard SQIsign signatures, hence have no impact on the security.

11.5.1 Adding seeds for the torsion basis in the signature

We revisit an idea that was previously present in the original SQIsign verification [125], but no longer in [78] or [126], and highlight its particular merits whenever $f \geq \lambda$, as enabled by signing with extension fields. So far, we have assumed that completing or sampling a basis for $E[2^f]$ is done by deterministically sampling points. Recall from Section 11.4.1 that sampling x_P resp. x_Q (when not pushing Q) on average requires the computation of several Legendre symbols resp. square roots. We instead suggest using a seed to find x_P (when pushing Q) or x_P and x_Q (otherwise), which we include in the signature, so that the verifier saves all of the above cost for finding x_P, resp. x_Q. Finding these seeds adds negligible overhead for the signer, while verification performance improves. Signer and verifier are assumed to agree upon all precomputed values.

Seeding a point *not* above $(0,0)$. For x_P *not* above $(0,0)$, we fix a large enough $k > 0$ and precompute the 2^k smallest values $u_j \in \mathbb{F}_p$ such that $u_j + i \in \mathbb{F}_{p^2}$ is non-square (where i is the same as in Section 11.4). During signing, we pick the smallest u_j such that $x_P = u_j + i$ is the x-coordinate of a point $P \in E(\mathbb{F}_{p^2})$, and add the index j to the signature as a seed for x_P. Theorem 11.4.1 ensures that any $P \in E(\mathbb{F}_{p^2})$ for non-square x_P is a point with full 2^f-torsion not above $(0,0)$. This furthermore has the advantage of fast scalar multiplication for x_P as the x-coordinate is very small.

Seeding a point *above* $(0,0)$. As noted above, when f is large, it is faster to deterministically compute a point of order 2^f above $(0,0)$ than to push Q through φ. We propose a similar seed here for fixed large enough $k > 0$, using Theorem 11.4.1 and the "direct sampling" approach from Section 11.4.1. During signing, we pick the smallest $j \leq 2^k$ such that $x_Q = j$ is the x-coordinate of a point $Q \in E(\mathbb{F}_{p^2})$ and $x_Q - \alpha$ is non-square. We add $x_Q = j$ to the signature as a seed.

[7] See https://blog.cloudflare.com/sizing-up-post-quantum-signatures/.

Note that when using both seeding techniques, we do not explicitly compute $[\frac{p+1}{2^f}]P$ or $[\frac{p+1}{2^f}]Q$, but rather use the seeded points P and Q as an implicit basis, as described in Section 11.4.3.

Size of seeds. Per seeded point, we add k bits to the signature size. Thus, we must balance k so that signatures do not become too large, while failure probabilities for not finding a suitable seed are small enough. In particular, seeding x_P resp. x_Q via direct sampling has a failure probability of $\frac{1}{2}$ resp. $\frac{3}{4}$ per precomputed value. For the sake of simplicity, we set $k = 8$ for both seeds, such that every seed can be encoded as a separate byte. This means that the failure rate for seeding Q is $(\frac{3}{4})^{256} \approx 2^{-106.25}$ for our choice, while for P it is 2^{-256}. Theoretically it is still possible that seeding failures occur. In such a case, we simply rerun SigningKLPT. Note that for equal failing rates the number of possible seeds for P can be chosen smaller than for Q, hence slightly decreasing the additional data sizes. We furthermore include similar seeds for the torsion basis on E_{pk} and E_{chall}, giving a size increase of $2(n+1)$ bytes.

The synergy with large f now becomes apparent. The larger f gets, the fewer blocks n are required, hence adding fewer seeds overall. For $f = 75$, the seeds require an additional 28 bytes when seeding both P and Q. For $f = 122$, 140, 163, 195, and 244 this drops to 18, 16, 14, 12, and 10 additional bytes, respectively, to the overall signature size of 177 bytes for NIST Level I security.

Remark 11.5.1. Instead of using direct sampling for Q with failure probability $\frac{3}{4}$, we can reduce it to $\frac{1}{2}$ via "smart sampling" (see Section 11.4.1). However, this requires the verifier to compute α via a square root to set $x_Q = z\alpha$ with seeded z. We thus prefer direct sampling for seeded Q, which incurs no such extra cost.

Before concluding with a discussion on uncompressed signatures, we summarise the last two sections by presenting an algorithm for the computation of a single block in verification with our new optimisations, given in Algorithm 11.3.

Algorithm 11.3 Single block in verification using improvements of Section 11.4 and Section 11.5

Input: Projective Montgomery coefficient $A \in \mathbb{F}_{p^2}$ describing elliptic curve E_A, a seed (n, m) and $s \in \mathbb{Z}/2^f\mathbb{Z}$ defining a kernel.
Output: Affine coefficient $A' \in \mathbb{F}_{p^2}$ describing $E_{A'}$ as the codomain of $E_A \to E_{A'}$ of degree 2^f.

```
1: $x_P \leftarrow \mathsf{SmallNonSquare}(m)$
2: $x_Q \leftarrow n$
3: $x_{P-Q} \leftarrow \mathsf{PointDifference}(x_P, x_Q, A)$        { implicit basis $x_P$, $x_Q$, $x_{P-Q}$ }
4: $K \leftarrow \mathsf{ThreePointLadder}(x_P, x_Q, x_{P-Q}, s, A)$
5: $K \leftarrow \mathsf{xMUL}(x_K, \frac{p+1}{2^f}, A)$        { semi-fast xMUL }
6: $A_{\text{proj}} \leftarrow \mathsf{FourIsogenyChain}(K, A)$
7: $A \leftarrow \mathsf{ProjectiveToAffine}(A_{\text{proj}})$
8: return $A$
```

11.5.2 Uncompressed signatures

When f is very large, and hence the number of blocks is small, in certain cases it is worthwhile to replace the value s in the signature by the full x-coordinate of $K = P + [s]Q$. In essence, this is the uncompressed version of the SQIsign signature σ, and we thus refer to this variant as *uncompressed* SQIsign.

Speed of uncompressed signatures. Adding the precise kernel point K removes the need for both FindBasis and FindKernel, leaving ComputeIsogeny as the sole remaining cost. This speed-up is significant, and leaves little room for improvement beyond optimizing the cost of computing isogenies. The cost of verification in this case is relatively constant, as computing an 2^e-isogeny given the kernels is only slightly affected by the size of f, as is visible in the black dashed line in Figure 11.2. This makes uncompressed SQIsign an attractive alternative in cases where the signature size, up to a certain bound, is less relevant.

Size of uncompressed signatures. Per step, this increases the size from $\log(s) \approx f$ to $2\log(p)$ bits, which is still relatively size efficient when f is close to $\log(p)$. For recomputing φ_{chall}, we take a slightly different approach than before. We add the Montgomery coefficient of E_{com} to the signature, and seeds for a basis of $E_{\text{com}}[2^f]$. From this, the verifier can compute the kernel generator of φ_{chall}, and verify that the j-invariant of its codomain matches E_{chall}. Hence, this adds $2\log(p)$ bits for E_{com} and two bytes for seeds to the signature, for a total of $(n+1)\cdot(\log(p)/4)+2$ bytes.

For $f = 244$, this approach less than doubles the signature size from 177 bytes to 322 bytes for NIST Level I security. For $f = 145$, the signature becomes approximately 514 bytes, while for the current NIST Level I prime with $f = 75$, the size would become 898 bytes. When adding the public key size of 64 bytes, especially the first two cases still appear to be reasonable alternatives to RSA-2048's combined data size of 528 bytes.

Remark 11.5.2. Uncompressed signatures significantly simplify verification, as many functionalities required for compressed signatures are not necessary. Hence, this allows for a much more compact code base, which might be important for use cases featuring embedded devices with strict memory requirements.

11.6 Primes and performance

In this section we show the performance of verification for varying f, using the optimisations from the previous sections. Further, we find specific primes with suitable f for $n = 4$ and $n = 7$, and report their signing performance using our SageMath implementation, comparing it with the current SQIsign (NIST) prime.

11.6.1 Performance of optimised verification

The optimisations for compressed variants from Section 11.4 and Section 11.5 allow for several variants of verification, depending on using seeds and pushing Q through isogenies, specifically optimised for $f \geq \lambda$. We summarise the four resulting approaches and measure their performance.

Pushing Q, sampling P without seed. This variant is closest to the original SQIsign (NIST) and SQIsign (LWXZ) implementations. It is the optimal version for non-seeded verification for

$f \leq 128$, using the general optimisations from Section 11.4.2 and the challenge optimisations from Section 11.4.4.

Not pushing Q, sampling both P and Q without seed. This variant competes with the previous version in terms of signature size. Due to Section 11.4.3, sampling a new Q is more efficient than pushing Q for large f.[8] This is the optimal version for non-seeded verification for $f > 128$, and additionally uses the optimisations from Section 11.4.3.

Pushing Q, sampling P with seed. This variant only adds seeds to the signature to describe x_P. As such, it lies between the other three variants in terms of both signature size and speed. The signature is 1 byte per block larger than the unseeded variants, and 1 byte per block smaller than the variant where x_Q is seeded too. In terms of speed, it is faster than the variants where P is unseeded, but slower than the variant where Q is seeded too. It uses the optimisations from Sections 11.4.2 and 11.4.4, but cannot benefit from the kernel computation via implicit bases from Section 11.4.3.

Not pushing Q and sampling both P and Q with seed. This is the fastest compressed version that we present in this work. Although it adds 2 bytes per block, the small number of blocks n for large f makes the total increase in signature size small. All the optimisations from Section 11.4 now apply: we additionally have fast xMUL for Q, as well as the optimised implicit basis method to compute the kernel and optimised challenge. An algorithmic description of a single block in this version is given in Algorithm 11.3.

11.6.2 Performance benchmark

To compare the verification performance of our optimised variants with compressed signatures to SQIsign (NIST) and SQIsign (LWXZ), we run benchmarks in the same setting as in Section 11.3.3. Our implementation of SQIsign (LWXZ) [221] is identical to SQIsign (NIST) except for the improved sampling of P described in Section 11.4.1. In particular, Figure 11.2 shows the cost of verification for the NIST Level I primes $p^{(f)}$ for $50 \leq f \leq 250$. As before, we sample random public key curves and signatures σ of the correct form instead of using signatures generated by the SQIsign signing procedure.

We benchmarked these four approaches according to our cost metric by taking the average over 1024 random signatures. The results are given in Figure 11.2 showing the significant increase in performance compared to SQIsign (NIST) and SQIsign (LWXZ), as well as the additional performance gained from seeding. For comparison, we also show the performance when using uncompressed signatures, serving as a lower bound for the cost.

11.6.3 Finding specific primes

We now give two example primes, one prime optimal for 4-block verification, as well as the best we found for 7-block verification. The "quality" of a prime p is measured using the cost metric $\textsc{SigningCost}_p$ defined in Section 11.2.3.

[8]Based on benchmarking results, we sample Q with $x = n\alpha$ for $f < 200$ and directly for $f \geq 200$.

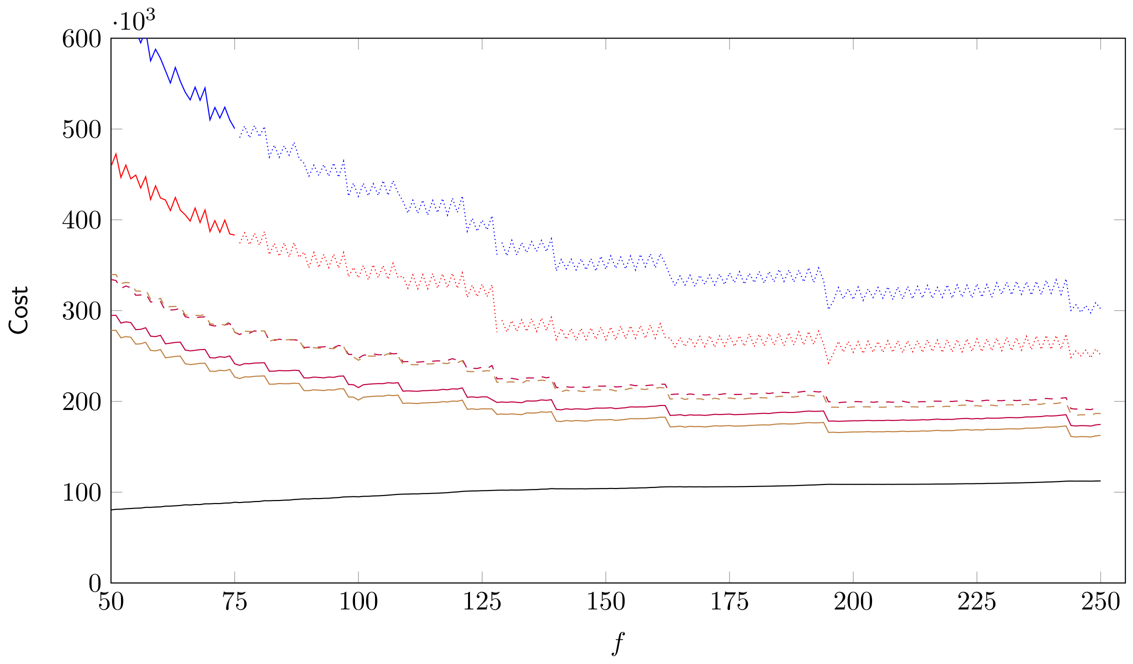


FIGURE 11.2: Extended version of Figure 11.1 showing the cost in $\mathbb{F}_p$-multiplications for verification at NIST-I security level, for varying f and $p^{(f)}$, averaged over 1024 runs per prime. In addition to SQIsign (NIST) in blue, it shows SQIsign (LWXZ) in red and all AprèsSQI variants: in purple is the performance of AprèsSQI when pushing Q, with dashed purple when not seeding P; in brown is the performance of AprèsSQI when not pushing Q, with dashed brown when not seeding P, Q; and the performance of uncompressed AprèsSQI is shown in black.

Optimal 4-block primes. For 4-block primes, taking $e = 975$ as a baseline, we need f bigger than 244. In other words, we are searching for primes of the form

$$p = 2^{244}N - 1,$$

where $N \in [2^4, 2^{12}]$ (accepting primes between 250 and 256 bits). This search space is quickly exhausted. For each prime of this form, we find the optimal torsion T to use, minimising the cost $\textsc{SigningCost}_p(T)$. The prime with the lowest total cost in this metric, which we denote p_4, is

$$p_4 = 2^{246} \cdot 3 \cdot 67 - 1.$$

The prime p_4 has $f = 246$, with T given below.

$$\begin{aligned} T = {} & 3^3 \cdot 5^2 \cdot 7^2 \cdot 11 \cdot 13 \cdot 17 \cdot 19 \cdot 23 \cdot 29 \cdot 31 \cdot 37 \cdot 41 \cdot 43 \cdot 47 \cdot 53 \cdot 59 \cdot 61 \cdot 67 \cdot 71 \\ & \cdot 73 \cdot 79 \cdot 83 \cdot 89 \cdot 97 \cdot 101 \cdot 103 \cdot 107 \cdot 109 \cdot 113 \cdot 127 \cdot 149 \cdot 151 \cdot 157 \cdot 163 \cdot 181 \\ & \cdot 197 \cdot 211 \cdot 229 \cdot 241 \cdot 271 \cdot 317 \cdot 397 \cdot 577 \cdot 593 \cdot 641 \cdot 661 \cdot 757 \cdot 1069 \cdot 2293 \end{aligned}$$

It has $\textsc{SigningCost}_{p_4}(T) = 9632.73$. The field of definition for the various torsion groups can be found in Table 11.1.

k	N	k	N
1	$67, 73, 757$	21	7^2
2	$317, 2293$	23	47
3	$37, 127, 1069$	25	151
4	593	26	53
5	$11, 31, 71, 661$	27	$109, 163, 271$
6	13	29	59
7	43	30	241
8	$17, 113$	35	211
9	$3^3, 19, 181, 577$	37	149
10	$5^2, 61, 641$	39	$79, 157$
11	$23, 89$	41	83
14	$29, 197$	48	97
18	397	50	101
19	229	51	103
20	41	53	107

TABLE 11.1: Torsion groups $E[N]$ and their minimal field $E(\mathbb{F}_{p^{2k}})$ for the prime p_4.

