Security on the Line:
Modern Curve-based Cryptography

ISBN: 978–94–6323–695–9
Typeset using LaTeX

Cover design: Ilse Modder – `www.ilsemodder.nl`
Printed by: Gildeprint – `www.gildeprint.nl`

Radboud University

Applied and
Engineering Sciences

This work is part of the research programme TYPHOON with project number 13499, which is (partly) financed by the Netherlands Organisation for Scientific Research (NWO).

Security on the Line:
Modern Curve-based Cryptography

Proefschrift
ter verkrijging van de graad van doctor
aan de Radboud Universiteit Nijmegen
op gezag van de rector magnificus prof. dr. J.H.J.M. van Krieken,
volgens besluit van het college van decanen
in het openbaar te verdedigen op maandag 1 juli 2019
om 14.30 uur precies

door
Joost Roland Renes

Promotor

Prof. dr. L. Batina

Manuscriptcommissie

Prof. dr. E.R. Verheul
Prof. dr. S.D. Galbraith (University of Auckland, Nieuw-Zeeland)
Prof. dr. A.J. Menezes (University of Waterloo, Canada)
Dr. N. Heninger (University of California San Diego, Verenigde Staten)
Dr. F. Vercauteren (KU Leuven, België)

Acknowledgements

A unique feature of doing a PhD is that by the end of it, one is expected to deliver a book detailing all personal contributions to the field of study. This inherently highlights its individual nature, yet one would be wrong to think all these years are spent alone in a dark office (if anything, because no PhD student would ever be given their own office). My academic journey has given me the opportunity to experience life in many different places, and to visit far too many to list here. I would like to take this chance to thank those who have been there along the way to make it all the more worthwhile.

First and foremost I would like to thank my promotor Lejla Batina. Her immediate enthusiasm convinced me to start on this path, and I have never felt a lack of professional or personal support during. I am very grateful to have had the chance to work with you and for the many exciting things that have happened because of it.

During my PhD I have been lucky enough to work with great people. A special thanks goes out to Craig Costello, whom I met at the very beginning of my PhD. We not only collaborated on my first paper (and others after that), but you also gave me the opportunity to spend three great summers at Microsoft Research. In extension, I would like to thank Michael Naehrig for being a great (co-)mentor during said internships. I am proud of the work I have done with both of you, and slightly embarrassed of the persistent failures on the soccer field. I would also like to thank Brian LaMacchia, and the rest of the team, for repeatedly inviting me to the group and creating a great work environment. A final thanks to Ben Smith, whom I had the pleasure to work with and whose impressively detailed yet simple way of describing many topics has tremendously improved my understanding numerous times.

I would like to thank the members of my reading committee Eric Verheul, Steven Galbraith, Alfred Menezes, Nadia Heninger and Fréderik Vercauteren for taking the time to go through this lengthy thesis. Also, I thank my co-authors Wouter Castryck, Huseyin Hisil, David Jao, Tanja Lange, Patrick Longa, Chloe Martindale, Lorenz

Panny, Peter Schwabe, David Urbanik and Fernando Virdia for their hard work and the great discussions along the way. A particular thanks to Fréderik Vercauteren and Steven Galbraith for inviting me to spend some time at their respective research groups, and to Lorenz Panny for his detailed and helpful comments on a preliminary version of the first part of this thesis.

Despite all the moving around, most of my time has still been spent in Nijmegen. I have been a proud member of the Digital Security Group, all of whose (past) members I would like to thank for their warmth and kindness through coffee breaks, Friday beers and game nights. I am proud to have created many friendships that I am sure will last beyond the brief scope of a PhD. In particular, a special thanks goes out to Pedro, who has literally been there since day one on the job. It has been a pleasure to have shared an office for all this time, and I consider it a miracle that I have not gained weight over the years. I would also like to thank Louiza for her friendship and advice through the years, whom I was honored to be a paranimph for. I thank Joan Daemen and Peter Schwabe for their support and nice discussions related to teaching and research.

I would like to thank my parents and my brothers for their continued support, for always giving me a place to return home to, and for being there during trying times.

Finally, a heartfelt thanks to Anna for her love and invaluable support during the concluding months of this thesis, which have without a doubt been the most challenging. I am proud to start the next journey by your side.

Joost Renes
Nijmegen, May 2019

Contents

Introduction

The main theme (and title) of this thesis is "modern curve-based cryptography". Indeed, the history of cryptography is long and leads back to the early Egyptians and Romans. Its principal goal was (and is) to provide means for secure communication. However, one must wonder what it means to be secure. A typical example sketches a Roman general during wartime sending a message to one of their soldiers. This leads to several questions. Firstly, is there a chance that this message may be intercepted? If so, is the content of the message sensitive and not to be read by any other party? If the answer to the second question is positive, the general could consider using a cipher to encrypt the message. Yet, the sensitivity of the content may depend on the timeliness of the interception. If the general wants to send the message "Attack in one day!" he likely does not care if the opponent deciphers it in two days. The cipher the general chooses therefore depends not only on how good the opponent is at deciphering it, but also on how much time they have to do so. During Roman wartime, there were no doubt many more factors to take into consideration. The main takeaway is that understanding the context in which cryptography is used is crucial to understand its security.

This inherently leads to the question of what context cryptography is used in present day. The answer, as usual, is that it depends. Much of our communication nowadays is done through the internet, say our laptops connecting to a server of a bank to perform a transaction. Hopefully, only a small trusted set of people can access our laptop while only the bank has access to its server.[1] Untrusted parties can observe the communication between the laptop and the server, but can not view computations executed locally. Such parties are typically called *passive* adversaries (since they do nothing but listen). At first glance the situation appears analogous to that of the Roman general and the soldier, but there is a crucial difference in assumptions. We can reasonably presume that the Roman army has had time to pre-

[1] Whether or not to trust banks is a separate issue that has lead to the development of *cryptocurrencies* (or awkwardly abbreviated as *crypto*), which is not to be confused with cryptography (or currency).

pare (say, in Rome) before going out into the battlefield. During this time, they could have communicated a secret value (or a *key*). Once out in adversarial territory, they can use this shared key to encrypt their messages. Note that such an assumption cannot sensibly be made for a person connecting to the server of their bank. For example, what if one lives outside of Europe and wants to connect to a server in the Netherlands? Even if one could bootstrap communication by having a private meeting once, one should consider what happens if the key is compromised. Moreover, is such a setup feasible for banks with huge customer bases? Indeed, this problem is difficult and is known as the *key distribution problem*.

It is (more or less) in this context that a major breakthrough in cryptography happened. In 1976 the (now) famous cryptographers Whitfield Diffie and Martin E. Hellman [DH76] proposed to split the cryptographic primitive in two parts; a private operation that is done locally by each party separately (on the laptop and on the server) using a *private key*, leading to a public piece of data called a *public key*. The laptop and the server can now exchange their public keys and derive a shared key from them. As long as the passive adversary cannot learn any information about the private keys or the shared key from learning the public keys, we can assume the two parties have exchanged a key that is only known to them and henceforth use a cipher for communication.

Although extremely elegant in its simplicity, it is not immediately obvious how such a system can be achieved. This explains why the work of Diffie and Hellman was such a big leap forward. Not only did they present the idea of public-key cryptography, they also provided a working instantiation. Although not as secure as initially believed, it is worthwhile noting that their original proposal still holds up today. However, other primitives have since gained in popularity. Besides the well-known RSA cryptosystem [RSA78] by Rivest, Shamir and Adleman that was found quickly after the introduction of public-key cryptography, the instantiations based on *elliptic curves* by Miller [Mil86] and Koblitz [Kob87] are increasingly being used. The latter systems are typically referred to as *elliptic-curve cryptography* (ECC) or (slightly more generally) as *curve-based cryptography*. Their main advantage compared to earlier proposals is that the keys remain small, even when adversaries are assumed to have relatively extensive computational power. In this thesis, we only concern ourselves with curve-based primitives and we refer to §I for a more technical introduction. We should emphasize that curve-based cryptography is ubiquitous. For example, it is present in the TLS [Res18] protocol that dictates the aforementioned secure internet communications (known mostly for the appearance of a green lock next to the URL in a web browser). Moreover, it is used for securing bio-

metric passports or identity cards [ICA15] (e. g. Dutch identity documents), popular messaging applications such as Whatsapp [Wha17] and Signal [Sig] and operating systems like Android [And] and iOS [App]. In short, there are too many examples to list; it is not unreasonable to expect that essentially every person reading this thesis uses curve-based cryptography on a daily basis.

We return to the question of context in modern-day cryptography. As a result of the protocol by Diffie and Hellman, we can assume adversaries to only access public data. The question now shifts to what "public" means. Certainly anything that is purposefully published by the communicating parties is public. However, an adversary could have access to the network over which this is communicated. In that case they can measure the time it takes for the messages to be delivered and, more interestingly, how long it takes a party to respond. This potentially leaks a little bit of information about the private computation. Such attacks are referred to as *timing* attacks and were first introduced by Paul Kocher [Koc96]. If an adversary has access to a device (e. g. a passport or identity card), not only can they measure the time the operations take, but also (say) measure the power consumption or electromagnetic emanation [KJJ99]. More generally, such unintended channels of potential leakage of private information are called *side channels*. The exploitation of side channels has turned out to be a very fruitful method of attack. Even stronger adversaries can, for example, try to inject faults in the (private) computation by optical means (i. e. shoot a laser) or by means of power spikes [BDL97]. These faults could cause an erroneous execution of the algorithm that leads to knowledge about the private key. An attacker with these capabilities is called *active*. As such, implementing curve-based protocols on devices with such strong (yet realistic!) adversaries is a non-trivial task. Regardless, the classical cryptographic schemes based on elliptic curves have proven to be resistant for decades (potentially with the use of appropriate additional countermeasures, see e. g. [Cor99]).

This leads us to the final and most recent adversarial model that has arisen due to the advances of quantum computation. Assuming that an adversary has access to a large enough quantum computer, the security of the public-key cryptosystems described in this section completely disappears due to an algorithm of Peter Shor [Sho97]. At this moment, significant resources are spent towards advancing the field of quantum computing. The largest quantum computer in existence is built by Google and consists of 72 (physical) quantum bits (qubits) [Kel18]. It is estimated that 2 330 (logical) qubits are required to run the algorithm of Shor to break moderate elliptic-curve cryptographic parameters [Roe+17, Table 2], but it is unclear when (and if) this number will be reached. Notice the discrepancy between physical and

logical qubits; it is expected that many physical qubits are necessary to build a single logical qubit.

Although this adversarial model is not yet realistic (as far as we know), the field of cryptography should be prepared when it becomes so. For that reason, the National Institute of Standards and Technology (NIST) has initiated standardization for public-key cryptographic schemes that resist quantum adversaries [Nat16]. It is not immediate that protocols based on (elliptic) curves can be adapted to be secure against quantum adversaries. Indeed, significantly different techniques are employed [Cou06; RS06; JDF11] and the resulting research direction is referred to as *isogeny*-based cryptography. In particular, it underlies SIKE [Jao+16], one of the 82 submitted proposals to the standardization effort of NIST and one of 26 proposals remaining in the (currently ongoing) second round. Again, the main advantage of the protocol based on elliptic curves is the small size of the (public) keys. However, the isogeny-based protocols incur a much more significant slowdown compared to other schemes.

In the first part of this thesis we work towards simply and securely implementing curve-based primitives against classical (i. e. non-quantum) adversaries. Although minimizing the time spent on computing keys and signatures is tempting to optimize for, it will typically come at a cost. For example, the fastest formulas for most (if not all) forms of elliptic curves have exceptional cases that may be exploited by an adversary. Moreover, one must be even more careful in the presence of adversaries that have access to side-channels. One may instead want to opt for formulas that are simpler and easier to implement, thereby more naturally excluding certain attack vectors. Due to the increasing number of online devices with low resources, another typical trade-off is between speed, code size and memory usage. We believe that the most important feature of an implementation is its security, which is strongly connected to the simplicity and size of the underlying code. In the second part we look towards isogeny-based protocols, conjectured to be secure against quantum adversaries (i. e. *post-quantum* secure). Although similar trade-offs can be made, these protocols are less standard and much more sensitive to change. As such, the focus of these chapters is aimed towards understanding the related theory, and thereby improving the efficiency and size of the primitives. In fact, in Chapter IX we propose a new post-quantum primitive. In short, this thesis is divided into three parts;

Part 1. The first part (Chapters I & II) gives an introduction to the theory that underlies the rest of the thesis. That is, we discuss the basics of elliptic and hyperelliptic curves and their usage in cryptography. There is no claim of novelty in this part, and its intention is simply to provide the necessary background.

Part 2. The second part (Chapters III–VI) considers contributions to classical cryptographic protocols. More precisely, protocols that are secure under the assumption that the adversary does not have access to a quantum computer. In essence, we improve the security and efficiency of protocols for key exchange and digital signatures based on the discrete logarithm problem in the (Kummer variety of the) Jacobian of curves of genus 1 or 2.

Part 3. The final part (Chapters VII–IX) makes contributions to protocols based on isogeny problems, which are thought to be secure against quantum adversaries. We improve the efficiency of the SIDH protocol and its key compression methods, and present the new primitive CSIDH.

Contributions

This work is organized as a sequence of (mostly published) papers, with minor modifications. These changes are made to align notation and lay-out of the different papers, and to combine overlapping material. This is done with the intent to improve readability and should not affect the content. As such, there is no reason to read the thesis sequentially (although the order is mostly chronological). In particular, references to tables, figures and equations do not include chapter numbers. In all cases this means that the reference points to within the chapter itself. For any exception the chapter number will be explicitly included. Many chapters also include software, which is all made available (most of which in the public domain) at

`https://joostrenes.nl`

unless mentioned otherwise. In the rest of this section we briefly summarize the contents of each chapter, and highlight our own contribution.

Complete Addition Formulas (Chapter III)

In the first chapter we consider the arithmetic of elliptic curves present in many standards. That is, curves in (short) Weierstrass form defined over a prime field $\mathbb{F}_p$ whose group of rational points over $\mathbb{F}_p$ has prime order. Such curves were known to have complete addition formulas (i. e. formulas that work on all pairs of points as input), but they incurred a tremendous slowdown compared to the incomplete formulas. In this chapter we significantly improve these formulas. Although a minor loss of efficiency remains (of about 30–40% in software depending on parameters),

the implementation naturally simplifies and should give users more confidence in their security. The chapter is mostly based on the paper

> Joost Renes, Craig Costello, and Lejla Batina. "Complete Addition Formulas for Prime Order Elliptic Curves". In: *Advances in Cryptology – EUROCRYPT 2016*. Ed. by Marc Fischlin and Jean-Sébastien Coron. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 403–428.

A convenient property is the parallelizability of the formulas, which is especially exploitable in hardware. We work out efficient algorithms for 1–6 cores and apply them in a hardware implementation in

> Pedro Maat C. Massolino, Joost Renes, and Lejla Batina. "Implementing Complete Formulas on Weierstrass Curves in Hardware". In: *Security, Privacy, and Applied Cryptography Engineering*. Ed. by Claude Carlet, M. Anwar Hasan, and Vishal Saraswat. Cham: Springer International Publishing, 2016, pp. 89–108.

This essentially comprises of §5 of Chapter III.

Contribution. I am the main author of the work in the first paper. I developed the optimized formulas and algorithms in §2, including their inclusion in OpenSSL in §4.3 and their Magma implementations. Moreover, I wrote and implemented (in Magma) the parallel versions of the second paper that appear in §5 and the appendix. I have also written the optimality analysis in §3.

Digital Signatures from Kummer Varieties (Chapter IV & V)

The next two chapters consider the practicality of signature schemes based on the efficient arithmetic on Kummer varieties. In Chapter IV we present the results of

> Joost Renes, Peter Schwabe, Benjamin Smith, and Lejla Batina. "μKummer: Efficient Hyperelliptic Signatures and Key Exchange on Microcontrollers". In: *Cryptographic Hardware and Embedded Systems – CHES 2016*. Ed. by Benedikt Gierlichs and Axel Y. Poschmann. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 301–320,

which provides implementations of the Diffie–Hellman key exchange and Schnorr signature scheme based on a Kummer surface of a genus-2 hyperelliptic curve. The software is written for the AVR ATmega and ARM Cortex M0 architectures, outperforming all other existing schemes and demonstrating its applicability for low-resource devices. In this work we follow the approach of Chung, Costello and Smith

[CCS17] of performing all scalar multiplications on the Kummer surface via projecting and recovering. However, this leaves some complex operations to be performed on the Jacobian, leading to increased code complexity and memory usage. An arguably more elegant approach slightly modifies the signature scheme itself to be naturally instantiated with a Kummer variety. This leads to the qDSA signature scheme, which is presented in

> Joost Renes and Benjamin Smith. "qDSA: Small and Secure Digital Signatures with Curve-Based Diffie–Hellman Key Pairs". In: *Advances in Cryptology – ASIACRYPT 2017*. Ed. by Tsuyoshi Takagi and Thomas Peyrin. Cham: Springer International Publishing, 2017, pp. 273–302.

On top of a theoretical security analysis of the scheme and highly efficient implementations on the same two architectures as before, we also show how to efficiently instantiate the necessary operations (i. e. signature verification and public-key compression) with genus-2 Kummer surfaces.

Contribution. I have developed the μKummer library on both the AVR ATmega and ARM Cortex M0 platforms, which is the main contribution of the first paper. I have also co-developed the qDSA signature scheme, and wrote its proof of security. Moreover, I created the qDSA library consisting of a C reference implementation and software for the AVR ATmega and ARM Cortex M0 platforms.

On Kummer Lines with Rational 2-torsion (Chapter VI)

The final chapter of Part 2 studies the relations of the various Kummer lines that have appeared in the literature. In the presence of full rational 2-torsion, we provide explicit maps between Montgomery curves, (twisted) Edwards models and (squared) Kummer lines. This significantly simplifies the treatment of Karati and Sarkar [KS17], who demonstrate the feasibility of the squared Kummer line on platforms with SIMD instructions. In

> Huseyin Hisil and Joost Renes. *On Kummer Lines With Full Rational 2-torsion and Their Usage in Cryptography*. Cryptology ePrint Archive, Report 2018/839. `https://eprint.iacr.org/2018/839`. 2018

we present an easy framework for moving between the different models with isomorphisms. This improves interoperability of the models and simplifies the task of an implementer. In particular, this allows for a straightforward generalization of the qDSA signature scheme to the squared Kummer line.

Contribution. I am a main contributor of the work that appears in this paper. In particular, I developed the library implementing qDSA based on the squared Kummer line on the ARM Cortex M0 platform.

Efficient Compression of SIDH Public Keys (Chapter VII)

The main advantage of SIDH compared to other post-quantum scheme is the relatively small size of its public keys. This chapter is based on

> Craig Costello, David Jao, Patrick Longa, Michael Naehrig, Joost Renes, and David Urbanik. "Efficient Compression of SIDH Public Keys". In: *Advances in Cryptology – EUROCRYPT 2017*. Ed. by Jean-Sébastien Coron and Jesper Buus Nielsen. Cham: Springer International Publishing, 2017, pp. 679–706,

which shows how to achieve even smaller public keys. It is based on the work by [Aza+16] but improves on it in multiple ways. We provide techniques for efficiently sampling torsion bases on a curve, significantly improve the pairing computations and obtain extremely efficient discrete logarithms in smooth cyclic groups. Moreover, we show how to compress the keys even further at essentially no cost.

Contribution. I am a main contributor to all content in this chapter. In particular, I created the optimized discrete logarithm algorithm of §4 and the (de)compression algorithms of §5. In addition, I have significantly contributed to the C and Magma libraries.

Computing Isogenies between Montgomery Curves (Chapter VIII)

The speed of supersingular-isogeny Diffie–Hellman is for a large part determined by the efficiency of the arithmetic of the elliptic-curve model and its isogeny formulas. A particularly popular form is the Montgomery form, which is used in the currently most optimal implementations of SIDH. This chapter is based on

> Joost Renes. "Computing Isogenies Between Montgomery Curves Using the Action of $(0,0)$". In: *Post-Quantum Cryptography*. Ed. by Tanja Lange and Rainer Steinwandt. Cham: Springer International Publishing, 2018, pp. 229–247,

and studies the isogeny formulas between elliptic curves in Montgomery form, expanding on and simplifying the work of [CH17]. That is, we show that the isogeny

formulas generalize to any group not containing the point $(0, 0)$ (and in particular 2-isogenies) and provide simplifications to the proofs. We also include potential new models that could lead to elegant isogeny formulas, though they do not lead to faster implementations of SIDH as of yet.

Contribution. I am the sole author of the work in this paper.

CSIDH (Chapter IX)

The last chapter proposes a new cryptographic primitive for key exchange, which is strongly related to the work of Couveignes [Cou06] and Rostovtsev and Stolbunov [RS06]. It is based on the paper

> Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. "CSIDH: An Efficient Post-Quantum Commutative Group Action". In: *Advances in Cryptology – ASIACRYPT 2018*. Ed. by Thomas Peyrin and Steven Galbraith. Cham: Springer International Publishing, 2018, pp. 395–427,

and replaces the class group action arising from the endomorphism ring of an ordinary elliptic curve by a class group action related to the $\mathbb{F}_p$-rational endomorphism ring of a supersingular elliptic curve. The main upside is the fact that the group of rational points over $\mathbb{F}_p$ has exactly size $p + 1$, allowing to easily select primes such that the curves have extremely smooth group orders. This leads to significantly faster evaluation of isogenies, while retaining the extremely small public keys. Moreover, we show how to efficiently validate public keys. As a result, the key exchange supports static public keys and is considered non-interactive.

Contribution. The main idea of this work is attributed to the first author. The paper is a collaborative effort, in which my contributions focus on the instantiation and its arithmetic, methods of public-key validation and the corresponding security analysis (§§4–8). The optimized C and assembly implementation is credited to the fourth author.

List of Symbols

ADD	Additive group operation.
$\mathbb{A}^n$	Affine space of dimension n.
$\bar{k}$	Algebraic closure of a field k.
$\mathcal{C}$	Algebraic curve.
$\mathcal{O}$	(1) Base point of an elliptic curve or (2) Endomorphism ring.
O	Big-O complexity.
char	Characteristic of a field.
cl	Class group.
$\|\|$	Concatenation.
$\mathbf{a}, \mathbf{s}, \mathbf{neg}$	Cost associated to a finite field addition (**a**), subtraction (**s**) and negation (**neg**).
$\mathbf{M}, \mathbf{S}, \mathbf{m_c}, \mathbf{E}, \mathbf{I}$	Cost associated to a finite field multiplication (**M**), squaring (**S**), multiplication by constant c ($\mathbf{m_c}$), exponentiation (**E**) and inversion (**I**).

DBL	Doubling operation in a group.
df	Differential of a rational function f.
ψ_n	n-division polynomial for some integer n.
Pic	Divisor class group.
$\mathrm{div}(f)$	Divisor of a non-zero rational function f.
Div	Abelian group of divisors.
Prin	Group of principal divisors.
$\widehat{x}$	Dual of an element x.
$\mathcal{E}\ell\ell_p$	Set of elliptic curves over a finite field $\mathbb{F}_p$ with a given endomorphism ring.
E	Elliptic curve.
k	Field.
$\mathbb{F}_q$	Finite field of order q.
Gal	Galois group.
$\mathfrak{a}, \mathfrak{b}, \mathfrak{c}, \mathfrak{l}$	Ideals in an order of a quadratic number field.
$\mathcal{J}$	Jacobian.
SK, PK, K	Private (SK), public (PK) and shared (K) keys.
$\mathcal{K}$	Kummer surface.
$\widetilde{\mathcal{K}}$	Kummer surface (general model).
log	Logarithm in base 2.
MADD	Mixed addition in group of points of an elliptic curve.
$\mathrm{N}(x)$	Norm of an element x.
∞	Point at infinity on an elliptic curve.
$\mathbf{x}, \mathbf{y}$	Projection maps to the x-coordinate or y-coordinate of an elliptic curve.

$\mathbb{P}^n$	Projective space of dimension n.
$\mathbb{Q}$	Rational numbers.
$\mathcal{L}$	Riemann-Roch space.
$\mathbb{Z}$	Ring of integers.
$\mathbb{Z}/N\mathbb{Z}$	Ring of integers modulo N.
$*$	Star operator denoting the action of a class group on the set of (isomorphism classes of) ordinary elliptic curves.
$\otimes$	Tensor product.

Part 1

Background

Chapter I

Elliptic and Hyperelliptic Curves

The main theme of this thesis is curve-based cryptography, so we begin by introducing the notions that are encountered throughout the different chapters. There are several great sources to find more extensive introductions to (hyper)elliptic curves and their cryptographic properties, for example the books by Silverman [Sil09] and Galbraith [Gal12]. The intent of this chapter is to summarize the relevant theory and fix notation, often referring back to established works for further details.

1 Algebraic Curves

Throughout this chapter (and indeed, the whole thesis) we follow [Sil09] and let k be a *perfect* field (i. e. every finite extension is separable). In a cryptographic context $k = \mathbb{F}_q$ is a finite field of q elements, in which this assumption holds true. The main consequence of interest is that the algebraic closure $\bar{k}$ is a *Galois* extension of k, in which case the *fixed field*

$$\bar{k}^{\mathrm{Gal}(\bar{k}/k)} = \left\{\lambda \in \bar{k} \mid \sigma(\lambda) = \lambda \text{ for all } \sigma \in \mathrm{Gal}(\bar{k}/k)\right\}$$

of $\bar{k}/k$ is the base field k. Moreover, for any finite extension K/k, the fixed field of the (necessarily Galois) extension $\bar{k}/K$ is K. This allows us to initially make many definitions and statements over $\bar{k}$, and only restrict to K whenever necessary (i. e. when talking about K-rationality). For example, we let $\mathbb{A}^n$ and $\mathbb{P}^n$ denote affine resp. projective n-space over $\bar{k}$, which have a canonical action of $\mathrm{Gal}(\bar{k}/K)$ (i. e. coordinate-wise). We write $\mathbb{A}^n(K)$ resp. $\mathbb{P}^n(K)$ for the points fixed under this action (i. e. their

K-rational points). This applies analogously for many concepts that we define, and we return to this in more detail at the end of the section.

Algebraic varieties. As is commonplace, we denote elements of $\mathbb{A}^n$ by comma-separated tuples $(x_1, \ldots, x_n)$, and elements of $\mathbb{P}^n$ by colon-separated capitalized tuples $(X_0 : \cdots : X_n)$. For any $i \in \{0, \ldots, n\}$ there exists an embedding

$$\begin{aligned} \chi_i : \mathbb{A}^n &\hookrightarrow \mathbb{P}^n \\ (x_1, \ldots, x_n) &\mapsto (x_1 : \cdots : x_{i-1} : 1 : x_{i+1} : \cdots x_n)\,, \end{aligned}$$

and since any element of $\mathbb{P}^n$ has a non-zero coordinate, projective space is covered by the union of the $\chi_i(\mathbb{A}^n)$. As usual, the affine and projective n-spaces carry the structure of a topological space via the Zariski topology [Har77, §I.1–I.2]. The closed subsets of affine (resp. projective) n-space are the sets of common zeroes of polynomials (resp. homogeneous polynomials) of (necessarily finitely generated by Hilbert's basis theorem [Hil90]) ideals I inside $\bar{k}[x_1, \ldots, x_n]$ (resp. $\bar{k}[X_0, \ldots, X_n]$), viewed as functions to $\bar{k}$. We define an affine (resp. projective) *algebraic variety* V to be a non-empty, closed and (topologically) irreducible subset of $\mathbb{A}^n$ (resp. $\mathbb{P}^n$).

Remark 1. Given a projective algebraic variety $V \subseteq \mathbb{P}^n$, we note that $V \cap \chi_i(\mathbb{A}^n)$ is an affine variety whose projective closure is V [Sil09, Proposition I.2.6]. We shall often identify the two without explicit mention. For example, although elliptic curves are often defined via their affine Weierstrass model in $\mathbb{A}^2$, we are always interested in their projective closure in $\mathbb{P}^2$ (which includes an extra point at infinity).

For a (projective algebraic) variety $V \subseteq \mathbb{P}^n$ defined by an ideal I, we define its *function field* $\bar{k}(V)$ as the set of fractions G/H of homogeneous polynomials (of equal degree) $G, H \in \bar{k}[X_0, \ldots, X_n]$ such that $H \notin I$ and under the equivalence relation defined by $G_0/H_0 = G_1/H_1$ if and only if $G_0H_1 - G_1H_0 \in I$ [Sil09, Remark II.2.9]. Elements of $\bar{k}(V)$ are referred to as *rational functions*. A rational function is said to be *regular* at a point $P \in V$ if there exists a representative in its equivalence class whose denominator does not vanish at P. A *rational map* ψ between projective varieties $V \subseteq \mathbb{P}^n$ and $W \subseteq \mathbb{P}^m$ is a map $\psi = (f_0 : \cdots : f_m) : V \to W$ where $f_i \in \bar{k}(V)$. Note that this map is only well-defined at a point $P \in V$ if there exists a rational function $g \in \bar{k}(V)$ such that all of the gf_i are regular at P. In that case, $\psi(P) = (gf_0(P) : \cdots : gf_n(P))$ lies in W, and we say that ψ is *regular at* P. The map ψ is a *morphism* whenever it is regular at all points in V. It is an *isomorphism* if it is a morphism and there exists a morphism $\chi : W \to V$ such that $\psi \circ \chi$ and $\chi \circ \psi$ are the identity map. Crucially, the choice of g is not necessarily the same for every point in

V. However, for any point $P \in V$ there exists an open neighborhood $U \subseteq V$ of P and functions $g_0, \ldots, g_m \in \bar{k}(V)$ regular on U such that $\psi(Q) = (g_0(Q) : \cdots : g_m(Q))$ for all $Q \in U$ [Har77, Lemma I.3.6]. In other words, any morphism can be locally described by well-defined fractions of homogeneous polynomials (of equal degree).

The *dimension* of an algebraic variety is its dimension as a topological space (and coincides with the *Krull dimension* of its coordinate ring). A *curve* $\mathcal{C}$ is a projective algebraic variety of dimension 1. It is called *hyperelliptic* when there exists a morphism $\mathcal{C} \to \mathbb{P}^1$ of degree 2. Every curve considered in this thesis is hyperelliptic (and, in fact, any curve of genus 1 or 2 is hyperelliptic). A curve $\mathcal{C} \subseteq \mathbb{P}^n$ is *smooth at* P whenever the rank of the Jacobian matrix (of a set of generators of the ideal defining $\mathcal{C}$) evaluated at P is $n - 1$ [Sil09, § I.1], and is called *smooth* (or *non-singular*) when it is smooth at all points. In this thesis we shall mostly work with smooth curves. A *surface* is a projective algebraic variety of dimension 2.