Balanced primes. Additionally, we look for primes that get above the significant $f > 128$ line, while minimizing $\textsc{SigningCost}_p(T)$. To do this, we adopt the sieve-and-boost technique used to find the current SQIsign primes [78, §5.2.1]. However, instead of looking for divisors of $p^2 - 1$, we follow Theorem 11.2.3 and look for divisors of

$$\prod_{n=1}^{k} \Phi_n(p^2)/2$$

to find a list of good candidate primes. This list is then sorted with respect to their signing cost according to $\textsc{SigningCost}_p$. The prime with the lowest signing cost we could find, which we call p_7, is

$$p_7 = 2^{145} \cdot 3^9 \cdot 59^3 \cdot 311^3 \cdot 317^3 \cdot 503^3 - 1.$$

The prime p_7 is used for a verification with $n = 7$ blocks. It achieves $f = 145$, with T given below.

$$\begin{aligned} T = {} & 3^7 \cdot 5^4 \cdot 7^2 \cdot 11 \cdot 13 \cdot 17 \cdot 19^2 \cdot 23 \cdot 29 \cdot 31 \cdot 37 \cdot 41 \cdot 43 \cdot 47 \cdot 53^2 \cdot 59^3 \cdot 61 \cdot 67 \\ & \cdot 71 \cdot 73 \cdot 79 \cdot 109 \cdot 113 \cdot 131 \cdot 157 \cdot 181 \cdot 193 \cdot 223 \cdot 239 \cdot 241 \cdot 271 \cdot 283 \cdot 311^3 \\ & \cdot 317^3 \cdot 331 \cdot 349 \cdot 503^2 \cdot 859 \cdot 997 \end{aligned}$$

It has $\textsc{SigningCost}_{p_7}(T) = 4137.91$. The field of definition for the various torsion groups we work with can be found in Table 11.2.

Remark 11.6.1. This method of searching for primes is optimised for looking for divisors of $p^2 - 1$, hence it might be suboptimal in the case of allowing torsion in higher extension fields. We leave it as future work to find methods which further take advantage of added flexibility in the prime search.

k	N
1	$3^7, 53^2, 59^3, 61, 79, 283, 311^3, 317^3, 349, 503^2, 859, 997$
3	$13, 109, 223, 331$
4	17
5	$11, 31, 71, 241, 271$
6	157
7	$7^2, 29, 43, 239$
8	113
9	19^2
10	$5^4, 41$
11	$23, 67$
12	193
13	131
15	181
18	$37, 73$
23	47

TABLE 11.2: Torsion groups $E[N]$ and their minimal field $E(\mathbb{F}_{p^{2k}})$ for the prime p_7.

p	Largest $N \mid T$	Largest $\mathbb{F}_{p^{2k}}$	SigningCost$_p(T)$	Adj. Cost	Timing
p_{1973}	1973	$k = 1$	8371.7	1956.5	11m, 32s
p_7	997	$k = 23$	4137.9	-	9m, 20s
p_4	2293	$k = 53$	9632.7	-	15m, 52s

TABLE 11.3: Comparison between estimated cost of signing for three different primes.

11.6.4 Performance for specific primes

We now compare the performance of the specific primes p_4, p_7, as well as the current NIST Level I prime p_{1973} used in SQIsign (NIST).

Signing performance. We give a summary of the estimated signing costs in Table 11.3. For p_{1973}, we include the metric "Adjusted Cost", which we compute as SigningCost with the N-isogeny computations scaling as $\sqrt{N}\log N$ to (rather optimistically) account for the benefit of $\sqrt{\text{é}}\text{lu}$. Further, we ran our proof-of-concept SageMath implementation on the three primes, using SageMath 9.8, on a laptop with an Intel-Core i5-1038NG7 processor, averaged over five runs. An optimised C implementation will be orders of magnitude faster; we use these timings simply for comparison.

We note that the SigningCost-metric correctly predicts the ordering of the primes, though the performance difference is smaller than predicted. A possible explanation for this is that the SigningCost-metric ignores all overhead, such as quaternion operations, which roughly adds similar amounts of cost per prime.

Our implementation uses $\sqrt{\text{é}}\text{lu}$ whenever the kernel generator is defined over $\mathbb{F}_{p^2}$ and N is bigger than a certain crossover point. This mainly benefits p_{1973}, as this prime only uses kernel generators defined over $\mathbb{F}_{p^2}$. The crossover point is experimentally found to be around $N > 300$ in our implementation, which is not optimal, compared to an optimised C implementation. For instance, work by Adj, Chi-Domínguez, and Rodríguez-Henríquez [1] gives the crossover point at $N > 89$, although for isogenies defined over $\mathbb{F}_p$. Nevertheless, we believe that these timings, together with the cost metrics, provide sufficient evidence that extension field signing in an optimised implementation stays in the same order of magnitude for signing time as staying over $\mathbb{F}_{p^2}$.

Verification performance. In Table 11.4, we summarise the performance of verification for p_{1973}, p_7, and p_4, both in terms of speed, and signature sizes.

Two highlights of this work lie in using p_7, both with and without seeds, having (almost) the same signature sizes as the current SQIsign signatures, but achieving a speed-up of factor 2.37 resp. 2.80 in comparison to SQIsign (NIST) and 1.82 resp. 2.15 in comparison to SQIsign (LWXZ), using p_{1973}. Another interesting alternative is using uncompressed p_4, at the cost of roughly double signature sizes, giving a speed-up of factor 4.46 in comparison to SQIsign (NIST) and 3.41 in comparison to SQIsign (LWXZ).

p	f	Implementation	Variant	Verif. cost	Sig. size
p_{1973}	75	SQIsign (NIST) [78]	-	500.4	177 B
		SQIsign (LWXZ) [221]	-	383.1	177 B
		AprèsSQI	unseeded	276.1	177 B
		AprèsSQI	seeded	226.8	195 B
p_7	145	AprèsSQI	unseeded	211.0	177 B
		AprèsSQI	seeded	178.6	193 B
		AprèsSQI	uncompressed	103.7	514 B
p_4	246	AprèsSQI	unseeded	185.2	177 B
		AprèsSQI	seeded	160.8	187 B
		AprèsSQI	uncompressed	112.2	322 B

TABLE 11.4: Comparison between verification cost for different variants and different primes, with cost given in terms of 10^3 $\mathbb{F}_p$-multiplications, using $\mathbf{S} = 0.8\mathbf{M}$.

Remark 11.6.2. We analyse and optimise the cost of verification with respect to $\mathbb{F}_p$-operations. However, primes of the form $p = 2^f \cdot c - 1$ are considered to be particularly suitable for fast optimised finite field arithmetic, especially when f is large [11]. Hence, we expect primes like p_4 to improve significantly more in comparison to p_{1973} in low-level field arithmetic, leading to a larger speed-up than predicted in Table 11.4. Furthermore, other low-level improvements, such as fast non-constant time GCD for inversions or Legendre symbols, will improve the performance of primes in terms of cycles, which is unaccounted for by our cost metric.

Remark 11.6.3. In recent joint work with Eriksen, Meyer and Rodríguez-Henríquez [277], we find better AprèsSQI-friendly primes according to a more accurate cost metric.

Chapter 12

Conclusion

In this thesis, we explored three facets of isogeny-based cryptography: the security of the underlying hardness assumptions; the efficiency of two-dimensional isogenies; and the design and efficiency of SQIsign.

We began in Part II by exploring the concrete security of the dimension-g superspecial isogeny problem for $g = 1$ and 2. In Chapter 7, we focus on the case $g = 1$ and introduced SuperSolver, a new attack against the isogeny problem which boasts a better concrete complexity than the Delfs–Galbraith algorithm. More specifically, for cryptographic-sized primes $p \approx 2^{256}$, we obtain around an 8 times reduction in complexity. Obtaining a grasp on the concrete complexity of SuperSolver allows us to better understand the security of isogeny-based schemes, such as SQIsign.

The attack strategy from this chapter uses only one-dimensional isogenies. In contrast, the SIDH attacks exhibit that higher dimensional isogenies are crucial to understand the dimension-1 superspecial isogeny graph $\mathcal{X}_1(\overline{\mathbb{F}}_p)$. Though this has only been exploited when extra information is available to the attacker, it would be interesting to revisit the work in Chapter 7 with this renewed perspective. In particular, it is natural to ask whether using higher dimensional isogenies can extend the SuperSolver attack to be more powerful. This leads us to the following open question:

> *Can higher dimensional isogenies help us understand the hardness of the dimension-*1 *superspecial isogeny problem?*

Moving forward to Chapter 8, we then studied the dimension-2 superspecial isogeny problem. We introduced SplitSearcher, a classical attack with lower concrete complexity than the previous state-of-the-art: the Costello–Smith algorithm. The improvement exhibited in this chapter is greater: for cryptographic-sized primes $p \approx 2^{128}$, we decrease the complexity by a factor of around 25 to 29. We remark that the algorithms in this chapter also enable a user to efficiently traverse the $(2, 2)$-isogeny graph $\mathcal{X}_2(\overline{\mathbb{F}}_p, 2)$ and explore the expansion properties of $\mathcal{X}_2(\overline{\mathbb{F}}_p, 2)$.

Though the work in Chapter 8 makes a big step towards understanding the dimension-2 graph, there is still much more to study. To accompany the increased use of higher dimensional isogenies in cryptography, it is crucial to obtain a better grasp of the security of the isogeny problem in higher dimensions. Another approach we could take to achieve this is to instead study the endomorphism ring problem in dimension 2 and determine how it relates to the isogeny problem. This is known for dimension 1, but remains unclear for higher dimensions.

> *Is the dimension-*2 *superspecial isogeny problem equivalent to the endomorphism ring problem for p.p. abelian surfaces?*

Turning towards constructive applications of two-dimensional isogenies, in Part III we gave efficient formulæ for $(3, 3)$-isogenies. In joint work with Flynn [94], we extend these ideas to give a description of a general method to construct (N, N)-isogenies between Kummer surfaces with efficiently computable biquadratic forms, for any odd N, using purely algebraic methods. This makes the method described in Section 9.2 effective for all odd N. These techniques could prove useful in developing algorithms to explore the graphs $\mathcal{X}_2(\bar{\mathbb{F}}_p, N)$ for prime N and gain fruitful experimental data on its expansion properties. Furthermore, as isogeny-based cryptography increasingly relies on two-dimensional isogenies, our formulæ could prove useful for future constructions.

> *Can we construct efficient isogeny-based primitives (other than hash functions) using (N, N)-isogenies for primes $N \neq 2$?*

In the final part of this thesis, we focused on a particular signature scheme built from isogenies: SQIsign, which boasts the smallest combined public key and signature size of all post-quantum signature schemes. The main disadvantage of SQIsign is that its signing and verification algorithms are slow, especially when compared to lattice-based alternatives. In Part IV, we presented work that aims to alleviate some inefficiencies in SQIsign. Firstly, in Chapter 10, we proposed new SQIsign-friendly parameters for all security levels, which mainly target applications where fast signing is a priority.

Due to its inherent characteristics, SQIsign is also attractive in scenarios that require small signatures and fast verification. With this in mind, in Chapter 11, we propose AprèsSQI, a variant of SQIsign that prioritises verification without sacrificing the compactness of the signatures. In recent joint work with Eriksen, Meyer and Rodríguez-Henríquez [277], we use similar methods to those described in Chapter 10 to obtain new parameters for SQIsign adapted for efficient verification. These parameters were used in the Round 1 NIST submission of SQIsign. Since AprèsSQI was published, two-dimensional variants of SQIsign, called SQIsign2D [17, 142, 247], were made public. SQIsign2D has faster signing and verification, and more compact signatures than SQIsign. In light of this, due to the lack of an optimised implementation of AprèsSQI in a low-level language (at the time of writing), it is unclear whether AprèsSQI can still provide an interesting solution for applications where verification time is crucial, particularly when considering uncompressed signatures. A natural question arises:

> *For what applications is one-dimensional SQIsign, or its variant AprèsSQI, the best choice?*

A recent paper on constructing ring signatures [46] gives interesting insight to this question: the authors construct a linear ring signature from a variant of SQIsign, rather than SQIsign2D. Indeed, the method for proving that the underlying identification scheme of SQIsign2D is zero-knowledge is not compatible with the current methods of constructing secure ring signatures. We further remark that, if future research develops better KLPT algorithms (where the output ideal has close-to-optimal norm), one-dimensional SQIsign verification will likely become more efficient than that of its higher dimensional variants. Thus, research on one-dimensional SQIsign remains important.

We end by taking a step back and viewing SQIsign in light of its submission to NIST's alternate call for signature schemes. Working with isogenies requires fairly significant mathematical knowledge, just as working with pairings did before the development of an abstraction of a bilinear map. As such, constructing isogeny-based protocols presents a high barrier of entry for cryptographers not familiar with these particular areas of mathematics. If SQIsign is to be standardised, we must endeavour to close the gap between theoreticians and the practitioners who will eventually be tasked to implement these algorithms securely and efficiently. This is crucial for the cryptographic community to be confident in the security SQIsign, thus enabling its widespread adoption. On that note, we end this thesis with a question:

> *Can we obtain a satisfying abstraction of* SQIsign *(and its variants) that will help practitioners understand and implement its algorithms more easily and securely?*

Bibliography

[1] Gora Adj, Jesús-Javier Chi-Domínguez, and Francisco Rodríguez-Henríquez. “Karatsuba-based square-root Vélu’s formulas applied to two isogeny-based protocols”. In: *Journal of Cryptographic Engineering* 13.1 (2023), pp. 89–106. DOI: 10.1007/S13389-022-00293-Y.

[2] Yusuke Aikawa, Ryokichi Tanaka, and Takuya Yamauchi. “Isogeny graphs on superspecial abelian varieties: eigenvalues and connection to Bruhat–Tits buildings”. In: *Canadian Journal of Mathematics. Journal Canadien de Mathématiques* 76.6 (2024), pp. 1891–1916. ISSN: 0008-414X,1496-4279. DOI: 10.4153/S0008414X23000676.

[3] Navid Alamati, Luca De Feo, Hart Montgomery, and Sikhar Patranabis. “Cryptographic Group Actions and Applications”. In: *ASIACRYPT 2020, Part II*. Ed. by Shiho Moriai and Huaxiong Wang. Vol. 12492. LNCS. Springer, Cham, Dec. 2020, pp. 411–439. DOI: 10.1007/978-3-030-64834-3_14.

[4] Tom M. Apostol. “Resultants of cyclotomic polynomials”. In: *Proceedings of the American Mathematical Society* 24 (1970), pp. 457–462. ISSN: 0002-9939,1088-6826. DOI: 10.2307/2037387.

[5] Sarah Arpin, Catalina Camacho-Navarro, Kristin Lauter, Joelle Lim, Kristina Nelson, Travis Scholl, and Jana Sotáková. “Adventures in supersingularland”. In: *Experimental Mathematics* 32.2 (2023), pp. 241–268. ISSN: 1058-6458,1944-950X. DOI: 10.1080/10586458.2021.1926009.

[6] Michael Artin. *Algebra*. Prentice Hall Inc., Englewood Cliffs, NJ, 1991, pp. xviii+618. ISBN: 0-13-004763-5.

[7] Shahla Atapoor, Karim Baghery, Daniele Cozzo, and Robi Pedersen. “CSI-SharK: CSI-FiSh with Sharing-friendly Keys”. In: *ACISP 23*. Ed. by Leonie Simpson and Mir Ali Rezazadeh Baee. Vol. 13915. LNCS. Springer, Cham, July 2023, pp. 471–502. DOI: 10.1007/978-3-031-35486-1_21.