The Jacobian of a curve and its Kummer variety. A particularly useful property of curves is that there exists a well-defined notion of zeroes and poles of rational functions at smooth points, and every non-zero rational function only contains finitely many such points [Sil09, Proposition II.1.2]. Hence if $\mathcal{C}$ is a smooth curve, for each point $P \in \mathcal{C}$ we can define a valuation $v_P : \bar{k}(\mathcal{C})^* \to \mathbb{Z}$ [Gal12, Lemma 7.4.14] which assigns to each function its order of vanishing at P. In other words, to any function $f \in \bar{k}(\mathcal{C})^*$ we can assign a finite formal sum of points

$$\operatorname{div}(f) = \sum_{P \in \mathcal{C}} v_P(P)(P) .$$

More generally, any formal sum of points

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

where only finitely many n_P are non-zero is called a *divisor*. Its *degree* is $\deg(D) = \sum_{P \in \mathcal{C}} n_P$. The divisor D is called *effective* whenever n_P is non-negative for all $P \in \mathcal{C}$, and we define the so-called *Riemann–Roch space* $\mathcal{L}(D)$ as

$$\mathcal{L}(D) = \left\{ f \in \bar{k}(\mathcal{C})^* \mid \operatorname{div}(f) + D \text{ is effective} \right\} .$$

It is a finite-dimensional $\bar{k}$-vector space [Sil09, Proposition II.5.2]. The set of divisors is denoted $\operatorname{Div}(\mathcal{C})$ and forms an (abelian) group under addition with identity 0 (the divisor with $n_P = 0$ for all $P \in \mathcal{C}$), and the set $\operatorname{Div}^0(\mathcal{C})$ of divisors of degree

zero is a subgroup. The divisors of the form $\mathrm{div}(f)$ for $f \in \bar{k}(\mathcal{C})^*$ are called *principal divisors*, and the set of such divisors is denoted $\mathrm{Prin}(\mathcal{C})$. They are contained in $\mathrm{Div}^0(\mathcal{C})$ [Gal12, Theorem 8.3.14], and in fact form a subgroup [Gal12, Lemma 7.7.6]. The *divisor class group* (or *Picard group*) is the quotient group

$$\mathrm{Pic}^0(\mathcal{C}) = \mathrm{Div}^0(\mathcal{C}) / \mathrm{Prin}(\mathcal{C}) .$$

The set of points of $\mathrm{Pic}^0(\mathcal{C})$ is a projective algebraic variety,[1] which we refer to as the *Jacobian* $\mathcal{J}_\mathcal{C}$ of $\mathcal{C}$. Carrying over the group structure of $\mathrm{Pic}^0(\mathcal{C})$, one can show that $\mathcal{J}_\mathcal{C}$ is in fact an *abelian* variety. The set $\mathcal{K}_\mathcal{C} = \mathcal{J}_\mathcal{C}/\pm$ (i. e. the set of points where elements are identified with their inverses) is also an algebraic variety, which we refer to as the *Kummer variety* $\mathcal{K}_\mathcal{C}$ of $\mathcal{C}$. The image of a point $P \in \mathcal{J}_\mathcal{C}$ in $\mathcal{K}_\mathcal{C}$ is denoted by $\pm P$. The Kummer variety does not inherit the abelian group structure of $\mathcal{J}_\mathcal{C}$. However, the operation $\{\pm P, \pm Q\} \mapsto \{\pm(P+Q), \pm(P-Q)\}$ is well-defined via lifting to $\mathcal{J}_\mathcal{C}$. Moreover, since scalar multiplication on $\mathcal{J}_\mathcal{C}$ commutes with the negation map, the Kummer variety *does* inherit a well-defined pseudo-scalar multiplication $(m, \pm P) \mapsto \pm[m]P$. See §2 for more details.

Differentials and the genus. Let $\mathcal{C}$ be a smooth curve. We define the *space of differentials* $\Omega_\mathcal{C}$ on $\mathcal{C}$ to be the 1-dimensional $\bar{k}$-vector space [Sil09, Proposition II.4.2] generated by formal elements df for $f \in \bar{k}(\mathcal{C})$ with the relations [Sil09, §II.4]

$$\begin{aligned}
&(1) \quad d(f+g) = df + dg \quad \text{for all } f, g \in \bar{k}(\mathcal{C}) \ , \\
&(2) \quad d(fg) = gdf + fdg \quad \text{for all } f, g \in \bar{k}(\mathcal{C}) \, , \\
&(3) \quad dx = 0 \qquad\qquad\quad \text{for all } x \in \bar{k} \, .
\end{aligned}$$

The notions of poles and zeroes are also well-defined on non-zero differentials $\omega \in \Omega_\mathcal{C}$ [Sil09, Proposition II.4.3], giving rise to divisors $\mathrm{div}(\omega)$ in the same way as for principal divisors. The divisors of elements of $\Omega_\mathcal{C}$ all lie in the same divisor class. This is called the *canonical divisor class*, and any element of that class is called a *canonical divisor*. Although differentials are only explicitly used in Chapter VIII, the degree of canonical divisors has a direct connection to the genus of a curve and is therefore implicit throughout this whole thesis. The *genus* g of $\mathcal{C}$ is the (non-negative) integer $g = \deg(D) + 1 + \dim_{\bar{k}} \mathcal{L}(\mathrm{div}(\omega) - D) - \dim_{\bar{k}} \mathcal{L}(D)$, where $\omega \in \Omega_\mathcal{C}$ is an arbitrary non-zero differential.

[1] This is not immediately obvious in general. For elliptic curves the Picard group can always be embedded into $\mathbb{P}^2$ using the Weierstrass model [Sil09, Proposition III.3.4], while the Jacobian of genus-2 curves can be described as the locus of 72 quadratic equations inside $\mathbb{P}^{15}$ [CF96, §2]. These are the only two cases we care about in this thesis.

Rationality. Until now, we have worked merely over the algebraically closed field $\bar{k}$. However, for cryptographic purposes the finite fields $\mathbb{F}_q$ of interest are not algebraically closed. As promised, we now discuss rationality of the objects discussed so far over the perfect base field and its finite extensions.

We say that an algebraic variety $V \subseteq \mathbb{P}^n$ is *defined over* k when its generating ideal has a generating set contained in $k[X_0, \ldots, X_n]$. In that case, for any finite field extension K/k, there is an action of $\mathrm{Gal}(\bar{k}/K)$ on the points $P = (Z_0 : \cdots : Z_n)$ of V where $\sigma(P) = (\sigma(Z_0) : \cdots : \sigma(Z_n))$ for any $\sigma \in \mathrm{Gal}(\bar{k}/K)$. We define $V(K)$, called the set of *K-rational points* of V, to be the set of points invariant under the action of $\mathrm{Gal}(\bar{k}/K)$. Moreover, suppose that V is defined over k and let $W \subseteq \mathbb{P}^m$ be another variety defined over k. Recall that a morphism $\varphi : V \to W$ is locally given by rational functions with coefficients in $\bar{k}$. Therefore, any $\sigma \in \mathrm{Gal}(\bar{k}/K)$ acts on φ by acting on its coefficients. We say that φ is *defined over* K when $\varphi(P) = \sigma(\varphi)(P)$ for all $P \in V$ and all $\sigma \in \mathrm{Gal}(\bar{k}/K)$. Finally, there is an action of $\mathrm{Gal}(\bar{k}/K)$ on any divisor $D \in \mathrm{Div}(\mathcal{C})$ by acting on the points in its support [Sil09, §II.3] and we say that D is *defined over* K if $\sigma(D) = D$ for all $\sigma \in \mathrm{Gal}(\bar{k}/K)$. This action is well-defined on $\mathrm{Pic}^0(\mathcal{C})$. We define $\mathrm{Pic}^0_K(\mathcal{C})$ to be the $\mathrm{Gal}(\bar{k}/K)$-stable subgroup of $\mathrm{Pic}^0(\mathcal{C})$, and note that for all intents and purposes we shall always work with $\mathrm{Pic}^0_K(\mathcal{C})$. It is in bijection with $\mathcal{J}_{\mathcal{C}}(K)$ for any finite extension K/k.

2 Curves of Genus 1 and 2

We now turn our attention to the cryptographically most relevant curves. Note that we could immediately restrict everything to the case where k is a finite field (i. e. a field of prime order or its quadratic extension), but the general treatment is usually not much different, while some results in this thesis hold in the general case too. We elaborate on special properties of elliptic curves over finite fields, and in particular their endomorphism rings, at the end of §2.1.

Classically, elliptic-curve cryptography is typically[2] based on the hardness of the discrete logarithm problem in $\mathrm{Pic}^0_{\mathbb{F}_p}(\mathcal{C})$ of a curve $\mathcal{C}$ defined over the prime field $\mathbb{F}_p$. This can be solved via Pollard's rho algorithm [Pol75] of time complexity $O(\sqrt{N})$, where $N = \#\mathrm{Pic}^0_{\mathbb{F}_p}(\mathcal{C})$. Alternatively, the discrete logarithm problem in the Picard group of a curve of genus g over a field $\mathbb{F}_p$ can be solved with time complexity $\tilde{O}(p^{2-2/g})$ [Gau+07, Theorem 1], improving on the time complexity of the rho attack for $g \geq 3$. As such, all curves of genus $g \geq 3$ require relatively large parameters

[2] We do not consider binary fields at all in this thesis.

to be securely implemented and are much less efficient than their genus 1 and 2 counterparts for which the rho method is the best known attack. Therefore we do not consider the case $g \geq 3$ in this thesis.

For protocols based on the hardness of isogeny problems, it is not entirely obvious that curves of higher genus do not hold value. However, although they are an interesting direction of research, we only consider isogeny graphs of elliptic curves in this thesis (which have already received cryptographic interest, see for example the ongoing standardization process [Nat16] by NIST).

2.1 Elliptic Curves

There exist various more and less general definitions of elliptic curves in the literature, whose usefulness depends on the context they are being used in. In this thesis we do not a priori restrict the chosen embedding into projective space (although we always assume it exists), while we possibly want to consider points that are not defined over the field of definition of the curve. Therefore, we follow [Sil09] and simply define an elliptic curve E (over $\bar{k}$) to be a smooth projective curve of genus 1 with a specified base point $\mathcal{O} \in E$. We say that E is defined over k whenever E is defined over k as a curve and $\mathcal{O} \in E(k)$.

The abelian group of points. Perhaps the most well-known property of elliptic curves is the natural bijection of the set of points on E with $\mathrm{Pic}^0(E)$, giving a simple description of $\mathrm{Pic}^0(E)$ as an abelian variety (i. e. E is its own Jacobian). More explicitly, the map κ that sends P to the class of the divisor $(P) - (\mathcal{O})$ is a bijection of sets [Sil09, Proposition III.3.4], and the obvious abelian group structure of $\mathrm{Pic}^0(E)$ transfers to E. The identity element of the group of points on E is the specified base point $\mathcal{O}$. If E is defined over k, then κ induces a bijection between $E(K)$ and $\mathrm{Pic}^0_K(E)$ for any finite extension K/k.

For any (non-zero) integer $m \in \mathbb{Z}$ there is an m-torsion subgroup $E[m]$, which is the set of points in E that are mapped to $\mathcal{O}$ under the multiplication-by-m map $[m] : E \to E$. If m is non-zero in k, then $E[m] \cong \mathbb{Z}/m\mathbb{Z} \times \mathbb{Z}/m\mathbb{Z}$. Otherwise, if $\mathrm{char}(k) = p$ for a prime p and r a positive integer, then $E[p^r]$ is isomorphic to either $\{\mathcal{O}\}$ or $\mathbb{Z}/p^r\mathbb{Z}$ [Sil09, Corollary III.6.4].

Weierstrass form. Let E be defined over k. Given functions $x, y \in k(E)$ such that

$$\mathcal{L}(2[\mathcal{O}]) = \langle 1, x \rangle \,,\ \mathcal{L}(3[\mathcal{O}]) = \langle 1, x, y \rangle$$

as $\bar{k}$-vector spaces [Sil09, Proposition II.5.8], we obtain the classical embedding $P \mapsto (x(P) : y(P) : 1)$ onto the (projective closure of the) locus defined by the *Weierstrass* equation

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6\,, \quad a_1, \ldots, a_6 \in k \tag{1}$$

inside the projective plane $\mathbb{P}^2$ [Sil09, Proposition III.3.1(a)] such that $\mathcal{O} \mapsto (0 : 1 : 0)$. If $\mathrm{char}(k) \neq 2, 3$ then one can apply an additional transformation to ensure that $a_1 = a_2 = a_3 = 0$, and we refer to it as the *short Weierstrass form*.

The group law inherited from $\mathrm{Pic}^0(E)$ now has a simple geometric description in $\mathbb{P}^2$ [Sil09, Proposition 3.4(e)]. Three points $P, Q, R \in E$ satisfy $P + Q + R = \mathcal{O}$ if and only if there exists a line intersecting E at all three points. In particular, choosing $R = \mathcal{O}$ implies that for any affine point $P = (s, t) \in E$ we have $-P = (s, -t)$, and we note that these are the only two affine points whose x-coordinate is s. That is, the degree-2 morphism $E \to \mathbb{P}^1$ mapping $(X : Y : Z) \mapsto (X : Z)$ factors through $\mathcal{K}_E$ and induces an isomorphism $\mathcal{K}_E \cong \mathbb{P}^1$ (of algebraic varieties).[3] For that reason, we refer to $\mathcal{K}_E$ as the Kummer *line* of E. Moreover, it immediately implies that E is a hyperelliptic curve.

The group operation $E \times E \to E$ that maps $(P, Q) \mapsto P + Q$ can be described by rational functions and, in fact, is a morphism of algebraic varieties [Sil09, Theorem III.3.6]. The same is true for the negation map $P \mapsto -P$. However, we immediately note that morphisms only have local descriptions as (tuples of) rational functions and are not necessarily well-defined on all of $E \times E$. Instead, they are only well-defined on an open subset of $E \times E$. Rational maps that compute the group law on an open subset of $E \times E$ are called *addition formulas* [BL95]. Indeed, Bosma and Lenstra [BL95, Theorem 1] prove that the full group law can not be described by a single addition formula. For example, the following addition formulas [Sil09, §III.2] that add two points $P = (s, t)$ and $Q = (u, v)$ as $R = (w, z)$ defined by

$$\begin{aligned} w &= \lambda^2 + a_1\lambda - a_2 - s - u \\ z &= -(\lambda + a_1)w - \nu - a_3 \end{aligned}\,, \quad \text{where} \quad \begin{aligned} \lambda &= (v - t)/(u - s) \\ \nu &= (tu - sv)/(u - s) \end{aligned}$$

are only defined on the open subset where P and Q are both affine and $s \neq u$. As shown by Lange and Ruppert [LR85], the space of addition formulas forms a 3-dimensional $\bar{k}$-vector space, and a basis for this space has been given by Bosma and Lenstra [BL95, §5]. Although a single addition formula does not suffice for $E \times E$, often one is only interested in K-rational points of E for some finite extension K/k. Interestingly, if there exist addition formulas defined on an open subset U of $E \times E$

[3] Since $x = X/Z$ has a pole of order 2 at $\mathcal{O} = (0 : 1 : 0)$, we have $\mathcal{O} \mapsto (1 : 0)$.

such that $(E \times E)(K) \subseteq U$, then this single addition formula will suffice for computing the group law on $E(K)$. In that case we call the addition formulas K-complete. Such examples have been given, see e. g. [BL95, §5] or [AKR12, Remark 4.4]. This is the main topic of discussion in Chapter III, in the cryptographic setting where k is a finite field $\mathbb{F}_q$ and $E(\mathbb{F}_q)$ does not contain any points of even order.

Montgomery form. Suppose that $\mathrm{char}(k) \neq 2, 3$ and that E is defined over k. Then one can embed E into $\mathbb{P}^2$ as the (projective closure of the) curve

$$by^2 = x^3 + ax^2 + x\,,$$

which is known as a *Montgomery curve* [Mon87], with unique point at infinity $\mathcal{O} = (0 : 1 : 0)$. It follows by smoothness of E that $b(a^2 - 4) \neq 0$. It is not immediate that one can guarantee that $a, b \in k$ and, indeed, this is not true in general. For example, if $k = \mathbb{F}_p$ then one can find $a, b \in k$ if and only if E or its (quadratic) twist contains an $\mathbb{F}_p$-rational point of order 4 [Ber+08, Theorem 3.3]. Every Montgomery curve has a k-rational point $Q = (0, 0)$ of order 2 whose action by translation on affine points acts like inversion on the x-coordinate. Moreover, for any $Q_4 \in E(\bar{k})$ such that $[2]Q_4 = Q$ we have $Q_4 \in \left\{(1, \pm\sqrt{(a+2)/b}), (-1, \pm\sqrt{(a-2)/b})\right\}$.

The geometric description of the group law with identity $\mathcal{O}$ as described for the Weierstrass model in $\mathbb{P}^2$ carries over to Montgomery form. In particular, the inverse of any affine point $S = (s, t)$ is $-S = (s, -t)$ and the map $E \to \mathbb{P}^1$ by sending $(X : Y : Z) \mapsto (X : Z)$ is again a morphism of degree 2. Although the formulas for computing the group law on a curve in Montgomery form are typically lengthy, the arithmetic significantly simplifies when we move to the Kummer line $\mathcal{K}_E \cong \mathbb{P}^1$. That is [Mon87, §10], given the abscissas x_P resp. x_Q of two affine points P resp. Q such that $P \neq \pm Q$ and $P, Q \notin E[2]$ and given the abscissa x_{P-Q} of their difference, we find

$$\begin{aligned} x_{P+Q} &= (x_P x_Q - 1)^2 / \left[x_{P-Q}(x_P - x_Q)^2\right], \\ x_{[2]P} &= (x_P^2 - 1)^2 / \left[4x_P(x_P^2 + ax_P + 1)\right]. \end{aligned}$$

Montgomery curves also have very efficiently computable isogenies on the x-line. This is the main topic of Chapter VIII.

(Twisted) Edwards form. Let $c \in \bar{k}$ such that $c^5 \neq c$ and $E : x^2 + y^2 = c^2(1 + x^2y^2)$ is a smooth curve of genus 1. This is technically only a subset of the set of curves of the form $x^2 + y^2 = c^2(1 + dx^2y^2)$ originally defined as *Edwards curves* by Bernstein

and Lange [BL07]. However, in this thesis we are only concerned with the case $d = 1$, which corresponds to the form introduced by Edwards [Edw07], who was the first to observe that its arithmetic with respect to the base point $\mathcal{O} = (0, c)$ is extremely symmetric.

Embedding the curve into $\mathbb{P}^2$ via $(x, y) \mapsto (x : y : 1)$ gives two singularities at $(1 : 0 : 0)$ and $(0 : 1 : 0)$. We can resolve these by blowing up (see e. g. [His10, §2.3.4] or [Gal12, Lemma 9.12.18]) to obtain the curve

$$E/k = V(X^2 + Y^2 - c^2(Z^2 + T^2), XY - TZ) \subseteq \mathbb{P}^3$$

and embedding $(x, y) \mapsto (xy : x : y : 1)$. When referring to Edwards curves, we will mean their embedding into $\mathbb{P}^3$. For affine points we will sometimes use the affine notation and expect that this should not cause confusion. Note that this is a purely theoretical tool, since once all is said and done, the cryptographically relevant arithmetic is performed in a prime order subgroup in which all points are affine. At infinity, an Edwards curve contains the elements

$$\Theta_1 = (1 : c : 0 : 0)\,, \qquad \theta_1 = (1 : 0 : c : 0)\,,$$
$$\Theta_2 = (1 : -c : 0 : 0)\,, \qquad \theta_2 = (1 : 0 : -c : 0)\,,$$

where Θ_1, Θ_2 resp. θ_1, θ_2 have orders 2 resp. 4. Observe that if E is defined over k, then $\mathcal{O} \in E(k)$ implies that $c \in k$. Moreover, if $i \in \bar{k}$ is an element such that $i^2 = -1$, then $E[4] \cong \mathbb{Z}/4\mathbb{Z} \times \mathbb{Z}/4\mathbb{Z}$ has generating set $\langle \theta_1, (i, 1) \rangle$. That is, the 4-torsion is rational over k or over a quadratic extension of k.

The geometric description of the group law differs from the Weierstrass model [Arè+11, §4]. For example, given any point $P = (P_0 : P_1 : P_2 : P_3) \in E$ such that $P \neq \Theta_1, \Theta_2$ there exists a hyperplane $H : P_3 Y - P_2 Z = 0 \subseteq \mathbb{P}^3$ that intersects E in P, Θ_1, Θ_2 and a unique fourth point $Q = (-P_0 : -P_1 : P_2 : P_3)$. It follows that $\operatorname{div}(H/(Y - cZ)) = (P) + (Q) - 2(\mathcal{O})$, and hence $Q = -P$. In particular, the inverse of an affine point (s, t) is given by $(-s, t)$. Consequently, the projection to the Kummer line is now obtained by projection to the *y-coordinate*

$$E \to \mathbb{P}^1 : (T : X : Y : Z) \mapsto (Y : Z)\,.$$

Notably, $(\Theta_1, \Theta_2) \mapsto ((1 : c), (1 : -c))$. In other words, the projection to the Kummer line corresponds to *projecting away* from $\{\Theta_1, \Theta_2\}$ onto $\mathbb{P}^1$.

In general, one can not expect for $E[4]$ to be rational over k or a quadratic extension. As such, not every curve admits an Edwards model over k. As a partial

resolution, instead we can take $\alpha, \delta \in k$ such that $\alpha\delta(\alpha - \delta) \neq 0$ and let

$$\alpha x^2 + y^2 = 1 + \delta x^2 y^2$$

be (the affine part of) a smooth projective curve of genus one. This is commonly referred to as the *twisted Edwards* model [Ber+08], where the base point is chosen as $\mathcal{O} = (0, 1)$. It is a more general model than the Edwards model and is closely related to a Montgomery curve [Ber+08, Theorem 3.2(i)]. As above, we use the smooth model inside $\mathbb{P}^3$ containing the elements

$$\begin{aligned} \Omega_1 &= (1 : \sqrt{\delta/\alpha} : 0 : 0), & \omega_1 &= (1 : 0 : \sqrt{\delta} : 0), \\ \Omega_2 &= (1 : -\sqrt{\delta/\alpha} : 0 : 0), & \omega_2 &= (1 : 0 : -\sqrt{\delta} : 0), \end{aligned}$$

where Ω_1, Ω_2 have order 2 and ω_1, ω_2 have order 4. Again, we have a projection to $\mathbb{P}^1$ by projecting to the y-coordinate, which in particular maps

$$(\Omega_1, \Omega_2) \mapsto \left((1 : \sqrt{\delta/\alpha}), (1 : -\sqrt{\delta/\alpha})\right).$$

As opposed to Montgomery curves, the arithmetic on the curve itself is very symmetric. That is, given two affine points $P = (s, t)$ and $Q = (u, v)$ on a twisted Edwards curve, their sum $R = P + Q$ is given by

$$R = \left(\frac{sv + tu}{1 + \delta stuv}, \frac{tv - \alpha su}{1 - \delta stuv}\right).$$

Similar to the case of Montgomery curves, the arithmetic on the Kummer line is also very elegant. In Chapter VI we provide very simple isomorphisms between the Kummer lines of Montgomery curves, (twisted) Edwards curves and Kummer lines arising from the theory of theta functions [GL09, §6].

Isogenies. An *isogeny* between elliptic curves $(E_0, \mathcal{O}_0)$ and $(E_1, \mathcal{O}_1)$ defined over k is a non-constant morphism $\varphi : E_0 \to E_1$ such that $\varphi(\mathcal{O}_0) = \mathcal{O}_1$. If such an isogeny exists, we say that E_0 and E_1 are *isogenous*. For any isogeny $\varphi : E_0 \to E_1$ there exists a unique isogeny $\widehat{\varphi} : E_1 \to E_0$ such that $\varphi \circ \widehat{\varphi} = \widehat{\varphi} \circ \varphi = [\deg \varphi]$ and we call $\widehat{\varphi}$ the *dual isogeny* of φ [Sil09, Theorem III.6.1]. The set of elliptic curves isogenous to a given curve E_0 is an equivalence class which we call the *isogeny class* of E_0. We say that φ is defined over k if it is defined over k as a morphism of algebraic curves. It is an *isomorphism of elliptic curves* if there exists an isogeny ψ from $(E_1, \mathcal{O}_1)$ to $(E_0, \mathcal{O}_0)$ such that $\varphi \circ \psi$ and $\psi \circ \varphi$ are the identity maps.

Elliptic curves are, up to isomorphism over $\bar{k}$, classified by their *j-invariant* [Sil09, Proposition III.1.4(b)]. A *twist* of an elliptic curve E_0/k is an elliptic curve over k which is not isomorphic to E_0 over k, but shares the same j-invariant. A morphism $\varphi : E_0 \to E_1$ induces a field embedding $\varphi^* : k(E_1) \to k(E_0)$ by pulling back rational functions, which gives a finite extension $k(E_0)/\varphi^* k(E_1)$ [Sil09, Theorem II.2.4(a)]. We say that φ is *separable* whenever the corresponding field extension is, and define the degree of φ as $\deg(\varphi) = [k(E_0) : \varphi^* k(E_1)]$ [Gal12, Definition 8.1.6]. Any separable isogeny over k has a finite kernel $G \subseteq E_0$, which is a $\mathrm{Gal}(\bar{k}/k)$-stable group [Gal12, Exercise 9.6.5] of size $\deg(\varphi)$ [Sil09, Theorem III.4.10]. In fact, any finite $\mathrm{Gal}(\bar{k}/k)$-stable group $H \subseteq E_0$ gives rise to a separable isogeny $\psi : E_0 \to \widetilde{E}$ defined over k such that $\ker \psi = H$ and $\widetilde{E}$ is defined over k, which is unique up to post-composition with an isomorphism [Gal12, Theorem 9.6.19]. In that case we write $\widetilde{E} = E/H$. One can obtain an explicit description of ψ and E/H from H through Vélu's formulas [Vél71]; in Chapter VIII we discuss alternatives when the action of ψ on one affine point is known.

An *endomorphism* of an elliptic curve E/k is an isogeny $\psi : E \to E$. The set of endomorphisms, together with the zero map, forms a ring $\mathrm{End}(E)$ (under pointwise addition and composition of isogenies) called the *endomorphism ring*. It contains the ring of endomorphisms defined over k, denoted $\mathrm{End}_k(E)$, as a subring. Both these rings clearly contain $\mathbb{Z}$ since they contain the maps $[m] : E \to E$ [Sil09, Corollary III.5.4]. The algebra $\mathrm{End}(E) \otimes_{\mathbb{Z}} \mathbb{Q}$ is called the *endomorphism algebra*. If $\mathrm{End}(E) \neq \mathbb{Z}$ we say that E has *complex multiplication*. In fact, there are only two possible structures for $\mathrm{End}(E)$ if E has complex multiplication; it is an order in $\mathrm{End}(E) \otimes \mathbb{Q}$, which is either a quadratic imaginary number field or a quaternion algebra [Sil09, Theorem III.9.3]. In the first case we call E *ordinary*, in the latter case we say that E is *supersingular*. Given an isogeny $\varphi : E_0 \to E_1$, the endomorphism rings of E_0 and E_1 are not necessarily isomorphic, but the induced map

$$\begin{aligned} \mathrm{End}(E_0) \otimes \mathbb{Q} &\to \mathrm{End}(E_1) \otimes \mathbb{Q} \\ \psi &\mapsto (\varphi \circ \psi \circ \widehat{\varphi}) \,/\, [\deg \varphi] \end{aligned}$$

does give rise to an isomorphism of endomorphism *algebras* [Koh96, pp. 7]. Consequently, the property of being ordinary or supersingular is well-defined on isogeny classes.

Elliptic curves over finite fields. Finally we restrict to the cryptographically most interesting case when $k = \mathbb{F}_q$ is a finite field of size $q = p^n$ and E an elliptic curve

defined over $\mathbb{F}_q$. The Frobenius map on $\overline{\mathbb{F}}_q$ induces a (purely inseparable) map $\pi : E \to E^{(q)}$ by applying the Frobenius map to the coordinates of points on E, where $E^{(q)}$ is the elliptic curve defined by the equation of E after applying the Frobenius map to its coefficients. If E is defined over $\mathbb{F}_q$, then $E^{(q)} = E$ and we refer to π as the *Frobenius endomorphism*. For any non-negative integer m, the endomorphism $\pi^m - 1$ is separable with kernel $E(\mathbb{F}_{q^m})$ [Sil09, Corollary III.5.5], i. e. the $\mathbb{F}_{q^m}$-rational points of E are simply those fixed by π^m. There exists a $t \in \mathbb{Z}$ such that $\pi^2 - t\pi + q = 0$ in $\mathrm{End}(E)$ and we refer to t as the *trace of Frobenius*. By considering the action π on the Tate module [Sil09, §III.7] for any prime $\ell \neq p$, one shows that $t = q + 1 - \#E(\mathbb{F}_q)$ [Sil09, Theorem V.2.3.1]. The number of possible traces is relatively small; Hasse's theorem [Sil09, Theorem V.1.1] tells us that $|t| \leq \sqrt{2q}$. The endomorphism algebra $\mathrm{End}(E) \otimes \mathbb{Q}$ contains the imaginary quadratic number field $\mathbb{Q}(\pi)$ of discriminant $D = t^2 - 4q$, so any elliptic curve over $\mathbb{F}_q$ has complex multiplication. If E is ordinary, then $\mathrm{End}(E) \otimes \mathbb{Q} \cong \mathbb{Q}(\pi)$. Otherwise, $\mathrm{End}(E) \otimes \mathbb{Q}$ is strictly bigger and E is supersingular. This happens if and only if $t \equiv 0 \pmod{p}$ [Sil09, Exercise V.5.10].

Isogeny graphs over finite fields. Let $E/\mathbb{F}_q$ be an elliptic curve over a field of characteristic p and let $\ell \neq p$ be a prime. Given a finite extension $K/\mathbb{F}_q$, we denote by $G_{K,\ell}$ the ℓ-isogeny graph over K, i. e. the graph whose nodes are K-isomorphism classes of elliptic curves and whose edges are (separable) isogenies of degree ℓ over K (up to post-composition with K-isomorphisms). When $K = \overline{\mathbb{F}}_q$, the isomorphism classes can be represented by their j-invariants. Moreover, since $E(\overline{\mathbb{F}}_q)[\ell] \cong \mathbb{Z}/\ell\mathbb{Z} \times \mathbb{Z}/\ell\mathbb{Z}$, the curve E contains exactly $\ell + 1$ subgroups of order ℓ. These correspond to $\ell + 1$ outgoing isogenies from $j(E)$. Note that typically (i. e. when $j(E) \neq \{0, 1728\}$, see [Gal12, Remark 25.3.2]) every outgoing edge has a corresponding incoming edge, since every isogeny induces a dual isogeny. Therefore, although isogenies technically have a direction, the isogeny graph can (almost) be thought of as an undirected graph. The ℓ-isogeny graph consists of ordinary and supersingular components.

Suppose that $E/\mathbb{F}_q$ is an ordinary curve with Frobenius endomorphism π of trace t. By Tate's theorem [Tat66, §3], the set of such curves forms an isogeny class. The ring of $\mathbb{F}_q$-rational endomorphisms $\mathrm{End}_{\mathbb{F}_q}(E)$ is an order $\mathcal{O} \subset \mathbb{Q}(\sqrt{t^2 - 4q})$ containing $\mathbb{Z}[\pi]$ [Sil09, Theorem III.9.3]. Given an ideal $\mathfrak{a} \subset \mathcal{O}$ we define $\varphi_\mathfrak{a} : E \to E/\mathfrak{a}$ to be an isogeny with $\ker \varphi_\mathfrak{a} = \bigcap_{\alpha \in \mathfrak{a}} \ker \alpha$. It is immediate that $\ker \varphi_\mathfrak{a}$ is $\mathrm{Gal}(\overline{\mathbb{F}}_q/\mathbb{F}_q)$-stable, and thus $\varphi_\mathfrak{a}$ is defined over $\mathbb{F}_q$ and $E/\mathfrak{a}$ is well-defined up to $\mathbb{F}_q$-isomorphism. As principal ideals in $\mathcal{O}$ correspond to endomorphisms, we obtain a well-defined action $*$ of the ideal class group $\mathrm{cl}(\mathcal{O})$ on the set X of $\mathbb{F}_q$-isomorphism classes of curves

of trace t with endomorphism ring $\mathcal{O}$ by defining $[\mathfrak{a}] * E = E/\mathfrak{a}$, identifying the curves with their $\mathbb{F}_q$-isomorphism classes. This action is free and transitive [Sch87; Wat69, Theorem 4.5], and we say that X is a *principal homogeneous space* for $\mathrm{cl}(\mathcal{O})$. Note that typically one does not require the isogeny class to have fixed trace t, in which case there are two orbits of the action of $\mathrm{cl}(\mathcal{O})$ on the set of $\mathbb{F}_q$-isomorphism classes with endomorphism ring $\mathcal{O}$ (the second orbit are the classes of curves with trace of Frobenius $-t$). However, these are simply two isomorphic copies of the same graph (see [DFKS18, §2.2] for more details) and so we may identify them.