[8] Shahla Atapoor, Karim Baghery, Daniele Cozzo, and Robi Pedersen. “Practical Robust DKG Protocols for CSIDH”. In: *ACNS 23International Conference on Applied Cryptography and Network Security, Part II*. Ed. by Mehdi Tibouchi and Xiaofeng Wang. Vol. 13906. LNCS. Springer, Cham, June 2023, pp. 219–247. DOI: 10.1007/978-3-031-33491-7_9.

[9] Roland Auer and Jaap Top. “Legendre elliptic curves over finite fields”. In: *Journal of Number Theory* 95.2 (2002), pp. 303–312. ISSN: 0022-314X,1096-1658.

[10] Reza Azarderakhsh, Matthew Campagna, Craig Costello, Luca De Feo, Basil Hess, Amir Jalali, David Jao, Brian Koziel, Brian LaMacchia, Patrick Longa, et al. “Supersingular isogeny key encapsulation”. In: *Submission to the NIST Post-Quantum Standardization project* 152 (2017), pp. 154–155.

[11] Jean-Claude Bajard and Sylvain Duquesne. “Montgomery-friendly primes and applications to cryptography”. In: *Journal of Cryptographic Engineering* 11.4 (2021), pp. 399–415. DOI: 10.1007/s13389-021-00260-z.

[12] Gustavo Banegas, Daniel J. Bernstein, Fabio Campos, Tung Chou, Tanja Lange, Michael Meyer, Benjamin Smith, and Jana Sotáková. “CTIDH: faster constant-time CSIDH”. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2021.4 (2021), pp. 351–387. DOI: 10.46586/TCHES.V2021.I4.351-387.

[13] Gustavo Banegas, Valerie Gilchrist, Anaëlle Le Dévéhat, and Benjamin Smith. “Fast and Frobenius: Rational Isogeny Evaluation over Finite Fields”. In: *LATINCRYPT 2023*. Ed. by Abdelrahaman Aly and Mehdi Tibouchi. Vol. 14168. LNCS. Springer, Cham, Oct. 2023, pp. 129–148. DOI: 10.1007/978-3-031-44469-2_7.

[14] Efrat Bank, Catalina Camacho-Navarro, Kirsten Eisenträger, Travis Morrison, and Jennifer Park. “Cycles in the supersingular ℓ-isogeny graph and corresponding endomorphisms”. In: *Research directions in number theory—Women in Numbers IV*. Vol. 19. Assoc. Women Math. Ser. Springer, Cham, 2019, pp. 41–66. ISBN: 978-3-030-19478-9; 978-3-030-19477-2. DOI: 10.1007/978-3-030-19478-9_2.

[15] William D. Banks and Igor E. Shparlinski. “Integers with a large smooth divisor”. In: *Integers. Electronic Journal of Combinatorial Number Theory* 7 (2007), A17, 11. ISSN: 1553-1732.

[16] Andrea Basso. *POKE: A Framework for Efficient PKEs, Split KEMs, and OPRFs from Higher-dimensional Isogenies*. Cryptology ePrint Archive, Paper 2024/624. 2024. URL: https://eprint.iacr.org/2024/624.

[17] Andrea Basso, Pierrick Dartois, Luca De Feo, Antonin Leroux, Luciano Maino, Giacomo Pope, Damien Robert, and Benjamin Wesolowski. “SQIsign2D–West”. In: *Advances in Cryptology – ASIACRYPT 2024*. Ed. by Kai-Min Chung and Yu Sasaki. Singapore: Springer Nature Singapore, 2025, pp. 339–370. DOI: 10.1007/978-981-96-0891-1_11.

[18] Andrea Basso and Tako Boris Fouotsa. “New SIDH Countermeasures for a More Efficient Key Exchange”. In: *ASIACRYPT 2023, Part VIII*. Ed. by Jian Guo and Ron Steinfeld. Vol. 14445. LNCS. Springer, Singapore, Dec. 2023, pp. 208–233. DOI: 10.1007/978-981-99-8742-9_7.

[19] Andrea Basso, Luciano Maino, and Giacomo Pope. “FESTA: Fast Encryption from Supersingular Torsion Attacks”. In: *ASIACRYPT 2023, Part VII*. Ed. by Jian Guo and Ron Steinfeld. Vol. 14444. LNCS. Springer, Singapore, Dec. 2023, pp. 98–126. DOI: 10.1007/978-981-99-8739-9_4.

[20] Mihir Bellare and Phillip Rogaway. “Random Oracles are Practical: A Paradigm for Designing Efficient Protocols”. In: *ACM CCS 93*. Ed. by Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby. ACM Press, Nov. 1993, pp. 62–73. DOI: 10.1145/168588.168596.

[21] Mihir Bellare and Phillip Rogaway. “The Security of Triple Encryption and a Framework for Code-Based Game-Playing Proofs”. In: *EUROCRYPT 2006*. Ed. by Serge Vaudenay. Vol. 4004. LNCS. Springer, Berlin, Heidelberg, 2006, pp. 409–426. DOI: 10.1007/11761679_25.

[22] Benjamin Bencina, Péter Kutas, Simon-Philipp Merz, Christophe Petit, Miha Stopar, and Charlotte Weitkämper. “Improved Algorithms for Finding Fixed-Degree Isogenies Between Supersingular Elliptic Curves”. In: *Advances in Cryptology – CRYPTO 2024 – 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2024, Proceedings, Part V*. Ed. by Leonid Reyzin and Douglas Stebila. Vol. 14924. Lecture Notes in Computer Science. Springer, 2024, pp. 183–217. DOI: 10.1007/978-3-031-68388-6_8.

[23] Elwyn R. Berlekamp. “Factoring polynomials”. In: *Proceedings of the Third Southeastern Conference on Combinatorics, Graph Theory and Computing (Florida Atlantic Univ., Boca Raton, Fla., 1972)*. Florida Atlantic University, Boca Raton, FL, 1972, pp. 1–7.

[24] Daniel J. Bernstein. *Differential addition chains*. 2006. URL: http://cr.yp.to/ecdh/diffchain-20060219.pdf.

[25] Daniel J. Bernstein. *Differential addition chains*. 2006. URL: http://cr.yp.to/ecdh/diffchain-20060219.pdf.

[26] Daniel J. Bernstein, Chitchanok Chuengsatiansup, Tanja Lange, and Peter Schwabe. “Kummer Strikes Back: New DH Speed Records”. In: *ASIACRYPT 2014, Part I*. Ed. by Palash Sarkar and Tetsu Iwata. Vol. 8873. LNCS. Springer, Berlin, Heidelberg, Dec. 2014, pp. 317–337. DOI: 10.1007/978-3-662-45611-8_17.

[27] Daniel J. Bernstein, Luca De Feo, Antonin Leroux, and Benjamin Smith. “Faster computation of isogenies of large prime degree”. In: *ANTS XIV—Proceedings of the Fourteenth Algorithmic Number Theory Symposium*. Vol. 4. Open Book Series. Math. Sci. Publ., Berkeley, CA, 2020, pp. 39–55. ISBN: 978-1-935107-08-8; 978-1-935107-07-1. DOI: 10.2140/obs.2020.4.39.

[28] Daniel J. Bernstein, Andreas Hülsing, Stefan Kölbl, Ruben Niederhagen, Joost Rijneveld, and Peter Schwabe. “The SPHINCS+ Signature Framework”. In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*. Ed. by Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz. ACM, 2019, pp. 2129–2146. DOI: 10.1145/3319535.3363229.

[29] Ward Beullens. “Breaking Rainbow Takes a Weekend on a Laptop”. In: *CRYPTO 2022, Part II*. Ed. by Yevgeniy Dodis and Thomas Shrimpton. Vol. 13508. LNCS. Springer, Cham, Aug. 2022, pp. 464–479. DOI: 10.1007/978-3-031-15979-4_16.

[30] Ward Beullens, Lucas Disson, Robi Pedersen, and Frederik Vercauteren. “CSI-RAShi: Distributed Key Generation for CSIDH”. In: *Post-Quantum Cryptography - 12th International Workshop, PQCrypto 2021*. Ed. by Jung Hee Cheon and Jean-Pierre Tillich. Springer, Cham, 2021, pp. 257–276. DOI: 10.1007/978-3-030-81293-5_14.

[31] Ward Beullens, Shuichi Katsumata, and Federico Pintore. "Calamari and Falafl: Logarithmic (Linkable) Ring Signatures from Isogenies and Lattices". In: *ASIACRYPT 2020, Part II*. Ed. by Shiho Moriai and Huaxiong Wang. Vol. 12492. LNCS. Springer, Cham, Dec. 2020, pp. 464–492. DOI: 10.1007/978-3-030-64834-3_16.

[32] Ward Beullens, Thorsten Kleinjung, and Frederik Vercauteren. "CSI-FiSh: Efficient Isogeny Based Signatures Through Class Group Computations". In: *ASIACRYPT 2019, Part I*. Ed. by Steven D. Galbraith and Shiho Moriai. Vol. 11921. LNCS. Springer, Cham, Dec. 2019, pp. 227–247. DOI: 10.1007/978-3-030-34578-5_9.

[33] Jean-François Biasse, David Jao, and Anirudh Sankar. "A Quantum Algorithm for Computing Isogenies between Supersingular Elliptic Curves". In: *INDOCRYPT 2014*. Ed. by Willi Meier and Debdeep Mukhopadhyay. Vol. 8885. LNCS. Springer, Cham, Dec. 2014, pp. 428–442. DOI: 10.1007/978-3-319-13039-2_25.

[34] Timo Bingmann. *TLX: Collection of Sophisticated C++ Data Structures, Algorithms, and Miscellaneous Helpers*. https://panthema.net/tlx, retrieved Oct. 7, 2020. 2018.

[35] Gaëtan Bisson. "Endomorphism rings in cryptography". PhD thesis. Institut National Polytechnique de Lorraine-INPL; Technische Universiteit Eindhoven, 2011.

[36] Gaëtan Bisson, Romain Cosset, and Damien Robert. AVIsogenies – a library for computing isogenies between abelian varieties. URL: http://avisogenies.gforge.inria.fr. 2012.

[37] Oskar Bolza. "On binary sextics with linear transformations into themselves". In: *American Journal of Mathematics* 10.1 (1887), pp. 47–70. ISSN: 0002-9327. DOI: 10.2307/2369402.

[38] Dan Boneh, Xavier Boyen, and Hovav Shacham. "Short Group Signatures". In: *CRYPTO 2004*. Ed. by Matthew Franklin. Vol. 3152. LNCS. Springer, Berlin, Heidelberg, Aug. 2004, pp. 41–55. DOI: 10.1007/978-3-540-28628-8_3.

[39] Dan Boneh, Özgür Dagdelen, Marc Fischlin, Anja Lehmann, Christian Schaffner, and Mark Zhandry. "Random Oracles in a Quantum World". In: *ASIACRYPT 2011*. Ed. by Dong Hoon Lee and Xiaoyun Wang. Vol. 7073. LNCS. Springer, Berlin, Heidelberg, Dec. 2011, pp. 41–69. DOI: 10.1007/978-3-642-25385-0_3.

[40] Dan Boneh and Matthew K. Franklin. "Identity-Based Encryption from the Weil Pairing". In: *CRYPTO 2001*. Ed. by Joe Kilian. Vol. 2139. LNCS. Springer, Berlin, Heidelberg, Aug. 2001, pp. 213–229. DOI: 10.1007/3-540-44647-8_13.

[41] Dan Boneh, Dmitry Kogan, and Katharine Woo. "Oblivious Pseudorandom Functions from Isogenies". In: *ASIACRYPT 2020, Part II*. Ed. by Shiho Moriai and Huaxiong Wang. Vol. 12492. LNCS. Springer, Cham, Dec. 2020, pp. 520–550. DOI: 10.1007/978-3-030-64834-3_18.

[42] Dan Boneh, Ben Lynn, and Hovav Shacham. "Short Signatures from the Weil Pairing". In: *ASIACRYPT 2001*. Ed. by Colin Boyd. Vol. 2248. LNCS. Springer, Berlin, Heidelberg, Dec. 2001, pp. 514–532. DOI: 10.1007/3-540-45682-1_30.

[43] Dan Boneh and Hovav Shacham. "Group Signatures With Verifier-Local Revocation". In: *ACM CCS 2004*. Ed. by Vijayalakshmi Atluri, Birgit Pfitzmann, and Patrick McDaniel. ACM Press, Oct. 2004, pp. 168–177. DOI: 10.1145/1030083.1030106.

[44] Xavier Bonnetain and André Schrottenloher. “Quantum Security Analysis of CSIDH”. In: *EUROCRYPT 2020, Part II.* Ed. by Anne Canteaut and Yuval Ishai. Vol. 12106. LNCS. Springer, Cham, May 2020, pp. 493–522. DOI: 10.1007/978-3-030-45724-2_17.

[45] Zenon I. Borevich and Igor R. Shafarevich. *Number theory.* Vol. Vol. 20. Pure and Applied Mathematics. Translated from the Russian by Newcomb Greenleaf. Academic Press, New York-London, 1966, pp. x+435.

[46] Giacomo Borin, Yi-Fu Lai, and Antonin Leroux. *Erebor and Durian: Full Anonymous Ring Signatures from Quaternions and Isogenies.* Cryptology ePrint Archive, Paper 2024/1185. 2024. URL: https://eprint.iacr.org/2024/1185.

[47] Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. “CRYSTALS-Kyber: a CCA-secure module-lattice-based KEM”. In: *2018 IEEE European Symposium on Security and Privacy, EuroS&P 2018, London, United Kingdom, April 24-26, 2018.* IEEE. IEEE, 2018, pp. 353–367. DOI: 10.1109/EUROSP.2018.00032.

[48] Joppe W. Bos, Craig Costello, Hüseyin Hisil, and Kristin E. Lauter. “Fast Cryptography in Genus 2”. In: *Journal of Cryptology* 29.1 (Jan. 2016), pp. 28–60. DOI: 10.1007/s00145-014-9188-7.

[49] Wieb Bosma, John Cannon, and Catherine Playoust. “The Magma algebra system. I. The user language”. In: vol. 24. 3-4. Computational algebra and number theory (London, 1993). 1997, pp. 235–265. DOI: 10.1006/jsco.1996.0125.

[50] Jean-Benoît Bost and Jean-François Mestre. “Moyenne arithmético-géométrique et périodes des courbes de genre 1 et 2”. In: *Gazette des Mathématiciens* 38 (1988), pp. 36–64. ISSN: 0224-8999,2275-0622.

[51] Nicolas Bourbaki. *Algebra I. Chapters 1–3.* Elements of Mathematics (Berlin). Springer-Verlag, Berlin, 1998, pp. xxiv+709. ISBN: 3-540-64243-9.

[52] Bradley Wayne Brock. *Superspecial curves of genera two and three.* Thesis (Ph.D.)–Princeton University. ProQuest LLC, Ann Arbor, MI, 1993, p. 69.

[53] Reinier Bröker, Everett W. Howe, Kristin E. Lauter, and Peter Stevenhagen. “Genus-2 curves and Jacobians with a given number of points”. In: *LMS Journal of Computation and Mathematics* 18.1 (2015), pp. 170–197. ISSN: 1461-1570. DOI: 10.1112/S1461157014000461.

[54] Fouazou Lontouo Perez Broon and Emmanuel Fouotsa. *Analogue of Vélu’s Formulas for Computing Isogenies over Hessian Model of Elliptic Curves.* Cryptology ePrint Archive, Report 2019/1480. 2019. URL: https://eprint.iacr.org/2019/1480.

[55] N. G. de Bruijn. “On the number of positive integers $\leq x$ and free prime factors $> y$. II”. In: *Indag. Math.* 28 (1966). Nederl. Akad. Wetensch. Proc. Ser. A **69**, pp. 239–247.

[56] Nils Bruin and Kevin Doerksen. *Electronic resources.* http://www.cecm.sfu.ca/~nbruin/splitigusa/. Accessed Septemeber 2022. 2011.

[57] Nils Bruin and Kevin Doerksen. “The arithmetic of genus two curves with $(4,4)$-split Jacobians”. In: *Canadian Journal of Mathematics. Journal Canadien de Mathématiques* 63.5 (2011), pp. 992–1024. ISSN: 0008-414X,1496-4279. DOI: 10.4153/CJM-2011-039-3.

[58] Nils Bruin, E. Victor Flynn, and Damiano Testa. “Descent via $(3, 3)$-isogeny on Jacobians of genus 2 curves”. In: *Acta Arithmetica* 165.3 (2014), pp. 201–223. ISSN: 0065-1036,1730-6264. DOI: 10.4064/aa165-3-1.

[59] Jan Hendrik Bruinier, Ken Ono, and Andrew V. Sutherland. “Class polynomials for non-holomorphic modular functions”. In: *Journal of Number Theory* 161 (2016), pp. 204–229. ISSN: 0022-314X,1096-1658. DOI: 10.1016/j.jnt.2015.07.002.

[60] Giacomo Bruno, Maria Corte-Real Santos, Craig Costello, Jonathan Komada Eriksen, Michael Meyer, Michael Naehrig, and Bruno Sterner. “Cryptographic smooth neighbors”. In: *Advances in Cryptology - ASIACRYPT 2023 - 29th International Conference on the Theory and Application of Cryptology and Information Security, Guangzhou, China, December 4-8, 2023, Proceedings, Part VII.* Vol. 14444. Lecture Notes in Computer Science. Springer, Singapore, 2023, pp. 190–221. ISBN: 978-981-99-8738-2; 978-981-99-8739-9. DOI: 10.1007/978-981-99-8739-9_7.

[61] Duncan A. Buell. *Binary quadratic forms.* Classical theory and modern computations. Springer-Verlag, New York, 1989, pp. x+247. ISBN: 0-387-97037-1. DOI: 10.1007/978-1-4612-4542-1.

[62] Jeffrey Burdges and Luca De Feo. “Delay Encryption”. In: *EUROCRYPT 2021, Part I.* Ed. by Anne Canteaut and François-Xavier Standaert. Vol. 12696. LNCS. Springer, Cham, Oct. 2021, pp. 302–326. DOI: 10.1007/978-3-030-77870-5_11.

[63] Peter J. Cameron. *Introduction to algebra.* Second. Oxford University Press, Oxford, 2008, pp. x+342. ISBN: 978-0-19-852793-0.

[64] David G. Cantor. “Computing in the Jacobian of a hyperelliptic curve”. In: *Mathematics of Computation* 48.177 (1987), pp. 95–101. ISSN: 0025-5718,1088-6842. DOI: 10.2307/2007876.

[65] David G. Cantor and Hans Zassenhaus. “A new algorithm for factoring polynomials over finite fields”. In: *Mathematics of Computation* 36.154 (1981), pp. 587–592. ISSN: 0025-5718,1088-6842. DOI: 10.2307/2007663.

[66] J. W. S. Cassels and E. Victor Flynn. *Prolegomena to a middlebrow arithmetic of curves of genus* 2. Vol. 230. London Mathematical Society Lecture Note Series. Cambridge University Press, Cambridge, 1996, pp. xiv+219. ISBN: 0-521-48370-0. DOI: 10.1017/CBO9780511526084.

[67] Wouter Castryck and Thomas Decru. “An Efficient Key Recovery Attack on SIDH”. In: *EUROCRYPT 2023, Part V.* Ed. by Carmit Hazay and Martijn Stam. Vol. 14008. LNCS. Springer, Cham, Apr. 2023, pp. 423–447. DOI: 10.1007/978-3-031-30589-4_15.

[68] Wouter Castryck and Thomas Decru. “CSIDH on the Surface”. In: *Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020.* Ed. by Jintai Ding and Jean-Pierre Tillich. Springer, Cham, 2020, pp. 111–129. DOI: 10.1007/978-3-030-44223-1_7.

[69] Wouter Castryck and Thomas Decru. *Multiradical isogenies.* Cryptology ePrint Archive, Report 2021/1133. 2021. URL: https://eprint.iacr.org/2021/1133.

[70] Wouter Castryck and Thomas Decru. “Multiradical isogenies”. In: *Arithmetic, geometry, cryptography, and coding theory 2021*. Vol. 779. Contemporary Mathematics. Amer. Math. Soc., Providence, RI, 2022, pp. 57–89. ISBN: 978-1-4704-6794-4. DOI: 10.1090/conm/779/15671.

[71] Wouter Castryck, Thomas Decru, Marc Houben, and Frederik Vercauteren. “Horizontal Racewalking Using Radical Isogenies”. In: *ASIACRYPT 2022, Part II*. Ed. by Shweta Agrawal and Dongdai Lin. Vol. 13792. LNCS. Springer, Cham, Dec. 2022, pp. 67–96. DOI: 10.1007/978-3-031-22966-4_3.

[72] Wouter Castryck, Thomas Decru, and Benjamin Smith. “Hash functions from superspecial genus-2 curves using Richelot isogenies”. In: *Journal of Mathematical Cryptology* 14.1 (2020), pp. 268–292. ISSN: 1862-2976,1862-2984. DOI: 10.1515/jmc-2019-0021.

[73] Wouter Castryck, Thomas Decru, and Frederik Vercauteren. “Radical Isogenies”. In: *ASIACRYPT 2020, Part II*. Ed. by Shiho Moriai and Huaxiong Wang. Vol. 12492. LNCS. Springer, Cham, Dec. 2020, pp. 493–519. DOI: 10.1007/978-3-030-64834-3_17.

[74] Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. “CSIDH: An Efficient Post-Quantum Commutative Group Action”. In: *ASIACRYPT 2018, Part III*. Ed. by Thomas Peyrin and Steven Galbraith. Vol. 11274. LNCS. Springer, Cham, Dec. 2018, pp. 395–427. DOI: 10.1007/978-3-030-03332-3_15.

[75] Daniel Cervantes-Vázquez, Mathilde Chenu, Jesús-Javier Chi-Domínguez, Luca De Feo, Francisco Rodríguez-Henríquez, and Benjamin Smith. “Stronger and Faster Side-Channel Protections for CSIDH”. In: *LATINCRYPT 2019*. Ed. by Peter Schwabe and Nicolas Thériault. Vol. 11774. LNCS. Springer, Cham, Oct. 2019, pp. 173–193. DOI: 10.1007/978-3-030-30530-7_9.

[76] Denis Xavier Charles, Kristin E. Lauter, and Eyal Z. Goren. “Cryptographic Hash Functions from Expander Graphs”. In: *Journal of Cryptology* 22.1 (Jan. 2009), pp. 93–113. DOI: 10.1007/s00145-007-9002-x.

[77] Jorge Chávez-Saab, Jesús-Javier Chi-Domínguez, Samuel Jaques, and Francisco Rodríguez-Henríquez. “The SQALE of CSIDH: sublinear Vélu quantum-resistant isogeny action with low exponents”. In: *Journal of Cryptographic Engineering* 12.3 (2022), pp. 349–368. DOI: 10.1007/S13389-021-00271-W.

[78] Jorge Chavez-Saab, Maria Corte-Real Santos, Jonathan Komada Eriksen, Basil Hess, David Kohel, Antonin Leroux, Patrick Longa, Michael Meyer, Lorenz Panny, Sikhar Patranabis, Christophe Petit, Francisco Rodríguez Henríquez, Sina Schaeffler, and Benjamin Wesolowski. *SQIsign: Algorithm specifications and supporting documentation*. 2023. URL: https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/round-1/spec-files/sqisign-spec-web.pdf.

[79] Mingjie Chen, Antonin Leroux, and Lorenz Panny. “SCALLOP-HD: Group Action from 2-Dimensional Isogenies”. In: *PKC 2024, Part II*. Ed. by Qiang Tang and Vanessa Teague. Vol. 14603. LNCS. Springer, Cham, Apr. 2024, pp. 190–216. DOI: 10.1007/978-3-031-57725-3_7.

[80] Jesús-Javier Chi-Domínguez and Krijn Reijnders. "Fully Projective Radical Isogenies in Constant-Time". In: *CT-RSA 2022*. Ed. by Steven D. Galbraith. Vol. 13161. LNCS. Springer, Cham, Mar. 2022, pp. 73–95. DOI: 10.1007/978-3-030-95312-6_4.

[81] Jesús-Javier Chi-Domínguez and Francisco Rodríguez-Henríquez. "Optimal strategies for CSIDH". In: *Advances in Mathematics of Communications* 16.2 (2022), pp. 383–411. ISSN: 1930-5346,1930-5338. DOI: 10.3934/amc.2020116.

[82] Andrew Childs, David Jao, and Vladimir Soukharev. "Constructing elliptic curve isogenies in quantum subexponential time". In: *Journal of Mathematical Cryptology* 8.1 (2014), pp. 1–29. ISSN: 1862-2976,1862-2984. DOI: 10.1515/jmc-2012-0016.

[83] David V. Chudnovsky and Gregory V. Chudnovsky. "Sequences of numbers generated by addition in formal groups and new primality and factorization tests". In: *Adv. in Appl. Math.* 7.4 (1986), pp. 385–434. ISSN: 0196-8858,1090-2074. DOI: 10.1016/0196-8858(86)90023-0.

[84] Henri Cohen. *A course in computational algebraic number theory*. Vol. 138. Graduate Texts in Mathematics. Springer-Verlag, Berlin, 1993, pp. xii+534. ISBN: 3-540-55640-0. DOI: 10.1007/978-3-662-02945-9.

[85] Paul M. Cohn. *Algebra, Vol. 1*. John Wiley & Sons, London-New York-Sydney, 1974, pp. xii+321.

[86] J. B. Conrey, M. A. Holmstrom, and T. L. McLaughlin. "Smooth neighbors". In: *Experimental Mathematics* 22.2 (2013), pp. 195–202. ISSN: 1058-6458,1944-950X. DOI: 10.1080/10586458.2013.768483.

[87] John B. Conrey and Mark A. Holmstrom. "Smooth values of quadratic polynomials". In: *Experimental Mathematics* 30.4 (2021), pp. 447–452. ISSN: 1058-6458,1944-950X. DOI: 10.1080/10586458.2018.1559775.

[88] Maria Corte-Real Santos, Craig Costello, and Sam Frengley. "An Algorithm for Efficient Detection of (N, N)-Splittings and Its Application to the Isogeny Problem in Dimension 2". In: *Public-Key Cryptography - PKC 2024 - 27th IACR International Conference on Practice and Theory of Public-Key Cryptography, Sydney, NSW, Australia, April 15-17, 2024, Proceedings, Part III*. Ed. by Qiang Tang and Vanessa Teague. Vol. 14603. Lecture Notes in Computer Science. Springer, 2024, pp. 157–189. DOI: 10.1007/978-3-031-57725-3_6.

[89] Maria Corte-Real Santos, Craig Costello, and Michael Naehrig. "On Cycles of Pairing-Friendly Abelian Varieties". In: *Advances in Cryptology - CRYPTO 2024 - 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2024, Proceedings, Part IX*. Ed. by Leonid Reyzin and Douglas Stebila. Vol. 14928. Lecture Notes in Computer Science. Springer, 2024, pp. 221–253. DOI: 10.1007/978-3-031-68400-5_7.

[90] Maria Corte-Real Santos, Craig Costello, and Jia Shi. "Accelerating the Delfs-Galbraith algorithm with fast subfield root detection". In: *Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15-18, 2022, Proceedings, Part III*. Vol. 13509. Lecture Notes in Computer

Science. Springer, Cham, 2022, pp. 285–314. ISBN: 978-3-031-15981-7; 978-3-031-15982-4. DOI: 10.1007/978-3-031-15982-4_10.

[91] Maria Corte-Real Santos, Craig Costello, and Benjamin Smith. “Efficient $(3, 3)$-isogenies on fast Kummer surfaces”. In: (2024). arXiv: 2402.01223 [cs.CR]. URL: https://arxiv.org/abs/2402.01223.

[92] Maria Corte-Real Santos, Jonathan Komada Eriksen, Michael Meyer, and Krijn Reijnders. “AprèsSQI: extra fast verification for SQIsign using extension-field signing”. In: *Advances in Cryptology - EUROCRYPT 2024 - 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland, May 26-30, 2024, Proceedings, Part I.* Vol. 14651. Lecture Notes in Computer Science. Springer, Cham, 2024, pp. 63–93. ISBN: 978-3-031-58715-3; 978-3-031-58716-0. DOI: 10.1007/978-3-031-58716-0_3.

[93] Maria Corte-Real Santos, Jonathan Komada Eriksen, Michael Meyer, and Krijn Reijnders. *AprèsSQI: Extra Fast Verification for SQIsign Using Extension-Field Signing.* Artifact at EUROCRYPT 2024. 2024. URL: https://artifacts.iacr.org/eurocrypt/2024/a4/.

[94] Maria Corte-Real Santos and E. Victor Flynn. *Isogenies on Kummer Surfaces.* 2024. arXiv: 2409.14819 [math.NT]. URL: https://arxiv.org/abs/2409.14819.

[95] Maria Corte-Real Santos and Krijn Reijnders. *Return of the Kummer: a Toolbox for Genus-2 Cryptography.* Cryptology ePrint Archive, Paper 2024/948. 2024. URL: https://eprint.iacr.org/2024/948.

[96] Romain Cosset. “Applications des fonctions thêta à la cryptographie sur courbes hyperelliptiques”. PhD thesis. Université Henri Poincaré - Nancy, 2011. URL: https://theses.hal.science/tel-00642951.

[97] Romain Cosset. “Factorization with genus 2 curves”. In: *Mathematics of Computation* 79.270 (2010), pp. 1191–1208. ISSN: 0025-5718,1088-6842. DOI: 10.1090/S0025-5718-09-02295-9.

[98] Romain Cosset and Damien Robert. “Computing (ℓ, ℓ)-isogenies in polynomial time on Jacobians of genus 2 curves”. In: *Mathematics of Computation* 84.294 (2015), pp. 1953–1975. ISSN: 0025-5718,1088-6842. DOI: 10.1090/S0025-5718-2014-02899-8.

[99] Craig Costello. “B-SIDH: Supersingular Isogeny Diffie-Hellman Using Twisted Torsion”. In: *ASIACRYPT 2020, Part II.* Ed. by Shiho Moriai and Huaxiong Wang. Vol. 12492. LNCS. Springer, Cham, Dec. 2020, pp. 440–463. DOI: 10.1007/978-3-030-64834-3_15.

[100] Craig Costello. “Computing Supersingular Isogenies on Kummer Surfaces”. In: *ASIACRYPT 2018, Part III.* Ed. by Thomas Peyrin and Steven Galbraith. Vol. 11274. LNCS. Springer, Cham, Dec. 2018, pp. 428–456. DOI: 10.1007/978-3-030-03332-3_16.

[101] Craig Costello. *Pairings for beginners.* Technical Report. 2024. URL: https://static1.squarespace.com/static/5fdbb09f31d71c1227082339/t/5ff394720493bd28278889c6/1609798774687/PairingsForBeginners.pdf.

[102] Craig Costello and Hüseyin Hisil. “A Simple and Compact Algorithm for SIDH with Arbitrary Degree Isogenies”. In: *ASIACRYPT 2017, Part II.* Ed. by Tsuyoshi Takagi and Thomas Peyrin. Vol. 10625. LNCS. Springer, Cham, Dec. 2017, pp. 303–329. DOI: 10.1007/978-3-319-70697-9_11.

[103] Craig Costello, David Jao, Patrick Longa, Michael Naehrig, Joost Renes, and David Urbanik. “Efficient Compression of SIDH Public Keys”. In: *EUROCRYPT 2017, Part I.* Ed. by Jean-Sébastien Coron and Jesper Buus Nielsen. Vol. 10210. LNCS. Springer, Cham, 2017, pp. 679–706. DOI: 10.1007/978-3-319-56620-7_24.