Now let $E/\overline{\mathbb{F}}_q$ be supersingular. There exist only finitely many isomorphism classes of supersingular elliptic curves, and their j-invariants are contained in $\mathbb{F}_{p^2}$ [Sil09, Theorem V.3.1(a)]. As a result, every isomorphism class can be represented by a supersingular elliptic curve defined over $\mathbb{F}_{p^2}$. More precisely, the number of isomorphism classes is exactly $\lfloor p/12 \rfloor + \varepsilon_p$ [Gal12, Theorem 9.11.11], where

$$\varepsilon_p = \begin{cases} 0 & \text{if } p \equiv 1 \pmod{12}, \\ 1 & \text{if } p \equiv 5, 7 \pmod{12}, \\ 2 & \text{if } p \equiv 11 \pmod{12}. \end{cases}$$

The set of supersingular isomorphism classes forms a connected component in $G_{\bar{k},\ell}$ [Koh96, Corollary 78], necessarily $(\ell+1)$-regular almost everywhere (i. e. away from $j = 0, 1728$) and satisfies the *Ramanujan* property [Piz90]. Note that although setting $k = \mathbb{F}_{p^2}$ would suffice to have $G_{k,\ell}$ contain all supersingular isomorphism classes as nodes, this does not necessarily imply that all ℓ-isogenies are defined over k. We expand on this in a cryptographic context in §II.2.

Finally, suppose that E is supersingular and defined over $\mathbb{F}_p$ for $p \geq 5$. Its $\mathbb{F}_p$-rational endomorphism ring $\mathrm{End}_{\mathbb{F}_p}(E)$ is an order $\mathcal{O} \subset \mathbb{Q}(\sqrt{-p})$ containing $\mathbb{Z}[\pi]$. The graph of such curves is contained in the supersingular isogeny graph over $\mathbb{F}_{p^2}$, but much more closely resembles the (volcano) structure of the ordinary case [DG16]. In Chapter IX we elaborate on this case, and show how to obtain an efficient non-interactive key exchange protocol.

Pairings. Finally, we briefly discuss bilinear pairings on elliptic curves. Although the literature is vast, including all sorts of applications in cryptography, we only consider the Weil and (reduced) Tate-Lichtenbaum pairing. Only the latter is used in this thesis (see Chapter VII), to efficiently solve discrete logarithm problems on supersingular elliptic curves of smooth order.

Let $E/\mathbb{F}_q$ be an elliptic curve and m a positive integer not divisible by $p =$

$\mathrm{char}(\mathbb{F}_q)$. Given any divisor $D = \sum_{P \in E} n_P(P)$ and rational function $f \in \overline{\mathbb{F}}_q(E)$ such that $\mathrm{div}(f)$ has support disjoint from D, we define $f(D) = \prod_{P \in E} f(P)^{n_P}$. We denote by μ_m the group of m-th roots of unity of $\overline{\mathbb{F}}_q$, and define the *Weil pairing* $w_m : E[m] \times E[m] \to \mu_m$ as $w_m(P, Q) = f_P(D_Q)/f_Q(D_P)$, where $D_P, D_Q \in \mathrm{Div}^0(E)$ and $f_P, f_Q \in \overline{\mathbb{F}}_q(E)$ are such that

$$D_P \sim (P) - (\mathcal{O}), \quad \mathrm{div}(f_P) = mD_P,$$
$$D_Q \sim (Q) - (\mathcal{O}), \quad \mathrm{div}(f_Q) = mD_Q,$$

and D_P and D_Q have disjoint support. Note that there exist different definitions of the Weil pairing [Sil09, §III.8], which can be shown to be equivalent [Sil09, Exercise 3.16]. It is a bilinear, alternating, non-degenerate and Galois invariant map [Sil09, Proposition III.8.1]. The bilinearity and alternating properties lead to a strong relation to (two-dimensional) discrete logarithm problems. That is, if $E[m] = \langle P, Q \rangle$ and $R = [a]P + [b]Q$ for some $a, b \in \mathbb{Z}/m\mathbb{Z}$, then it follows that

$$w_m(P, R) = w_m(P, Q)^b, \quad w_m(Q, R) = w_m(Q, P)^a.$$

In other words, the Weil pairing allows to reduce discrete logarithms in E to discrete logarithms in μ_m. It is not immediately obvious that this leads to a computational advantage. First, if m is exponentially large, then so are the degrees of f_P and f_Q. However, w_m can be computed with complexity logarithmic in m due to an algorithm of Miller [Mil04]. More problematic is the fact that μ_m is typically not contained in $\mathbb{F}_q$, but only in an extension $\mathbb{F}_{q^k}$. The smallest positive k is called the *embedding degree* of m in $\mathbb{F}_q$. Although k is generally large, one can show that $k \leq 6$ whenever E is supersingular (see e. g. [MOV91, Table 1]). In this thesis we only consider the case where E is indeed supersingular (with $k = 2$), and hence do not need to worry about the embedding degree.

A computationally more appealing pairing is the *Tate-Lichtenbaum pairing* [Lic69]. Let K be a finite extension of $\mathbb{F}_q$ such that $E[m] \subseteq E(K)$. Then we define the pairing as the function

$$t_m : E[m] \times E(K)/mE(K) \to K^*/(K^*)^m, \quad (P, Q) \mapsto f_P(D_Q).$$

That is, the computation can again be performed via Miller's algorithm, but requires fewer Miller functions. Note that since K is a finite extension of $\mathbb{F}_q$ (say of degree k), its multiplicative group K^* is cyclic of order $q^k - 1$. Moreover, since $E[m] \subseteq E(K)$ it follows that $m \mid q^k - 1$, and in particular $K^*/(K^*)^m \cong \mu_m$ through the (group)

isomorphism $\tau : x \mapsto x^{(q^k-1)/m}$. Although this causes a computational overhead, the isomorphism is typically used to avoid working with cosets. We refer to $\widetilde{e}_m : E[m] \times E(K)/mE(K) \to \mu_m$ defined by $\widetilde{e}_m = \tau \circ t_m$ as the *reduced* Tate-Lichtenbaum pairing. The (reduced) pairing is bilinear and non-degenerate [Eng14, Theorem 9]. Note, however, that the relation with the discrete logarithms above relied on the pairing being alternating. Indeed, the (reduced) Tate-Lichtenbaum pairing does not have this property in general (in fact, the definition of being alternating does not even make sense since $E[m]$ and $E(K)/mE(K)$ are distinct objects). However, in the cases of our interest we shall have $E(K) \cong (\mathbb{Z}/m\mathbb{Z})^2 \times (\mathbb{Z}/\ell\mathbb{Z})^2$ for some prime $\ell \neq m$ (see §2.1). In that case it is immediate that

$$mE(K) \cong (\mathbb{Z}/\ell\mathbb{Z})^2\,, \quad E(K)/mE(K) \cong E[m]\,.$$

We write $e_m : E[m] \times E[m] \to \mu_m$ for the resulting pairing (which we also refer to as the reduced Tate-Lichtenbaum pairing). Moreover, we shall always assume e_m to be alternating and therefore to be interchangeable with the Weil pairing for all our purposes (this can easily be checked in specific cases).

2.2 Hyperelliptic Curves of Genus 2

Finally, we consider the case where $\mathcal{C}$ is a smooth curve of genus 2 over a field k such that $\mathrm{char}(k) \neq 2$. In this thesis, we only use genus-2 curves in protocols based on the hardness of the discrete logarithm problem. In fact, only a single genus-2 curve is used. Although everything we describe works for other genus-2 hyperelliptic curves (say given in Rosenhain form), it does not lead to cryptographically secure protocols unless $\#\,\mathrm{Pic}^0_{\mathbb{F}_p}(\mathcal{C})$ is divisible by a large prime. Such curves are not easy to find and, in fact, the only one (satisfying also certain other cryptographic properties) that has been found to date is the Gaudry–Schost curve. Therefore we do not lose much generality by restricting our attention to this curve. As such, we fix $p = 2^{127} - 1$ and let $k = \mathbb{F}_p$. Given the constants

$$\begin{aligned}\lambda &= \texttt{0x155555555555555555555555555555552}\,,\\ \mu &= \texttt{0x73E334FBB315130E05A505C31919A746}\,,\\ \nu &= \texttt{0x552AB1B63BF799716B5806482D2D21F3}\,,\end{aligned}$$

we define $\mathcal{C}/\mathbb{F}_p$ to be the genus-2 curve defined by its *Rosenhain form* $y^2 = x(x-1)(x-\lambda)(x-\mu)(x-\nu)$. It is (the quadratic twist of) the twist-secure curve found by Gaudry and Schost [GS12] for which $\#\mathcal{J}_{\mathcal{C}}(\mathbb{F}_p)$ equals $16 \cdot N$ for a 250-bit prime N.

The Picard group. Contrary to the case of elliptic curves, there exists no bijection between elements of $\mathrm{Pic}^0_{\mathbb{F}_p}(\mathcal{C})$ and $\mathcal{C}(\mathbb{F}_p)$. However, every divisor class of $\mathrm{Pic}^0_{\mathbb{F}_p}(\mathcal{C})$ can be uniquely represented by a divisor that is one of

$$\begin{aligned} &(1) \quad 0\ , \\ &(2) \quad [R] - [\mathcal{O}]\ , \\ &(3) \quad [P] + [Q] - 2[\mathcal{O}]\,, \end{aligned}$$

where P, Q and R are affine points such that $x_P \neq x_Q$ and $\mathcal{O}$ is the (unique) point at infinity [Gal12, Theorem 10.4.1]. This is called a *reduced* divisor [Can87, §2]. A reduced divisor is commonly described by its *Mumford representation* [Mum93], which simply keeps track of the affine points that identify it. That is, it is a pair of polynomials $\langle u(x), v(x)\rangle$ where

$$\begin{aligned} &(1) \quad \langle u(x), v(x)\rangle = \langle 1, 0\rangle\ , \\ &(2) \quad \langle u(x), v(x)\rangle = \langle x - x_R, y_R\rangle\ , \\ &(3) \quad \langle u(x), v(x)\rangle = \langle (x - x_P)(x - x_Q), \tfrac{y_Q - y_P}{x_Q - x_P} x + \tfrac{x_Q y_P - x_P y_Q}{x_Q - x_P}\rangle\ . \end{aligned}$$

An element of $\mathcal{J}_\mathcal{C}(\mathbb{F}_p)$ can therefore be represented by the coefficients of $u(x)$ and $v(x)$, requiring (at most) 4 elements of $\mathbb{F}_p$ [Gal12, Lemma 10.3.10].

An algorithm by Cantor performs the group operation in the divisor class group on reduced divisors in Mumford representation [Can87] and this theoretically suffices to construct the most interesting cryptographic protocols. However, there are some downsides to this algorithm. Not only is it rather slow, it is also hard to make constant-time (i. e. with running time independent of its inputs). Alternatively, the Picard group can be embedded into $\mathbb{P}^{15}$ [CF96, §2.1–2.2], giving an algebraic description of the Jacobian $\mathcal{J}_\mathcal{C}$ and its group law in terms of the curve coeffients [CF96, Eq. 3.9.5]. Unfortunately the involved rational functions very quickly become unwieldy (i. e. much more computationally heavy than its genus-1 counterpart, see for example the formulas provided in the appendix of [CF96]). As such, arithmetic on the Jacobian is best avoided as much as possible (see e. g. Chapter IV and Chapter V).

Kummer surfaces. The situation significantly improves by projecting to the Kummer variety, i. e. by projecting onto the first 4 coordinates of the Jacobian in $\mathbb{P}^{15}$, and by making some assumptions on the curve model [Duq04, §4]. The locus of the Kummer variety in $\mathbb{P}^3$ is described by a quartic

$$\widetilde{\mathcal{K}}_\mathcal{C} : K_2 T^2 + K_1 T + K_0 = 0\,,$$

where $K_i(X, Y, Z)$ for $i \in \{0, 1, 2\}$ are homogeneous polynomials of degree $4 - i$ with coefficients in $k[\lambda, \mu, \nu]$ [CF96, Eq. 3.1.9]. It is a 2-dimensional algebraic variety, so we refer to $\widetilde{\mathcal{K}}_{\mathcal{C}}$ as the Kummer *surface*. More precisely, we call this model the *general model* of the Kummer surface of $\mathcal{J}_{\mathcal{C}}$ (denoted by the tilde over $\mathcal{K}_{\mathcal{C}}$). Although the performance improves over working directly on $\mathcal{J}_{\mathcal{C}}$, it is still not more efficient than the analogous (and much simpler) elliptic-curve operations.

Finally, for use in cryptographic applications Gaudry [Gau07] proposed an alternative embedding into $\mathbb{P}^3$ as a quartic surface $\mathcal{K}_{\mathcal{C}}$ with locus described by

$$X^4 + Y^4 + Z^4 + T^4 + 2EXYZT = \begin{array}{c} F(X^2T^2 + Y^2Z^2) + G(X^2Z^2 + Y^2T^2) \\ + H(X^2Y^2 + Z^2T^2) \end{array},$$

where the constants $E, F, G, H \in \bar{k}$ are determined by the curve $\mathcal{C}$. The surface is related to the theory of theta functions [CC86] and leads to much faster arithmetic. We refer to $\mathcal{K}_{\mathcal{C}}$ as the *fast* Kummer surface,[4] though different variants exist (see e. g. Chapter V). It is not necessarily true that E, F, G and H will lie in k. Indeed, not every Kummer surface of a genus-2 hyperelliptic curve defined over k admits a fast model over k. For instance, all 16 of the 2-torsion points of $\mathcal{J}_{\mathcal{C}}$ must be defined over k.

[4] Notice a slight abuse of notation, since we use $\mathcal{K}_{\mathcal{C}}$ for both the Kummer variety as well as the fast model of a Kummer surface. This should not cause confusion.

Chapter II

Curve-based Cryptographic Protocols

Although the field of cryptography is broad, in this thesis we focus on two important public-key cryptographic primitives: two-party key exchange and digital signatures. Classically, this is achieved via the Diffie–Hellman key exchange and the Schnorr signature scheme based on the hardness of the *discrete logarithm problem*. We introduce these in §1. However, the introduction of large-scale quantum computers would break these schemes through Shor's (polynomial-time) algorithm [Sho97]. As a result, alternative protocols based on the hardness of the *isogeny problem* were introduced. We elaborate on them in §2.

1 Classical Cryptography

In this section protocols always take place in cyclic prime-order groups, so we fix notation first. Let $\mathcal{J}$ be an (additive) abelian group of prime order N with identity element $\mathcal{O}$, and let P be a non-zero element of $\mathcal{J}$ (i. e. such that $\mathcal{J} = \langle P \rangle$). For any integer $m \in \mathbb{Z}$, scalar multiplication is denoted by a map $[m] : \mathcal{J} \to \mathcal{J}$ such that $[m] : (m, P) \mapsto [m]P$. Let $\mathcal{K} = \mathcal{J}/\pm$ be the set of its elements under the equivalence relation where Q and R are in the same class whenever $Q = \pm R$. As usual, the image of an element $Q \in \mathcal{J}$ in $\mathcal{K}$ is denoted by $\pm Q$. The notation suggests that $\mathcal{J}$ can be instantiated as the (prime order subgroup of the rational points of a) Jacobian of a hyperelliptic curve over $\mathbb{F}_q$, while $\mathcal{K}$ is its Kummer variety. Indeed, in this thesis this is always the case.

1.1 Diffie–Hellman

In their seminal work Diffie and Hellman [DH76] first proposed the idea of public key cryptosystems, and provided an instantiation based on the discrete logarithm problem in the multiplicative subgroup of finite fields. A decade later, Diffie–Hellman groups were constructed as the group of rational points of an elliptic curve (or the Jacobian of a hyperelliptic curve [Kob88]) over a finite field [Mil86; Kob87].

Diffie–Hellman in $\mathcal{J}$. The protocol works in an arbitrary cyclic group $\mathcal{J} = \langle P \rangle$, which we assume to be of prime order N. Two parties Alice and Bob both choose their respective *private keys* SK_A and SK_B as an element of $(\mathbb{Z}/N\mathbb{Z})^*$, and publish their *public keys* $\mathrm{PK}_A = [\mathrm{SK}_A]P$ and $\mathrm{PK}_B = [\mathrm{SK}_B]P$. Both parties can now derive the shared secret $\mathrm{K}_{AB} = [\mathrm{SK}_A]\mathrm{PK}_B = [\mathrm{SK}_B]\mathrm{PK}_A$.

Diffie–Hellman in $\mathcal{K}$. As remarked by Miller [Mil86], a completely analogous construction works in the Kummer variety $\mathcal{K}$. Given the image $\pm P$ of a generator of $\mathcal{J}$, Alice (resp. Bob) again chooses their private key SK_A (resp. SK_B) as an element of $(\mathbb{Z}/N\mathbb{Z})^*$. Their public key is now $\mathrm{PK}_A = \pm[\mathrm{SK}_A]P$ (resp. $\mathrm{PK}_B = \pm[\mathrm{SK}_B]P$), while the shared secret can be derived by both parties as $\mathrm{K}_{AB} = \pm[\mathrm{SK}_A \cdot \mathrm{SK}_B]P$.

Security. We only discuss the security of the Diffie–Hellman protocol in the group $\mathcal{J}$. The Diffie–Hellman problems for the protocol in $\mathcal{K}$ all reduce to the analogous Diffie–Hellman problems in $\mathcal{J}$, with a minor security loss in the reduction. The security of the Diffie–Hellman protocol relates to several problems:

Discrete Logarithm Problem. The discrete logarithm problem supposes to be given non-zero $P, Q \in \mathcal{J}$ and asks to find an $m \in (\mathbb{Z}/N\mathbb{Z})^*$ such that $Q = [m]P$. In terms of Diffie–Hellman, it is the problem of retrieving a secret SK from a public key PK. The best-known (generic) classical algorithm for solving the discrete logarithm problem is Pollard's rho algorithm [Pol75] using $O(\sqrt{m})$ group operations with negligible memory requirement. Note that we assume that index calculus attacks [COS86] do not apply to the groups under consideration. The problem is solved in polynomial time by Shor's algorithm [Sho97] under the assumption of having a large enough quantum computer.

Computational Diffie–Hellman. The *computational* Diffie–Hellman problem (abbreviated CDH) requires to compute K_{AB}, given all public information of the protocol. This problem is no harder than the discrete logarithm problem, but is not known to be equivalent. However, it is typically conjectured to be (see

e. g. [Boe90; MW99]) and the currently best-known attacks on curve-based Diffie–Hellman are attacks on the discrete logarithm problem (i. e. Pollard rho).

Decisional Diffie–Hellman. The *decisional* Diffie–Hellman problem (abbreviated as DDH) requires to distinguish K_{AB} from a random element, given all public information of the protocol. It is no harder than DDH, and indeed there exist groups where CDH is hard yet DDH is easy. A large prime order group of rational points of a supersingular elliptic curve would fall in this class through the use of pairings (see §2.1).

Computational aspects. The group operation of $\mathcal{J}$ is written as a function $\text{ADD} : \mathcal{J} \times \mathcal{J} \to \mathcal{J}$ that maps $(P, Q) \mapsto P + Q$. For the special case where $P = Q$ we define the function $\text{DBL} : \mathcal{J} \to \mathcal{J}$ as $\text{DBL}(P) = \text{ADD}(P, P)$. Although naïvely $[m]P = P + P + \cdots + P$, this would have $O(m)$ computational complexity (in the number of group operations). It can more conveniently be computed via the *double-and-add* algorithm of complexity $O(\log_2 m)$. There exist many variants of this algorithm, most notably ones whose sequence of operations is independent of m under the assumption that we know an upper bound on m (such algorithms lend themselves to easy *constant-time* implementations).

The set $\mathcal{K}$ does not (generally) inherit an abelian group structure from $\mathcal{J}$. Note that although $\mathcal{K}$ has a well-defined operation $\text{XDBL} : \pm P \mapsto \pm[2]P$ coming from DBL on $\mathcal{J}$, the ADD operation becomes slightly more complicated. That is, we have a function

$$\text{XADD} : \{\pm P, \pm Q, \pm(P - Q)\} \mapsto \pm(P + Q)$$

referred to as *differential addition* (i. e. we can only compute the sum of two points if we are given the difference). Since scalar multiplication on $\mathcal{J}$ commutes with the negation map, the Kummer variety has a well-defined pseudo-scalar multiplication $\text{LADDER} : (m, \pm P) \mapsto \pm[m]P$. This can be computed with $O(\log_2 m)$ calls to XADD and XDBL through the *Montgomery ladder* [Mon85]. Typically one defines a function XDBLADD that simultaneously computes XDBL and XADD at lower cost.

1.2 Schnorr Signatures

There exist many different (variants of) signature schemes based on the discrete logarithm problem. Here we describe the *Schnorr signature scheme* [Sch90], which is (arguably) the most natural.

Alice		**Bob**
$\mathrm{SK}_A \leftarrow_R (\mathbb{Z}/N\mathbb{Z})^*$		$\mathrm{SK}_B \leftarrow_R (\mathbb{Z}/N\mathbb{Z})^*$
$\mathrm{PK}_A \leftarrow [\mathrm{SK}_A]P$		$\mathrm{PK}_B \leftarrow [\mathrm{SK}_B]P$
	$\xleftarrow{\mathrm{PK}_A, \mathrm{PK}_B}$	
$\mathrm{K}_{AB} \leftarrow [\mathrm{SK}_A]\mathrm{PK}_B$		$\mathrm{K}_{AB} \leftarrow [\mathrm{SK}_B]\mathrm{PK}_A$

Figure 1. Diffie–Hellman key exchange in a cyclic group $\mathcal{J} = \langle P \rangle$ of (large) prime order N.

Schnorr identification and signatures. We first define the *Schnorr identification protocol* (see Figure 2) in $\mathcal{J}$. Suppose that Alice has a secret key $\mathrm{SK}_A \in (\mathbb{Z}/N\mathbb{Z})^*$ and an accompanying public key $\mathrm{PK}_A = [\mathrm{SK}_A]P$. She wants to prove knowledge of her secret key SK_A to Bob, without revealing any information about it (i. e. this is a *zero-knowledge* protocol). By tying her identity to her public key PK_A (e. g. via a certificate authority), this provides a form of authentication from Alice to Bob. Alice starts by selecting an ephemeral secret $r \in (\mathbb{Z}/N\mathbb{Z})^*$ and computing $R = [r]P$, and she sends her commitment R to Bob. Upon receiving R, Bob selects a random challenge $c \in \mathbb{Z}/N\mathbb{Z}$ and returns it to Alice. In turn, Alice computes

$$s = (r - c \cdot \mathrm{SK}_A) \bmod N$$

and sends s to Bob. Bob accepts if and only if R equals $[s]P + [c]\mathrm{PK}_A$.

Notice that this is an *interactive* authentication protocol. It can be made *non-interactive* via the Fiat–Shamir heuristic [FS87]. That is, Alice can generate the challenge herself based on the commitment R by setting $c = H(R)$, where H is a *random oracle* [BR93]. By instead selecting $c = H(R \mid\mid M)$ for some message M, Alice can compute a *Schnorr signature* (R, s) on M that can be validated by any party that has access to her public key.

Security. A signature scheme in this thesis is considered to be secure if any party that does not know SK_A is unable to forge signatures on any message not signed before. That is, a signature scheme is secure when it is *existentially unforgeable under adaptive chosen message attacks*. Any signature scheme constructed by applying the Fiat–Shamir heuristic to a *sigma protocol* has this property [PS96]. A sigma protocol is a three-round protocol that satisfies the properties of *completeness*, *special soundness* and *honest-verifier zero-knowledge* [HL10, §6.2]. The Schnorr identification scheme is such a sigma protocol, which is typically proved via the *Forking Lemma* [PS96, Lemma 2] in the random oracle model. In Chapter V we construct a sigma proto-

col similar to the Schnorr identification scheme that works in $\mathcal{K}$, and we define a signature scheme in $\mathcal{K}$ via the Fiat–Shamir heuristic.

Alice		**Bob**
$r \leftarrow_R (\mathbb{Z}/N\mathbb{Z})^*$		
$R \leftarrow [r]P$	$\xrightarrow{R}$	
	$\xleftarrow{c}$	$c \in \mathbb{Z}/N\mathbb{Z}$
$s \leftarrow (r - c \cdot \mathrm{SK}_A) \bmod N$	$\xrightarrow{s}$	
		$R \stackrel{?}{=} [s]P + [c]\mathrm{PK}_A$

Figure 2. Schnorr identification scheme in a cyclic group $\mathcal{J} = \langle P \rangle$ of (large) prime order N.

2 Post-Quantum Cryptography

The protocols in this section are based on (variants of) the *isogeny problem*, i. e. the hardness of finding an isogeny between two curves in the same isogeny class. However, the structure of the isogeny graph is highly dependent on whether we are in an ordinary or supersingular component, which results in significantly different protocols (and attacks). We discuss them separately.

2.1 Supersingular Isogeny Diffie–Hellman

The key exchange based on (a variant of) the supersingular isogeny problem closely resembles the Diffie–Hellman key exchange, and is therefore referred to as *Supersingular Isogeny Diffie–Hellman* (SIDH). It was introduced by Jao and De Feo [JDF11] in 2011 and has since received a lot of attention, resulting in the *SIKE* submission [Jao+16] to the post-quantum standardization effort by NIST [Nat16]. The description of the SIDH protocol here includes some of the choices made in the SIKE submission, simplifying the treatment.

As before, we assume to have two parties Alice and Bob that wish to exchange a secret. An important remark to make is that SIDH is not completely symmetric. That is, it begins by having Alice and Bob choose between two distinct (small) primes ℓ and m. In what follows we assume Alice to have chosen the prime ℓ and Bob the prime m. In practice this requires communication between Alice and Bob (as opposed to regular Diffie–Hellman). For that reason, we denote the secret–public key pair of Alice (resp. Bob) by $(\mathrm{SK}_\ell, \mathrm{PK}_\ell)$ (resp. $(\mathrm{SK}_m, \mathrm{PK}_m)$).

SIDH. Let ℓ and m be two small primes, and e_ℓ and e_m be two positive integers such that $p = \ell^{e_\ell} \cdot m^{e_m} - 1$ is prime. Let $E_0/\mathbb{F}_{p^2}$ be a supersingular elliptic curve with the Frobenius map $\pi : (x, y) \mapsto (x^{p^2}, y^{p^2})$ having trace $t = -2p$. Then $\#E_0(\mathbb{F}_{p^2}) = (p+1)^2$ and the eigenvalues of the action of π on $E_0[\ell^{e_\ell}]$ and $E_0[m^{e_m}]$ are all 1. Thus, $E_0(\mathbb{F}_{p^2})[\ell^{e_\ell}] \cong (\mathbb{Z}/\ell^{e_\ell}\mathbb{Z})^2$ and $E_0(\mathbb{F}_{p^2})[m^{e_m}] \cong (\mathbb{Z}/m^{e_m}\mathbb{Z})^2$. As such, we can fix (public) bases $E_0[\ell^{e_\ell}] = \langle P_\ell, Q_\ell \rangle$ and $E_0[m^{e_m}] = \langle P_m, Q_m \rangle$ inside $E_0(\mathbb{F}_{p^2})$. The secret key SK_ℓ (resp. SK_m) is a random element of $\mathbb{Z}/\ell^{e_\ell}\mathbb{Z}$ (resp. $\mathbb{Z}/m^{e_m}\mathbb{Z}$). It determines a cyclic subgroup $\langle P_\ell + [\mathrm{SK}_\ell]Q_\ell \rangle$ (resp. $\langle P_m + [\mathrm{SK}_m]Q_m \rangle$) of E_0 of order ℓ^{e_ℓ} (resp. m^{e_m}). In turn, these determine (separable) isogenies (up to post-composition with an isomorphism) $\varphi_\ell : E_0 \to E_\ell$ and $\varphi_m : E_0 \to E_m$ of degree ℓ^{e_ℓ} resp. m^{e_m}, where $E_\ell = E_0/\langle P_\ell + [\mathrm{SK}_\ell]Q_\ell \rangle$ and $E_m = E_0/\langle P_m + [\mathrm{SK}_m]Q_m \rangle$. The public keys are $\mathrm{PK}_\ell = (E_\ell, \varphi_\ell(P_m), \varphi_\ell(Q_m))$ and $\mathrm{PK}_m = (E_m, \varphi_m(P_\ell), \varphi_m(Q_\ell))$, respectively, while the shared secret $\mathrm{K}_{\ell m}$ is $j(E_0/\langle P_\ell + [\mathrm{SK}_\ell]Q_\ell, P_m + [\mathrm{SK}_m]Q_m \rangle)$. In other words, the shared secret is the j-invariant of the curve $E_m/\langle \varphi_m(P_\ell + [\mathrm{SK}_\ell]Q_\ell) \rangle$ resp. $E_\ell/\langle \varphi_\ell(P_m + [\mathrm{SK}_m]Q_m) \rangle$ that can be computed by Alice resp. Bob.

Security. The security of SIDH (against passive attacks) relates to several problems that are analogous to classical Diffie–Hellman problems; the *Computational Supersingular Isogeny* (CSSI) problem asks to compute SK_ℓ from PK_ℓ [DFJP14, Problem 5.2], the *Supersingular Computational Diffie–Hellman* (SSCDH) problem asks to compute $\mathrm{K}_{\ell m}$ [DFJP14, Problem 5.3], while solving the *Supersingular Decisional Diffie–Hellman* (SSDDH) requires to distinguish $\mathrm{K}_{\ell m}$ from random [DFJP14, Problem 5.4]. Although these problems are not known to be equivalent, they are assumed to be. The best-known attacks against SIDH are indeed attacks against the CSSI problem. Following the security definitions of NIST (i. e. λ-bit security means breaking the problem is at least as hard as recovering a λ-bit AES key), the best-known attacks on SIDH are classical and are trivial meet-in-the-middle attacks of query and memory complexity $O(\sqrt[4]{p})$. As a result, for a λ-bit security level one chooses e_ℓ, e_m such that ℓ^{e_ℓ} and m^{e_m} are greater than $2^{2\lambda}$. We emphasize that on the one hand the latter choice is conservative; it is very hard to obtain efficient access to memory of size $\sqrt[4]{p}$. Algorithms that overcome this problem (i. e. Van Oorschot–Wiener [OW99]) lead to higher run-times [Adj+19]. On the other hand, the described problems that underly the security of SIDH are quite different from generic isogeny problems. For instance, the number of possible public keys equals the size of $\mathbb{Z}/\ell^{e_\ell}\mathbb{Z}$ (resp. $\mathbb{Z}/m^{e_m}\mathbb{Z}$), which is approximately $\sqrt{p}$. This is much smaller than the $\lfloor p/12 \rfloor + \varepsilon_p$ (see §2.1) supersingular isomorphism classes, making the isogeny problem easier. Note that this could be solved easily, but would naïvely require to move to (possibly large) extension

fields [Pet17, §2]. Finally, the inclusion of torsion points to the public key has led to serious active attacks [Gal+16].

Computational aspects. Typical choices for parameters are $\ell = 2$ and $m = 3$, in which case E_0 can be chosen as the Montgomery curve $y^2 = x^3 + x$ [CLN16a]. Such curves contain the 2-torsion point $(0, 0)$ whose action by translation is very simple. We show how one can use this action to efficiently compute isogenies on Montgomery curves in Chapter VIII. Moreover, although the bases for $E_0[2^{e_2}]$ and $E_0[3^{e_3}]$ can be chosen arbitrarily, we show that certain choices allow to completely avoid exceptional cases in the arithmetic.