[104] Craig Costello, Patrick Longa, and Michael Naehrig. “Efficient Algorithms for Supersingular Isogeny Diffie-Hellman”. In: *CRYPTO 2016, Part I.* Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9814. LNCS. Springer, Berlin, Heidelberg, Aug. 2016, pp. 572–601. DOI: 10.1007/978-3-662-53018-4_21.

[105] Craig Costello, Michael Meyer, and Michael Naehrig. “Sieving for Twin Smooth Integers with Solutions to the Prouhet-Tarry-Escott Problem”. In: *EUROCRYPT 2021, Part I.* Ed. by Anne Canteaut and François-Xavier Standaert. Vol. 12696. LNCS. Springer, Cham, Oct. 2021, pp. 272–301. DOI: 10.1007/978-3-030-77870-5_10.

[106] Craig Costello and Benjamin Smith. “Montgomery curves and their arithmetic - The case of large characteristic fields”. In: *Journal of Cryptographic Engineering* 8.3 (2018), pp. 227–240. DOI: 10.1007/S13389-017-0157-6.

[107] Craig Costello and Benjamin Smith. “The Supersingular Isogeny Problem in Genus 2 and Beyond”. In: *Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020.* Ed. by Jintai Ding and Jean-Pierre Tillich. Springer, Cham, 2020, pp. 151–168. DOI: 10.1007/978-3-030-44223-1_9.

[108] Jean-Marc Couveignes. *Hard Homogeneous Spaces.* Cryptology ePrint Archive, Paper 2006/291. 2006. URL: https://eprint.iacr.org/2006/291.

[109] Jean-Marc Couveignes and Tony Ezome. “Computing functions on Jacobians and their quotients”. In: *LMS Journal of Computation and Mathematics* 18.1 (2015), pp. 555–577. ISSN: 1461-1570. DOI: 10.1112/S1461157015000169.

[110] David Cox, John Little, and Donal O’Shea. *Ideals, varieties, and algorithms.* Third. Undergraduate Texts in Mathematics. An introduction to computational algebraic geometry and commutative algebra. Springer, New York, 2007, pp. xvi+551. ISBN: 978-0-387-35650-1; 0-387-35650-9. DOI: 10.1007/978-0-387-35651-8.

[111] David A. Cox. *Primes of the form x^2+ny^2.* Second. Pure and Applied Mathematics (Hoboken). Fermat, class field theory, and complex multiplication. John Wiley & Sons Inc., Hoboken, NJ, 2013, pp. xviii+356. ISBN: 978-1-118-39018-4. DOI: 10.1002/9781118400722.

[112] Daniele Cozzo and Nigel P. Smart. “Sashimi: Cutting up CSI-FiSh Secret Keys to Produce an Actively Secure Distributed Signing Protocol”. In: *Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020.* Ed. by Jintai Ding and Jean-Pierre Tillich. Springer, Cham, 2020, pp. 169–186. DOI: 10.1007/978-3-030-44223-1_10.

[113] Ivan Damgård. *On Σ-protocols.* Lecture Notes, University of Aarhus, Department for Computer Science. 2002. URL: https://www.cs.au.dk/~ivan/Sigma.pdf.

[114] Thinh Dang and Dustin Moody. *Twisted Hessian Isogenies.* Cryptology ePrint Archive, Report 2019/1003. 2019. URL: https://eprint.iacr.org/2019/1003.

[115] Pierrick Dartois. *Fast computation of 2-isogenies in dimension 4 and cryptographic applications.* Cryptology ePrint Archive, Paper 2024/1180. 2024. URL: https://eprint.iacr.org/2024/1180.

[116] Pierrick Dartois, Antonin Leroux, Damien Robert, and Benjamin Wesolowski. "SQIsignHD: New Dimensions in Cryptography". In: *EUROCRYPT 2024, Part I.* Ed. by Marc Joye and Gregor Leander. Vol. 14651. LNCS. Springer, Cham, May 2024, pp. 3–32. DOI: 10.1007/978-3-031-58716-0_1.

[117] Pierrick Dartois, Luciano Maino, Giacomo Pope, and Damien Robert. "An Algorithmic Approach to $(2, 2)$-Isogenies in the Theta Model and Applications to Isogeny-Based Cryptography". In: *Advances in Cryptology – ASIACRYPT 2024.* Ed. by Kai-Min Chung and Yu Sasaki. Singapore: Springer Nature Singapore, 2025, pp. 304–338. DOI: 10.1007/978-981-96-0891-1_10.

[118] Luca De Feo. "Exploring isogeny graphs". In: *Habilitation á diriger des recherches. Université de Versailles Saint-Quentin-en-Yvelines* (2018). URL: https://defeo.lu/hdr/.

[119] Luca De Feo, Samuel Dobson, Steven D. Galbraith, and Lukas Zobernig. "SIDH Proof of Knowledge". In: *ASIACRYPT 2022, Part II.* Ed. by Shweta Agrawal and Dongdai Lin. Vol. 13792. LNCS. Springer, Cham, Dec. 2022, pp. 310–339. DOI: 10.1007/978-3-031-22966-4_11.

[120] Luca De Feo, Tako Boris Fouotsa, Péter Kutas, Antonin Leroux, Simon-Philipp Merz, Lorenz Panny, and Benjamin Wesolowski. "SCALLOP: Scaling the CSI-FiSh". In: *Public-Key Cryptography - PKC 2023 - 26th IACR International Conference on Practice and Theory of Public-Key Cryptography, Atlanta, GA, USA, May 7-10, 2023, Proceedings, Part I.* Vol. 13940. Lecture Notes in Computer Science. Springer, Cham, 2023, pp. 345–375. ISBN: 978-3-031-31367-7; 978-3-031-31368-4. DOI: 10.1007/978-3-031-31368-4_13.

[121] Luca De Feo, Tako Boris Fouotsa, and Lorenz Panny. "Isogeny Problems with Level Structure". In: *EUROCRYPT 2024, Part VII.* Ed. by Marc Joye and Gregor Leander. Vol. 14657. LNCS. Springer, Cham, May 2024, pp. 181–204. DOI: 10.1007/978-3-031-58754-2_7.

[122] Luca De Feo and Steven D. Galbraith. "SeaSign: Compact Isogeny Signatures from Class Group Actions". In: *EUROCRYPT 2019, Part III.* Ed. by Yuval Ishai and Vincent Rijmen. Vol. 11478. LNCS. Springer, Cham, May 2019, pp. 759–789. DOI: 10.1007/978-3-030-17659-4_26.

[123] Luca De Feo, David Jao, and Jérôme Plût. "Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies". In: *Journal of Mathematical Cryptology* 8.3 (2014), pp. 209–247. DOI: 10.1515/JMC-2012-0015.

[124] Luca De Feo, Jean Kieffer, and Benjamin Smith. "Towards Practical Key Exchange from Ordinary Isogeny Graphs". In: *ASIACRYPT 2018, Part III.* Ed. by Thomas Peyrin and Steven Galbraith. Vol. 11274. LNCS. Springer, Cham, Dec. 2018, pp. 365–394. DOI: 10.1007/978-3-030-03332-3_14.

[125] Luca De Feo, David Kohel, Antonin Leroux, Christophe Petit, and Benjamin Wesolowski. "SQISign: Compact Post-quantum Signatures from Quaternions and Isogenies". In: *ASIACRYPT 2020, Part I*. Ed. by Shiho Moriai and Huaxiong Wang. Vol. 12491. LNCS. Springer, Cham, Dec. 2020, pp. 64–93. DOI: 10.1007/978-3-030-64837-4_3.

[126] Luca De Feo, Antonin Leroux, Patrick Longa, and Benjamin Wesolowski. "New Algorithms for the Deuring Correspondence - Towards Practical and Secure SQISign Signatures". In: *EUROCRYPT 2023, Part V*. Ed. by Carmit Hazay and Martijn Stam. Vol. 14008. LNCS. Springer, Cham, Apr. 2023, pp. 659–690. DOI: 10.1007/978-3-031-30589-4_23.

[127] Luca De Feo and Michael Meyer. "Threshold Schemes from Isogeny Assumptions". In: *PKC 2020, Part II*. Ed. by Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas. Vol. 12111. LNCS. Springer, Cham, May 2020, pp. 187–212. DOI: 10.1007/978-3-030-45388-6_7.

[128] Luca De Feo, Cyprien Delpech de Saint Guilhem, Tako Boris Fouotsa, Péter Kutas, Antonin Leroux, Christophe Petit, Javier Silva, and Benjamin Wesolowski. "Séta: supersingular encryption from torsion attacks". In: *Advances in Cryptology - ASIACRYPT 2021 - 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6-10, 2021, Proceedings, Part IV*. Vol. 13093. Lecture Notes in Computer Science. Springer, Cham, 2021, pp. 249–278. ISBN: 978-3-030-92067-8; 978-3-030-92068-5. DOI: 10.1007/978-3-030-92068-5_9.

[129] Thomas Decru. "Radical Vélu Isogeny Formulae". In: *Advances in Cryptology – CRYPTO 2024*. Ed. by Leonid Reyzin and Douglas Stebila. Springer Nature Switzerland, 2024, pp. 107–128. ISBN: 978-3-031-68388-6.

[130] Thomas Decru and Sabrina Kunzweiler. "Efficient Computation of $(3^n, 3^n)$-Isogenies". In: *AFRICACRYPT 23*. Ed. by Nadia El Mrabet, Luca De Feo, and Sylvain Duquesne. Vol. 14064. LNCS. Springer, Cham, July 2023, pp. 53–78. DOI: 10.1007/978-3-031-37679-5_3.

[131] Thomas Decru, Lorenz Panny, and Frederik Vercauteren. "Faster SeaSign Signatures Through Improved Rejection Sampling". In: *Post-Quantum Cryptography - 10th International Conference, PQCrypto 2019*. Ed. by Jintai Ding and Rainer Steinwandt. Springer, Cham, 2019, pp. 271–285. DOI: 10.1007/978-3-030-25510-7_15.

[132] Christina Delfs and Steven D. Galbraith. "Computing isogenies between supersingular elliptic curves over $\mathbb{F}_p$". In: *DCC* 78.2 (2016), pp. 425–440. DOI: 10.1007/s10623-014-0010-1.

[133] Max Deuring. "Die Typen der Multiplikatorenringe elliptischer Funktionenkörper". In: *Abhandlungen aus dem Mathematischen Seminar der Hansischen Universität* 14 (1941), pp. 197–272. ISSN: 0025-5858. DOI: 10.1007/BF02940746.

[134] Karl Dickman. "On the frequency of numbers containing prime factors of a certain relative magnitude". In: *Arkiv for matematik, astronomi och fysik* 22.10 (1930), A–10.

[135] Denis Diemert. "On the Tight Security of the Transport Layer Security (TLS) Protocol Version 1.3". PhD thesis. University of Wuppertal, Germany, 2023.

[136] Whitfield Diffie and Martin E. Hellman. "New directions in cryptography". In: *IEEE Transactions on Information Theory* IT-22.6 (1976), pp. 644–654. ISSN: 0018-9448,1557-9654. DOI: 10.1109/tit.1976.1055638.

[137] Martin Djukanović. *Families of* $(3, 3)$*-split Jacobians*. 2019. arXiv: 1811.10075 [math.AG]. URL: https://arxiv.org/abs/1811.10075.

[138] Martin Djukanović. "Split Jacobians and lower bounds on heights". PhD thesis. Université de Bordeaux; Universiteit Leiden (Leyde, Pays-Bas), 2017.

[139] Igor V. Dolgachev and David Lehavi. "On isogenous principally polarized abelian surfaces". In: *Curves and abelian varieties*. Vol. 465. Contemporary Mathematics. Amer. Math. Soc., Providence, RI, 2008, pp. 51–69. ISBN: 978-0-8218-4334-5. DOI: 10.1090/conm/465/09100.

[140] Léo Ducas, Eike Kiltz, Tancrede Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. "CRYSTALS-Dilithium: A Lattice-Based Digital Signature Scheme". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2018.1 (2018), pp. 238–268. DOI: 10.13154/TCHES.V2018.I1.238-268.

[141] Alina Dudeanu, Dimitar Jetchev, Damien Robert, and Marius Vuille. "Cyclic Isogenies for Abelian Varieties with Real Multiplication". In: *Moscow Mathematical Journal* 22.4 (2022), pp. 613–655.

[142] Max Duparc and Tako Boris Fouotsa. "SQIPrime: A Dimension 2 Variant of SQISignHD with Non-smooth Challenge Isogenies". In: *Advances in Cryptology – ASIACRYPT 2024*. Ed. by Kai-Min Chung and Yu Sasaki. Singapore: Springer Nature Singapore, 2025, pp. 396–429. DOI: 10.1007/978-981-96-0891-1_13.

[143] Max Duparc, Tako Boris Fouotsa, and Serge Vaudenay. *SILBE: an Updatable Public Key Encryption Scheme from Lollipop Attacks*. Cryptology ePrint Archive, Paper 2024/400. 2024. URL: https://eprint.iacr.org/2024/400.

[144] Sylvain Duquesne and Gerhard Frey. "Background on pairings". In: *Handbook of elliptic and hyperelliptic curve cryptography*. Discrete Mathematics and its Applications (Boca Raton). Chapman & Hall/CRC, Boca Raton, FL, 2006, pp. 115–124. ISBN: 978-1-58488-518-4; 1-58488-518-1.

[145] Edward Eaton, David Jao, Chelsea Komlo, and Youcef Mokrani. "Towards Post-Quantum Key-Updatable Public-Key Encryption via Supersingular Isogenies". In: *SAC 2021*. Ed. by Riham AlTawy and Andreas Hülsing. Vol. 13203. LNCS. Springer, Cham, 2022, pp. 461–482. DOI: 10.1007/978-3-030-99277-4_22.

[146] Bas Edixhoven, Gerard Van der Geer, and Ben Moonen. *Abelian varieties*. Book Project. 2012. URL: http://van-der-geer.nl/gerard/AV.pdf.

[147] Kirsten Eisenträger, Sean Hallgren, Kristin E. Lauter, Travis Morrison, and Christophe Petit. "Supersingular Isogeny Graphs and Endomorphism Rings: Reductions and Solutions". In: *EUROCRYPT 2018, Part III*. Ed. by Jesper Buus Nielsen and Vincent Rijmen. Vol. 10822. LNCS. Springer, Cham, 2018, pp. 329–368. DOI: 10.1007/978-3-319-78372-7_11.

[148] Kirsten Eisenträger, Sean Hallgren, Chris Leonardi, Travis Morrison, and Jennifer Park. "Computing endomorphism rings of supersingular elliptic curves and connections to path-finding in isogeny graphs". In: *ANTS XIV—Proceedings of the Fourteenth Algorithmic Number Theory Symposium.* Vol. 4. The Open Book Series. Math. Sci. Publ., Berkeley, CA, 2020, pp. 215–232. ISBN: 978-1-935107-08-8; 978-1-935107-07-1. DOI: 10.2140/obs.2020.4.215.

[149] Noam D. Elkies. "Elliptic and modular curves over finite fields and related computational issues". In: *Computational perspectives on number theory.* Vol. 7. AMS/IP Stud. Adv. Math. Amer. Math. Soc., Providence, RI, 1998, pp. 21–76. ISBN: 0-8218-0880-X. DOI: 10.1090/amsip/007/03.

[150] Jonathan Komada Eriksen, Lorenz Panny, Jana Sotáková, and Mattia Veroni. "Deuring for the people: supersingular elliptic curves with prescribed endomorphism ring in general characteristic". In: *LuCaNT: LMFDB, computation, and number theory.* Vol. 796. Contemporary Mathematics. Amer. Math. Soc., Providence, RI, 2024, pp. 339–373. ISBN: 978-1-4704-7260-3. DOI: 10.1090/conm/796/16008.

[151] Amos Fiat and Adi Shamir. "How to Prove Yourself: Practical Solutions to Identification and Signature Problems". In: *CRYPTO'86.* Ed. by Andrew M. Odlyzko. Vol. 263. LNCS. Springer, Berlin, Heidelberg, Aug. 1987, pp. 186–194. DOI: 10.1007/3-540-47721-7_12.

[152] Tom Fisher. *On families of 13-congruent elliptic curves.* 2019. arXiv: 1912.10777 [math.NT]. URL: https://arxiv.org/abs/1912.10777.

[153] Tom Fisher. *On pairs of 17-congruent elliptic curves.* 2021. arXiv: 2106.02033 [math.NT]. URL: https://arxiv.org/abs/2106.02033.

[154] Enric Florit and Benjamin Smith. "An atlas of the Richelot isogeny graph". In: *Theory and Applications of Supersingular Curves and Supersingular Abelian Varieties.* Vol. B90. RIMS Kôkyûroku Bessatsu. Res. Inst. Math. Sci. (RIMS), Kyoto, 2022, pp. 195–219.

[155] Enric Florit and Benjamin Smith. "Automorphisms and isogeny graphs of abelian varieties, with applications to the superspecial Richelot isogeny graph". In: *Arithmetic, geometry, cryptography, and coding theory 2021.* Vol. 779. Contemporatry Mathematics. Amer. Math. Soc., Providence, RI, 2022, pp. 103–132. ISBN: 978-1-4704-6794-4. DOI: 10.1090/conm/779/15672.

[156] E. Victor Flynn. "Curves of genus 2". PhD thesis. University of Cambridge, 1989.

[157] E. Victor Flynn. "Descent via $(5, 5)$-isogeny on Jacobians of genus 2 curves". In: *Journal of Number Theory* 153 (2015), pp. 270–282. ISSN: 0022-314X,1096-1658. DOI: 10.1016/j.jnt.2015.01.018.

[158] E. Victor Flynn. "The Jacobian and formal group of a curve of genus 2 over an arbitrary ground field". In: *Mathematical Proceedings of the Cambridge Philosophical Society* 107.3 (1990), pp. 425–441. ISSN: 0305-0041,1469-8064. DOI: 10.1017/S0305004100068729.

[159] E. Victor Flynn and Yan Bo Ti. "Genus Two Isogeny Cryptography". In: *Post-Quantum Cryptography - 10th International Conference, PQCrypto 2019.* Ed. by Jintai Ding and Rainer Steinwandt. Springer, Cham, 2019, pp. 286–306. DOI: 10.1007/978-3-030-25510-7_16.

[160] Tako Boris Fouotsa, Tomoki Moriya, and Christophe Petit. “M-SIDH and MD-SIDH: Countering SIDH Attacks by Masking Information”. In: *EUROCRYPT 2023, Part V*. Ed. by Carmit Hazay and Martijn Stam. Vol. 14008. LNCS. Springer, Cham, Apr. 2023, pp. 282–309. DOI: 10.1007/978-3-031-30589-4_10.

[161] Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Prest, Thomas Ricosset, Gregor Seiler, William Whyte, Zhenfei Zhang, et al. “Falcon: Fast-Fourier lattice-based compact signatures over NTRU”. In: *Submission to the NIST’s post-quantum cryptography standardization process* 36.5 (2018), pp. 1–75.

[162] Sam Frengley. “On 12-congruences of elliptic curves”. In: *International Journal of Number Theory* 20.2 (2024), pp. 565–601. ISSN: 1793-0421,1793-7310. DOI: 10.1142/S1793042124500301.

[163] Gerhard Frey and Ernst Kani. “Curves of genus 2 covering elliptic curves and an arithmetical application”. In: *Arithmetic algebraic geometry*. Vol. 89. Progr. Math. Birkhäuser Boston, Boston, MA, 1991, pp. 153–176. ISBN: 0-8176-3513-0. DOI: 10.1007/978-1-4612-0457-2_7.

[164] Gerhard Frey and Tanja Lange. “Background on curves and Jacobians”. In: *Handbook of elliptic and hyperelliptic curve cryptography*. Discrete Mathematics and its Applications (Boca Raton). Chapman & Hall/CRC, Boca Raton, FL, 2006, pp. 45–85. ISBN: 978-1-58488-518-4; 1-58488-518-1.

[165] Gerhard Frey, Michael Müller, and Hans-Georg Rück. “The Tate pairing and the discrete logarithm applied to elliptic curve cryptosystems”. In: *IEEE Transactions on Information Theory* 45.5 (1999), pp. 1717–1719. ISSN: 0018-9448,1557-9654. DOI: 10.1109/18.771254.

[166] Gerhard Frey and Hans-Georg Rück. “A remark concerning m-divisibility and the discrete logarithm in the divisor class group of curves”. In: *Mathematics of Computation* 62.206 (1994), pp. 865–874. ISSN: 0025-5718,1088-6842. DOI: 10.2307/2153546.

[167] Eiichiro Fujisaki and Tatsuaki Okamoto. “Secure Integration of Asymmetric and Symmetric Encryption Schemes”. In: *CRYPTO’99*. Ed. by Michael J. Wiener. Vol. 1666. LNCS. Springer, Berlin, Heidelberg, Aug. 1999, pp. 537–554. DOI: 10.1007/3-540-48405-1_34.

[168] William Fulton. *Algebraic curves*. Advanced Book Classics. An introduction to algebraic geometry, Notes written with the collaboration of Richard Weiss, Reprint of 1969 original. Addison-Wesley Publishing Company, Advanced Book Program, Redwood City, CA, 1989, pp. xxii+226. ISBN: 0-201-51010-3.

[169] Jenny Fuselier, Annamaria Iezzi, Mark Kozek, Travis Morrison, and Changningphaabi Namoijam. *Computing supersingular endomorphism rings using inseparable endomorphisms*. 2023. arXiv: 2306.03051 [math.NT]. URL: https://arxiv.org/abs/2306.03051.

[170] Steven Galbraith. *Mathematics of public key cryptography*. Cambridge University Press, Cambridge, 2012, pp. xiv+615. ISBN: 978-1-107-01392-6. DOI: 10.1017/CBO9781139012843.

[171] Steven Galbraith. “Pairings”. In: *Advances in elliptic curve cryptography*. Vol. 317. London Mathematical Society Lecture Note Series. Cambridge Univ. Press, Cambridge, 2005, pp. 183–213. ISBN: 0-521-60415-X. DOI: 10.1017/CBO9780511546570.011.

[172] Steven D. Galbraith, Florian Hess, and Frederik Vercauteren. “Hyperelliptic Pairings (Invited Talk)”. In: *PAIRING 2007*. Ed. by Tsuyoshi Takagi, Tatsuaki Okamoto, Eiji Okamoto, and Takeshi Okamoto. Vol. 4575. LNCS. Springer, Berlin, Heidelberg, July 2007, pp. 108–131. DOI: 10.1007/978-3-540-73489-5_7.

[173] Steven D. Galbraith, Christophe Petit, Barak Shani, and Yan Bo Ti. “On the Security of Supersingular Isogeny Cryptosystems”. In: *ASIACRYPT 2016, Part I*. Ed. by Jung Hee Cheon and Tsuyoshi Takagi. Vol. 10031. LNCS. Springer, Berlin, Heidelberg, Dec. 2016, pp. 63–91. DOI: 10.1007/978-3-662-53887-6_3.

[174] Steven D. Galbraith, Christophe Petit, and Javier Silva. “Identification Protocols and Signature Schemes Based on Supersingular Isogeny Problems”. In: *ASIACRYPT 2017, Part I*. Ed. by Tsuyoshi Takagi and Thomas Peyrin. Vol. 10624. LNCS. Springer, Cham, Dec. 2017, pp. 3–33. DOI: 10.1007/978-3-319-70694-8_1.

[175] Pierrick Gaudry. “Fast genus 2 arithmetic based on theta functions”. In: *Journal of Mathematical Cryptology* 1.3 (2007), pp. 243–265. ISSN: 1862-2976,1862-2984. DOI: 10.1515/JMC.2007.012.

[176] Pierrick Gaudry and Éric Schost. “On the invariants of the quotients of the Jacobian of a curve of genus 2”. In: *Applied algebra, algebraic algorithms and error-correcting codes*. Vol. 2227. Lecture Notes in Computer Science. Springer, Berlin, 2001, pp. 373–386.

[177] Josep González, Jordi Guàrdia, and Victor Rotger. “Abelian surfaces of GL_2-type as Jacobians of curves”. In: *Acta Arithmetica* 116.3 (2005), pp. 263–287. ISSN: 0065-1036,1730-6264. DOI: 10.4064/aa116-3-3.

[178] David Grant. “Formal groups in genus two”. In: *Journal für die Reine und Angewandte Mathematik* 411 (1990), pp. 96–121. ISSN: 0075-4102,1435-5345. DOI: 10.1515/crll.1990.411.96.

[179] David Gruenewald. “Computing Humbert surfaces and applications”. In: *Arithmetic, geometry, cryptography and coding theory 2009*. Vol. 521. Contemp. Math. Amer. Math. Soc., Providence, RI, 2010, pp. 59–69.

[180] Mike Hamburg. *Fast and compact elliptic-curve cryptography*. Cryptology ePrint Archive, Paper 2012/309. 2012. URL: https://eprint.iacr.org/2012/309.

[181] Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. “A Modular Analysis of the Fujisaki-Okamoto Transformation”. In: *TCC 2017, Part I*. Ed. by Yael Kalai and Leonid Reyzin. Vol. 10677. LNCS. Springer, Cham, Nov. 2017, pp. 341–371. DOI: 10.1007/978-3-319-70500-2_12.

[182] Roger A. Horn and Charles R. Johnson. *Topics in matrix analysis*. Cambridge University Press, Cambridge, 1991, pp. viii+607. ISBN: 0-521-30587-X. DOI: 10.1017/CBO9780511840371.

[183] Everett W. Howe. “Principally polarized ordinary abelian varieties over finite fields”. In: *Transactions of the American Mathematical Society* 347.7 (1995), pp. 2361–2401. ISSN: 0002-9947,1088-6850. DOI: 10.2307/2154828.

[184] Dale Husemöller. *Elliptic curves*. Second. Vol. 111. Graduate Texts in Mathematics. With appendices by Otto Forster, Ruth Lawrence and Stefan Theisen. Springer-Verlag, New York, 2004, pp. xxii+487. ISBN: 0-387-95490-2.

[185] Aaron Hutchinson and Koray Karabina. "Constructing multidimensional differential addition chains and their applications". In: *Journal of Cryptographic Engineering* 9.1 (2019), pp. 1–19. DOI: 10.1007/S13389-017-0177-2.

[186] Tomoyoshi Ibukiyama and Toshiyuki Katsura. "On the field of definition of superspecial polarized abelian varieties and type numbers". In: *Compositio Mathematica* 91.1 (1994), pp. 37–46. ISSN: 0010-437X,1570-5846.

[187] Tomoyoshi Ibukiyama, Toshiyuki Katsura, and Frans Oort. "Supersingular curves of genus two and class numbers". In: *Compositio Mathematica* 57.2 (1986), pp. 127–152. ISSN: 0010-437X,1570-5846.

[188] Jun-ichi Igusa. "Arithmetic variety of moduli for genus two". In: *Annals of Mathematics. Second Series* 72 (1960), pp. 612–649. ISSN: 0003-486X. DOI: 10.2307/1970233.

[189] Jun-ichi Igusa. "Class number of a definite quaternion with prime discriminant". In: *Proceedings of the National Academy of Sciences of the United States of America* 44 (1958), pp. 312–314. ISSN: 0027-8424. DOI: 10.1073/pnas.44.4.312.

[190] Jun-ichi Igusa. "On Siegel modular forms of genus two". In: *American Journal of Mathematics* 84 (1962), pp. 175–200. ISSN: 0002-9327,1080-6377. DOI: 10.2307/2372812.

[191] David Jao, Stephen D. Miller, and Ramarathnam Venkatesan. "Do All Elliptic Curves of the Same Order Have the Same Difficulty of Discrete Log?" In: *ASIACRYPT 2005*. Ed. by Bimal K. Roy. Vol. 3788. LNCS. Springer, Berlin, Heidelberg, Dec. 2005, pp. 21–40. DOI: 10.1007/11593447_2.

[192] Samuel Jaques and John M. Schanck. "Quantum Cryptanalysis in the RAM Model: Claw-Finding Attacks on SIKE". In: *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part I*. Ed. by Alexandra Boldyreva and Daniele Micciancio. Vol. 11692. Lecture Notes in Computer Science. Springer, 2019, pp. 32–61. DOI: 10.1007/978-3-030-26948-7_2.

[193] Don Johnson, Alfred Menezes, and Scott A. Vanstone. "The Elliptic Curve Digital Signature Algorithm (ECDSA)". In: *International Journal of Information Security* 1.1 (2001), pp. 36–63. DOI: 10.1007/s102070100002.

[194] Bruce W. Jordan and Yevgeny Zaytman. *Isogeny graphs of superspecial abelian varieties and Brandt matrices*. 2023. arXiv: 2005.09031.

[195] Simon Josefsson and Ilari Liusvaara. "Edwards-Curve Digital Signature Algorithm (EdDSA)". In: *RFC* 8032 (2017), pp. 1–60. DOI: 10.17487/RFC8032.

[196] Antoine Joux. "A one round protocol for tripartite Diffie-Hellman". In: *Algorithmic number theory (Leiden, 2000)*. Vol. 1838. Lecture Notes in Computer Science. Springer, Berlin, 2000, pp. 385–393. ISBN: 3-540-67695-3. DOI: 10.1007/10722028_23.

[197] Ernst Kani and Wolfgang Schanz. “Modular diagonal quotient surfaces”. In: *Mathematische Zeitschrift* 227.2 (1998), pp. 337–366. ISSN: 0025-5874,1432-1823. DOI: 10.1007/PL00004379.

[198] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. “Constant-Size Commitments to Polynomials and Their Applications”. In: *ASIACRYPT 2010*. Ed. by Masayuki Abe. Vol. 6477. LNCS. Springer, Berlin, Heidelberg, Dec. 2010, pp. 177–194. DOI: 10.1007/978-3-642-17373-8_11.

[199] Shuichi Katsumata, Yi-Fu Lai, Jason T. LeGrow, and Ling Qin. “CSI-Otter: Isogeny-Based (Partially) Blind Signatures from the Class Group Action with a Twist”. In: *CRYPTO 2023, Part III*. Ed. by Helena Handschuh and Anna Lysyanskaya. Vol. 14083. LNCS. Springer, Cham, Aug. 2023, pp. 729–761. DOI: 10.1007/978-3-031-38548-3_24.

[200] Jonathan Katz. *Digital signatures*. Springer, New York, 2010, pp. xiv+192. ISBN: 978-0-387-27711-0. DOI: 10.1007/978-0-387-27712-7.

[201] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. Third. Chapman & Hall/CRC Cryptography and Network Security. CRC Press, Boca Raton, FL, 2021, pp. xx+626. ISBN: 978-0-8153-5436-9; 978-1-351-13303-6.

[202] Neal Koblitz. “Elliptic curve cryptosystems”. In: *Mathematics of Computation* 48.177 (1987), pp. 203–209. ISSN: 0025-5718,1088-6842. DOI: 10.2307/2007884.

[203] David Kohel, Kristin Lauter, Christophe Petit, and Jean-Pierre Tignol. “On the quaternion ℓ-isogeny path problem”. In: *LMS Journal of Computation and Mathematics* 17 (2014), pp. 418–432. ISSN: 1461-1570. DOI: 10.1112/S1461157014000151.

[204] David R. Kohel. *Endomorphism rings of elliptic curves over finite fields*. Thesis (Ph.D.)–University of California, Berkeley. ProQuest LLC, Ann Arbor, MI, 1996, p. 117. ISBN: 978-0591-32123-4.

[205] Robert M. Kuhn. “Curves of genus 2 with split Jacobian”. In: *Transactions of the American Mathematical Society* 307.1 (1988), pp. 41–49. ISSN: 0002-9947,1088-6850. DOI: 10.2307/2000749.