The sizes of the public keys are naïvely about $8 \log_2 p$ bits, since the curve (in short Weierstrass form) can be represented by two elements of $\mathbb{F}_{p^2}$, while each point is represented by its x-coordinate in $\mathbb{F}_{p^2}$ (plus a sign bit). This can be improved to $6 \log_2 p$ bits by projecting the curve to the Kummer line [CLN16a]. Alternatively, one can observe that for the above parameters the curve can always by put in Montgomery form $y^2 = x^3 + Ax^2 + x$ (i. e. with $B = 1$) and can therefore simply be represented by the coefficient $A \in \mathbb{F}_{p^2}$ (see Remark VIII.8). It can be further compressed to $4 \log_2 p$ using point compression techniques [Aza+16]. The main idea is to transmit basis points P, Q such that $\langle P, Q \rangle = E(\mathbb{F}_{p^2})[n]$, where $n \in \{\ell^{e_\ell}, m^{e_m}\}$, as their two-dimensional scalar decomposition with respect to a fixed public basis $\langle R_1, R_2 \rangle = E(\mathbb{F}_{p^2})[n]$. Of course, the curve in each public key is different and thus there is no public basis that can be fixed once-and-for-all. The idea therefore relies on Alice and Bob being able to, on input of a given curve E, arrive at the same basis $\{R_1, R_2\}$ for $E(\mathbb{F}_{p^2})[n]$. Given such a basis, we can write $P = [\alpha_P]R_1 + [\beta_P]R_2$ and $Q = [\alpha_Q]R_1 + [\beta_Q]R_2$, and can solve for $(\alpha_P, \beta_P, \alpha_Q, \beta_Q) \in (\mathbb{Z}/n\mathbb{Z})^4$. This is feasible via the Pohlig-Hellman algorithm [PH78] since $\#E(\mathbb{F}_{p^2})[n] = n^2$ is extremely smooth. By noting that $w_n(R_1, P) = w_n(R_1, R_2)^{\beta_P}$ and $w_n(R_2, P) = w_n(R_1, R_2)^{-\alpha_P}$ (and similarly for α_Q, β_Q) they reduce the discrete logarithm computation to μ_n in $\mathbb{F}_{p^2}^*$, increasing efficiency. As $\log_2 n \approx \frac{1}{2} \log_2 p$, the size of $(\alpha_P, \beta_P, \alpha_Q, \beta_Q)$ is about $2 \log_2 p$. In Chapter VII we show how to decrease the public keys to $\frac{7}{2} \log_2 p$ bits while simultaneously significantly increasing the efficiency of the above procedures.

2.2 Ordinary Isogeny Diffie–Hellman

The key exchange based on the isogeny problem in ordinary isogeny graphs was originally introduced by Couveignes (see the abstract of [Cou06] for details) and was rediscovered by Rovstovstev–Stolbunov [RS06] a decade later. It is much easier to

describe (on a high level) than SIDH, and its security reduces to a much more natural problem. Moreover, the scheme is *non-interactive*. This makes it interesting as a direct replacement for classical Diffie–Hellman. Its major drawback is its inefficiency.

OIDH. Let $\mathbb{F}_q$ be a finite field and $E_0/\mathbb{F}_q$ an ordinary elliptic curve with endomorphism ring $\mathcal{O}$ and trace of Frobenius t. Let X be the set of $\mathbb{F}_q$-isomorphism classes of elliptic curves with trace of Frobenius t whose endomorphism ring is isomorphic to $\mathcal{O}$. Then the action $* : \mathrm{cl}(\mathcal{O}) \times X \to X$ such that $[\mathfrak{a}] * E = E/\mathfrak{a}$ is simply transitive, where $\mathfrak{a}$ is an arbitrary representative of its class and we identify curves with their $\mathbb{F}_q$-isomorphism class. As such, the secret keys SK_A and SK_B are chosen to be random elements of $\mathrm{cl}(\mathcal{O})$, and their corresponding public keys are $\mathrm{PK}_A = \mathrm{SK}_A * E_0$ and $\mathrm{PK}_B = \mathrm{SK}_B * E_0$. The shared secret K_{AB} is now simply $(\mathrm{SK}_A \cdot \mathrm{SK}_B) * E_0$.

Security. As usual, there is a separation between the key recovery problem and the (computational and decisional) Diffie–Hellman problems that are not known to be equivalent. However, we observe that the key recovery problem is simply the ordinary isogeny problem; given two isogenous ordinary curves E_0 and E_1 over a finite field $\mathbb{F}_q$ with endomorphism ring $\mathcal{O}$, find an isogeny between them. The isogeny problem can be phrased as a *hidden shift problem* [CJS14], which can be solved in subexponential time on a quantum computer [Kup05; Reg04]. Though these attacks do of course not lead to a complete break, they are what motivated the development of SIDH (see [JDF11, §1]).

Computational aspects. Although the protocol is much easier to describe from a high level, quite the opposite is true for the computation of the group action. In fact, the evaluation of the group action has sub-exponential complexity [CJS14]. Therefore, instead of randomly sampling secret keys, we construct them as classes of products of ideals with small norm. That is, we fix a set of (distinct small) *Elkies primes* $\ell_0, \ldots, \ell_s$ and ideals $\mathfrak{l}_0, \ldots, \mathfrak{l}_s$ such that $\mathfrak{l}_i\bar{\mathfrak{l}}_i = (\ell_i)$ and such that $\mathrm{cl}(\mathcal{O})$ is (expected to be) generated by the $[\mathfrak{l}_i]$. A secret key is now simply a tuple $\mathrm{SK} = (e_0, \ldots, e_s)$ of small elements of $\mathbb{Z}$, while the public key is $\mathrm{PK} = [\mathfrak{l}_0]^{e_0} \cdots [\mathfrak{l}_s]^{e_s} * E_0$. Now it remains to compute the action of ideals of norm ℓ_i, essentially reducing to computing separable isogenies of degree ℓ_i. Ideally, one finds a rational point and applies Vélu's formulas [Vél71]. However, such points are typically only defined over (large) extension fields, forcing one to resort to other methods. As a result, the protocol is extremely slow [DFKS18]. In Chapter IX we show how to overcome many of these issues by instantiating the protocol with supersingular elliptic curves over a prime field $\mathbb{F}_p$.

Part 2

Classical Cryptography

Chapter III

Complete Addition Formulas for Prime Order Elliptic Curves

An elliptic curve addition law is said to be *complete* if it correctly computes the sum of *any* two rational points in the elliptic curve group. One of the main reasons for the increased popularity of Edwards curves in the ECC community is that they can allow a complete group law that is also relatively efficient (e. g. when compared to all known addition laws on Edwards curves). Such complete addition formulas can simplify the task of an ECC implementer and, at the same time, can greatly reduce the potential vulnerabilities of a cryptosystem. Unfortunately, until now, complete addition laws that are relatively efficient have only been proposed on curves of composite order[1] and have thus been incompatible with all of the currently standardized prime order curves.

In this chapter we present optimized addition formulas that are complete on *every* prime order short Weierstrass curve defined over a field k such that $\mathrm{char}(k) \neq 2, 3$. Compared to their incomplete counterparts, these formulas require a larger number of field additions, but interestingly require fewer field multiplications. We discuss how these formulas can be used to achieve secure, exception-free implementations on *all* of the prime order curves in the NIST (and many other) standards.

[1] The *order* of an elliptic curve $E/\mathbb{F}_q$ is defined as $\#E(\mathbb{F}_q)$.

1 Introduction

Extending the works of Lange–Ruppert [LR85] and Bosma–Lenstra [BL95], Arène, Kohel and Ritzenthaler [AKR12] showed that, under any projective embedding of an elliptic curve E/k, *every* addition law has pairs of exceptional points in $(E \times E)(\bar{k})$. That is, over the algebraic closure of k, there are always pairs of points for which a given elliptic curve addition law does not work.

Fortunately, in elliptic curve cryptography (ECC), we are most often only concerned with the k-rational points on E. In this case it is possible to have a single addition law that is well-defined on all pairs of k-rational points, because its exceptional pairs are found in $(E \times E)(\bar{k})$, but not in $(E \times E)(k)$. A celebrated example of this is the Edwards model [Edw07]; when suitably chosen [BL07], an Edwards curve has a simple addition law that works for all pairs of k-rational points. This phenomenon was characterized more generally over elliptic curves by Kohel [Koh11], and further generalized to arbitrary abelian varieties in [AKR12]. For our purposes it suffices to state a special case of the more general results in [Koh11; AKR12]: namely, that every elliptic curve E over a finite field $\mathbb{F}_q$ (with $q \geq 5$) has an $\mathbb{F}_q$-complete addition law corresponding to the short Weierstrass model in $\mathbb{P}^2(\mathbb{F}_q)$.

Addition laws that are $\mathbb{F}_q$-complete are highly desirable in ECC. They can significantly simplify the task of an implementer and greatly reduce the potential vulnerabilities of a cryptosystem. We elaborate on this below.

Our contributions. In Algorithm 1 we present an optimized $\text{ADD} : E \times E \to E$ function, i. e. point addition formulas that correctly compute the sum of *any* two points on *any* odd order elliptic curve $E/\mathbb{F}_q\colon y^2 = x^3 + ax + b$ with $q \geq 5$. We do not claim credit for the complete formulas themselves, as these are exactly the formulas given by Bosma and Lenstra two decades ago [BL95]. What is novel in this chapter is optimizing the explicit computation of these formulas for cryptographic application. In particular, Table 1 shows that the computation of the Bosma–Lenstra complete additions can be performed using fewer general field multiplications than the best known (incomplete!) addition formulas on short Weierstrass curves: excluding multiplications by curve constants and field additions, the explicit formulas in this chapter compute additions in 12 field multiplications (12**M**), while the fastest known addition formulas in homogeneous coordinates require 14 field multiplications ($12\mathbf{M} + 2\mathbf{S}$) and the fastest known addition formulas in Jacobian coordinates require 16 field multiplications ($11\mathbf{M} + 5\mathbf{S}$). We immediately note, however, that our explicit formulas incur a much larger number of field additions than their incom-

Table 1. Summary of explicit formulas for the addition law on prime order short Weierstrass elliptic curves $E/k\colon y^2 = x^3 + ax + b$ in either homogeneous coordinates or Jacobian coordinates, and the corresponding exceptions (excep.) in both point doublings (DBL) and point additions (ADD). Here the operation counts include multiplications (**M**), squarings (**S**), multiplications by a ($\mathbf{m_a}$), multiplications by (small multiples of) b ($\mathbf{m_b}$), and additions (**a**), all in the ground field k. We note that various trade-offs exist with the above formulas [BLb].

Ref.	a	ADD(P,Q)						DBL(P)					
		Excep.	**M**	**S**	$\mathbf{m_a}$	$\mathbf{m_b}$	**a**	**Excep.**	**M**	**S**	$\mathbf{m_a}$	$\mathbf{m_b}$	**a**
This	any		12	0	3	2	23		8	3	3	2	15
	-3	—	12	0	0	2	29	—	8	3	0	2	21
	-0		12	0	0	2	19		6	2	0	1	9
[CMO98; BLb]	any		12	2	0	0	7		5	6	1	0	12
[CMO98; BLb]	-3	$Q \neq \pm P, \mathcal{O}$	12	2	0	0	7	$P \neq \mathcal{O}$	7	3	0	0	11
	-0				—						—		
[CMO98]	any		12	4	0	0	7		3	6	1	0	13
[CMO98; LG10]	-3	$Q \neq \pm P, \mathcal{O}$	12	4	0	0	7	—	4	4	0	0	8
[CMO98; HLX12]	-0		12	4	0	0	7		3	4	0	0	7

plete counterparts. Thus, as is discussed at length below, the relative performance of the complete additions will be highly dependent on the platform and/or scenario. However, we stress that outperforming the incomplete addition formulas is not the point of this chapter: our aim is to provide the fastest possible complete formulas for prime order curves.

Wide applicability. While the existence of an $\mathbb{F}_q$-complete addition law for prime order Weierstrass curves is not news to mathematicians (or to anyone that has read, e. g. [BL95; AKR12]), we hope it might be a pleasant surprise to ECC practitioners. In particular, the benefits of completeness are now accessible to anyone whose task it is to securely implement the prime order curves in the standards. These include:

– The example curves specified in the working drafts versions X9.62 and X9.63 [Acc99a; Acc99b] of the American National Standards Institute (ANSI).

– The five NIST prime curves specified in the current USA digital signature standard (DSS), i. e. FIPS 186-4 – see [Nat00; Nat13]. This includes Curve P-384, which is the National Security Agency (NSA) recommended curve in the most recent Suite B fact sheet for both key exchange and digital signatures [Nat15; Com15]; Curve P-256, which is the most widely supported curve in the Secure Shell (SSH) and Transport Layer Security (TLS) protocol [Bos+14, §3.2-3.3]; and

Curve P-192, which is the most common elliptic curve used in Austria's national e-ID cards [Bos+14, §3.4].

- The seven curves specified in the German brainpool standard [ECC05]. That is, `brainpoolPXXXr1` where `XXX` $\in \{160, 192, 224, 256, 320, 384, 512\}$.
- The eight curves specified by the UK-based company Certivox [Cer], namely `ssc-XXX`, where `XXX` $\in \{160, 192, 224, 256, 288, 320, 384, 512\}$.
- The curve `FRP256v1` recommended by the French Agence nationale de la sécurité des systèmes d'information (ANSSI) [Age14].
- The three curves specified (in addition to the above NIST prime curves) in the Certicom SEC 2 standard [Cer10]. This includes `secp256k1`, which is the curve used in the Bitcoin protocol.
- The recommended curve in the Chinese SM2 [Chi10] digital signature algorithm.
- The example curve in the Russian GOST R 34.10 standard [Gov01].

In particular, implementers can now write secure, exception-free code that supports all of the above curves without ever having to look further than the ADD function for curve arithmetic. Moreover, in §4.2 we show how ADD can easily be used to securely implement the two composite order curves, Curve25519 [Ber06a] and Ed448-Goldilocks [Ham15b], recently recommended for inclusion in future versions of TLS by the Internet Research Task Force Crypto Forum Research Group (IRTF CFRG).

Side-channel protection. Real-world implementations of ECC have a number of potential side-channel vulnerabilities that could fall victim to simple timing attacks [Koc96] or exceptional point attacks [IT02; FGV11]. One of the main reasons these attacks pose a threat is the *branching* that is inherent in the schoolbook short Weierstrass elliptic curve addition operation. For example, among the dozens of `if` statements in OpenSSL's[2] function `ec_GFp_simple_add` for point addition, the first three that check whether the input points are equal, opposite, or at infinity can cause timing variability (and therefore leak secret data) in ECDH or ECDSA. The complete formulas in this chapter remove these vulnerabilities and significantly decrease the attack surface of a cryptosystem. As Bernstein and Lange point out [BL09], completeness "eases implementations" and "avoids simple side-channel attacks".

[2] See `ecp_smpl.c` in `crypto/ec/` in the latest release at `https://openssl.org/source/`.

Although it is possible to use incomplete formulas safely, e. g. by carefully deriving uniform scalar multiplication algorithms that avoid exceptional pairs of inputs, implementing these routines in *constant-time* and in a provably correct way can be a cumbersome and painstaking process [Bos+16, §4]. Constant-time ECC implementations typically recode scalars from their binary encoding to some other form that allows a uniform execution path (c. f. Okeya-Tagaki [OT03] and Joye-Tunstall [JT09]), and these recodings can complicate the analysis of exceptional inputs to the point addition functions. For example, it can be difficult to prove that the running value in a scalar multiplication is never equal to (or the inverse of) elements in the lookup table; if this equality occurs before an addition, the incomplete addition function is likely to fail. Furthermore, guaranteeing exception-free, constant-time implementations of more exotic scalar multiplication routines, e. g. multiscalar multiplication for ECDSA verification, fixed-base scalar multiplications [LL94], scalar multiplications exploiting endomorphisms [GLV01], or scalar multiplications using common power analysis countermeasures [Cor99; FV12], is even more difficult; that is, unless the routine can call complete addition formulas.

Performance considerations. While the wide applicability and correctness of the ADD function is at the heart of this chapter, we have also aimed to cater to implementers that do not want to sacrifice free performance gains, particularly those concerned with supporting a special curve or special family of curves. To that end, Algorithms 2 and 3 give faster complete addition formulas in the special (and standardized) cases that the Weierstrass curve constant a is $a = -3$ or $a = 0$, and in the special case of point doublings (DBL); Table 1 summarizes the operation counts for all of these scenarios.

As we mentioned above, outperforming the (previously deployed) incomplete addition formulas is not the point of this chapter. Indeed, the high number of field additions present in our complete addition functions are likely to introduce an overall slowdown in many scenarios. To give an idea of this *performance hit* in a common software scenario, we plugged our complete addition functions into OpenSSL's implementation of the five NIST prime curves. Using the `openssl speed` function to benchmark the performance of the existing incomplete formulas and the new complete formulas shows that the latter incurs between a 1.34x and 1.44x slowdown in an average run of the elliptic curve Diffie-Hellman (ECDH) protocol (see Table 5 for the full details). As we discuss below, and in detail in §4.3, this factor slowdown should be considered an upper bound on the difference in performance between the fastest incomplete algorithms and our complete ones.

On the contrary, there are example scenarios where plugging in the complete formulas will result in an unnoticeable performance difference, or possibly even a speedup. For example, compared to the incomplete addition function used in the Bitcoin code[3] (`secp256k1_gej_add_var`), our complete addition function ADD in Algorithm 3 saves $4\mathbf{S}$ at the cost of $8\mathbf{a} + 1\texttt{mul\_int}$[4]; compared to Bitcoin's incomplete mixed addition (`secp256k1_gej_add_ge_var`), our complete mixed addition saves $3\mathbf{S}$ at the cost of $3\mathbf{M} + 2\mathbf{a} + 1\texttt{mul\_int}$; and, compared to Bitcoin's doubling function (`secp256k1_gej_double_var`), our formulas save $2\mathbf{S} + 5\texttt{mul\_int}$ at the cost of $3\mathbf{M} + 3\mathbf{a}$. In this case it is unclear which set of formulas would perform faster, but it is likely to be relatively close and to depend on the underlying field arithmetic and/or target platform. Furthermore, the overall speed is not just dependent on the formulas: the `if` statements present in the Bitcoin code also hamper performance. On the contrary, the complete formulas in this chapter have no `if` statements.

There are a number of additional real-world scenarios where the performance gap between the incomplete and the complete formulas will not be as drastic as the OpenSSL example above. The operation counts in Table 1 and Table 6 suggest that this will occur when the cost of field multiplications and squarings heavily outweighs the cost of field additions. The benchmarks above were obtained on a 64-bit processor, where the $\mathbf{M}/\mathbf{a}$ ratio tends to be much lower than that of low-end (e. g. 8-, 16-, and 32-bit) architectures. For example, field multiplications on wireless sensor nodes commonly require over 10 times more clock cycles than a field addition (see e. g. [Liu+13, Table 1] and [Szc+08, Table 1]), and in those cases the complete formulas in this chapter are likely to be very competitive in terms of raw performance.

In any case, we believe that many practitioners will agree that a small performance difference is a worthwhile cost to pay for branch-free point addition formulas that culminate in much simpler and more compact code, which *guarantees* correctness of the outputs and eliminates several side-channel vulnerabilities. We also note that the Bitcoin curve is not an isolated example of the more favorable formula comparison above: several families of pairing-friendly curves, including Barreto-Naehrig (BN) curves [BN06] which have appeared in recent IETF drafts, also have $a = 0$. In those cases, our specialized, exception-free formulas give implementers an easy way to correctly implement curve arithmetic in both $\mathbb{G}_1$ and $\mathbb{G}_2$ in the setting of cryptographic pairings. On a related note, we point that the word "prime" in our title can be relaxed to "odd"; the completeness of the Bosma–Lenstra formulas only

[3] See `https://github.com/bitcoin/bitcoin/tree/master/src/secp256k1`.

[4] `mul_int` denotes the cost of Bitcoin's specialized function that multiplies field elements by small integers.

requires the non-existence of rational two-torsion points (see §2), i. e. that the group order $\#E(\mathbb{F}_q)$ is not even. The BN curves define $\mathbb{G}_2$ as (being isomorphic to) a proper subgroup of a curve $E'/\mathbb{F}_{p^2}$, whose group order $\#E'(\mathbb{F}_{p^2})$ is the product of a large prime with odd integers [BN06, §3], meaning that our explicit formulas are not only complete in $\mathbb{G}_2 \subset E'(\mathbb{F}_{p^2})$, but also in $E'(\mathbb{F}_{p^2})$.

Related work. Complete addition laws have been found and studied on different models of elliptic curves, e. g. on the (twisted) Edwards [BL07; Ber+08] and (twisted) Hessian models [Ber+15b]. Unfortunately, in all of those scenarios, the models are not compatible with prime order curves and therefore all of the standardized curves mentioned above.

In terms of obtaining a complete and computationally efficient addition algorithm for prime order curves, there has been little success to date. Bernstein and Lange [BL09] found complete formulas on a non-Weierstrass model that would be compatible with, e. g. the NIST curves, reporting explicit formulas that (ignoring additions and multiplications by curve constants) cost $26\mathbf{M} + 8\mathbf{S}$. Bos et al. [Bos+16] considered applying the set of two Bosma–Lenstra addition laws to certain prime order Weierstrass curves, missing the observation (c. f. [AKR12, Remark 4.4]) that one of the addition laws is enough, and abandoning the high cost of computing both addition laws for an alternative but more complicated approach towards side-channel protection [Bos+16, Appendix C]. Brier and Joye [BJ02] developed *unified* formulas[5] for general Weierstrass curves, but these formulas still have exceptions and (again, ignoring additions and multiplications by curve constants) require $11\mathbf{M} + 6\mathbf{S}$, which is significantly slower than our complete algorithms.

Prime order curves can be safe. Several of the standardized curves of prime order mentioned above have recently been critiqued in [BLc], where they were deemed not to meet (some or all of) the four "ECC security" requirements: (i) Ladder, (ii) Twists, (iii) Completeness, and (iv) Indistinguishability.

On the contrary, this chapter shows that prime order curves have complete formulas that are comparably efficient. In addition, Brier and Joye [BJ02, §4] extended the Montgomery ladder to all short Weierstrass curves. In particular, when the curve $E/\mathbb{F}_q\colon y^2 = x^3 + ax + b$ has prime order, their formulas give rise to a function LADDER that computes $x([m]P) = \text{LADDER}(x(P), m, a, b)$ for the points $P \in E(\mathbb{F}_{q^2})$ with $(x, y) \in \mathbb{F}_q \times \mathbb{F}_{q^2}$. That is, a function that works for all $x \in \mathbb{F}_q$ and that does not distinguish whether x corresponds to a point on the curve E, or to a point on its

[5] These are addition formulas that also work for point doublings.

quadratic twist $E'\colon dy^2 = x^3 + ax + b$, where d is non-square in $\mathbb{F}_q$. If E is chosen to be twist-secure (this presents no problem in the prime order setting), then for all $x \in \mathbb{F}_q$, the function $\text{LADDER}(x, m, a, b)$ returns an instance of the discrete logarithm problem (whose solution is m) on a cryptographically strong curve, just like the analogous function on twist-secure Montgomery curves [Ber06a]. Finally, we note that Tibouchi [Tib14] presented a prime-order analogue of the encoding given for certain composite-order curves in [Ber+13], showing that the indistinguishability property can also be achieved on prime order curves.

As is discussed in [BLc], adopting the Brier-Joye ladder (or, in our case, the complete formulas) in place of the fastest formulas presents implementers with a trade-off between "simplicity, security and speed". However, these same trade-offs also exist on certain choices of Edwards curves, where, for example, the fastest explicit formulas are also not complete: the Curve41417 implementation chooses to sacrifice the fastest coordinate system for the sake of completeness [BCL14, §3.1], while the Goldilocks implementation goes to more complicated lengths to use the fastest formulas [Ham14; Ham15a; Ham15b]. Furthermore, there is an additional category that is not considered in [BLc], i. e. the non-trivial security issues related to having a cofactor h greater than 1 [Ham15a, §1.1].

Given the complete explicit formulas in this chapter, it is our opinion that well-chosen prime order curves can be considered safe choices for elliptic curve cryptography. It is well-known that curves with cofactors offer efficiency benefits in certain scenarios, but to our knowledge, efficiency and/or bandwidth issues are the only valid justifications for choosing a curve with a cofactor $h > 1$.

Organization. In §2 we present the complete addition function ADD. In §3 we give intuition as to why these explicit formulas are optimal, or close to optimal, for prime order curves in short Weierstrass form. In §4 we discuss how these formulas can be used in practice. We also provide Magma [BCP97] scripts that can be used to verify our explicit algorithms and operation counts (see `https://joostrenes.nl`).

2 Complete Addition Formulas

The complete addition formulas optimized in this section follow from the theorem of Bosma and Lenstra [BL95, Theorem 2], which states that, for any extension field K/k, there exists a 1–to–1 correspondence between lines in $\mathbb{P}^2(K)$ and addition laws of bidegree $(2, 2)$ on $E(K)$. Two points P and Q in $E(K)$ are then exceptional for an addition law if and only if $P - Q$ lies on the corresponding line. When $K = \bar{k}$,

the algebraic closure of k, *every* line intersects $E(K)$; thus, one consequence of this theorem is that every addition law of bidegree $(2,2)$ has an exceptional pair over the algebraic closure.

The addition law considered in this chapter is the addition law corresponding to the line $Y = 0$ in $\mathbb{P}^2$ in [BL95], specialized to the short Weierstrass embedding of E above. For two points $P = (X_1 : Y_1 : Z_1)$, $Q = (X_2 : Y_2 : Z_2)$ on E, the sum $(X_3 : Y_3 : Z_3) = P + Q$ is given by

$$\begin{aligned}
X_3 &= Y_1Y_2(X_1Y_2 + X_2Y_1) - aX_1X_2(Y_1Z_2 + Y_2Z_1) \\
&\quad - a(X_1Y_2 + X_2Y_1)(X_1Z_2 + X_2Z_1) - 3b(X_1Y_2 + X_2Y_1)Z_1Z_2 \\
&\quad - 3b(X_1Z_2 + X_2Z_1)(Y_1Z_2 + Y_2Z_1) + a^2(Y_1Z_2 + Y_2Z_1)Z_1Z_2\,, \\
Y_3 &= Y_1^2Y_2^2 + 3aX_1^2X_2^2 + 9bX_1X_2(X_1Z_2 + X_2Z_1) \\
&\quad - 2a^2X_1Z_2(X_1Z_2 + 2X_2Z_1) + a^2(X_1Z_2 + X_2Z_1)(X_1Z_2 - X_2Z_1) \\
&\quad - 3abX_1Z_1Z_2^2 - 3abX_2Z_1^2Z_2 - (a^3 + 9b^2)Z_1^2Z_2^2\,, \\
Z_3 &= 3X_1X_2(X_1Y_2 + X_2Y_1) + Y_1Y_2(Y_1Z_2 + Y_2Z_1) + a(X_1Y_2 + X_2Y_1)Z_1Z_2 \\
&\quad + a(X_1Z_2 + X_2Z_1)(Y_1Z_2 + Y_2Z_1) + 3b(Y_1Z_2 + Y_2Z_1)Z_1Z_2\,.
\end{aligned}$$

Bosma and Lenstra prove that a pair of points (P, Q) is exceptional for this addition law if and only if $P - Q$ is a point of order two.

Exceptions. Throughout this chapter, we fix $q \geq 5$ and assume throughout that $E(\mathbb{F}_q)$ has prime order to exclude $\mathbb{F}_q$-rational points of order two, so that the above formulas are complete. However, we note that the explicit algorithms that are derived in §2 will, firstly, be complete for any short Weierstrass curves of odd order, and secondly, also be exception-free for all pairs of points inside odd order subgroups on any short Weierstrass curve. In particular, this means that they can also be used to compute exception-free additions and scalar multiplications on certain curves with an even order. We come back to this in §4.2.

2.1 The General Case

Despite the attractive properties that come with completeness, this addition law seems to have been overlooked[6] due to its apparent inefficiency. We now show that these formulas are not as inefficient as they seem, to the point where the performance will be competitive with the fastest, incomplete addition laws in current

[6] Some (unpublished) results were obtained by Joye (see `http://joye.site88.net/techreps/complete.pdf`), but this chapter improves on it further.

Table 2. Operation counts for group operations on the short Weierstrass curve $E/\mathbb{F}_q : y^2 = x^3 + ax + b$.

	M	**S**	$\mathbf{m_a}$	$\mathbf{m_{3b}}$	**a**
ADD	12	0	3	2	23
MADD	11	0	3	2	17
DBL	8	3	3	2	15

implementations of prime order curves. We start by rewriting the above formulas as

$$\begin{aligned}
X_3 &= (X_1Y_2 + X_2Y_1)(Y_1Y_2 - a(X_1Z_2 + X_2Z_1) - 3bZ_1Z_2) \\
&\quad - (Y_1Z_2 + Y_2Z_1)(aX_1X_2 + 3b(X_1Z_2 + X_2Z_1) - a^2Z_1Z_2)\,, \\
Y_3 &= (3X_1X_2 + aZ_1Z_2)(aX_1X_2 + 3b(X_1Z_2 + X_2Z_1) - a^2Z_1Z_2) + \\
&\quad (Y_1Y_2 + a(X_1Z_2 + X_2Z_1) + 3bZ_1Z_2)(Y_1Y_2 - a(X_1Z_2 + X_2Z_1) - 3bZ_1Z_2)\,, \\
Z_3 &= (Y_1Z_2 + Y_2Z_1)(Y_1Y_2 + a(X_1Z_2 + X_2Z_1) + 3bZ_1Z_2) \\
&\quad + (X_1Y_2 + X_2Y_1)(3X_1X_2 + aZ_1Z_2)\,. \qquad (1)
\end{aligned}$$

The rewritten formulas still appear somewhat cumbersome, but a closer inspection of (1) reveals that several terms are repeated. Although they are sufficient for cryptographic implementations, performance gains can be obtained by specializing the point additions to the useful scenarios of mixed additions[7] (i. e. where $Z_2 = 1$) and/or point doublings (i. e. where $P = Q$). The mixed addition follows the same formulas as for point addition, while for a point $P = (X : Y : Z)$, doubling is computed as

$$\begin{aligned}
X_3 &= 2XY(Y^2 - 2aXZ - 3bZ^2) \\
&\quad - 2YZ(aX^2 + 6bXZ - a^2Z^2)\,, \\
Y_3 &= (Y^2 + 2aXZ + 3bZ^2)(Y^2 - 2aXZ - 3bZ^2) \\
&\quad + (3X^2 + aZ^2)(aX^2 + 6bXZ - a^2Z^2)\,, \\
Z_3 &= 8Y^3Z\,.
\end{aligned}$$

Throughout this chapter we write ADD for the function that maps $P = (X_1 : Y_1 : Z_1)$

[7] We note that it is not technically correct to call "mixed" additions complete, since $Z_2 = 1$ precludes the second point being the point at infinity. However, this is not a problem in practice as the second point is typically taken from a precomputed lookup table consisting of small multiples of the input point $P \neq \mathcal{O}$. For prime order curves, these small multiples can never be at infinity.

and $Q = (X_2 : Y_2 : Z_2)$ to $P + Q = (X_3 : Y_3 : Z_3)$, we denote by MADD the ADD function under the assumption that $Z_2 = 1$ and we write DBL for the ADD function assuming that $P = Q$. We summarize the costs of these algorithms in Table 2, and we refer to Algorithm 1 for a (non-unique) way of attaining these operation counts.

Algorithm 1. Group operations in the prime order group $E(\mathbb{F}_q)$ on an arbitrary short Weierstrass curve $E/\mathbb{F}_q : y^2 = x^3 + ax + b$

Function: ADD

1 $t_0 \leftarrow X_1 \cdot X_2$
2 $t_1 \leftarrow Y_1 \cdot Y_2$
3 $t_2 \leftarrow Z_1 \cdot Z_2$
4 $t_3 \leftarrow X_1 + Y_1$
5 $t_4 \leftarrow X_2 + Y_2$
6 $t_3 \leftarrow t_3 \cdot t_4$
7 $t_4 \leftarrow t_0 + t_1$
8 $t_3 \leftarrow t_3 - t_4$
9 $t_4 \leftarrow X_1 + Z_1$
10 $t_5 \leftarrow X_2 + Z_2$
11 $t_4 \leftarrow t_4 \cdot t_5$
12 $t_5 \leftarrow t_0 + t_2$
13 $t_4 \leftarrow t_4 - t_5$
14 $t_5 \leftarrow Y_1 + Z_1$
15 $X_3 \leftarrow Y_2 + Z_2$
16 $t_5 \leftarrow t_5 \cdot X_3$
17 $X_3 \leftarrow t_1 + t_2$
18 $t_5 \leftarrow t_5 - X_3$
19 $Z_3 \leftarrow a \cdot t_4$
20 $X_3 \leftarrow b_3 \cdot t_2$
21 $Z_3 \leftarrow X_3 + Z_3$
22 $X_3 \leftarrow t_1 - Z_3$
23 $Z_3 \leftarrow t_1 + Z_3$
24 $Y_3 \leftarrow X_3 \cdot Z_3$
25 $t_1 \leftarrow t_0 + t_0$
26 $t_1 \leftarrow t_1 + t_0$
27 $t_2 \leftarrow a \cdot t_2$
28 $t_4 \leftarrow b_3 \cdot t_4$
29 $t_1 \leftarrow t_1 + t_2$
30 $t_2 \leftarrow t_0 - t_2$
31 $t_2 \leftarrow a \cdot t_2$
32 $t_4 \leftarrow t_4 + t_2$
33 $t_0 \leftarrow t_1 \cdot t_4$
34 $Y_3 \leftarrow Y_3 + t_0$
35 $t_0 \leftarrow t_5 \cdot t_4$
36 $X_3 \leftarrow t_3 \cdot X_3$
37 $X_3 \leftarrow X_3 - t_0$
38 $t_0 \leftarrow t_3 \cdot t_1$
39 $Z_3 \leftarrow t_5 \cdot Z_3$
40 $Z_3 \leftarrow Z_3 + t_0$

Function: MADD

41 $t_0 \leftarrow X_1 \cdot X_2$
42 $t_1 \leftarrow Y_1 \cdot Y_2$
43 $t_3 \leftarrow X_2 + Y_2$
44 $t_4 \leftarrow X_1 + Y_1$
45 $t_3 \leftarrow t_3 \cdot t_4$
46 $t_4 \leftarrow t_0 + t_1$
47 $t_3 \leftarrow t_3 - t_4$
48 $t_4 \leftarrow X_2 \cdot Z_1$
49 $t_4 \leftarrow t_4 + X_1$
50 $t_5 \leftarrow Y_2 \cdot Z_1$
51 $t_5 \leftarrow t_5 + Y_1$
52 $Z_3 \leftarrow a \cdot t_4$
53 $X_3 \leftarrow b_3 \cdot Z_1$
54 $Z_3 \leftarrow X_3 + Z_3$
55 $X_3 \leftarrow t_1 - Z_3$
56 $Z_3 \leftarrow t_1 + Z_3$
57 $Y_3 \leftarrow X_3 \cdot Z_3$
58 $t_1 \leftarrow t_0 + t_0$
59 $t_1 \leftarrow t_1 + t_0$
60 $t_2 \leftarrow a \cdot Z_1$
61 $t_4 \leftarrow b_3 \cdot t_4$
62 $t_1 \leftarrow t_1 + t_2$
63 $t_2 \leftarrow t_0 - t_2$
64 $t_2 \leftarrow a \cdot t_2$
65 $t_4 \leftarrow t_4 + t_2$
66 $t_0 \leftarrow t_1 \cdot t_4$
67 $Y_3 \leftarrow Y_3 + t_0$
68 $t_0 \leftarrow t_5 \cdot t_4$
69 $X_3 \leftarrow t_3 \cdot X_3$
70 $X_3 \leftarrow X_3 - t_0$
71 $t_0 \leftarrow t_3 \cdot t_1$
72 $Z_3 \leftarrow t_5 \cdot Z_3$
73 $Z_3 \leftarrow Z_3 + t_0$

Function: DBL

74 $t_0 \leftarrow X \cdot X$
75 $t_1 \leftarrow Y \cdot Y$
76 $t_2 \leftarrow Z \cdot Z$
77 $t_3 \leftarrow X \cdot Y$
78 $t_3 \leftarrow t_3 + t_3$
79 $Z_3 \leftarrow X \cdot Z$
80 $Z_3 \leftarrow Z_3 + Z_3$
81 $X_3 \leftarrow a \cdot Z_3$
82 $Y_3 \leftarrow b_3 \cdot t_2$
83 $Y_3 \leftarrow X_3 + Y_3$
84 $X_3 \leftarrow t_1 - Y_3$
85 $Y_3 \leftarrow t_1 + Y_3$
86 $Y_3 \leftarrow X_3 \cdot Y_3$
87 $X_3 \leftarrow t_3 \cdot X_3$
88 $Z_3 \leftarrow b_3 \cdot Z_3$
89 $t_2 \leftarrow a \cdot t_2$
90 $t_3 \leftarrow t_0 - t_2$
91 $t_3 \leftarrow a \cdot t_3$
92 $t_3 \leftarrow t_3 + Z_3$
93 $Z_3 \leftarrow t_0 + t_0$
94 $t_0 \leftarrow Z_3 + t_0$
95 $t_0 \leftarrow t_0 + t_2$
96 $t_0 \leftarrow t_0 \cdot t_3$
97 $Y_3 \leftarrow Y_3 + t_0$
98 $t_2 \leftarrow Y \cdot Z$
99 $t_2 \leftarrow t_2 + t_2$
100 $t_0 \leftarrow t_2 \cdot t_3$
101 $X_3 \leftarrow X_3 - t_0$
102 $Z_3 \leftarrow t_2 \cdot t_1$
103 $Z_3 \leftarrow Z_3 + Z_3$
104 $Z_3 \leftarrow Z_3 + Z_3$

Table 3. Operation counts for group operations on the short Weierstrass curve $E/\mathbb{F}_q : y^2 = x^3 - 3x + b$.

	M	**S**	$\mathbf{m_b}$	**a**
ADD	12	0	2	29
MADD	11	0	2	23
DBL	8	3	2	21

Table 4. Operation counts for group operations on the short Weierstrass curve $E/\mathbb{F}_q : y^2 = x^3 + b$.

	M	**S**	$\mathbf{m_{3b}}$	**a**
ADD	12	0	2	19
MADD	11	0	2	13
DBL	6	2	1	9

2.2 The Case $a = -3$

Several standards (e. g. [Cer; Nat13; Kir+15; Age14; Cer10; ECC05]) adopt short Weierstrass curves with the constant a being $a = -3$, which gives rise to faster explicit formulas for point doubling.[8] In this case, the complete formulas in (1) specialize to

$$\begin{aligned}
X_3 &= (X_1Y_2 + X_2Y_1)(Y_1Y_2 + 3(X_1Z_2 + X_2Z_1 - bZ_1Z_2)) \\
&\quad - 3(Y_1Z_2 + Y_2Z_1)(b(X_1Z_2 + X_2Z_1) - X_1X_2 - 3Z_1Z_2)\,, \\
Y_3 &= 3(3X_1X_2 - 3Z_1Z_2)(b(X_1Z_2 + X_2Z_1) - X_1X_2 - 3Z_1Z_2) + \\
&\quad (Y_1Y_2 - 3(X_1Z_2 + X_2Z_1 - bZ_1Z_2))(Y_1Y_2 + 3(X_1Z_2 + X_2Z_1 - bZ_1Z_2))\,, \\
Z_3 &= (Y_1Z_2 + Y_2Z_1)(Y_1Y_2 - 3(X_1Z_2 + X_2Z_1 - bZ_1Z_2)) \\
&\quad + (X_1Y_2 + X_2Y_1)(3X_1X_2 - 3Z_1Z_2)\,.
\end{aligned}$$

The doubling formulas simplify to

$$\begin{aligned}
X_3 &= 2XY(Y^2 + 3(2XZ - bZ^2)) \\
&\quad - 6YZ(2bXZ - X^2 - 3Z^2)\,, \\
Y_3 &= (Y^2 - 3(2XZ - bZ^2))(Y^2 + 3(2XZ - bZ^2)) \\
&\quad + 3(3X^2 - 3Z^2)(2bXZ - X^2 - 3Z^2)\,, \\
Z_3 &= 8Y^3Z\,.
\end{aligned}$$

We describe the costs of these algorithms in Table 3, and we refer to Algorithm 2 for a (non-unique) way of achieving these operation counts.

[8] When $\mathbb{F}_q$ is a large prime field, the case $a = -3$ covers 1/2 resp. 1/4 of the isomorphism classes for $q \equiv 3 \pmod 4$ resp. $q \equiv 1 \pmod 4$ — see [BJ03, §3].

Algorithm 2. Group operations in the prime order group $E(\mathbb{F}_q)$ on the short Weierstrass curve $E/\mathbb{F}_q : y^2 = x^3 - 3x + b$

Function: ADD

1 $t_0 \leftarrow X_1 \cdot X_2$
2 $t_1 \leftarrow Y_1 \cdot Y_2$
3 $t_2 \leftarrow Z_1 \cdot Z_2$
4 $t_3 \leftarrow X_1 + Y_1$
5 $t_4 \leftarrow X_2 + Y_2$
6 $t_3 \leftarrow t_3 \cdot t_4$
7 $t_4 \leftarrow t_0 + t_1$
8 $t_3 \leftarrow t_3 - t_4$
9 $t_4 \leftarrow Y_1 + Z_1$
10 $X_3 \leftarrow Y_2 + Z_2$
11 $t_4 \leftarrow t_4 \cdot X_3$
12 $X_3 \leftarrow t_1 + t_2$
13 $t_4 \leftarrow t_4 - X_3$
14 $X_3 \leftarrow X_1 + Z_1$
15 $Y_3 \leftarrow X_2 + Z_2$
16 $X_3 \leftarrow X_3 \cdot Y_3$
17 $Y_3 \leftarrow t_0 + t_2$
18 $Y_3 \leftarrow X_3 - Y_3$
19 $Z_3 \leftarrow b \cdot t_2$
20 $X_3 \leftarrow Y_3 - Z_3$
21 $Z_3 \leftarrow X_3 + X_3$
22 $X_3 \leftarrow X_3 + Z_3$
23 $Z_3 \leftarrow t_1 - X_3$
24 $X_3 \leftarrow t_1 + X_3$
25 $Y_3 \leftarrow b \cdot Y_3$
26 $t_1 \leftarrow t_2 + t_2$
27 $t_2 \leftarrow t_1 + t_2$
28 $Y_3 \leftarrow Y_3 - t_2$
29 $Y_3 \leftarrow Y_3 - t_0$
30 $t_1 \leftarrow Y_3 + Y_3$
31 $Y_3 \leftarrow t_1 + Y_3$
32 $t_1 \leftarrow t_0 + t_0$
33 $t_0 \leftarrow t_1 + t_0$
34 $t_0 \leftarrow t_0 - t_2$
35 $t_1 \leftarrow t_4 \cdot Y_3$
36 $t_2 \leftarrow t_0 \cdot Y_3$
37 $Y_3 \leftarrow X_3 \cdot Z_3$
38 $Y_3 \leftarrow Y_3 + t_2$
39 $X_3 \leftarrow t_3 \cdot X_3$
40 $X_3 \leftarrow X_3 - t_1$
41 $Z_3 \leftarrow t_4 \cdot Z_3$
42 $t_1 \leftarrow t_3 \cdot t_0$
43 $Z_3 \leftarrow Z_3 + t_1$

Function: MADD

44 $t_0 \leftarrow X_1 \cdot X_2$
45 $t_1 \leftarrow Y_1 \cdot Y_2$
46 $t_3 \leftarrow X_2 + Y_2$
47 $t_4 \leftarrow X_1 + Y_1$
48 $t_3 \leftarrow t_3 \cdot t_4$
49 $t_4 \leftarrow t_0 + t_1$
50 $t_3 \leftarrow t_3 - t_4$
51 $t_4 \leftarrow Y_2 \cdot Z_1$
52 $t_4 \leftarrow t_4 + Y_1$
53 $Y_3 \leftarrow X_2 \cdot Z_1$
54 $Y_3 \leftarrow Y_3 + X_1$
55 $Z_3 \leftarrow b \cdot Z_1$
56 $X_3 \leftarrow Y_3 - Z_3$
57 $Z_3 \leftarrow X_3 + X_3$
58 $X_3 \leftarrow X_3 + Z_3$
59 $Z_3 \leftarrow t_1 - X_3$
60 $X_3 \leftarrow t_1 + X_3$
61 $Y_3 \leftarrow b \cdot Y_3$
62 $t_1 \leftarrow Z_1 + Z_1$
63 $t_2 \leftarrow t_1 + Z_1$
64 $Y_3 \leftarrow Y_3 - t_2$
65 $Y_3 \leftarrow Y_3 - t_0$
66 $t_1 \leftarrow Y_3 + Y_3$
67 $Y_3 \leftarrow t_1 + Y_3$
68 $t_1 \leftarrow t_0 + t_0$
69 $t_0 \leftarrow t_1 + t_0$
70 $t_0 \leftarrow t_0 - t_2$
71 $t_1 \leftarrow t_4 \cdot Y_3$
72 $t_2 \leftarrow t_0 \cdot Y_3$
73 $Y_3 \leftarrow X_3 \cdot Z_3$
74 $Y_3 \leftarrow Y_3 + t_2$
75 $X_3 \leftarrow t_3 \cdot X_3$
76 $X_3 \leftarrow X_3 - t_1$
77 $Z_3 \leftarrow t_4 \cdot Z_3$
78 $t_1 \leftarrow t_3 \cdot t_0$
79 $Z_3 \leftarrow Z_3 + t_1$

Function: DBL

80 $t_0 \leftarrow X \cdot X$
81 $t_1 \leftarrow Y \cdot Y$
82 $t_2 \leftarrow Z \cdot Z$
83 $t_3 \leftarrow X \cdot Y$
84 $t_3 \leftarrow t_3 + t_3$
85 $Z_3 \leftarrow X \cdot Z$
86 $Z_3 \leftarrow Z_3 + Z_3$
87 $Y_3 \leftarrow b \cdot t_2$
88 $Y_3 \leftarrow Y_3 - Z_3$
89 $X_3 \leftarrow Y_3 + Y_3$
90 $Y_3 \leftarrow X_3 + Y_3$
91 $X_3 \leftarrow t_1 - Y_3$
92 $Y_3 \leftarrow t_1 + Y_3$
93 $Y_3 \leftarrow X_3 \cdot Y_3$
94 $X_3 \leftarrow X_3 \cdot t_3$
95 $t_3 \leftarrow t_2 + t_2$
96 $t_2 \leftarrow t_2 + t_3$
97 $Z_3 \leftarrow b \cdot Z_3$
98 $Z_3 \leftarrow Z_3 - t_2$
99 $Z_3 \leftarrow Z_3 - t_0$
100 $t_3 \leftarrow Z_3 + Z_3$
101 $Z_3 \leftarrow Z_3 + t_3$
102 $t_3 \leftarrow t_0 + t_0$
103 $t_0 \leftarrow t_3 + t_0$
104 $t_0 \leftarrow t_0 - t_2$
105 $t_0 \leftarrow t_0 \cdot Z_3$
106 $Y_3 \leftarrow Y_3 + t_0$
107 $t_0 \leftarrow Y \cdot Z$
108 $t_0 \leftarrow t_0 + t_0$
109 $Z_3 \leftarrow t_0 \cdot Z_3$
110 $X_3 \leftarrow X_3 - Z_3$
111 $Z_3 \leftarrow t_0 \cdot t_1$
112 $Z_3 \leftarrow Z_3 + Z_3$
113 $Z_3 \leftarrow Z_3 + Z_3$

2.3 The Case $a = 0$

Short Weierstrass curves with $a = 0$, i. e. with j-invariant 0, have also appeared in the standards. For example, Certicom's SEC-2 standard [Cer10] specifies three such curves; one of these is `secp256k1`, which is the curve used in the Bitcoin protocol. In addition, in the case that pairing-based cryptography becomes standardized, it is possible that the curve choices will be short Weierstrass curves with $a = 0$, e. g. BN curves [BN06]. In this case, the complete additions simplify to

$$\begin{aligned} X_3 &= (X_1Y_2 + X_2Y_1)(Y_1Y_2 - 3bZ_1Z_2) - 3b(Y_1Z_2 + Y_2Z_1)(X_1Z_2 + X_2Z_1)\,, \\ Y_3 &= (Y_1Y_2 + 3bZ_1Z_2)(Y_1Y_2 - 3bZ_1Z_2) + 9bX_1X_2(X_1Z_2 + X_2Z_1)\,, \\ Z_3 &= (Y_1Z_2 + Y_2Z_1)(Y_1Y_2 + 3bZ_1Z_2) + 3X_1X_2(X_1Y_2 + X_2Y_1)\,. \end{aligned}$$

The doubling formulas are

$$\begin{aligned} X_3 &= 2XY(Y^2 - 9bZ^2)\,, \\ Y_3 &= (Y^2 - 9bZ^2)(Y^2 + 3bZ^2) + 24bY^2Z^2\,, \\ Z_3 &= 8Y^3Z\,. \end{aligned}$$

The costs of these algorithms can be found in Table 4, while Algorithm 3 provides a (non-unique) way of achieving these operation counts.

3 Some Intuition Towards Optimality

In this section we motivate the choice of the complete formulas in (1) that were taken from Bosma and Lenstra [BL95], by providing reasoning as to why, among the many possible complete addition laws on prime order curves, we chose the set corresponding to the line $Y = 0$ in $\mathbb{P}^2(k)$ under the straightforward homogeneous projection.

We do not claim that this choice is truly optimal, since proving that a certain choice of projective embedding and/or complete addition law for any particular prime order curve is faster than *all* of the other choices for that curve seems extremely difficult, if not impossible. We merely explain why, when aiming to write down explicit functions that will simultaneously be complete on all prime order short Weierstrass curves, choosing the Bosma–Lenstra formulas makes sense.

Furthermore, we also do not claim that our explicit algorithms to compute the addition law in (1) are computationally optimal. It is likely that trade-offs can be advantageously exploited on some platforms (c. f. [His10, §3.6]) or that alternative

Algorithm 3. Group operations in the prime order group $E(\mathbb{F}_q)$ on the short Weierstrass curve $E/\mathbb{F}_q : y^2 = x^3 + b$

Function: ADD

1 $t_0 \leftarrow X_1 \cdot X_2$
2 $t_1 \leftarrow Y_1 \cdot Y_2$
3 $t_2 \leftarrow Z_1 \cdot Z_2$
4 $t_3 \leftarrow X_1 + Y_1$
5 $t_4 \leftarrow X_2 + Y_2$
6 $t_3 \leftarrow t_3 \cdot t_4$
7 $t_4 \leftarrow t_0 + t_1$
8 $t_3 \leftarrow t_3 - t_4$
9 $t_4 \leftarrow Y_1 + Z_1$
10 $X_3 \leftarrow Y_2 + Z_2$
11 $t_4 \leftarrow t_4 \cdot X_3$
12 $X_3 \leftarrow t_1 + t_2$
13 $t_4 \leftarrow t_4 - X_3$
14 $X_3 \leftarrow X_1 + Z_1$
15 $Y_3 \leftarrow X_2 + Z_2$
16 $X_3 \leftarrow X_3 \cdot Y_3$
17 $Y_3 \leftarrow t_0 + t_2$
18 $Y_3 \leftarrow X_3 - Y_3$
19 $X_3 \leftarrow t_0 + t_0$
20 $t_0 \leftarrow X_3 + t_0$
21 $t_2 \leftarrow b_3 \cdot t_2$
22 $Z_3 \leftarrow t_1 + t_2$
23 $t_1 \leftarrow t_1 - t_2$
24 $Y_3 \leftarrow b_3 \cdot Y_3$
25 $X_3 \leftarrow t_4 \cdot Y_3$
26 $t_2 \leftarrow t_3 \cdot t_1$
27 $X_3 \leftarrow t_2 - X_3$
28 $Y_3 \leftarrow Y_3 \cdot t_0$
29 $t_1 \leftarrow t_1 \cdot Z_3$
30 $Y_3 \leftarrow t_1 + Y_3$
31 $t_0 \leftarrow t_0 \cdot t_3$
32 $Z_3 \leftarrow Z_3 \cdot t_4$
33 $Z_3 \leftarrow Z_3 + t_0$

Function: MADD

34 $t_0 \leftarrow X_1 \cdot X_2$
35 $t_1 \leftarrow Y_1 \cdot Y_2$
36 $t_3 \leftarrow X_2 + Y_2$
37 $t_4 \leftarrow X_1 + Y_1$
38 $t_3 \leftarrow t_3 \cdot t_4$
39 $t_4 \leftarrow t_0 + t_1$
40 $t_3 \leftarrow t_3 - t_4$
41 $t_4 \leftarrow Y_2 \cdot Z_1$
42 $t_4 \leftarrow t_4 + Y_1$
43 $Y_3 \leftarrow X_2 \cdot Z_1$
44 $Y_3 \leftarrow Y_3 + X_1$
45 $X_3 \leftarrow t_0 + t_0$
46 $t_0 \leftarrow X_3 + t_0$
47 $t_2 \leftarrow b_3 \cdot Z_1$
48 $Z_3 \leftarrow t_1 + t_2$
49 $t_1 \leftarrow t_1 - t_2$
50 $Y_3 \leftarrow b_3 \cdot Y_3$
51 $X_3 \leftarrow t_4 \cdot Y_3$
52 $t_2 \leftarrow t_3 \cdot t_1$
53 $X_3 \leftarrow t_2 - X_3$
54 $Y_3 \leftarrow Y_3 \cdot t_0$
55 $t_1 \leftarrow t_1 \cdot Z_3$
56 $Y_3 \leftarrow t_1 + Y_3$
57 $t_0 \leftarrow t_0 \cdot t_3$
58 $Z_3 \leftarrow Z_3 \cdot t_4$
59 $Z_3 \leftarrow Z_3 + t_0$

Function: DBL

60 $t_0 \leftarrow Y \cdot Y$
61 $Z_3 \leftarrow t_0 + t_0$
62 $Z_3 \leftarrow Z_3 + Z_3$
63 $Z_3 \leftarrow Z_3 + Z_3$
64 $t_1 \leftarrow Y \cdot Z$
65 $t_2 \leftarrow Z \cdot Z$
66 $t_2 \leftarrow b_3 \cdot t_2$
67 $X_3 \leftarrow t_2 \cdot Z_3$
68 $Y_3 \leftarrow t_0 + t_2$
69 $Z_3 \leftarrow t_1 \cdot Z_3$
70 $t_1 \leftarrow t_2 + t_2$
71 $t_2 \leftarrow t_1 + t_2$
72 $t_0 \leftarrow t_0 - t_2$
73 $Y_3 \leftarrow t_0 \cdot Y_3$
74 $Y_3 \leftarrow X_3 + Y_3$
75 $t_1 \leftarrow X \cdot Y$
76 $X_3 \leftarrow t_0 \cdot t_1$
77 $X_3 \leftarrow X_3 + X_3$

operation scheduling could reduce the number of field additions.[9]

3.1 Choice of $Y = 0$ for Bidegree $(2,2)$ Addition Laws

Let $L_{(\alpha,\beta,\gamma)}$ denote the line given by $\alpha X + \beta Y + \gamma Z = 0$ inside $\mathbb{P}^2(\mathbb{F}_q)$, and, under the necessary assumption that $L_{(\alpha,\beta,\gamma)}$ does not intersect the $\mathbb{F}_q$-rational points of the curve $E\colon Y^2Z = X^3 + aXZ^2 + bZ^3$, let $A_{(\alpha,\beta,\gamma)}$ denote the complete addition law of bidegree $(2,2)$ corresponding to $L_{(\alpha,\beta,\gamma)}$ given by [BL95, Theorem 2]. So far we have given optimizations for $A_{(0,1,0)}$, but the question remains as to whether there are other lines $L_{(\alpha,\beta,\gamma)}$ which give rise to even faster addition laws $A_{(\alpha,\beta,\gamma)}$.

We first point out that $L_{(0,1,0)}$ is the only line that does not intersect $E(\mathbb{F}_q)$ independently of a, b and q. That is, it is easy to show that any other line in $\mathbb{P}^2(\mathbb{F}_q)$ that does not intersect the group of $\mathbb{F}_q$-rational points of any elliptic curve E such that $\#E(\mathbb{F}_q)$ is odd will have a dependency on at least one of a, b and q, and the resulting addition law will therefore only be complete on a subset of prime order curves.

Nevertheless, it is possible that there is a better choice than $A_{(0,1,0)}$ for a given short Weierstrass curve, or that there are special choices of prime order curves that give rise to more efficient complete group laws. We now sketch some intuition as to why this is unlikely. For $A_{(\alpha,\beta,\gamma)}$ to be complete, it is necessary that, in particular, $L_{(\alpha,\beta,\gamma)}$ does not intersect E at the point at infinity $(0:1:0)$. This implies that $\beta \neq 0$. From [LR85; BL95], we know that the space of all addition laws has dimension 3 and that

$$A_{(\alpha,\beta,\gamma)} = \alpha A_{(1,0,0)} + \beta A_{(0,1,0)} + \gamma A_{(0,0,1)}\,,$$

where $A_{(1,0,0)}$, $A_{(0,1,0)}$ and $A_{(0,0,1)}$ are the three addition laws given in [BL95, p. 236-239], specialized to short Weierstrass curves. Given that $\beta \neq 0$, our only hope of finding a more simple addition law than $A_{(0,1,0)}$ is by choosing α and/or γ in a way that causes an advantageous cross-cancellation of terms. Close inspection of the formulas in [BL95] strongly suggests that no such cancellation exists.

Remark 1. Interestingly, both $A_{(1,0,0)}$ and $A_{(0,0,1)}$ vanish to zero when specialized to doubling. This means that any doubling formula in bidegree $(2,2)$ that is not exceptional at the point at infinity is a scalar multiple of $A_{(0,1,0)}$, i. e. the formulas used in this chapter.

Remark 2. Although a more efficient addition law might exist for larger bidegrees, it is worth reporting that our experiments to find higher bidegree analogues of the Bosma and Lenstra formulas suggest that this, too, is unlikely. The complexity (and

[9] Our experimentation did suggest that computing (1) in any reasonable way with fewer than 12 generic multiplications appears to be difficult.

computational cost) of the explicit formulas grows rapidly as the bidegree increases, which is most commonly the case across all models of elliptic curves and projective embeddings (c. f. [His10]). We could hope for an addition law of bidegree lower than $(2, 2)$, but in [BL95, §3] Bosma and Lenstra prove that this is not possible under the short Weierstrass embedding[10] of E.

3.2 Jacobian Coordinates

Since first suggested for short Weierstrass curves by Miller in his seminal paper [Mil86, p. 424], Jacobian coordinates have proven to offer significant performance advantages over other coordinate systems. Given their ubiquity in real-world ECC code, and the fact that their most commonly used sets of efficient point doubling formulas turn out to be exception-free on prime order curves (see Table 1), it is highly desirable to go searching for a Jacobian coordinate analogue of the Bosma–Lenstra (homogeneous coordinates) addition law. Unfortunately, we now show that such addition formulas in Jacobian coordinates must have a higher bidegree, intuitively making them slower to compute.

For the remainder of this section only, let $E(\mathbb{F}_q) \subset \mathbb{P}(2, 3, 1)(\mathbb{F}_q)$ have odd order. If an addition law $f = (f_X : f_Y : f_Z)$ has f_Z of bidegree (μ, ν), then the bidegrees of f_X and f_Y are $(2\mu, 2\nu)$ and $(3\mu, 3\nu)$, respectively. Below we show that any complete formulas must have $\mu, \nu \geq 3$.

Consider the addition of two points $P = (X_1 : Y_1 : Z_1)$ and $Q = (X_2 : Y_2 : Z_2)$, using the addition law

$$f(P, Q) = (f_X(P, Q) : f_Y(P, Q) : f_Z(P, Q)),$$

with f_Z of bidegree (μ, ν). Suppose that f is complete, and that $\mu < 3$. Then f_Z, viewed as a polynomial in X_1, Y_1, Z_1, has degree $\mu < 3$, and in particular cannot contain Y_1. Now, since $-P = (X_1 : -Y_1 : Z_1)$ on E, it follows that $f_Z(P, Q) = f_Z(-P, Q)$ for all possible Q, and in particular when $Q = P$. But given that P cannot have order 2, we have $f_Z(P, P) \neq 0$ and $f_Z(-P, P) = 0$, a contradiction. We conclude that $\mu \geq 3$, and (by symmetry) that $\nu \geq 3$. It follows that f_X and f_Y have bidegrees at least $(6, 6)$ and $(9, 9)$, respectively, which destroys any hope of comparable efficiency to the homogeneous Bosma–Lenstra formulas.

[10] Lower bidegree addition laws are possible for other embeddings (i. e. models) of E in the case where E has a k-rational torsion structure – see [Koh11].

4 Using These Formulas in Practice

In this section we discuss the practical application of the complete formulas in this chapter. We discuss how they can be used for both the prime order curves (§4.1) and composite order curves (§4.2) in the standards. In §4.3, we give performance numbers that shed light on the expected cost of completeness in certain software scenarios, before discussing why this cost is likely to be significantly reduced in many other scenarios, e. g. in hardware.

4.1 Application to Prime Order Curves

Using the ADD function in Algorithm 1 as a black-box point addition routine, non-experts now have a straightforward way to implement the standardized prime order elliptic curves. So long as scalars are *recoded* correctly, the subsequent scalar multiplication routine will always compute the correct result.

Given the vulnerabilities exposed in already deployed ECC implementations (see §1), we now provide some implementation recommendations, e. g. for an implementer whose task it is to (re)write a simple and timing-resistant scalar multiplication routine for prime order curves from scratch. The main point is that branches (e. g. `if` statements) inside the elliptic curve point addition algorithms can now be avoided entirely. Our main recommendation is that more streamlined versions of the ADD function should only be introduced to an implementation if they are guaranteed to be exception-free; subsequently, we stress that branching should never be introduced into any point addition algorithms.

Assuming access to branch-free, constant-time field arithmetic in $\mathbb{F}_q$, a first step is to implement the ADD in Algorithm 1 to be used for *all* point (doubling and addition) operations, working entirely in homogeneous projective space. The natural next step is to implement a basic scalar recoding (e. g. [OT03; JT09]) that gives rise to a fixed, uniform, scalar-independent main loop. This typically means that the main loop repeats the same pattern of a fixed number of doublings followed by a single table lookup/extraction and, subsequently, an addition. The important points are that this table lookup must be done in a cache-timing resistant manner (c. f. [Käs12, §3.4]), and that the basic scalar recoding must itself be performed in a uniform manner.

Once the above routine is running correctly, an implementer that is seeking further performance gains can start by viewing stages of the routine where ADD can safely be replaced by its specialized, more efficient variants. If the code is intended to support only short Weierstrass curves with either $a = -3$ or $a = 0$, then Algorithm 1

should be replaced by (the faster and more compact) Algorithm 2 or Algorithm 3, respectively. If the performance gains warrant the additional code, then at all stages where the addition function is called to add a point to itself (i. e. the point doubling stages), the respective exception-free point doubling routine(s) DBL in Algorithms 1–3 should be implemented and called there instead.

Incomplete short Weierstrass addition routines (e. g. the prior works summarized in Table 1) should only be introduced for further performance gains if the implementer can guarantee that exceptional pairs of points can never be input into the algorithms, and subsequently can implement them without introducing any branches. For example, Bos et al. [Bos+16, §4.1] proved that, under their particular choice of scalar multiplication algorithm, all-but-one of the point additions in a variable-base scalar multiplication can be performed without exception using an incomplete addition function. The high-level argument used there was that such additions almost always took place between elements of the lookup table and a running value that had just been output from a point doubling, the former being small odd multiples of the input point (e. g. P, $[3]P$, $[5]P$, etc.) and the latter being some even multiple. Subsequently, they showed that the only possible time when the input points to the addition algorithm could coincide with (or be inverses of) each other is in the final addition, ruling out the exceptional points in all prior additions. On the other hand, as we mentioned in §1 and as was encountered in [Bos+16, §4.1], it can be significantly more complicated to rule out exceptional input points in more exotic scalar multiplication scenarios like fixed-base scalar multiplications, multiscalar multiplications, or those that exploit endomorphisms. In those cases, it could be that the only option to rule out any exceptional points is to *always* call complete addition algorithms.

Remark 3 (The best of both worlds?). We conclude this subsection by mentioning one more option that may be of interest to implementers who want to combine the fastest complete point addition algorithms with the fastest exception-free point doubling algorithms. Recall from Table 1 that the fastest doubling algorithms for short Weierstrass curves work in Jacobian coordinates and happen to be exception-free in the prime order setting, but recall from §3.2 that there is little hope of obtaining relatively efficient complete addition algorithms in Jacobian coordinates. This prompts the question as to whether the doubling algorithms that take place in $\mathbb{P}(2,3,1)(k)$ can be combined with our complete addition algorithms that take place in $\mathbb{P}^2(k)$. Generically, we can map the elliptic curve point $(X : Y : Z) \in \mathbb{P}(2,3,1)(k)$ to $(XZ : Y : Z^3) \in \mathbb{P}^2(k)$, and conversely, we can map the point $(X : Y : Z) \in \mathbb{P}^2(k)$ to $(XZ : YZ^2 : Z) \in \mathbb{P}(2,3,1)(k)$; both maps cost $2\mathbf{M} + 1\mathbf{S}$. We note that in the

first direction there are no exceptions: in particular, the point at infinity $(1:1:0) \in \mathbb{P}(2,3,1)(k)$ correctly maps to $(0:1:0) \in \mathbb{P}^2(k)$. However, in the other direction, the point at infinity $(0:1:0) \in \mathbb{P}^2(k)$ does not correctly map to $(1:1:0) \in \mathbb{P}(2,3,1)(k)$, but rather to the point $(0:0:0) \notin \mathbb{P}(2,3,1)(k)$.

For a variable-base scalar multiplication using a fixed window of width w, one option would be to store the precomputed lookup table in $\mathbb{P}^2(k)$ (or in $\mathbb{A}^2(k)$ if normalizing for the sake of complete mixed additions is preferred), and to compute the main loop as follows. After computing each of the w consecutive doublings in $\mathbb{P}(2,3,1)(k)$, the running value is converted to $\mathbb{P}^2(k)$ at a cost of $2\mathbf{M} + 1\mathbf{S}$, then the result of a complete addition (between the running value and a lookup table element) is converted back to $\mathbb{P}(2,3,1)(k)$ at a cost of $2\mathbf{M} + 1\mathbf{S}$. Even for small window sizes that result in additions (and thus the back-and-forth conversions) occurring relatively often, the operation counts in Table 1 suggest that this trade-off will be favorable; and, for larger window sizes, the resulting scalar multiplication will be significantly faster than one that works entirely in $\mathbb{P}^2(k)$.

The only possible exception that could occur in the above routine is when the result of an addition is the point at infinity $(0:1:0) \in \mathbb{P}^2(k)$, since the conversion back to $\mathbb{P}(2,3,1)(k)$ fails here. Thus, this strategy should only be used if the scalar multiplication routine is such that the running value is never the inverse of any element in the lookup table, or if the conversion from $\mathbb{P}^2(k)$ to $\mathbb{P}(2,3,1)(k)$ is written to handle this possible exception in a constant-time fashion. In the former case, if (as in [Bos+16, §4.1]) this can only happen in the final addition, then the workaround is easy: either guarantee that the scalars cannot be a multiple of the group order (which rules out this possibility), or else do not apply the conversion back to $\mathbb{P}(2,3,1)(k)$ after the final addition.

4.2 Interoperability With Composite Order Curves

The IRTF CFRG recently selected two composite order curves as a recommendation to the TLS working group for inclusion in upcoming versions of TLS: Bernstein's Curve25519 [Ber06a] and Hamburg's Goldilocks [Ham15b]. The current IETF internet draft[11] specifies the wire format for these curves to be the u-coordinate corresponding to a point (u, v) on the Montgomery model of these curves $E_{\mathrm{M}}/\mathbb{F}_q \colon v^2 = u^3 + Au^2 + u$. Curve25519 has $q = 2^{255} - 19$ with $A = 486662$ and Goldilocks has $q = 2^{448} - 2^{224} - 1$ with $A = 156326$. Since our complete formulas are likely to be of interest to practitioners concerned with global interoperability, e. g. those invest-

[11] See `https://datatracker.ietf.org/doc/draft-irtf-cfrg-curves/`.

ing a significant budget into one implementation that may be intended to support as many standardized curves as possible, we now show that ADD in Algorithm 1 can be adapted to interoperate with the composite order curves in upcoming TLS ciphersuites. We make no attempt to disguise the fact that this will come with a significant performance penalty over the Montgomery ladder, but in this case we are assuming that top performance is not the priority.

A trivial map $E_M \to E$ from the Montgomery curve to a short Weierstrass curve is $(u, v) \mapsto (x, y) = (u + A/3, v)$; here the short Weierstrass curve is $E\colon y^2 = x^3 + ax + b$, with $a = 1 - A^2/3$ and $b = A(2A^2 - 9)/27$. Thus, a dedicated short Weierstrass implementation can interoperate with Curve25519 (resp. Goldilocks) as follows. After receiving the u-coordinate on the wire, set $x = u + A/3$ (i. e. add a fixed, global constant), and decompress to compute the corresponding y-coordinate on E via the square root $y = \sqrt{x^3 + ax + b}$ as usual; the choice of square root here does not matter. Setting $P = (x, y)$ and validating that $P \in E$, we can then call ADD to compute 3 (resp. 2) successive doublings to get Q. This is in accordance with the scalars being defined with 3 (resp. 2) fixed zero bits to clear the cofactor [Ber06a]. The point Q is then multiplied by the secret part of the scalar (using, e. g. the methods we just described in §4.1), then normalized to give $Q = (x', y')$, and the Montgomery u-coordinate of the result is output as $u' = x' - A/3$.

Note that the above routine is exception free: ADD only fails to add the points P_1 and P_2 when $P_1 - P_2$ is a point of exact order 2. Thus, it can be used for point doublings on all short Weierstrass curves (including those of even order). Furthermore, the point Q is in the prime order subgroup, so the subsequent scalar multiplication (which only encounters multiples of Q) cannot find a pair of points that are exceptional to ADD.

Finally, we note that although neither Curve25519 or Goldilocks are isomorphic to a Weierstrass curve with $a = -3$, both curves have simple isomorphisms to Weierstrass curves with small a values, e. g. $a = 2$ and $a = 1$, respectively. Making use of this would noticeably decrease the overhead of our complete formulas.

4.3 An OpenSSL Implementation

In Table 5 we report the factor slowdown obtained when substituting the MADD and DBL functions in Algorithm 2 for OpenSSL's MADD (`ec_GFp_simple_add`) and DBL (`ec_GFp_simple_dbl`) functions inside the OpenSSL scalar multiplication routine for the five NIST prime curves (which all have $a = -3$). We intentionally left OpenSSL's scalar multiplication routines unaltered in order to provide an unbiased

Table 5. Number of ECDH operations in 10 seconds for the OpenSSL implementation of the five NIST prime curves, using complete and incomplete addition formulas. Timings were obtained by running the "`openssl speed ecdhpXXX`" command on an Intel Core i5-5300 CPU @ 2.30GHz, averaged over 100 trials of 10s each.

NIST curve	No. of ECDH operations (per 10 s)		Factor slowdown
	Complete	Incomplete	
P-192	35 274	47 431	1.34×
P-224	24 810	34 313	1.38×
P-256	21 853	30 158	1.38×
P-384	10 109	14 252	1.41×
P-521	4 580	6 634	1.44×

upper bound on the performance penalty that our complete algorithms will introduce. For the remainder of this subsection, we discuss why the performance difference is unlikely to be this large in many practical scenarios.

Referring to Table 6 (which, as well as the counts given in Table 1, includes the operation counts for mixed additions), we see that the mixed addition formulas in Jacobian coordinates (i. e. [CMO98; BLb; HLX12]) are $4\mathbf{M} + 1\mathbf{S}$ faster than full additions, while for our complete formulas the difference is only $1\mathbf{M} + 6\mathbf{a}$. Thus, in Jacobian coordinates, it is often advantageous to normalize the lookup table (using one shared inversion [Mon87]) in order to save $4\mathbf{M} + 1\mathbf{S}$ per addition. On the other hand, in the case of the complete formulas, this will not be a favorable trade-off and (assuming there is ample cache space) it is likely to be better to leave all of the lookup elements in $\mathbb{P}^2$. The numbers reported in Table 5 use OpenSSL's scalar multiplication which does normalize the lookup table to use mixed additions, putting the complete formulas at a disadvantage.

As we mentioned in §1, the slowdowns reported in Table 5 (which were obtained on a 64-bit machine) are likely to be significantly less on low-end architectures where the relative cost of field additions drops. Furthermore, in embedded scenarios where implementations must be protected against more than just timing attacks, a common countermeasure is to randomize the projective coordinates of intermediate points [Cor99]. In these cases, normalized lookup table elements could also give rise to side-channel vulnerabilities [FV12, §3.4–3.6], which would take mixed additions out of the equation. As Table 6 suggests, when full additions are used throughout, our complete algorithms will give much better performance relative to their incomplete counterparts.

We remark that runtime is not the only metric of concern to ECC practitioners;

Table 6. Operation counts for the prior incomplete addition algorithms and our complete ones, with the inclusion of mixed addition formulas. Credits for the incomplete formulas are the same as in Table 1, except for the additional mixed formulas which are, in homogeneous coordinates, due to Cohen, Miyaji and Ono [CMO98], and in Jacobian coordinates, due to Hankerson, Menezes and Vanstone [HMV06, p. 91].

a	Ref.	ADD					MADD					DBL				
		M	**S**	$\mathbf{m_a}$	$\mathbf{m_b}$	**a**	**M**	**S**	$\mathbf{m_a}$	$\mathbf{m_b}$	**a**	**M**	**S**	$\mathbf{m_a}$	$\mathbf{m_b}$	**a**
any	**This**	12	0	3	2	23	11	0	3	2	17	8	3	3	2	15
	[CMO98; BLb]	12	4	0	0	7	8	3	0	0	7	3	6	1	0	13
	[CMO98]	12	2	0	0	7	9	2	0	0	7	5	6	1	0	12
-3	**This**	12	0	0	2	29	11	0	0	2	23	8	3	0	2	21
	[CMO98; BLb]	12	4	0	0	7	8	3	0	0	7	4	4	0	0	8
	[CMO98; LG10]	12	2	0	0	7	9	2	0	0	7	7	3	0	0	11
0	**This**	12	0	0	2	19	11	0	0	2	13	6	2	0	1	9
	[CMO98; HLX12]	12	4	0	0	7	8	3	0	0	7	3	4	0	0	7

in fact, there was wide consensus (among both speakers and panelists) at the recent NIST workshop[12] that security and simplicity are far more important in real-world ECC than raw performance. While our complete algorithms are likely to be slower in some scenarios, we reiterate that complete formulas reign supreme in all other aspects, including total code size, ease of implementation, and issues relating to side-channel resistance.

5 Hardware Implementations

In the setting of hardware implementations, different assumptions on the cost of operations should be made. That is, hardware implementations of ECC typically rely on using general field hardware multipliers that are often based on the algorithm of Montgomery [Mon85]. These types of hardware modules use a multiplier for both multiplications and squarings [CBC07; GP08], meaning that the squarings our addition algorithms save (over the prior formulas) are full multiplications. Moreover, hardware architectures that are based on Montgomery multiplication can benefit from modular additions/subtractions computed as non-modular operations. The concept is explained in [BBMÖ04], which is a typical ECC hardware architecture using the "relaxed" Montgomery parameter such that the conditional subtraction (from the original algorithm of Montgomery) can be omitted. In this way, the modu-

[12] See `https://www.nist.gov/itl/csd/ct/ecc-workshop.cfm`.

Table 7. Dependencies of multiplications inside the complete addition formulas.

Stage	Result	Multiplication	Dependent on
0	ℓ_0	$X_1 \cdot X_2$	—
0	ℓ_1	$Y_1 \cdot Y_2$	—
0	ℓ_2	$Z_1 \cdot Z_2$	—
0	ℓ_3	$(X_1 + Y_1) \cdot (X_2 + Y_2)$	—
0	ℓ_4	$(X_1 + Z_1) \cdot (X_2 + Z_2)$	—
0	ℓ_5	$(Y_1 + Z_1) \cdot (Y_2 + Z_2)$	—
1	ℓ_6	$b_3 \cdot \ell_2$	ℓ_2
1	ℓ_7	$a \cdot \ell_2$	ℓ_2
1	ℓ_8	$a \cdot (\ell_4 - \ell_0 - \ell_2)$	ℓ_0, ℓ_2, ℓ_4
1	ℓ_9	$b_3 \cdot (\ell_4 - \ell_0 - \ell_2)$	ℓ_0, ℓ_2, ℓ_4
2	ℓ_{10}	$a \cdot (\ell_0 - \ell_7)$	ℓ_0, ℓ_7
2	ℓ_{11}	$(\ell_3 - \ell_0 - \ell_1) \cdot (\ell_1 - \ell_8 - \ell_6)$	$\ell_0, \ell_1, \ell_3, \ell_6, \ell_8$
2	ℓ_{13}	$(\ell_1 + \ell_8 + \ell_6) \cdot (\ell_1 - \ell_8 - \ell_6)$	ℓ_1, ℓ_6, ℓ_8
2	ℓ_{15}	$(\ell_5 - \ell_1 - \ell_2) \cdot (\ell_1 + \ell_8 + \ell_6)$	$\ell_1, \ell_2, \ell_5, \ell_6, \ell_8$
2	ℓ_{16}	$(\ell_3 - \ell_0 - \ell_1) \cdot (3\ell_0 + \ell_7)$	$\ell_0, \ell_1, \ell_3, \ell_7$
3	ℓ_{12}	$(\ell_5 - \ell_1 - \ell_2) \cdot (\ell_{10} + \ell_9)$	$\ell_1, \ell_2, \ell_5, \ell_9, \ell_{10}$
3	ℓ_{14}	$(3\ell_0 + \ell_7) \cdot (\ell_{10} + \ell_9)$	$\ell_0, \ell_7, \ell_9, \ell_{10}$

lar addition/subtraction is implemented not just very efficiently, but also as a time-constant operation. Using this approach implies the only cost to be taken into account is the one of modular multiplication, i. e. modular additions come almost for free. Similar conclusions apply for multiplications with a constant as they can be implemented very efficiently in hardware, assuming a constant is predefined and hence "hardwired".

A second benefit of implementing the algorithms in hardware is that one can efficiently run multiple (say n) processors in parallel. Here we focus on the ADD function in Algorithm 1, but an analogous analysis can be performed when $a = -3$

Table 8. Efficiency approximation of the number of Montgomery multipliers against the area used.

n	Cost	$n \times$ Cost	Ref.
1	$17\mathbf{M} + 23\mathbf{a}$	$17\mathbf{M} + 23\mathbf{a}$	Alg. 1
2	$9\mathbf{M}_2 + 12\mathbf{a}_2$	$18\mathbf{M} + 24\mathbf{a}$	Appendix A
3	$6\mathbf{M}_3 + 8\mathbf{a}_3$	$18\mathbf{M} + 24\mathbf{a}$	Appendix A
4	$5\mathbf{M}_4 + 7\mathbf{a}_4$	$20\mathbf{M} + 28\mathbf{a}$	Appendix A
5	$4\mathbf{M}_5 + 6\mathbf{a}_5$	$20\mathbf{M} + 30\mathbf{a}$	Appendix A
6	$3\mathbf{M}_6 + 6\mathbf{a}_6$	$18\mathbf{M} + 36\mathbf{a}$	Appendix A

or $a = 0$. As explained in the previous paragraph, we assume the cost of this function to be 17**M**. These multiplications are of course not independent, but we can easily work out their interdependencies (see Table 7). Using this table we can write down algorithms for implementations running n processors in parallel. Denote by $\mathbf{M}_n$ resp. $\mathbf{a}_n$ the cost of doing n multiplications resp. additions (or subtractions) in parallel. In Table 8 we present the costs for $1 \leq n \leq 6$ (for $n = 6$ we assume to have the constant a^2 precomputed). Note that typically $\mathbf{M}_n > \mathbf{M}$ and $\mathbf{a}_n > \mathbf{a}$. For example, a larger number of Montgomery multipliers can result in scheduling overhead. Verification code in Magma for all algorithms can be found in Appendix A.

A Magma Verification Code for Parallel ADD

```
function ADD_two(X1,Y1,Z1,X2,Y2,Z2,a,b3)
    t0 := X1+Y1;    t1 := X2+Y2;                    //  1
    t2 := Y1+Z1;    t3 := Y2+Z2;                    //  2
    t0 := t0*t1;    t1 := t2*t3;                    //  3
    t4 := X1*X2;    t6 := Z1*Z2;                    //  4
    t2 := X1+Z1;    t3 := X2+Z2;                    //  5
    t0 := t0-t4;    t1 := t1-t6;                    //  6
    t5 := Y1*Y2;    t2 := t2*t3;                    //  7
    t7 := a*t6;     t8 := b3*t6;                    //  8
    t9 := t4-t7;    t10 := t4+t4;                   //  9
    t11 := t4+t7;   t2 := t2-t4;                    // 10
    t0 := t0-t5;    t1 := t1-t5;                    // 11
    t2 := t2-t6;    t10 := t10+t11;                 // 12
    t9 := a*t9;     t11 := b3*t2;                   // 13
    t2 := a*t2;                                     // 14
    t9 := t9+t11;   t8 := t2+t8;                    // 15
    t6 := t5-t8;    t5 := t5+t8;                    // 16
    t3 := t1*t9;    t9 := t9*t10;                   // 17
    t10 := t0*t10;  t0 := t0*t6;                    // 18
    t6 := t5*t6;    t1 := t1*t5;                    // 19
    X3 := t0-t3;    Y3 := t6+t9;                    // 20
    Z3 := t1+t10;                                   // 21
    return X3,Y3,Z3;
end function;

function ADD_three(X1,Y1,Z1,X2,Y2,Z2,a,b3);
    t0 := X1*X2;    t1 := Y1*Y2;    t2 := Z1*Z2;    //  1
    t3 := X1+Y1;    t4 := X2+Y2;    t5 := Y1+Z1;    //  2
    t6 := Y2+Z2;    t7 := X1+Z1;    t8 := X2+Z2;    //  3
    t9 := t3*t4;    t10 := t5*t6;   t11 := t7*t8;   //  4
    t3 := t0+t1;    t4 := t1+t2;    t5 := t0+t2;    //  5
    t6 := b3*t2;    t8 := a*t2;                     //  6
    t2 := t9-t3;    t9 := t0+t0;    t3 := t10-t4;   //  7
    t10 := t9+t0;   t4 := t11-t5;   t7 := t0-t8;    //  8
```

```
    t0 := a*t4;     t5 := b3*t4;    t9 := a*t7;     //  9
    t4 := t0+t6;    t7 := t5+t9;    t0 := t8+t10;   // 10
    t5 := t1-t4;    t6 := t1+t4;                    // 11
    t1 := t5*t6;    t4 := t0*t7;    t8 := t3*t7;    // 12
    t9 := t2*t5;    t10 := t3*t6;   t11 := t0*t2;   // 13
    X3 := t9-t8;    Y3 := t1+t4;    Z3 := t10+t11;  // 14
    return X3,Y3,Z3;
end function;

function ADD_four(X1,Y1,Z1 ,X2 ,Y2 ,Z2,a,b3);
    t0 := X1+Y1;    t1 := X2+Y2;
    t2 := Y1+Z1;    t3 := Y2+Z2;                    //  1
    t0 := t0*t1;    t1 := t2*t3;
    t4 := X1*X2;    t6 := Z1*Z2;                    //  2
    t2 := X1+Z1;    t3 := X2+Z2;
    t0 := t0-t4;    t1 := t1-t6;                    //  3
    t5 := Y1*Y2;    t2 := t2*t3;
    t7 := a*t6;;    t8 := b3*t6;                    //  4
    t9 := t4-t7;    t10 := t4+t4;
    t11 := t4+t7;   t2 := t2-t4;                    //  5
    t0 := t0-t5;    t1 := t1-t5;
    t2 := t2-t6;    t10 := t10+t11;                 //  6
    t9 := a*t9;     t11 := b3*t2;   t2 := a*t2;     //  7
    t9 := t9+t11;                                   //  8
    t3 := t1*t9;    t9 := t9*t10;
    t10 := t0*t10;  t8 := t2+t8;                    //  9
    t6 := t5-t8;    t5 := t5+t8;                    // 10
    t0 := t0*t6;    t6 := t5*t6;    t1 := t1*t5;    // 11
    X3 := t0-t3;    Y3 := t6+t9;    Z3 := t1+t10;   // 12
    return X3,Y3,Z3;
end function;

function ADD_five(X1,Y1,Z1,X2,Y2,Z2,a,b3);
    t5 := X1+Y1;    t6 := X2+Y2;    t7 := X1+Z1;
    t8 := X2+Z2;    t9 := Y1+Z1;                    //  1
    t0 := X1*X2;    t1 := Y1*Y2;    t2 := Z1*Z2;
    t3 := t5*t6;    t4 := t7*t8;                    //  2
```

```
    t10 := Y2+Z2;   t3 := t3-t0;    t4 := t4-t0;
    t11 := t0+t0;                                       //  3
    t3 := t3-t1;    t4 := t4-t2;    t11 := t11+t0;      //  4
    t5 := t9*t10;   t6 := b3*t2;    t7 := a*t2;
    t8 := a*t4;     t9 := b3*t4;                        //  5
    t5 := t5-t1;    t11 := t11+t7;  t4 := t0-t7;
    t10 := t6+t8;                                       //  6
    t0 := a*t4;     t6 := t3*t11;                       //  7
    t0 := t0+t9;    t7 := t1-t10;   t10 := t1+t10;
    t5 := t5-t2;                                        //  8
    t1 := t3*t7;    t2 := t5*t0;    t4 := t10*t7;
    t8 := t11*t0;   t9 := t5*t10;                       //  9
    X3 := t1-t2;    Y3 := t4+t8;    Z3 := t9+t6;        // 10
    return X3,Y3,Z3;
end function;

function ADD_six(X1,Y1,Z1,X2,Y2,Z2,a,b3)
    t0 := X1+Y1;    t1 := X2+Y2;    t2 := Y1+Z1;
    t3 := Y2+Z2;    t4 := X1+Z1;    t5 := X2+Z2;        //  1
    t0 := t0*t1;    t1 := t2*t3;    t2 := t4*t5;
    t3 := X1*X2;    t4 := Y1*Y2;    t5 := Z1*Z2;        //  2
    t0 := t0-t3;    t1 := t1-t4;    t2 := t2-t5;        //  3
    t0 := t0-t4;    t1 := t1-t5;    t2 := t2-t3;        //  4
    t6 := b3*t5;    t7 := a*t5;     t8 := a*t2;
    t9 := b3*t2;    t10 := a*t3;    t11 := a^2*t5;      //  5
    t6 := t6+t8;    t7 := t3+t7;    t8 := t3+t3;
    t9 := t9+t10;                                       //  6
    t9 := t9-t11;   t8 := t8+t7;    t7 := t4-t6;
    t6 := t4+t6;                                        //  7
    t3 := t0*t7;    t4 := t0*t8;    t5 := t1*t9;
    t8 := t8*t9;    t7 := t6*t7;    t6 := t1*t6;        //  8
    X3 := t3-t5;    Y3 := t7+t8;    Z3 := t6+t4;        //  9
    return X3,Y3,Z3;
end function;
```

Chapter IV

μKummer: Efficient Hyperelliptic Signatures and Key Exchange on Microcontrollers

We describe the design and implementation of efficient signature and key-exchange schemes for the AVR ATmega and ARM Cortex M0 microcontrollers, targeting the 128-bit security level. Our algorithms are based on an efficient Montgomery ladder scalar multiplication on the Kummer surface of Gaudry and Schost's genus-2 hyperelliptic curve [GS12], combined with the Jacobian point recovery technique of Chung, Costello, and Smith [CCS17]. Our results are the first to show the feasibility of software-only hyperelliptic cryptography on constrained platforms, and represent a significant improvement on the elliptic-curve state-of-the-art for both key exchange and signatures on these architectures. Notably, our key-exchange scalar-multiplication software runs in under 9520k cycles on the ATmega and under 2640k cycles on the Cortex M0, improving on the current speed records by 32% and 75% respectively.

1 Introduction

The current state of the art in asymmetric cryptography, not only on microcontrollers, is elliptic-curve cryptography; the most widely accepted reasonable security is the 128-bit security level. All current speed records for 128-bit secure key

exchange and signatures on microcontrollers are held — until now — by elliptic-curve-based schemes. Outside the world of microcontrollers, it is well known that genus-2 hyperelliptic curves and their Kummer surfaces present an attractive alternative to elliptic curves [Ber06c; Bos+13]. For example, at Asiacrypt 2014 Bernstein, Chuengsatiansup, Lange and Schwabe [Ber+14] presented speed records for timing-attack-protected 128-bit-secure scalar multiplication on a range of architectures with Kummer-based software. These speed records are currently only being surpassed by the elliptic-curve-based FourQ software by Costello and Longa [CL15] presented at Asiacrypt 2015, which makes heavy use of efficiently computable endomorphisms (i. e. of additional structure of the underlying elliptic curve). The Kummer-based speed records in [Ber+14] were achieved by exploiting the computational power of vector units of recent "large" processors such as Intel Sandy Bridge, Ivy Bridge, and Haswell, or the ARM Cortex-A8. Surprisingly, very little attention has been given to Kummer surfaces on embedded processors. Indeed, this is the first work showing the feasibility of software-only implementations of hyperelliptic-curve based crypto on constrained platforms. There have been some investigations of binary hyperelliptic curves targeting the much lower 80-bit security level, but those are actually examples of software-hardware co-design showing that using hardware acceleration for field operations was necessary to get reasonable performance figures (see e. g. [Bat+05; Hod+07]).

In this chapter we investigate the potential of genus-2 hyperelliptic curves for both key exchange and signatures on the "classical" 8-bit AVR ATmega architecture, and the more modern 32-bit ARM Cortex M0 processor. The former has the most previous results to compare to, while ARM is becoming more relevant in real-world applications. We show that not only are hyperelliptic curves competitive, they clearly outperform state-of-the art elliptic-curve schemes in terms of speed and size. For example, our variable-basepoint scalar multiplication on a 127-bit Kummer surface is 31% faster on AVR and 26% faster on the M0 than the recently presented speed records for Curve25519 software by Düll, Haase, Hinterwälder, Hutter, Paar, Sánchez, and Schwabe [Dül+15]; our implementation is also smaller, and requires less RAM.

We use a recent result by Chung, Costello, and Smith [CCS17] to also set new speed records for 128-bit secure signatures. Specifically, we present a new signature scheme based on fast Kummer surface arithmetic. It is inspired by the EdDSA construction by Bernstein, Duif, Lange, Schwabe, and Yang [Ber+12]. On the ATmega, it produces shorter signatures, achieves higher speeds and needs less RAM than the Ed25519 implementation presented in [NLD15].

Table 1. Cycle counts and stack usage in bytes of all functions related to the signature and key exchange schemes, for the AVR ATmega and ARM Cortex M0 microcontrollers.

Func.	AVR		ARM	
	Cycles	Stack	Cycles	Stack
KEYGEN	10 206 181	812 B	2 774 087	1 056 B
SIGN	10 404 033	926 B	2 865 351	1 360 B
VERIFY	16 240 510	992 B	4 453 978	1 432 B
DH_EXCHANGE	9 739 059	429 B	2 644 604	584 B

Our routines handling secret data are constant-time, and are thus naturally resistant to timing attacks. These algorithms are built around the Montgomery ladder, which improves resistance against simple power analysis (SPA) attacks. Resistance to DPA attacks can easily be added to the implementation by randomizing the scalar and/or Jacobian points. Re-randomizing the latter after each ladder step would also guarantee resistance against horizontal types of attacks.

Organization. We begin by describing the details of our signature and key exchange schemes, explaining the choices we made in their design. Concrete implementation details appear in §3 and §4 below. Experimental results and comparisons follow in §5.

2 High-level Overview

2.1 Signatures

Our signature scheme, defined at the end of this section, adheres closely to the proposal of [CCS15, §8], which in turn is a type of Schnorr signature [Sch90]. There are however some differences and trade-offs, which we discuss below.

Group structure. We build the signature scheme on top of the group structure from the Jacobian $\mathcal{J}_{\mathcal{C}}(\mathbb{F}_p)$ of a genus-2 hyperelliptic curve $\mathcal{C}$. More specifically, $\mathcal{C}$ is the Gaudry–Schost curve over the prime field $\mathbb{F}_p$ with $p = 2^{127} - 1$ (c. f. §3.2). The Jacobian is a group of order $\#\mathcal{J}_{\mathcal{C}}(\mathbb{F}_p) = 2^4 N$, where

$$N = 2^{250} - \texttt{0x334D69820C75294D2C27FC9F9A154FF47730B4B840C05BD}$$

is a 250-bit prime. For more details on the Jacobian and its elements, see §3.3.

Hash function. We may use any hash function H with a 128-bit security level. For our purposes, $H(M) = \text{SHAKE128}(M, 512)$ suffices [Dwo15]. Although SHAKE128 has variable-length output, we only use the 512-bit output implementation.

Encoding. At the highest level, we operate on points Q in $\mathcal{J}_{\mathcal{C}}(\mathbb{F}_p)$. To minimize communication costs, we compress the usual 508-bit representation of Q into a 256-bit encoding $\underline{Q}$ (see §3.3). (This notation is the same as in [Ber+12].)

Public generator. The public generator can be any element P of $\mathcal{J}_{\mathcal{C}}(\mathbb{F}_p)$ such that $[N]P = 0$. In our implementation we have made the arbitrary choice $P = (X^2 + u_1X + u_0, v_1X + v_0)$, where

$$\begin{aligned} u_1 &= \texttt{0x7D5D9C3307E959BF27B8C76211D35E8A}, \\ u_0 &= \texttt{0x2703150F9C594E0CA7E8302F93079CE8}, \\ v_1 &= \texttt{0x444569AF177A9C1C721736D8F288C942}, \\ v_0 &= \texttt{0x7F26CFB225F42417316836CFF8AEFB11}. \end{aligned}$$

This is the point which we use the most for scalar multiplication. Since it remains fixed, we assume we have its decompressed representation precomputed, so as to avoid having to perform the relatively expensive decompression operation whenever we need a scalar multiplication; this gives a low-cost speed gain. We further assume we have a "wrapped" representation of the projection of P to the Kummer surface, which is used to speed up the XDBLADD function. See §4.1 for more details on the XWRAP function.

Public keys. In contrast to the public generator, we assume public keys are compressed: they are communicated much more frequently, and we therefore benefit much more from smaller keys. Moreover, we include the public key in one of the hashes during the SIGN operation [KW03; MRa99], computing $h = H(\underline{R} \mid\mid \underline{Q} \mid\mid M)$ instead of the $h = H(\underline{R} \mid\mid M)$ originally suggested by Schnorr [Sch90]. This protects against adversaries attacking multiple public keys simultaneously.

Compressed signatures. Schnorr [Sch90] mentions the option of compressing signatures by hashing one of their two components: the hash size only needs to be $b/2$ bits, where b is the key length. Following this suggestion, our signatures are 384-bit values of the form $(h_{128} \mid\mid s)$, where h_{128} means the lowest 128 bits of $h = H(\underline{R} \mid\mid \underline{Q} \mid\mid M)$, and s is a 256-bit scalar. The most obvious upside is that signatures are

smaller, reducing communication overhead. Another big advantage is that we can exploit the half-size scalar to speed up signature verification. On the other hand, we lose the possibility of efficient batch verification.

Verification efficiency. The most costly operation in signature verification is the two-dimensional scalar multiplication $T = [s]P + [h_{128}]Q$. In [CCS17], the authors propose an algorithm relying on the differential addition chains presented in [Ber06b]. However, since we are using compressed signatures, we have a small scalar h_{128}. Unfortunately the two-dimensional algorithm in [CCS17] cannot directly exploit this fact, therefore not obtaining much benefit from the compressed signature. On the other hand, we can simply compute $[s]P$ and $[h_{128}]Q$ separately using the fast scalar multiplication on the Kummer surface and finally add them together on the Jacobian. Here $[s]P$ is a 256-bit scalar multiplication, whereas $[h_{128}]Q$ is only a 128-bit scalar multiplication. Not only do we need fewer cycles compared to the two-dimensional routine, but we also reduce code size by reusing the one-dimensional scalar multiplication routine.

The scheme. We now define our signature scheme, taking the above into account.

Key generation (KEYGEN). Let d be a 256-bit secret key, and P the public generator. Compute $(d' \mid\mid d'') \leftarrow H(d)$ (with d' and d'' both 256 bits), then $Q \leftarrow [16d']P$. The public key is $\underline{Q}$.

Signing (SIGN). Let M be a message, d a 256-bit secret key, P the public generator, and $\underline{Q}$ a compressed public key. Compute $(d' \mid\mid d'') \leftarrow H(d)$ (with d' and d'' both 256 bits), then $r \leftarrow H(d'' \mid\mid M)$, then $R \leftarrow [r]P$, then $h \leftarrow H(\underline{R} \mid\mid \underline{Q} \mid\mid M)$, and finally $s \leftarrow (r - 16h_{128}d') \bmod N$. The signature is $(h_{128} \mid\mid s)$.

Verification (VERIFY). Let M be a message with a signature $(h_{128} \mid\mid s)$ corresponding to a public key $\underline{Q}$, and let P be the public generator. Compute $T \leftarrow [s]P + [h_{128}]Q$, then $g \leftarrow H(\underline{T} \mid\mid \underline{Q} \mid\mid M)$. The signature is correct if $g_{128} = h_{128}$, and incorrect otherwise.

Remark 1. We note that there may be faster algorithms to compute the "one-and-a-half-dimensional" scalar multiplication in VERIFY, especially since we do not have to worry about being constant-time. One option might be to adapt Montgomery's PRAC [Sta03, §3.3.1] to make use of the half-size scalar. But while this may lead to a speedup, it would also cause an increase in code size compared to simply re-using the one-dimensional scalar multiplication. We have chosen not to pursue this line, preferring the solid benefits of reduced code size instead.

2.2 Diffie-Hellman Key Exchange.

For key exchange it is not necessary to have a group structure; it is enough to have a pseudo-multiplication. We can therefore carry out our the key exchange directly on the Kummer surface $\mathcal{K}_\mathcal{C} = \mathcal{J}_\mathcal{C}/\pm$, gaining efficiency by not projecting from and recovering to the Jacobian $\mathcal{J}_\mathcal{C}$. If Q is a point on $\mathcal{J}_\mathcal{C}$, then its image in $\mathcal{K}_\mathcal{C}$ is $\pm Q$. The common representation for points in $\mathcal{K}_\mathcal{C}(\mathbb{F}_p)$ is a 512-bit 4-tuple of field elements. For input points (i. e. the generator or public keys), we prefer the 384-bit "wrapped" representation (see §3.5). This not only reduces key size, but it also allows a speedup in the core XDBLADD subroutine. The wrapped representation of a point $\pm Q$ on $\mathcal{K}_\mathcal{C}$ is denoted by $\underline{\pm Q}$.

Key exchange (DH_EXCHANGE). Let d be a 256-bit secret key, and $\underline{\pm P}$ the public generator (respectively public key). Compute $\pm Q \leftarrow \pm[d]P$. The generated public key (respectively shared secret) is $\underline{\pm Q}$.

Remark 2. While it might be possible to reduce the key size even further to 256 bits, we would then have to pay the cost of compressing and decompressing, and also wrapping for XDBLADD (see the discussion in [CCS15, App. A]). We therefore choose to keep the 384-bit representation, which is consistent with [Ber+14].

3 Algorithms and Their Implementation

We begin by presenting the finite field $\mathbb{F}_{2^{127}-1}$ in §3.1. We then define the curve $\mathcal{C}$ in §3.2, before giving basic methods for the elements of $\mathcal{J}_\mathcal{C}$ in §3.3. We then present the fast Kummer $\mathcal{K}_\mathcal{C}$ and its differential addition operations in §3.4.

3.1 The Field $\mathbb{F}_p$

We work over the prime finite field $\mathbb{F}_p$, where p is the Mersenne prime $p = 2^{127} - 1$. For complete field arithmetic we implement modular reduction, addition, subtraction, multiplication, and inversion. We comment on some important aspects here, giving cycle counts in Table 2. We can represent elements of $\mathbb{F}_p$ as 127-bit values; but since the ATmega and Cortex M0 work with 8- and 32-bit words, respectively, the obvious choice is to represent field elements with 128 bits. That is, an element $g \in \mathbb{F}_p$ is represented as $g = \sum_{i=0}^{15} g_i 2^{8i}$ on the AVR ATmega platform and as $g = \sum_{i=0}^{3} \widetilde{g}_i 2^{32i}$ on the Cortex M0, where $g_i \in \{0, \ldots, 2^8 - 1\}$, $\widetilde{g}_i \in \{0, \ldots, 2^{32} - 1\}$.

Working with the prime field $\mathbb{F}_p$, we need integer reduction modulo p; this is implemented as BIGINT_RED. Reduction is very efficient because $2^{128} \equiv 2 \pmod{p}$,

which enables us to reduce using only shifts and integer additions. Given this reduction, we implement addition and subtraction operations for $\mathbb{F}_p$ (as GFE_ADD and GFE_SUB, respectively) in the obvious way.

The most costly operations in $\mathbb{F}_p$ are multiplication (GFE_MUL) and squaring (GFE_SQR), which are implemented as 128×128-bit big integer operations (named BIGINT_MUL and BIGINT_SQR) followed by a call to BIGINT_RED. Since we are working on the same platforms as [Dül+15] in which both of these operations are already highly optimized, we took the necessary code from those implementations:

- On the AVR ATmega the authors of [HS15] implement a 3-level Karatsuba multiplication of two 256-bit integers, representing elements f of $\mathbb{F}_{2^{255}-19}$ as $f = \sum_{i=0}^{31} f_i 2^{8i}$ with $f_i \in \{0, \ldots, 2^8 - 1\}$. Since the first level of Karatsuba relies on a 128×128-bit integer multiplication routine named MUL128, we simply lift this function out to form a 2-level 128×128-bit Karatsuba multiplication. Similarly, their 256×256-bit squaring relies on a 128×128-bit routine SQR128, which we can (almost) directly use. Since the 256×256-bit squaring is 2-level Karatsuba, the 128×128-bit squaring is 1-level Karatsuba.

- On the ARM Cortex M0 the authors of [Dül+15] use optimized Karatsuba multiplication and squaring. Their assembly code does not use subroutines, but fully inlines 128×128-bit multiplication and squaring. The 256×256-bit multiplication and squaring are both 3-level Karatsuba implementations. Hence, using these, we end up with 2-level 128×128-bit Karatsuba multiplication and squaring.

The function GFE_INVERT computes inversions in $\mathbb{F}_p$ as exponentiations, using the fact that $g^{-1} = g^{p-2}$ for all g in $\mathbb{F}_p^*$. To do this efficiently we use an addition chain for $p - 2$, doing the exponentiation in $10\mathbf{M} + 126\mathbf{S}$. Finally, to speed up our Jacobian point decompression algorithms, we define a function GFE_POWMINHALF which computes $g \mapsto g^{-1/2}$ for g in $\mathbb{F}_p$ (up to a choice of sign). To do this, we note that $g^{-1/2} = \pm g^{-(p+1)/4} = \pm g^{(3p-5)/4}$ in $\mathbb{F}_p$; this exponentiation can be done with an addition chain of length 136, using $11\mathbf{M} + 125\mathbf{S}$. We can then define a function GFE_SQRTINV, which given (x, y) and a bit b, computes $(\sqrt{x}, 1/y)$ as $(\pm xyz, xyz^2)$ where $z =$ GFE_POWMINHALF(xy^2), choosing the sign so that the square root has least significant bit b. Including the GFE_POWMINHALF call, this costs $15\mathbf{M} + 126\mathbf{S} + 1\mathbf{neg}$.

Table 2. Cycle counts of field arithmetic (including function-call overhead) on the AVR ATmega and ARM Cortex M0 platforms.

	AVR	ARM	Symbolic cost
BIGINT_MUL	1 654	410	
BIGINT_SQR	1 171	260	
BIGINT_RED	438	71	
GFE_MUL	1 952	502	**M**
GFE_SQR	1469	353	**S**
GFE_MULCONST	569	83	$\mathbf{m_c}$
GFE_ADD	400	62	**a**
GFE_SUB	401	66	**s**
GFE_INVERT	169 881	46 091	**I**
GFE_POWMINHALF	169 881	46 294	$11\mathbf{M} + 125\mathbf{S}$
GFE_SQRTINV	178 041	48 593	$15\mathbf{M} + 126\mathbf{S} + 1\mathbf{neg}$

3.2 The Curve $\mathcal{C}$ and Its Theta Constants

We define the curve $\mathcal{C}$ "backwards", starting from its (squared) theta constants

$$a = -11\,, \quad b = 22\,, \quad c = 19\,, \quad \text{and} \quad d = 3 \quad \text{in } \mathbb{F}_p\,.$$

From these, we define the dual theta constants

$$A = a + b + c + d = 33\,, \qquad B = a + b - c - d = -11\,,$$
$$C = a - b + c - d = -17\,, \qquad D = a - b - c + d = -49\,.$$

Observe that projectively, $(1/a : 1/b : 1/c : 1/d) = (114 : -57 : -66 : -418)$ and $(1/A : 1/B : 1/C : 1/D) = (-833 : 2499 : 1617 : 561)$. Crucially, all of these constants can be represented using just 16 bits each. Since Kummer arithmetic involves many multiplications by these constants, we implement a separate 16×128-bit multiplication function GFE_MULCONST. For the AVR ATmega, we store the constants in two 8-bit registers. For the Cortex M0, the values fit into a halfword; this works well with the 16×16-bit multiplication. Multiplication by any of these 16-bit constants costs $\mathbf{m_c}$.

Continuing, we define $e/f := (1 + \alpha)/(1 - \alpha)$, where $\alpha^2 = CD/AB$ (we take the

square root with least significant bit 0), and thus

$$\lambda := ac/bd = \texttt{0x155555555555555555555555555555552},$$
$$\mu := ce/df = \texttt{0x73E334FBB315130E05A505C31919A746},$$
$$\nu := ae/bf = \texttt{0x552AB1B63BF799716B5806482D2D21F3}.$$

These are the *Rosenhain invariants* of the curve $\mathcal{C}$, found by Gaudry and Schost [GS12], which we are (finally!) ready to define as

$$\mathcal{C} : Y^2 = f_{\mathcal{C}}(X) := X(X-1)(X-\lambda)(X-\mu)(X-\nu).$$

The curve constants are the coefficients of $f_{\mathcal{C}}(X) = \sum_{i=0}^{5} f_i X^i$; so $f_0 = 0$, $f_5 = 1$,

$$f_1 = \texttt{0x1EDD6EE48E0C2F16F537CD791E4A8D6E},$$
$$f_2 = \texttt{0x73E799E36D9FCC210C9CD1B164C39A35},$$
$$f_3 = \texttt{0x4B9E333F48B6069CC47DC236188DF6E8},$$
$$f_4 = \texttt{0x219CC3F8BB9DFE2B39AD9E9F6463E172}.$$

We store the squared theta constants $(a : b : c : d)$, along with $(1/a : 1/b : 1/c : 1/d)$ and $(1/A : 1/B : 1/C : 1/D)$; the Rosenhain invariants λ, μ, and ν, together with $\lambda\mu$ and $\lambda\nu$; and the curve constants f_1, f_2, f_3, and f_4 for use in our Kummer and Jacobian arithmetic functions. Obviously, none of the Rosenhain or curve constants are small; multiplying by these costs a full **M**.

3.3 Compressed and Decompressed Elements of $\mathcal{J}_{\mathcal{C}}$

Our algorithms use the usual Mumford representation for elements of $\mathcal{J}_{\mathcal{C}}(\mathbb{F}_p)$; they correspond to pairs $\langle u(X), v(X)\rangle$, where u and v are polynomials over $\mathbb{F}_p$ with u monic, $\deg v < \deg u \leq 2$, and $v(X)^2 \equiv f_{\mathcal{C}}(X) \pmod{u(X)}$. We compute the group operation $+$ in $\mathcal{J}_{\mathcal{C}}(\mathbb{F}_p)$ using a function ADD, which implements[1] the algorithm found in [HC14] (after a change of coordinates to meet their Assumption 1) at a cost of $28\mathbf{M} + 2\mathbf{S} + 11\mathbf{a} + 24\mathbf{s} + \mathbf{I}$. For transmission, we compress the 508-bit Mumford representation to a 256-bit form. Our functions COMPRESS and DECOMPRESS (Algorithm 1) implement Stahlke's compression technique (see [Sta04] and [CCS15, App. A] for details).

[1] We only call ADD once in our algorithms, while its very cumbersome to write down. We refer to the code (on either platform) for its implementation.

Algorithm 1. Point (de)compression on $\mathcal{J}_{\mathcal{C}}$.

Function: COMPRESS
Input: $\langle X^2 + u_1X + u_0, v_1X + v_0\rangle = P \in \mathcal{J}_{\mathcal{C}}$
Output: A string $b_0 \cdots b_{255}$ of 256 bits
Cost: $3\mathbf{M} + 1\mathbf{S} + 2\mathbf{a} + 2\mathbf{s}$

1 $w \leftarrow 4((u_1 \cdot v_0 - u_0 \cdot v_1) \cdot v_1 - v_0^2)$
2 $b_0 \leftarrow \text{LEASTSIGNIFICANTBIT}(v_1)$
3 $b_{128} \leftarrow \text{LEASTSIGNIFICANTBIT}(w)$
4 **return** $b_0 \,||\, u_0 \,||\, b_{128} \,||\, u_1$

– –

Function: DECOMPRESS
Input: A string $b_0 \cdots b_{255}$ of 256 bits.
Output: $\langle X^2 + u_1X + u_0, v_1X + v_0\rangle = P \in \mathcal{J}_{\mathcal{C}}$
Cost: $46\mathbf{M} + 255\mathbf{S} + 17\mathbf{a} + 12\mathbf{s} + 6\mathbf{neg}$

5 $U_1 = b_{129} \cdots b_{256}$ as an element of $\mathbb{F}_p$
6 $U_0 = b_1 \cdots b_{127}$ as an element of $\mathbb{F}_p$
7 $T_1 \leftarrow U_1^2$
8 $T_2 \leftarrow U_0 - T_1$
9 $T_3 \leftarrow U_0 + T_2$
10 $T_4 \leftarrow U_0 \cdot (T_3 \cdot f_4 + (U_1 \cdot f_3 - 2f_2))$
11 $T_3 \leftarrow -T_3$
12 $T_1 \leftarrow T_3 - U_0$
13 $T_4 \leftarrow 2(T_4 + (T_1 \cdot U_0 + f_1) \cdot U_1)$
14 $T_1 \leftarrow 2(T_1 - U_0))$
15 $T_5 \leftarrow ((U_0 - (f_3 + U_1 \cdot (U_1 - f_4))) \cdot U_0 + f_1)^2$
16 $T_5 \leftarrow T_4^2 - 2T_5 \cdot T_1$
17 $(T_6, T_5) \leftarrow \text{GFE\_SQRTINV}(T_5, T_1, b_1)$
18 $T_4 \leftarrow (T_5 - T_4) \cdot T_6$
19 $T_5 \leftarrow -f_4 \cdot T_2 - ((T_3 - f_3) \cdot U_1) + f_2 + T_4$
20 $T_6 = \text{GFE\_POWMINHALF}(4T_6)$
21 $V_1 \leftarrow 2T_5 \cdot T_6$
22 **if** $b_0 \neq \text{LEASTSIGNIFICANTBIT}(V_1)$ **then** $(V_1, T_6) \leftarrow (-V_1, -T_6)$
23 $T_5 \leftarrow (U_1 \cdot f_4 + (T_2 - f_3)) \cdot U_0$
24 $V_0 \leftarrow (U_1 \cdot T_4 + T_5 + f_1) \cdot T_6$
25 **return** $\langle X^2 + U_1X + U_0, V_1X + V_0\rangle$

3.4 The Kummer Surface $\mathcal{K}_\mathcal{C}$

The Kummer surface of $\mathcal{C}$ is the quotient $\mathcal{K}_\mathcal{C} = \mathcal{J}_\mathcal{C}/\pm$; points on $\mathcal{K}_\mathcal{C}$ correspond to points on $\mathcal{J}_\mathcal{C}$ taken up to sign. If P is a point in $\mathcal{J}_\mathcal{C}$, then we write

$$\pm P = (x_P : y_P : z_P : t_P)$$

for its image in $\mathcal{K}_\mathcal{C}$. To avoid subscript explosion, we make the following convention: when points P and Q on $\mathcal{J}_\mathcal{C}$ are clear from the context, we write

$$\pm(P+Q) = (x_\oplus : y_\oplus : z_\oplus : t_\oplus) \quad \text{and} \quad \pm(P-Q) = (x_\ominus : y_\ominus : z_\ominus : t_\ominus)\,.$$

The Kummer surface of this $\mathcal{C}$ has a "fast" model in $\mathbb{P}^3$ defined by

$$\mathcal{K}_\mathcal{C} : E \cdot xyzt = \begin{pmatrix} (x^2+y^2+z^2+t^2) \\ -F\cdot(xt+yz) - G\cdot(xz+yt) - H\cdot(xy+zt) \end{pmatrix}^2$$

where

$$F = \frac{a^2-b^2-c^2+d^2}{ad-bc}\,, \quad G = \frac{a^2-b^2+c^2-d^2}{ac-bd}\,, \quad H = \frac{a^2+b^2-c^2-d^2}{ab-cd}\,,$$

and $E = 4abcd\,(ABCD/((ad-bc)(ac-bd)(ab-cd)))^2$ (see e. g. [CC86], [Cos11] and [Gau07]). The identity point $\langle 1, 0\rangle$ of $\mathcal{J}_\mathcal{C}$ maps to $\pm 0_{\mathcal{J}_\mathcal{C}} = (a : b : c : d)$. The PROJECT function (Algorithm 2) maps general points from $\mathcal{J}_\mathcal{C}(\mathbb{F}_p)$ into $\mathcal{K}_\mathcal{C}$. The "special" case where u is linear is treated in [CCS15, §7.2]; this is not implemented, since PROJECT only operates on public generators and keys, none of which are special.

3.5 Pseudo-addition on $\mathcal{K}_\mathcal{C}$

While the points of $\mathcal{K}_\mathcal{C}$ do not form a group, we have a pseudo-addition operation (differential addition), which computes $\pm(P+Q)$ from $\pm P$, $\pm Q$, and $\pm(P-Q)$. The function XADD (Algorithm 3) implements the standard differential addition. The special case where $P = Q$ yields a pseudo-doubling operation. To simplify the presentation of our algorithms, we define three operations on points in $\mathbb{P}^3$. First, $\mathcal{M} : \mathbb{P}^3 \times \mathbb{P}^3 \to \mathbb{P}^3$ multiplies corresponding coordinates. That is, it takes as input a pair of vectors $(x_1 : y_1 : z_1 : t_1)$ and $(x_2 : y_2 : z_2 : t_2)$ and returns the coordinate-wise product vector $(x_1x_2 : y_1y_2 : z_1z_2 : t_1t_2)$. The special case $(x_1 : y_1 : z_1 : t_1) = (x_2 : y_2 : z_2 : t_2)$ is denoted by $\mathcal{S} : (x : y : z : t) \mapsto (x^2 : y^2 : z^2 : t^2)$. Finally, the Hadamard

Algorithm 2. Projection from $\mathcal{J}_\mathcal{C}$ to $\mathcal{K}_\mathcal{C}$ and point (un)wrapping on $\mathcal{K}_\mathcal{C}$.

Function: PROJECT
Input: $\langle X^2 + u_1X + u_0, v_1X + v_0\rangle = P \in \mathcal{J}_\mathcal{C}$
Output: $(x_P : y_P : z_P : t_P) = \pm P \in \mathcal{K}_\mathcal{C}$
Cost: $8\mathbf{M} + 1\mathbf{S} + 4\mathbf{m_c} + 7\mathbf{a} + 4\mathbf{s}$

1 $(T_1, T_2, T_3, T_4) \leftarrow (\mu - u_0, \lambda\nu - u_0, \nu - u_0, \lambda\mu - u_0)$
2 $T_5 \leftarrow \lambda + u_1$
3 $T_7 \leftarrow u_0 \cdot ((T_5 + \mu) \cdot T_3)$
4 $T_5 \leftarrow u_0 \cdot ((T_5 + \nu) \cdot T_1)$
5 $(T_6, T_8) \leftarrow (u_0 \cdot ((\mu + u_1) \cdot T_2 + T_2), u_0 \cdot ((\nu + u_1) \cdot T_4 + T_4))$
6 $T_1 \leftarrow v_0^2$
7 $(T_5, T_6, T_7, T_8) \leftarrow (T_5 - T_1, T_6 - T_1, T_7 - T_1, T_8 - T_1)$
8 **return** $(a \cdot T_5 : b \cdot T_6 : c \cdot T_7 : d \cdot T_8)$

– –

Function: XWRAP
Input: $(x : y : z : t) \in \mathbb{P}^3$
Output: $(x/y, x/z, x/t) \in \mathbb{F}_p^3$
Cost: $7\mathbf{M} + \mathbf{I}$

9 $V_1 \leftarrow y \cdot z$
10 $V_2 \leftarrow x/(V_1 \cdot t)$
11 $V_3 \leftarrow V_2 \cdot t$
12 **return** $(V_3 \cdot z, V_3 \cdot y, V_1 \cdot V_2)$

– –

Function: XUNWRAP
Input: $(u, v, w) \in \mathbb{F}_p^3$ s.t. $u = x_P/y_P, v = x_P/z_P, w = x_P/t_P$ for $\pm P \in \mathcal{K}_\mathcal{C}$
Output: $(\pm[m]P, \pm[m+1]P) \in \mathcal{K}_\mathcal{C}^2$ $(x_P : y_P : z_P : t_P) \in \mathbb{P}^3$
Cost: $4\mathbf{M}$

13 $(T_1, T_2, T_3) \leftarrow (v \cdot w, u \cdot w, u \cdot v)$
14 **return** $(T_3 \cdot w : T_1 : T_2 : T_3)$

transform[2] is defined by

$$\mathcal{H} : (x : y : z : t) \mapsto \begin{pmatrix} x+y+z+t : x+y-z-t : \\ x-y+z-t : x-y-z+t \end{pmatrix}.$$

Clearly $\mathcal{M}$ and $\mathcal{S}$ cost 4**M** and 4**S**, respectively. The Hadamard transform can easily be implemented with 4**a** + 4**s**. However, the additions and subtractions are relatively cheap, making function call overhead a large factor. To minimize this we inline the Hadamard transform, trading a bit of code size for efficiency.

Lines 6 and 7 of Algorithm 3 only involve the third argument, $\pm(P - Q)$; essentially, they compute the point $(y_\ominus z_\ominus t_\ominus : x_\ominus z_\ominus t_\ominus : x_\ominus y_\ominus t_\ominus : x_\ominus y_\ominus z_\ominus)$ (which is projectively equivalent to $(1/x_\ominus : 1/y_\ominus : 1/z_\ominus : 1/t_\ominus)$, but requires no inversions; note that this is generally *not* a point on $\mathcal{K}_\mathcal{C}$). In practice, the pseudoadditions used in our scalar multiplication all use a fixed third argument, so it makes sense to precompute this "inverted" point and to scale it by $x_\ominus$ so that the first coordinate is 1, thus saving 7**M** in each subsequent differential addition for a one-off cost of 1**I**. The resulting data can be stored as the 3-tuple $(x_\ominus/y_\ominus, x_\ominus/z_\ominus, x_\ominus/t_\ominus)$, ignoring the trivial first coordinate: this is the *wrapped* form of $\pm(P - Q)$. The function XWRAP (Algorithm 2) applies this transformation. The differential double-and-add XDBLADD (Algorithm 3) combines the pseudo-doubling with the differential addition, sharing intermediate operands. This is the fundamental building block of the Montgomery ladder.

4 Scalar Multiplication

All of our cryptographic routines are built around scalar multiplication in $\mathcal{J}_\mathcal{C}$ and pseudo-scalar multiplication in $\mathcal{K}_\mathcal{C}$. We implement pseudo-scalar multiplication using the classic Montgomery ladder in §4.1. In §4.2, we extend this to full scalar multiplication on $\mathcal{J}_\mathcal{C}$ using the point recovery technique proposed in [CCS17].

4.1 Pseudomultiplication on $\mathcal{K}_\mathcal{C}$

Since $[m](-P) = -[m]P$ for all m and P, we have a pseudo-scalar multiplication operation $(m, \pm P) \mapsto \pm[m]P$ on $\mathcal{K}_\mathcal{C}$, which we compute using Algorithm 4 (the Montgomery ladder), implemented as CRYPTO_SCALARMULT. The loop of Algorithm 4

[2] Note that $(A : B : C : D) = \mathcal{H}((a : b : c : d))$ and $(a : b : c : d) = \mathcal{H}((A : B : C : D))$.

Algorithm 3. Doubling and differential addition on $\mathcal{K}_{\mathcal{C}}$.

Function: XADD
Input: $(\pm P, \pm Q, \pm(P-Q)) \in \mathcal{K}_{\mathcal{C}}^3$ for some P and Q on $\mathcal{J}_{\mathcal{C}}$.
Output: $\pm(P+Q) \in \mathcal{K}_{\mathcal{C}}$
Cost: $14\mathbf{M} + 4\mathbf{S} + 4\mathbf{m_c} + 12\mathbf{a} + 12\mathbf{s}$

$(V_1, V_2) \leftarrow (\mathcal{H}(\pm P), \mathcal{H}(\pm Q))$
$V_1 \leftarrow \mathcal{M}(V_1, V_2)$
$V_1 \leftarrow \mathcal{M}(V_1, (1/A : 1/B : 1/C : 1/D))$
$V_1 \leftarrow \mathcal{H}(V_1)$
$V_1 \leftarrow \mathcal{S}(V_1)$
$(C_1, C_2) \leftarrow (z_\ominus \cdot t_\ominus, x_\ominus \cdot y_\ominus)$
$V_2 \leftarrow \mathcal{M}((C_1 : C_1 : C_2 : C_2), (y_\ominus : x_\ominus : t_\ominus : z_\ominus))$
return $\mathcal{M}(V_1, V_2)$

– –

Function: XDBLADD
Input: $(\pm P, \pm Q, (x_\ominus/y_\ominus, x_\ominus/z_\ominus, x_\ominus/t_\ominus)) \in \mathcal{K}_{\mathcal{C}}^2 \times \mathbb{F}_p^3$
Output: $(\pm[2]P, \pm(P+Q)) \in \mathcal{K}_{\mathcal{C}}^2$
Cost: $7\mathbf{M} + 12\mathbf{S} + 12\mathbf{m_c} + 16\mathbf{a} + 16\mathbf{s}$

$(V_1, V_2) \leftarrow (\mathcal{S}(\pm P), \mathcal{S}(\pm Q))$
$(V_1, V_2) \leftarrow (\mathcal{H}(V_1), \mathcal{H}(V_2))$
$(V_1, V_2) \leftarrow (\mathcal{S}(V_1), \mathcal{M}(V_1, V_2))$
$(V_1, V_2) \leftarrow (\mathcal{M}(V_1, (\frac{1}{A} : \frac{1}{B} : \frac{1}{C} : \frac{1}{D})), \mathcal{M}(V_2, (\frac{1}{A} : \frac{1}{B} : \frac{1}{C} : \frac{1}{D})))$
$(V_1, V_2) \leftarrow (\mathcal{H}(V_1), \mathcal{H}(V_2))$
return $(\mathcal{M}(V_1, (\frac{1}{a} : \frac{1}{b} : \frac{1}{c} : \frac{1}{d})), \mathcal{M}(V_2, (1 : \frac{x_\ominus}{y_\ominus} : \frac{x_\ominus}{y_\ominus} : \frac{x_\ominus}{t_\ominus})))$

maintains the following invariant: at the end of iteration i we have

$$(V_1, V_2) = (\pm[k]P, \pm[k+1]P) \quad \text{where} \quad k = \textstyle\sum_{j=i}^{\beta-1} m_j 2^{\beta-1-i}\,.$$

Hence, at the end we return $\pm[m]P$, and also $\pm[m+1]P$ as a (free) byproduct. We suppose we have a constant-time conditional swap CSWAP defined as

$$\text{CSWAP} : (b, (V_1, V_2)) \mapsto (V_{1+b}, V_{2-b})\,.$$

This makes the execution of Algorithm 4 uniform and constant-time, and thus suitable for use with secret m.

Algorithm 4. Uniform and constant-time scalar multiplication on $\mathcal{K}_\mathcal{C}$ and $\mathcal{J}_\mathcal{C}$.

Function: CRYPTO_SCALARMULT
Input: $(m = \sum_{i=0}^{\beta-1} m_i 2^i, (x_P/y_P, x_P/z_P, x_P/t_P)) \in [0, 2^\beta) \times \mathbb{F}_p^3$ for $\pm P$ in $\mathcal{K}_\mathcal{C}$
Output: $(\pm[m]P, \pm[m+1]P) \in \mathcal{K}_\mathcal{C}^2$
Cost: $(7\beta+4)\mathbf{M} + 12\beta\mathbf{S} + 12\beta\mathbf{m_c} + 16\beta\mathbf{a} + 16\beta\mathbf{s}$

$V_1 \leftarrow (a : b : c : d)$
$V_2 \leftarrow \text{XUNWRAP}(x_P/y_P, x_P/z_P, x_P/t_P)$
for $i = \beta - 1$ down to 0 **do**
  $(V_1, V_2) \leftarrow \text{CSWAP}(m_i, (V_1, V_2))$
  $(V_1, V_2) \leftarrow \text{XDBLADD}(V_1, V_2, (x_P/y_P, x_P/z_P, x_P/t_P))$
  $(V_1, V_2) \leftarrow \text{CSWAP}(m_i, (V_1, V_2))$
return (V_1, V_2)

Function: JACOBIAN_SCALARMULT
Input: $(m, P, (x_P/y_P, x_P/z_P, x_P/t_P)) \in [0, 2^\beta) \times \mathcal{J}_\mathcal{C} \times \mathbb{F}_p^3$
Output: $[m]P \in \mathcal{J}_\mathcal{C}$
Cost: $(7\beta+143)\mathbf{M} + (12\beta+12)\mathbf{S} + (12\beta+4)\mathbf{m_c} + (16\beta+70)\mathbf{a} + (16\beta+22)\mathbf{s} + 3\mathbf{neg} + \mathbf{I}$

$(X_0, X_1) \leftarrow \text{CRYPTO\_SCALARMULT}(m, (x_P/y_P, x_P/z_P, x_P/t_P))$
$V \leftarrow \text{XUNWRAP}((x_P/y_P, x_P/z_P, x_P/t_P))$
return $\text{RECOVERFAST}(P, V, X_0, X_1)$

Our implementation of CRYPTO_SCALARMULT assumes that its input Kummer point $\pm P$ is wrapped. This follows the approach of [Ber+14]. Indeed, many calls to CRYPTO_SCALARMULT involve Kummer points that are stored or transmitted in wrapped form. However, CRYPTO_SCALARMULT does require the unwrapped point internally—if only to initialize one variable. We therefore define a function XUNWRAP (Algorithm 2) to invert the XWRAP transformation at a cost of only $4\mathbf{M}$.

4.2 Point Recovery from $\mathcal{K}_\mathcal{C}$ to $\mathcal{J}_\mathcal{C}$

Point recovery means efficiently computing $[m]P$ on $\mathcal{J}_\mathcal{C}$ given $\pm[m]P$ on $\mathcal{K}_\mathcal{C}$ and some additional information. In our case, the additional information is the base point P and the second output of the Montgomery ladder, $\pm[m+1]P$. The RECOVERFAST function (Algorithm 5) implements the point recovery described in [CCS17]. This is the genus-2 analogue of the elliptic-curve methods in [LD99; OS01; BJ02]. We refer

Algorithm 5. Recovery from $\mathcal{K}_\mathcal{C}$ to $\mathcal{J}_\mathcal{C}$.

Function: RECOVERFAST
Input: $(P, \pm P, \pm Q, \pm(P+Q)) \in \mathcal{J}_\mathcal{C} \times \mathcal{K}_\mathcal{C}^3$ for some P, Q in $\mathcal{J}_\mathcal{C}$.
Output: $Q \in \mathcal{J}_\mathcal{C}$
Cost: $139\mathbf{M} + 12\mathbf{S} + 4\mathbf{m_c} + 70\mathbf{a} + 22\mathbf{s} + 3\mathbf{neg} + \mathbf{I}$

$g_P \leftarrow$ FAST2GENPARTIAL$(\pm P)$
$g_Q \leftarrow$ FAST2GENFULL$(\pm Q)$
$g_S \leftarrow$ FAST2GENPARTIAL$(\pm(P+Q))$
$x_D \leftarrow$ XADD$(\pm P, \pm Q, \pm(P+Q))$
$g_D \leftarrow$ FAST2GENPARTIAL(x_D)
return RECOVERGENERAL(P, g_P, g_Q, g_S, g_D)

Function: FAST2GENFULL
Input: $\pm P \in \mathcal{K}_\mathcal{C}$
Output: $\widetilde{\pm P} \in \widetilde{\mathcal{K}}_\mathcal{C}$
Cost: $15\mathbf{M} + 12\mathbf{a}$

$\widetilde{x}_P \leftarrow x_P + (L_{12}/L_{11})y_P + (L_{13}/L_{11})z_P + (L_{14}/L_{11})t_P$
$\widetilde{y}_P \leftarrow (L_{21}/L_{11})x_P + (L_{22}/L_{11})y_P + (L_{23}/L_{11})z_P + (L_{24}/L_{11})t_P$
$\widetilde{z}_P \leftarrow (L_{31}/L_{11})x_P + (L_{32}/L_{11})y_P + (L_{33}/L_{11})z_P + (L_{34}/L_{11})t_P$
$\widetilde{t}_P \leftarrow (L_{41}/L_{11})x_P + (L_{42}/L_{11})y_P + (L_{43}/L_{11})z_P + (L_{44}/L_{11})t_P$
return $(\widetilde{x}_P : \widetilde{y}_P : \widetilde{z}_P : \widetilde{t}_P)$

Function: FAST2GENPARTIAL
Input: $\pm P \in \mathcal{K}_\mathcal{C}$
Output: $(\widetilde{x}_P : \widetilde{y}_P : \widetilde{z}_P) \in \mathbb{P}^2$
Cost: $11\mathbf{M} + 9\mathbf{a}$

$\widetilde{x}_P \leftarrow x_P + (L_{12}/L_{11})y_P + (L_{13}/L_{11})z_P + (L_{14}/L_{11})t_P$
$\widetilde{y}_P \leftarrow (L_{21}/L_{11})x_P + (L_{22}/L_{11})y_P + (L_{23}/L_{11})z_P + (L_{24}/L_{11})t_P$
$\widetilde{z}_P \leftarrow (L_{31}/L_{11})x_P + (L_{32}/L_{11})y_P + (L_{33}/L_{11})z_P + (L_{34}/L_{11})t_P$
return $(\widetilde{x}_P : \widetilde{y}_P : \widetilde{z}_P)$

the reader to [CCS17] for technical details on this method, but there is one important mathematical detail that we should mention (since it is reflected in the structure of our code): point recovery is more natural starting from the general Flynn model $\widetilde{\mathcal{K}}_{\mathcal{C}}$ of the Kummer, because it is more closely related to the Mumford model for $\mathcal{J}_{\mathcal{C}}$. The RECOVERFAST function (Algorithm 5) therefore proceeds in two steps: first the functions FAST2GENFULL and FAST2GENPARTIAL map the problem into $\widetilde{\mathcal{K}}_{\mathcal{C}}$, and then we recover from $\widetilde{\mathcal{K}}_{\mathcal{C}}$ to $\mathcal{J}_{\mathcal{C}}$ using RECOVERGENERAL (Algorithm 6).

Since the general Kummer $\widetilde{\mathcal{K}}_{\mathcal{C}}$ only appears briefly in our recovery procedure (we never use its relatively slow arithmetic operations), we will not investigate it in detail here—but the curious reader may refer to [CF96] for the general theory. For our purposes, it suffices to recall that $\widetilde{\mathcal{K}}_{\mathcal{C}}$ is, like $\mathcal{K}_{\mathcal{C}}$, embedded in $\mathbb{P}^3$; and the isomorphism $\mathcal{K}_{\mathcal{C}} \to \widetilde{\mathcal{K}}_{\mathcal{C}}$ is defined (in e. g. [CCS15, §7.4]) by the linear transformation

$$(x_P : y_P : z_P : t_P) \mapsto (\widetilde{x}_P : \widetilde{y}_P : \widetilde{z}_P : \widetilde{t}_P) := (x_P : y_P : z_P : t_P)L\,,$$

where L is (any scalar multiple of) the matrix

$$\begin{pmatrix} a^{-1}(\nu-\lambda) & a^{-1}(\mu\nu-\lambda) & a^{-1}\lambda\nu(\mu-1) & a^{-1}\lambda\nu(\mu\nu-\lambda) \\ b^{-1}(\mu-1) & b^{-1}(\mu\nu-\lambda) & b^{-1}\mu(\nu-\lambda) & b^{-1}\mu(\mu\nu-\lambda) \\ c^{-1}(\lambda-\mu) & c^{-1}(\lambda-\mu\nu) & c^{-1}\lambda\mu(1-\nu) & c^{-1}\lambda\mu(\lambda-\mu\nu) \\ d^{-1}(1-\nu) & d^{-1}(\lambda-\mu\nu) & d^{-1}\nu(\lambda-\mu) & d^{-1}\nu(\lambda-\mu\nu) \end{pmatrix},$$

which we precompute and store. If $\pm P$ is a point on $\mathcal{K}_{\mathcal{C}}$, then $\widetilde{\pm P}$ denotes its image on $\widetilde{\mathcal{K}}_{\mathcal{C}}$; we compute $\widetilde{\pm P}$ using the function FAST2GENFULL (Algorithm 5). Sometimes we only require the first three coordinates of $\widetilde{\pm P}$. We save $4\mathbf{M} + 3\mathbf{a}$ per point in the function FAST2GENPARTIAL (Algorithm 5) by not computing $\widetilde{t}_P$.

4.3 Full Scalar Multiplication on $\mathcal{J}_{\mathcal{C}}$

We now combine the pseudo-scalar multiplication CRYPTO_SCALARMULT with the point-recovery function RECOVERFAST to define a full scalar multiplication function JACOBIAN_SCALARMULT (Algorithm 4) on $\mathcal{J}_{\mathcal{C}}$.

Remark 3. The function JACOBIAN_SCALARMULT takes not only a scalar m and a Jacobian point P in its Mumford representation, but also the wrapped form of $\pm P$ as an auxiliary argument: that is, we assume that $x_P \leftarrow \text{PROJECT}(P)$ and $\text{XWRAP}(x_P)$ have already been carried out. This saves redundant PROJECT and XWRAP calls when operating on fixed base points, as is often the case in our protocols. Nevertheless, JACOBIAN_SCALARMULT could easily be converted to a "pure" Jacobian scalar mul-

Algorithm 6. A map from $\widetilde{\mathcal{K}}_C$ to $\mathcal{J}_\mathcal{C}$.

Function: RECOVERGENERAL

Input: $(P, \widetilde{\pm P}, \widetilde{\pm Q}, \widetilde{\pm(P+Q)}, \widetilde{\pm(P-Q)}) \in \mathcal{J}_\mathcal{C} \times \widetilde{\mathcal{K}}_\mathcal{C}^4$ for some P and Q in $\mathcal{J}_\mathcal{C}$.

Output: $Q \in \mathcal{J}_\mathcal{C}$

Cost: $77\mathbf{M} + 8\mathbf{S} + 19\mathbf{a} + 10\mathbf{s} + 3\mathbf{neg} + \mathbf{I}$

$(Z_1, Z_2) \leftarrow (\widetilde{y}_P \cdot \widetilde{x}_Q - \widetilde{x}_Q \cdot \widetilde{y}_P, \widetilde{x}_P \cdot \widetilde{z}_Q - \widetilde{z}_P \cdot \widetilde{x}_Q)$

$T_1 \leftarrow Z_1 \cdot \widetilde{z}_P$

$Z_3 \leftarrow Z_2 \cdot \widetilde{y}_P + T_1$

$D \leftarrow Z_2^2 \cdot \widetilde{x}_P + Z_3 \cdot Z_1$

$T_2 \leftarrow Z_1 \cdot Z_2$

$T_3 \leftarrow \widetilde{x}_P \cdot \widetilde{x}_Q$

$E \leftarrow T_3 \cdot (T_3 \cdot (f_2 \cdot Z_2^2 - f_1 \cdot T_2) + \widetilde{t}_Q \cdot D)$

$E \leftarrow E + Z_3 \cdot \widetilde{x}_Q^2 \cdot (f_3 \cdot Z_2 \cdot \widetilde{x}_P + f_4 \cdot Z_3)$

$E \leftarrow E + Z_3 \cdot \widetilde{x}_Q \cdot (Z_3 \cdot \widetilde{y}_Q - Z_2 \cdot \widetilde{x}_P \cdot \widetilde{z}_Q)$

$X_1 \leftarrow \widetilde{x}_P \cdot (Z_2 \cdot v_1(P) - Z_1 \cdot v_0(P))$

$T_4 \leftarrow Z_1 \cdot \widetilde{y}_P + Z_2 \cdot \widetilde{x}_P$

$X_2 \leftarrow T_1 \cdot v_1(P) + T_4 \cdot v_0(P)$

$C_5 \leftarrow Z_1^2 - T_4 \cdot \widetilde{x}_Q$

$C_6 \leftarrow T_1 \cdot \widetilde{x}_Q + T_2$

$T_5 \leftarrow \widetilde{z}_\oplus \cdot \widetilde{x}_\ominus - \widetilde{x}_\oplus \cdot \widetilde{z}_\ominus$

$X_3 \leftarrow X_1 \cdot T_5 - X_2 \cdot (\widetilde{x}_\oplus \cdot \widetilde{y}_\ominus - \widetilde{y}_\oplus \cdot \widetilde{x}_\ominus)$

$(X_5, X_6) \leftarrow (X_3 \cdot C_5, X_3 \cdot C_6)$

$X_4 \leftarrow T_3 \cdot (X_1 \cdot (\widetilde{z}_\oplus \cdot \widetilde{y}_\ominus - \widetilde{y}_\oplus \cdot \widetilde{z}_\ominus) + T_5 \cdot X_2)$

$(X_7, X_8) \leftarrow (X_5 + Z_1 \cdot X_4, X_6 + Z_2 \cdot Z_4)$

$T_6 \leftarrow \widetilde{x}_\oplus \cdot \widetilde{x}_\ominus$

$E \leftarrow -T_6 \cdot T_3 \cdot (E \cdot \widetilde{x}_P^2 + (X_1 \cdot T_3)^2)$

$(X_9, X_{10}) \leftarrow (E \cdot X_7, E \cdot X_8)$

$F \leftarrow X_2 \cdot (\widetilde{x}_\oplus \cdot \widetilde{y}_\ominus + \widetilde{y}_\oplus \cdot \widetilde{x}_\ominus) + X_1 \cdot (\widetilde{z}_\oplus \cdot \widetilde{x}_\ominus + \widetilde{x}_\oplus \cdot \widetilde{z}_\ominus)$

$F \leftarrow X_1 \cdot F + 2(X_2^2 \cdot T_6)$

$F \leftarrow -2(F \cdot D \cdot T_6 \cdot T_3 \cdot T_3^2 \cdot \widetilde{x}_P)$

$(U_1, U_0) \leftarrow (-F \cdot \widetilde{y}_Q, F \cdot \widetilde{z}_Q)$

$F_i \leftarrow 1/(F \cdot \widetilde{x}_Q)$

$(u_1', u_0', v_1', v_0') \leftarrow (F_i \cdot U_1, F_i \cdot U_0, F_i \cdot X_9, F_i \cdot X_{10})$

return $\langle X^2 + u_1' X + u_0', v_1' X + v_0' \rangle$

Table 3. Operation and cycle counts of basic functions on the Kummer and Jacobian.

	M	**S**	$\mathbf{m_c}$	**a**	**s**	**neg**	**I**	**AVR**	**ARM**
ADD	28	2	0	11	24	0	1	228 552	62 886
PROJECT	8	1	4	7	8	0	0	20 205	5 667
XWRAP	7	0	0	0	0	0	1	182 251	49 609
XUNWRAP	4	0	0	0	0	0	0	7 297	2 027
XADD	14	4	4	12	12	0	0	34 774	9 598
XDBLADD	7	12	12	16	16	0	0	36 706	9 861
RECOVERGENERAL	77	8	0	19	10	3	1	318 910	88 414
FAST2GENPARTIAL	11	0	0	9	0	0	0	21 339	6 110
FAST2GENFULL	15	0	0	12	0	0	0	29 011	8 333
RECOVERFAST	139	12	4	70	22	5	1	447 176	124 936
COMPRESS	3	1	0	2	2	0	0	8 016	2 186
DECOMPRESS	46	255	0	17	12	6	0	386 524	106 013

tiplication function (with no auxiliary input) by inserting appropriate PROJECT and XWRAP calls at the start, and removing the XUNWRAP call at Line 2, increasing the total cost by $11\mathbf{M} + 1\mathbf{S} + 4\mathbf{m_c} + 7\mathbf{a} + 8\mathbf{s} + 1\mathbf{I}$.

5 Results and Comparison

The high-level cryptographic functions for our signature scheme are named KEYGEN, SIGN and VERIFY. Their implementations contain no surprises: they do exactly what was specified in §2.1, calling the lower-level functions described in §3 and §4 as required. Our Diffie-Hellman key generation and key exchange use only the function DH_EXCHANGE, which implements exactly what we specified in §2.2: one call to CRYPTO_SCALARMULT plus a call to XWRAP to convert to the correct 384-bit representation. Table 1 (in the introduction) presents the cycle counts and stack usage for all of our high-level functions.

Code and compilation. For our experiments, we compiled our AVR ATmega code with `avr-gcc -O2`, and our ARM Cortex M0 code with `clang -O2` (the optimization levels `-O3`, `-O1`, and `-Os` gave fairly similar results). The total program size is 20 242 bytes for the AVR ATmega, and 19 606 bytes for the ARM Cortex M0. This consists of the full signature and key-exchange code, including the reference implementation of the hash function SHAKE128 with 512-bit output.[3]

[3] We used the reference C implementation for the Cortex M0, and the assembly implementation for AVR; both are available from [Ber+16]. The only change required is to the padding, which must take domain separation into account according to [Dwo15, p.28].

Table 4. Comparison of scalar multiplication routines on the AVR ATmega architecture at the 128-bit security level. **S** denotes signature-compatible full scalar multiplication; **DH** denotes Diffie–Hellman pseudo-scalar multiplication. The code size and stack size are measured in bytes. The implementation marked * also contains a fixed-basepoint scalar multiplication routine, whereas the implementation marked † does not report code size for the separated scalar multiplication.

	Implementation	Object	Clock cycles	Code size	Stack
DH	[LWG14]	256-bit curve	$\approx$ 21 078 200	*14 700 B	556 B
S, DH	[WUW13]	NIST P-256	$\approx$ 34 930 000	16 112 B	590 B
DH	[HS13]	Curve25519	22 791 579	†—	677 B
DH	[Dül+15]	Curve25519	13 900 397	17 710 B	494 B
DH	**This**	$\mathcal{K}_{\mathcal{C}}$	9 513 536	$\approx$ 9 490 B	99 B
S	**This**	$\mathcal{J}_{\mathcal{C}}$	9 968 127	$\approx$ 16 516 B	735 B

Basis for comparison. As we believe ours to be the first genus-2 hyperelliptic curve implementation on both the AVR ATmega and the ARM Cortex M0 architectures, we can only compare with elliptic curve-based alternatives at the same 128-bit security level: notably [LWG14; HS13; WUW13; Dül+15]. This comparison is not superficial: the key exchanges in [LWG14; HS13; Dül+15] use the highly efficient x-only arithmetic on Montgomery elliptic curves, while [WUW13] uses similar techniques for Weierstrass elliptic curves, and x-only arithmetic is the exact elliptic-curve analogue of Kummer surface arithmetic. To provide full scalar multiplication in a group, [WUW13] appends y-coordinate recovery to its x-only arithmetic (using the approach of [BJ02]); again, this is the elliptic-curve analogue of our methods.

Results for AVR ATmega. Looking at Table 4, on the AVR ATmega architecture we reduce the cycle count for Diffie–Hellman by about 32% compared with the current record [Dül+15], again roughly halving the code size, and reducing stack usage by about 80%. The cycle count for Jacobian scalar multiplication (for signatures) is reduced by 71% compared with [WUW13], while increasing the stack usage by 25%.

Finally we can compare to the current fastest full signature implementation by Nascimento, López and Dahab [NLD15], shown in Table 5. We almost halve the number of cycles, while reducing stack usage by a decent margin (code size is not reported in [NLD15]).

Results for ARM Cortex M0. As we see in Table 6, genus-2 techniques give great results for Diffie–Hellman key exchange on the ARM Cortex M0 architecture. Compared with the current fastest implementation [Dül+15], we reduce the number of

Table 5. Comparison of signature schemes on the AVR ATmega architecture at the 128-bit security level. The stack size is measured in bytes.

Implementation	Object	Function	Clock cycles	Stack
[NLD15]	Ed25519	Sig. Gen.	19 047 706	1 473 B
This	$\mathcal{J}_\mathcal{C}$	SIGN	10 404 033	926 B
[NLD15]	Ed25519	Sig. Ver.	30 776 942	1 226 B
This	$\mathcal{J}_\mathcal{C}$	VERIFY	16 240 510	992 B

clock cycles by about 27%, while roughly halving code size and stack space. For signatures, the state-of-the-art is [WUW13]: here we reduce the cycle count for the underlying scalar multiplications by a very impressive 75%, at the cost of an increase in code size and stack usage.

Table 6. Comparison of scalar multiplication routines on the ARM Cortex M0 architecture at the 128-bit security level. **S** denotes signature-compatible full scalar multiplication; **DH** denotes Diffie–Hellman pseudo-scalar multiplication. The code size and stack size are measured in bytes.

	Implementation	Object	Clock cycles	Code size	Stack
S, DH	[WUW13]	NIST P-256	$\approx$ 10 730 000	7 168 B	540 B
DH	[Dül+15]	Curve25519	3 589 850	7 900 B	548 B
DH	**This**	$\mathcal{K}_\mathcal{C}$	2 633 662	$\approx$ 4 328 B	248 B
S	**This**	$\mathcal{J}_\mathcal{C}$	2 709 401	$\approx$ 9 874 B	968 B

Chapter V

qDSA: Small and Secure Digital Signatures with Curve-based Diffie-Hellman Key Pairs

The qDSA protocol is a high-speed, high-security signature scheme that facilitates implementations with a very small memory footprint, a crucial requirement for embedded systems and IoT devices, and that uses the same public keys as modern Diffie–Hellman schemes based on Montgomery curves (such as Curve25519) or Kummer surfaces. It resembles an adaptation of EdDSA to the world of Kummer varieties, which are quotients of algebraic groups by ± 1. Interestingly, qDSA does not require any full group operations or point recovery: all computations, including signature verification, occur on the quotient where there is no group law. We include details on four implementations of qDSA, using Montgomery and fast Kummer surface arithmetic on the 8-bit AVR ATmega and 32-bit ARM Cortex M0 platforms. We find that qDSA significantly outperforms state-of-the-art signature implementations in terms of stack usage and code size. We also include an efficient compression algorithm for points on fast Kummer surfaces, reducing them to the same size as compressed elliptic curve points for the same security level.

1 Introduction

Modern asymmetric cryptography based on elliptic and hyperelliptic curves [Kob87; Mil86] achieves two important goals. The first is efficient key exchange using the Diffie–Hellman protocol [DH76], using the fact that the (Jacobian of the) curve carries the structure of an abelian group. But in fact, as Miller observed [Mil86], we do not need the full group structure for Diffie–Hellman: the associated *Kummer variety* (the quotient by ± 1) suffices, which permits more efficiently-computable arithmetic [Mon87; Gau07]. A well-known example is Curve25519 [Ber06a], which offers fast scalar multiplications based on x-only arithmetic.

The second objective is efficient digital signatures, which are critical for authentication. There are several group-based signature schemes, the most important of which are ECDSA [Acc99a], Schnorr [Sch90], and now EdDSA [Ber+12] signatures. In contrast to the Diffie–Hellman protocol, all of these signature schemes explicitly require the group structure of the (Jacobian of the) curve. An unfortunate side-effect of this is that users essentially need two public keys to support both curve-based protocols. Further, basic cryptographic libraries need to provide implementations for arithmetic on both the Jacobian and the Kummer variety, thus complicating and increasing the size of the trusted code base. For example, the NaCl library [BLS12] uses Ed25519 [Ber+12] for signatures, and Curve25519 [Ber06a] for key exchange. This problem is worse for genus-2 hyperelliptic curves, where the Jacobian is significantly harder to use safely than its Kummer surface.

There have been several partial solutions to this problem. By observing that elements of the Kummer variety are elements of the Jacobian *up to sign*, one can build scalar multiplication on the Jacobian based on the fast Kummer arithmetic [OS01; CCS17]. This avoids the need for a separate scalar multiplication on the Jacobian, but does not avoid the need for its group law; it also introduces the need for projecting to and recovering from the Kummer. In any case, it does not solve the problem of having different public key types. Another proposal is XEdDSA [Per], which uses the public key on the Kummer variety to construct EdDSA signatures. In essence, it creates a key pair on the Jacobian by appending a sign bit to the public key on the Kummer variety, which can then be used for signatures. In [Ham12] Hamburg shows that one can actually verify signatures using only the x-coordinates of points on an elliptic curve, which is applied in the recent STROBE framework [Ham17]. We generalize this approach to allow Kummer varieties of curves of higher genera, and naturally adapt the scheme by only allowing challenges up to sign. This allows us to provide a proof of security, which has thus far not been attempted (in [Ham12]

Hamburg remarks that verifying up to sign does "probably not impact security at all"). Similar techniques have been applied for batch verification of ECDSA signatures [KD14], using the theory of summation polynomials [Sem04].

In this chapter we show that there is no intrinsic reason why Kummer varieties cannot be used for signatures. We present qDSA, a signature scheme relying only on Kummer arithmetic, and prove it secure in the random oracle model. It should not be surprising that the reduction in our proof is slightly weaker than the standard proof of security of Schnorr signatures [PS96], but not by more than we should expect. There is no difference between public keys for qDSA and Diffie–Hellman. We also provide an efficient compression method for points on fast Kummer surfaces, solving a long-standing open problem [Ber06c]. Our technique means that qDSA public keys for $g = 2$ can be efficiently compressed to 32 bytes, and that qDSA signatures fit into 64 bytes; it also finally reduces the size of Kummer-based Diffie–Hellman public keys from 48 to 32 bytes. Finally, we provide constant-time software implementations of genus-1 and genus-2 qDSA instances for the AVR ATmega and ARM Cortex M0 platforms. The performance of all four qDSA implementations comfortably beats earlier implementations in terms of stack usage and code size.

Organization. After an abstract presentation in §2, we give a detailed description of elliptic-curve qDSA instances in §3. We then move on to genus-2 instances based on fast Kummer surfaces, which give better performance. The necessary arithmetic appears in §4, before §5 describes the new verification algorithm. The method for compression of points on fast Kummer surfaces appears in §6, and the performance results for our implementations appear in §7.

2 The qDSA Signature Scheme

In this section we define qDSA, the *quotient Digital Signature Algorithm*. We start by recalling the basics of Kummer varieties in §2.1 and defining key operations in §2.2. The rest of the section is dedicated to the definition of the qDSA signature scheme, which is presented in full in Algorithm 1, and its proof of security, which follows Pointcheval and Stern [PS96; PS00]. The qDSA scheme closely resembles the Schnorr signature scheme [Sch90], as it results from applying the Fiat–Shamir heuristic [FS87] to an altered Schnorr identification protocol, together with a few standard changes as in EdDSA [Ber+12]. We comment on some special properties of qDSA in §2.5. Throughout, we work over finite fields $\mathbb{F}_p$ with $p > 3$.

2.1 The Kummer Variety Setting

Let $\mathcal{C}$ be a (hyper)elliptic curve and $\mathcal{J}$ its Jacobian.[1] The Jacobian is a commutative algebraic group with group operation $+$, inverse $-$ and identity 0. We assume $\mathcal{J}$ has a subgroup of large prime order N. The associated *Kummer variety* $\mathcal{K}$ is the quotient $\mathcal{K} = \mathcal{J}/\pm$. By definition, working with $\mathcal{K}$ corresponds to working on $\mathcal{J}$ *up to sign*. If P is an element of $\mathcal{J}$, we denote its image in $\mathcal{K}$ by $\pm P$. In this chapter we take $\log_2 N \approx 256$, and consider two important cases.

Genus 1. Here $\mathcal{J} = \mathcal{C}/\mathbb{F}_p$ is an elliptic curve with $\log_2 p \approx 256$, while $\mathcal{K} = \mathbb{P}^1$ is the x-line. We choose $\mathcal{C}$ to be Curve25519 [Ber06a], which is the topic of §3.

Genus 2. Here $\mathcal{J}$ is the Jacobian of a genus-2 curve $\mathcal{C}/\mathbb{F}_p$, where $\log_2 p \approx 128$, and $\mathcal{K}$ is a *Kummer surface*. We use the Gaudry–Schost parameters [GS12] for our implementations. The Kummer arithmetic, including some new constructions we need for signature verification and compression, is described in §4-6.

A point $\pm P$ in $\mathcal{K}(\mathbb{F}_p)$ is the image of a pair of points $\{P, -P\}$ on $\mathcal{J}$. It is important to note that P and $-P$ are not necessarily in $\mathcal{J}(\mathbb{F}_p)$; if not, then they are conjugate points in $\mathcal{J}(\mathbb{F}_{p^2})$, and correspond to points in $\mathcal{J}'(\mathbb{F}_p)$, where $\mathcal{J}'$ is the *quadratic twist* of $\mathcal{J}$. Both $\mathcal{J}$ and $\mathcal{J}'$ always have the same Kummer variety; we return to this fact, and its implications for our scheme, in §2.5 below.

2.2 Basic Operations

While a Kummer variety $\mathcal{K}$ has no group law, the operation that maps $\{\pm P, \pm Q\} \mapsto \{\pm(P+Q), \pm(P-Q)\}$ is well-defined. We can therefore define a *pseudo-addition* operation by $\text{XADD} : (\pm P, \pm Q, \pm(P-Q)) \mapsto \pm(P+Q)$. The special case where $\pm(P-Q) = \pm 0$ is the *pseudo-doubling* $\text{XDBL} : \pm P \mapsto \pm[2]P$. In our applications we can often improve efficiency by combining two of these operations in a single function $\text{XDBLADD} : (\pm P, \pm Q, \pm(P-Q)) \mapsto (\pm[2]P, \pm(P+Q))$. For any integer m, the scalar multiplication $[m]$ on $\mathcal{J}$ induces the key cryptographic operation of *pseudomultiplication* on $\mathcal{K}$, defined by $\text{LADDER} : (m, \pm P) \mapsto \pm[m]P$. As its name suggests, we compute LADDER using Montgomery's famous ladder algorithm [Mon87], i. e. as a uniform sequence of XDBLADDs and constant-time conditional swaps.[2] This

[1] In what follows, we could replace $\mathcal{J}$ by an arbitrary abelian group and all the proofs would be completely analogous. For simplicity we restrict to the cryptographically most interesting case of a Jacobian.

[2] In contemporary implementations such as NaCl, the LADDER function is sometimes named CRYPTO_SCALARMULT.

constant-time nature will be important for signing. Our signature verification requires a function CHECK on $\mathcal{K}^3$ defined by

$$\text{CHECK} : (\pm P, \pm Q, \pm R) \mapsto \begin{cases} \texttt{True} & \text{if } \pm R \in \{\pm(P+Q), \pm(P-Q)\} \\ \texttt{False} & \text{otherwise} \end{cases}.$$

Since we are working with projective points, we need a way to uniquely represent them. Moreover, we want this representation to be as small as possible, to minimize communication overhead. For this purpose we define the function COMPRESS : $\mathcal{K}(\mathbb{F}_p) \to \{0,1\}^{256}$ and, writing $\overline{\pm P} = \text{COMPRESS}(\pm P)$, the function

$$\text{DECOMPRESS} : \{0,1\}^{256} \to \mathcal{K}(\mathbb{F}_p) \cup \{\perp\}$$

such that $\text{DECOMPRESS}(\overline{\pm P}) = \pm P$ for $\pm P$ in $\mathcal{K}(\mathbb{F}_p)$ and $\text{DECOMPRESS}(X) = \perp$ for $X \in \{0,1\}^{256} \setminus \text{Im}(\text{COMPRESS})$. For the remainder of this section we assume that LADDER, CHECK, COMPRESS, and DECOMPRESS are defined. Their implementation depends on whether we are in the genus 1 or 2 setting; we return to this in later sections.

2.3 The qID Identification Protocol

Let P be a generator of a prime order subgroup of $\mathcal{J}$ of order N, and let $\pm P$ be its image in $\mathcal{K}$. Let $\mathbb{Z}_N^+$ denote the subset of $\mathbb{Z}_N$ with zero least significant bit (where we identify elements of $\mathbb{Z}_N$ with their representatives in $[0, N-1]$). Note that since N is odd, $\texttt{LSB}(-x) = 1 - \texttt{LSB}(x)$ for all $x \in \mathbb{Z}_N^*$. The private key is an element $d \in \mathbb{Z}_N$. Let $Q = [d]P$ and let the public key be $\pm Q$. Now consider the following Schnorr-style identification protocol, which we call qID:

1. The prover sets $r \leftarrow_R \mathbb{Z}_N^*$, $\pm R \leftarrow \pm[r]P$ and sends $\pm R$ to the verifier;
2. The verifier sets $c \leftarrow_R \mathbb{Z}_N^+$ and sends c to the prover;
3. The prover sets $s \leftarrow (r - cd) \bmod N$ and sends s to the verifier;
4. The verifier accepts only if $\pm R \in \{\pm([s]P + [c]Q), \pm([s]P - [c]Q)\}$.

There are some important differences between qID and the basic Schnorr identification protocol in [Sch90].

Scalar multiplications on $\mathcal{K}$. It is well-known that one can use $\mathcal{K}$ to perform the scalar multiplication [OS01; CCS17] within a Schnorr identification or signa-

ture scheme, but with this approach one must always lift back to an element of a group. In contrast, in our scheme this recovery step is not necessary.

Verification on $\mathcal{K}$. The original verification [Sch90] requires checking $R = [s]P + [c]Q$ for some $R, [s]P, [c]Q \in \mathcal{J}$. Working on $\mathcal{K}$, we only have these values up to sign (i. e. $\pm R$, $\pm[s]P$ and $\pm[c]Q$), which is not enough to check that $R = [s]P + [c]Q$. Instead, we only verify whether $\pm R = \pm([s]P \pm [c]Q)$.

Challenge from $\mathbb{Z}_N^+$. A Schnorr protocol using the weaker verification above would not satisfy the special soundness property: although the transcripts $(\pm R, c, s)$ and $(\pm R, -c, s)$ are both valid, they do not allow us to extract a witness. Choosing c from $\mathbb{Z}_N^+$ instead of $\mathbb{Z}$ eliminates this possibility, and allows a security proof (this is the main difference with Hamburg's STROBE [Ham17]).

Proposition 1. *The `qID` identification protocol is a sigma protocol.*

Proof. We prove the required properties (see [HL10, §6]).

Completeness. If the protocol is followed, then $r = s + cd$, and therefore $[r]P = [s]P + [c]Q$ on $\mathcal{J}$. Mapping to $\mathcal{K}$, it follows that