[206] Abhinav Kumar. “Hilbert modular surfaces for square discriminants and elliptic subfields of genus 2 function fields”. In: *Research in the Mathematical Sciences* 2 (2015), Art. 24, 46. ISSN: 2522-0144,2197-9847. DOI: 10.1186/s40687-015-0042-9.

[207] Sabrina Kunzweiler. “Efficient computation of $(2^n, 2^n)$-isogenies”. In: *DCC* 92.6 (2024), pp. 1761–1802. DOI: 10.1007/s10623-024-01366-1.

[208] Sabrina Kunzweiler, Luciano Maino, Tomoki Moriya, Christophe Petit, Giacomo Pope, Damien Robert, Miha Stopar, and Yan Bo Ti. *Radical 2-isogenies and cryptographic hash functions in dimensions 1, 2 and 3*. Cryptology ePrint Archive, Paper 2024/1732. 2024. URL: https://eprint.iacr.org/2024/1732.

[209] Sabrina Kunzweiler, Yan Bo Ti, and Charlotte Weitkämper. “Secret Keys in Genus-2 SIDH”. In: *SAC 2021*. Ed. by Riham AlTawy and Andreas Hülsing. Vol. 13203. LNCS. Springer, Cham, 2022, pp. 483–507. DOI: 10.1007/978-3-030-99277-4_23.

[210] Sabrina Kunzweiler, Yan Bo Ti, and Charlotte Weitkämper. *Secret Keys in Genus-2 SIDH*. Cryptology ePrint Archive, Paper 2021/990. 2021. URL: https://eprint.iacr.org/2021/990.

[211] Péter Kutas, Simon-Philipp Merz, Christophe Petit, and Charlotte Weitkämper. "One-Way Functions and Malleability Oracles: Hidden Shift Attacks on Isogeny-Based Protocols". In: *EUROCRYPT 2021, Part I*. Ed. by Anne Canteaut and François-Xavier Standaert. Vol. 12696. LNCS. Springer, Cham, Oct. 2021, pp. 242–271. DOI: 10.1007/978-3-030-77870-5_9.

[212] Serge Lang. *Abelian varieties*. Vol. No. 7. Interscience Tracts in Pure and Applied Mathematics. Interscience Publishers, Inc., New York; Interscience Publishers Ltd., London, 1959, pp. xii+256.

[213] Tanja Lange. "Efficient Arithmetic on Hyperelliptic Curves". Universität-Gesamthochschule Essen, 2001. URL: https://eprint.iacr.org/2002/107.pdf.

[214] Derrick H. Lehmer. "On a problem of Störmer". In: *Illinois Journal of Mathematics* 8 (1964), pp. 57–79. ISSN: 0019-2082. URL: http://projecteuclid.org/euclid.ijm/1256067456.

[215] Hendrik W. Lenstra. "Factoring integers with elliptic curves". In: *Annals of Mathematics. Second Series* 126.3 (1987), pp. 649–673. ISSN: 0003-486X,1939-8980. DOI: 10.2307/1971363.

[216] Christopher Leonardi. "Security Analysis of Isogeny-Based Cryptosystems". PhD thesis. University of Waterloo, Ontario, Canada, 2020. URL: https://hdl.handle.net/10012/16149.

[217] Antonin Leroux. "Quaternion Algebra and isogeny-based cryptography. (Algèbres de quaternions et cryptographie à base d'isogénies)". PhD thesis. Polytechnic Institute of Paris, France, 2022. URL: https://tel.archives-ouvertes.fr/tel-03886810.

[218] Antonin Leroux. *Verifiable random function from the Deuring correspondence and higher dimensional isogenies*. Cryptology ePrint Archive, Paper 2023/1251. 2023. URL: https://eprint.iacr.org/2023/1251.

[219] Stephen Lichtenbaum. "Duality theorems for curves over p-adic fields". In: *Inventiones Mathematicae* 7 (1969), pp. 120–136. ISSN: 0020-9910,1432-1297. DOI: 10.1007/BF01389795.

[220] Rudolf Lidl and Harald Niederreiter. *Introduction to finite fields and their applications*. Cambridge University Press, Cambridge, 1994, pp. xii+416. ISBN: 0-521-46094-8. DOI: 10.1017/CBO9781139172769.

[221] Kaizhan Lin, Weize Wang, Zheng Xu, and Chang-An Zhao. "A Faster Software Implementation of SQIsign". In: *IEEE Transactions on Information Theory* 70.9 (2024), pp. 1–1. DOI: 10.1109/TIT.2024.3423675.

[222] Jonathan Love and Dan Boneh. *Supersingular Curves With Small Non-integer Endomorphisms*. 2020. arXiv: 1910.03180 [math.NT]. URL: https://arxiv.org/abs/1910.03180.

[223] David Lubicz and Damien Robert. "Arithmetic on abelian and Kummer varieties". In: *Finite Fields and their Applications* 39 (2016), pp. 130–158. ISSN: 1071-5797,1090-2465. DOI: 10.1016/j.ffa.2016.01.009.

[224] David Lubicz and Damien Robert. “Fast change of level and applications to isogenies”. In: *Research in Number Theory* 9.1 (2023), Paper No. 7, 28. ISSN: 2522-0160,2363-9555. DOI: 10.1007/s40993-022-00407-9.

[225] Florian Luca and Filip Najman. “On the largest prime factor of $x^2 - 1$”. In: *Mathematics of Computation* 80.273 (2011), pp. 429–435. ISSN: 0025-5718,1088-6842. DOI: 10.1090/S0025-5718-2010-02381-6.

[226] Kay Magaard, Tanush Shaska, and Helmut Völklein. “Genus 2 curves that admit a degree 5 map to an elliptic curve”. In: *Forum Mathematicum* 21.3 (2009), pp. 547–566. ISSN: 0933-7741,1435-5337. DOI: 10.1515/FORUM.2009.027.

[227] Luciano Maino, Chloe Martindale, Lorenz Panny, Giacomo Pope, and Benjamin Wesolowski. “A Direct Key Recovery Attack on SIDH”. In: *EUROCRYPT 2023, Part V*. Ed. by Carmit Hazay and Martijn Stam. Vol. 14008. LNCS. Springer, Cham, Apr. 2023, pp. 448–471. DOI: 10.1007/978-3-031-30589-4_16.

[228] Chloe Martindale and Lorenz Panny. *How to not break SIDH*. Cryptology ePrint Archive, Report 2019/558. 2019. URL: https://eprint.iacr.org/2019/558.

[229] Alfred J. Menezes, Tatsuaki Okamoto, and Scott A. Vanstone. “Reducing elliptic curve logarithms to logarithms in a finite field”. In: *IEEE Transactions on Information Theory* 39.5 (1993), pp. 1639–1646. ISSN: 0018-9448,1557-9654. DOI: 10.1109/18.259647.

[230] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of applied cryptography*. CRC Press Series on Discrete Mathematics and its Applications. With a foreword by Ronald L. Rivest. CRC Press, Boca Raton, FL, 1997, pp. xxviii+780. ISBN: 0-8493-8523-7.

[231] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of applied cryptography*. CRC Press Series on Discrete Mathematics and its Applications. With a foreword by Ronald L. Rivest. CRC Press, Boca Raton, FL, 1997, pp. xxviii+780. ISBN: 0-8493-8523-7.

[232] Jean-François Mestre. “La méthode des graphes. Exemples et applications”. In: *Proceedings of the international conference on class numbers and fundamental units of algebraic number fields (Katata, 1986)*. Nagoya Univ., Nagoya, 1986, pp. 217–242.

[233] Jean-François Mestre. “Construction de courbes de genre 2 à partir de leurs modules”. In: *Effective methods in algebraic geometry*. Vol. 94. Progr. Math. Birkhäuser Boston, Boston, MA, 1991, pp. 313–334. ISBN: 0-8176-3546-7.

[234] Michael Meyer and Steffen Reith. “A Faster Way to the CSIDH”. In: *INDOCRYPT 2018*. Ed. by Debrup Chakraborty and Tetsu Iwata. Vol. 11356. LNCS. Springer, Cham, Dec. 2018, pp. 137–152. DOI: 10.1007/978-3-030-05378-9_8.

[235] Victor S. Miller. “The Weil Pairing, and Its Efficient Calculation”. In: *Journal of Cryptology* 17.4 (Sept. 2004), pp. 235–261. DOI: 10.1007/s00145-004-0315-8.

[236] Victor S. Miller. “Use of Elliptic Curves in Cryptography”. In: *Advances in Cryptology - CRYPTO ’85, Santa Barbara, California, USA, August 18-22, 1985, Proceedings*. Ed. by Hugh C. Williams. Vol. 218. Lecture Notes in Computer Science. Springer, 1985, pp. 417–426. DOI: 10.1007/3-540-39799-X_31.

[237] James S. Milne. “Abelian varieties”. In: *Arithmetic geometry*. Springer, New York, 1986, pp. 103–150. ISBN: 0-387-96311-1.

[238] James S. Milne. *Fields and Galois theory*. Kea Books, Ann Arbor, MI, 2022, pp. viii+189. ISBN: 979-8-218-07399-2.

[239] James S. Milne. “Jacobian varieties”. In: *Arithmetic geometry*. Springer, New York, 1986, pp. 167–212. ISBN: 0-387-96311-1.

[240] Arno Mittelbach and Marc Fischlin. *The Theory of Hash Functions and Random Oracles - An Approach to Modern Cryptography*. Information Security and Cryptography. Springer, 2021. ISBN: 978-3-030-63286-1. DOI: 10.1007/978-3-030-63287-8.

[241] Peter L. Montgomery. “Speeding the Pollard and elliptic curve methods of factorization”. In: *Mathematics of Computation* 48.177 (1987), pp. 243–264. ISSN: 0025-5718,1088-6842. DOI: 10.2307/2007888.

[242] Dustin Moody and Daniel Shumow. “Analogues of Vélu’s formulas for isogenies on alternate models of elliptic curves”. In: *Mathematics of Computation* 85.300 (2016), pp. 1929–1951. ISSN: 0025-5718,1088-6842. DOI: 10.1090/mcom/3036.

[243] Tomoki Moriya, Hiroshi Onuki, and Tsuyoshi Takagi. “How to Construct CSIDH on Edwards Curves”. In: *CT-RSA 2020*. Ed. by Stanislaw Jarecki. Vol. 12006. LNCS. Springer, Cham, Feb. 2020, pp. 512–537. DOI: 10.1007/978-3-030-40186-3_22.

[244] David Mumford. *Abelian varieties*. Vol. 5. Tata Institute of Fundamental Research Studies in Mathematics. Tata Institute of Fundamental Research, Bombay; by Oxford University Press, London, 1970, pp. viii+242.

[245] David Mumford. *Tata lectures on theta. II*. Modern Birkhäuser Classics. Jacobian theta functions and differential equations, With the collaboration of C. Musili, M. Nori, E. Previato, M. Stillman and H. Umemura, Reprint of the 1984 original. Birkhäuser Boston, Inc., Boston, MA, 2007, pp. xiv+272. ISBN: 978-0-8176-4569-4; 0-8176-4569-1. DOI: 10.1007/978-0-8176-4578-6.

[246] Kohei Nakagawa and Hiroshi Onuki. “QFESTA: Efficient Algorithms and Parameters for FESTA Using Quaternion Algebras”. In: *Advances in Cryptology – CRYPTO 2024*. Ed. by Leonid Reyzin and Douglas Stebila. Springer Nature Switzerland, 2024, pp. 75–106. ISBN: 978-3-031-68388-6.

[247] Kohei Nakagawa, Hiroshi Onuki, Wouter Castryck, Mingjie Chen, Riccardo Invernizzi, Gioella Lorenzon, and Frederik Vercauteren. “SQIsign2D-East: A New Signature Scheme Using 2-Dimensional Isogenies”. In: *Advances in Cryptology – ASIACRYPT 2024*. Ed. by Kai-Min Chung and Yu Sasaki. Singapore: Springer Nature Singapore, 2025, pp. 272–303. DOI: 10.1007/978-981-96-0891-1_9.

[248] Chris Nicholls. “Descent methods and torsion on Jacobians of higher genus curves”. PhD thesis. University of Oxford, 2018.

[249] Ryo Ohashi. “On the Rosenhain forms of superspecial curves of genus two”. In: *Finite Fields Appl.* 97 (2024), Paper No. 102445, 25. ISSN: 1071-5797,1090-2465. DOI: 10.1016/j.ffa.2024.102445.

[250] Katsuyuki Okeya, Hiroyuki Kurumatani, and Kouichi Sakurai. “Elliptic Curves with the Montgomery-Form and Their Cryptographic Applications”. In: *PKC 2000*. Ed. by Hideki Imai and Yuliang Zheng. Vol. 1751. LNCS. Springer, Berlin, Heidelberg, Jan. 2000, pp. 238–257. DOI: 10.1007/978-3-540-46588-1_17.

[251] Hiroshi Onuki and Tomoki Moriya. “Radical Isogenies on Montgomery Curves”. In: *PKC 2022, Part I*. Ed. by Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe. Vol. 13177. LNCS. Springer, Cham, Mar. 2022, pp. 473–497. DOI: 10.1007/978-3-030-97121-2_17.

[252] Hiroshi Onuki and Kohei Nakagawa. “Ideal-to-Isogeny Algorithm Using 2-Dimensional Isogenies and Its Application to SQIsign”. In: *Advances in Cryptology – ASIACRYPT 2024*. Ed. by Kai-Min Chung and Yu Sasaki. Singapore: Springer Nature Singapore, 2025, pp. 243–271. DOI: 10.1007/978-981-96-0891-1_8.

[253] Paul C. van Oorschot and Michael J. Wiener. “Parallel collision search with cryptanalytic applications”. In: *Journal of Cryptology* 12.1 (1999), pp. 1–28. ISSN: 0933-2790,1432-1378. DOI: 10.1007/PL00003816.

[254] Frans Oort. “A stratification of a moduli space of abelian varieties”. In: *Moduli of abelian varieties*. Vol. 195. Progr. Math. Birkhäuser, Basel, 2001, pp. 345–416. ISBN: 3-7643-6517-X. DOI: 10.1007/978-3-0348-8303-0_13.

[255] Frans Oort. “Subvarieties of moduli spaces”. In: *Inventiones Mathematicae* 24 (1974), pp. 95–119. ISSN: 0020-9910,1432-1297. DOI: 10.1007/BF01404301.

[256] Frans Oort and Kenji Ueno. “Principally polarized abelian varieties of dimension two or three are Jacobian varieties”. In: *Journal of the Faculty of Science. University of Tokyo. IA Math.* 20 (1973), pp. 377–381. ISSN: 0040-8980.

[257] Aurel Page and Benjamin Wesolowski. “The Supersingular Endomorphism Ring and One Endomorphism Problems are Equivalent”. In: *EUROCRYPT 2024, Part VI*. Ed. by Marc Joye and Gregor Leander. Vol. 14656. LNCS. Springer, Cham, May 2024, pp. 388–417. DOI: 10.1007/978-3-031-58751-1_14.

[258] Chris Peikert. “He Gives C-Sieves on the CSIDH”. In: *EUROCRYPT 2020, Part II*. Ed. by Anne Canteaut and Yuval Ishai. Vol. 12106. LNCS. Springer, Cham, May 2020, pp. 463–492. DOI: 10.1007/978-3-030-45724-2_16.

[259] Christophe Petit. “Faster Algorithms for Isogeny Problems Using Torsion Point Images”. In: *ASIACRYPT 2017, Part II*. Ed. by Tsuyoshi Takagi and Thomas Peyrin. Vol. 10625. LNCS. Springer, Cham, Dec. 2017, pp. 330–353. DOI: 10.1007/978-3-319-70697-9_12.

[260] Christophe Petit and Spike Smith. “An improvement to the quaternion analogue of the l-isogeny problem”. In: *Presentation at MathCrypt* (2018). URL: https://crypto.iacr.org/2018/affevents/mathcrypt/page.html.

[261] Arnold K. Pizer. “Ramanujan graphs and Hecke operators”. In: *American Mathematical Society. Bulletin. New Series* 23.1 (1990), pp. 127–137. ISSN: 0273-0979,1088-9485. DOI: 10.1090/S0273-0979-1990-15918-X.

[262] Victoria de Quehen, Péter Kutas, Chris Leonardi, Chloe Martindal, Lorenz Panny, Christophe Petit, and Katherine E. Stange. “Improved torsion-point attacks on SIDH variants”. In: *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part III.* Vol. 12827. Lecture Notes in Computer Science. Springer, Cham, 2021, pp. 432–470. ISBN: 978-3-030-84251-2; 978-3-030-84252-9. DOI: 10.1007/978-3-030-84252-9_15.

[263] Srinivasa Rao Subramanya Rao. “Three Dimensional Montgomery Ladder, Differential Point Tripling on Montgomery Curves and Point Quintupling on Weierstrass’ and Edwards Curves”. In: *AFRICACRYPT 16.* Ed. by David Pointcheval, Abderrahmane Nitaj, and Tajjeeddine Rachidi. Vol. 9646. LNCS. Springer, Cham, Apr. 2016, pp. 84–106. DOI: 10.1007/978-3-319-31517-1_5.

[264] Miles Reid. *Undergraduate algebraic geometry.* Vol. 12. London Mathematical Society Student Texts. Cambridge University Press, Cambridge, 1988, pp. viii+129. ISBN: 0-521-35559-1; 0-521-35662-8. DOI: 10.1017/CBO9781139163699.

[265] Krijn Reijnders. “Effective Pairings in Isogeny-Based Cryptography”. In: *LATINCRYPT 2023.* Ed. by Abdelrahaman Aly and Mehdi Tibouchi. Vol. 14168. LNCS. Springer, Cham, Oct. 2023, pp. 109–128. DOI: 10.1007/978-3-031-44469-2_6.

[266] Joost Renes. “Computing Isogenies Between Montgomery Curves Using the Action of (0, 0)”. In: *Post-Quantum Cryptography - 9th International Conference, PQCrypto 2018.* Ed. by Tanja Lange and Rainer Steinwandt. Springer, Cham, 2018, pp. 229–247. DOI: 10.1007/978-3-319-79063-3_11.

[267] Joost Renes, Peter Schwabe, Benjamin Smith, and Lejla Batina. “μKummer: Efficient Hyperelliptic Signatures and Key Exchange on Microcontrollers”. In: *CHES 2016.* Ed. by Benedikt Gierlichs and Axel Y. Poschmann. Vol. 9813. LNCS. Springer, Berlin, Heidelberg, Aug. 2016, pp. 301–320. DOI: 10.1007/978-3-662-53140-2_15.

[268] Joost Renes and Benjamin Smith. “qDSA: Small and Secure Digital Signatures with Curve-Based Diffie-Hellman Key Pairs”. In: *ASIACRYPT 2017, Part II.* Ed. by Tsuyoshi Takagi and Thomas Peyrin. Vol. 10625. LNCS. Springer, Cham, Dec. 2017, pp. 273–302. DOI: 10.1007/978-3-319-70697-9_10.

[269] Friedrich Julius Richelot. “De transformatione integralium Abelianorum primi ordinis commentatio. Caput secundum. De computatione integralium Abelianorum primi ordinis”. In: *Journal für die Reine und Angewandte Mathematik* 16 (1837), pp. 285–341. ISSN: 0075-4102,1435-5345. DOI: 10.1515/crll.1837.16.285.

[270] Ronald Rivest, Adi Shamir, and Len Adleman. “A method for obtaining digital signatures and public-key cryptosystems (1978)”. In: *Ideas that created the future—classic papers of computer science.* Reprinted from [0700103]. MIT Press, Cambridge, MA, 2021, pp. 463–474. ISBN: [9780262045308].

[271] Damien Robert. “Breaking SIDH in Polynomial Time”. In: *EUROCRYPT 2023, Part V*. Ed. by Carmit Hazay and Martijn Stam. Vol. 14008. LNCS. Springer, Cham, Apr. 2023, pp. 472–503. DOI: 10.1007/978-3-031-30589-4_17.

[272] Damien Robert. “Efficient algorithms for abelian varieties and their moduli spaces”. PhD thesis. Université de Bordeaux (UB), 2021. URL: https://hal.science/tel-03498268.

[273] Damien Robert. “Theta functions and cryptographic applications”. PhD thesis. Université Henri Poincaré - Nancy, 2010.

[274] Phillip Rogaway and Thomas Shrimpton. “Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance”. In: *Fast Software Encryption, 11th International Workshop, FSE 2004, Delhi, India, February 5-7, 2004, Revised Papers*. Ed. by Bimal K. Roy and Willi Meier. Vol. 3017. Lecture Notes in Computer Science. Springer, 2004, pp. 371–388. DOI: 10.1007/978-3-540-25937-4_24.

[275] Alexander Rostovtsev and Anton Stolbunov. *Public-Key Cryptosystem based on Isogenies*. Cryptology ePrint Archive, Paper 2006/145. 2006. URL: https://eprint.iacr.org/2006/145.

[276] Cyprien Delpech de Saint Guilhem and Robi Pedersen. “New proof systems and an OPRF from CSIDH”. In: *Public-Key Cryptography - PKC 2024 - 27th IACR International Conference on Practice and Theory of Public-Key Cryptography, Sydney, NSW, Australia, April 15-17, 2024, Proceedings, Part III*. Vol. 14603. Lecture Notes in Computer Science. Springer, Cham, 2024, pp. 217–251. ISBN: 978-3-031-57724-6; 978-3-031-57725-3. DOI: 10.1007/978-3-031-57725-3_8.

[277] Maria Corte-Real Santos, Jonathan Komada Eriksen, Michael Meyer, and Francisco Rodríguez-Henríquez. “Finding Practical Parameters for Isogeny-based Cryptography”. In: *IACR Communications in Cryptology* 1.3 (Oct. 7, 2024). ISSN: 3006-5496. DOI: 10.62056/ayojbhey6b.

[278] René Schoof. “Counting points on elliptic curves over finite fields”. In: vol. 7. 1. Les Dix-huitièmes Journées Arithmétiques. 1995, pp. 219–254. URL: http://jtnb.cedram.org/item?id=JTNB_1995__7_1_219_0.

[279] René Schoof. “Counting points on elliptic curves over finite fields”. In: vol. 7. 1. Les Dix-huitièmes Journées Arithmétiques. 1995, pp. 219–254. URL: http://jtnb.cedram.org/item?id=JTNB_1995__7_1_219_0.

[280] René Schoof. “Elliptic curves over finite fields and the computation of square roots mod p”. In: *Mathematics of Computation* 44.170 (1985), pp. 483–494. ISSN: 0025-5718,1088-6842. DOI: 10.2307/2007968.

[281] Michael Scott. *A note on the calculation of some functions in finite fields: Tricks of the Trade*. Cryptology ePrint Archive, Paper 2020/1497. 2020. URL: https://eprint.iacr.org/2020/1497.

[282] Michael Scott. *Elliptic Curve Cryptography for the masses: Simple and fast finite field arithmetic*. Cryptology ePrint Archive, Paper 2024/779. 2024. URL: https://eprint.iacr.org/2024/779.

[283] Tanush Shaska. "Curves of genus 2 with (N, N) decomposable Jacobians". In: *Journal of Symbolic Computation* 31.5 (2001), pp. 603–617. ISSN: 0747-7171,1095-855X. DOI: 10.1006/jsco.2001.0439.

[284] Tanush Shaska. *Curves of genus two covering elliptic curves*. University of Florida, 2001.

[285] Tanush Shaska. "Genus 2 fields with degree 3 elliptic subfields". In: *Forum Mathematicum* 16.2 (2004), pp. 263–280. ISSN: 0933-7741,1435-5337. DOI: 10.1515/form.2004.013.

[286] Tanush Shaska and Helmut Völklein. "Elliptic subfields and automorphisms of genus 2 function fields". In: *Algebra, arithmetic and geometry with applications*. Springer, Berlin, 2004, pp. 703–723. ISBN: 3-540-00475-0.

[287] Tanush Shaska, G. S. Wijesiri, Steve Wolf, and Lindsey Woodland. "Degree 4 coverings of elliptic curves by genus 2 curves". In: *Albanian Journal of Mathematics* 2.4 (2008), pp. 307–318. ISSN: 1930-1235.

[288] Peter W. Shor. "Algorithms for quantum computation: discrete logarithms and factoring". In: *35th Annual Symposium on Foundations of Computer Science (Santa Fe, NM, 1994)*. IEEE Comput. Soc. Press, Los Alamitos, CA, 1994, pp. 124–134. ISBN: 0-8186-6580-7. DOI: 10.1109/SFCS.1994.365700.

[289] Victor Shoup. *A computational introduction to number theory and algebra*. Second. Cambridge University Press, Cambridge, 2009, pp. xviii+580. ISBN: 978-0-521-51644-0.

[290] Victor Shoup. "Efficient computation of minimal polynomials in algebraic extensions of finite fields". In: *Proceedings of the 1999 International Symposium on Symbolic and Algebraic Computation (Vancouver, BC)*. ACM, New York, 1999, pp. 53–58. DOI: 10.1145/309831.309859.

[291] *Signal Protocol Specification*. https://signal.org/docs/. Accessed: 2024-08-25.

[292] Joseph H. Silverman. *The Arithmetic of Elliptic Curves*. Vol. 106. Graduate Texts in Mathematics. Springer-Verlag, New York, 1986, pp. xii+400. ISBN: 0-387-96203-4. DOI: 10.1007/978-1-4757-1920-8.

[293] Benjamin Smith. "Computing low-degree isogenies in genus 2 with the Dolgachev-Lehavi method". In: *Arithmetic, geometry, cryptography and coding theory*. Vol. 574. Contemporary Mathematics. Amer. Math. Soc., Providence, RI, 2012, pp. 159–170. ISBN: 978-0-8218-7572-8. DOI: 10.1090/conm/574/11418.

[294] Benjamin Smith. "Explicit endomorphisms and correspondences". PhD thesis. 2005.

[295] Damien Stehlé and Paul Zimmermann. "A Binary Recursive Gcd Algorithm". In: *Algorithmic Number Theory*. Ed. by Duncan Buell. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 411–425. ISBN: 978-3-540-24847-7.

[296] Bruno Sterner. *Towards Optimally Small Smoothness Bounds for Cryptographic-Sized Twin Smooth Integers and their Isogeny-based Applications*. Cryptology ePrint Archive, Paper 2023/1576. 2023. URL: https://eprint.iacr.org/2023/1576.

[297] Ian Stewart. *Galois theory.* Fourth. CRC Press, Boca Raton, FL, 2015, pp. xxii+321. ISBN: 978-1-4822-4582-0.

[298] Henning Stichtenoth. *Algebraic function fields and codes.* Universitext. Springer-Verlag, Berlin, 1993, pp. x+260. ISBN: 3-540-56489-6.

[299] Anton Stolbunov. "Constructing public-key cryptographic schemes based on class group action on a set of isogenous elliptic curves". In: *Advances in Mathematics of Communications* 4.2 (2010), pp. 215–235. ISSN: 1930-5346,1930-5338. DOI: 10.3934/amc.2010.4.215.

[300] Carl Størmer. "Quelques théorèmes sur l'équation de Pell $x^2 - Dy^2 = \pm 1$ et leurs applications". In: *Christiania Videnskabens Selskabs Skrifter, Math. Nat. Kl* 2 (1897), p. 48.

[301] Andrew V. Sutherland. *Modular Polynomials.* https://math.mit.edu/~drew/ClassicalModPolys.html. Online; accessed 30 September 2021.

[302] Andrew V. Sutherland. "On the evaluation of modular polynomials". In: *ANTS X—Proceedings of the Tenth Algorithmic Number Theory Symposium.* Vol. 1. The Open Book Series. Math. Sci. Publ., Berkeley, CA, 2013, pp. 531–555. ISBN: 978-1-935107-01-9; 978-1-935107-00-2. DOI: 10.2140/obs.2013.1.531.

[303] Katsuyuki Takashima. "Efficient algorithms for isogeny sequences and their cryptographic applications". In: *Mathematical modelling for next-generation cryptography.* Vol. 29. Mathematics for Industry (Tokyo). Springer, Singapore, 2018, pp. 97–114. ISBN: 978-981-10-5064-0; 978-981-10-5065-7. DOI: 10.1007/978-981-10-5065-7_6.

[304] Katsuyuki Takashima and Reo Yoshida. "An algorithm for computing a sequence of Richelot isogenies". In: *Bulletin of the Korean Mathematical Society* 46.4 (2009), pp. 789–802. ISSN: 1015-8634,2234-3016. DOI: 10.4134/BKMS.2009.46.4.789.

[305] Seiichiro Tani. "Claw finding algorithms using quantum walk". In: *Theoretical Computer Science* 410.50 (2009), pp. 5285–5297. ISSN: 0304-3975,1879-2294. DOI: 10.1016/j.tcs.2009.08.030.

[306] John Tate. "Endomorphisms of abelian varieties over finite fields". In: *Inventiones Mathematicae* 2 (1966), pp. 134–144. ISSN: 0020-9910,1432-1297. DOI: 10.1007/BF01404549.

[307] Gérald Tenenbaum. "Integers with a large friable component". In: *Acta Arithmetica* 124.3 (2006), pp. 287–291. ISSN: 0065-1036,1730-6264. DOI: 10.4064/aa124-3-6.

[308] Gérald Tenenbaum. *Introduction à la théorie analytique et probabiliste des nombres.* Second. Vol. 1. Cours Spécialisés. Société Mathématique de France, Paris, 1995, pp. xv+457. ISBN: 2-85629-032-9.

[309] The National Institute of Standards and Technology (NIST). *Call for Additional Digital Signature Schemes for the Post-Quantum Cryptography Standardization Process.* October, 2022.

[310] The National Institute of Standards and Technology (NIST). *Submission Requirements and Evaluation Criteria for the Post-Quantum Cryptography Standardization Process.* Dec. 2016.

[311] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 9.2).* https://www.sagemath.org. 2021.

[312] Kiminori Tsukazaki. “Explicit isogenies of elliptic curves”. PhD thesis. University of Warwick, 2013.

[313] Dominique Unruh. “Non-Interactive Zero-Knowledge Proofs in the Quantum Random Oracle Model”. In: *EUROCRYPT 2015, Part II.* Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9057. LNCS. Springer, Berlin, Heidelberg, Apr. 2015, pp. 755–784. DOI: 10.1007/978-3-662-46803-6_25.

[314] Jacques Vélu. “Isogénies entre courbes elliptiques”. In: *Comptes Rendus Hebdomadaires des Séances de l’Académie des Sciences. Séries A et B* 273 (1971), A238–A241. ISSN: 0151-0509.

[315] Daniele Venturi. *Zero-knowledge proofs and applications.* Lecture Notes, University of Rome. 2005. URL: https://danieleventuri.altervista.org/files/zero-knowledge.pdf.

[316] John Voight. *Quaternion algebras.* Vol. 288. Graduate Texts in Mathematics. Springer, Cham, 2021. ISBN: 978-3-030-56692-0; 978-3-030-56694-4. DOI: 10.1007/978-3-030-56694-4.

[317] William C. Waterhouse. “Abelian varieties over finite fields”. In: *Annales Scientifiques de l’École Normale Supérieure. Quatrième Série* 2 (1969), pp. 521–560. ISSN: 0012-9593.

[318] Benjamin Wesolowski. “The supersingular isogeny path and endomorphism ring problems are equivalent”. In: *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science—FOCS 2021.* IEEE Computer Soc., Los Alamitos, CA, 2022, pp. 1100–1111. ISBN: 978-1-6654-2055-6. DOI: 10.1109/FOCS52979.2021.00109.

[319] *WhatsApp Encryption Overview.* Technical Report. 2024. URL: https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf.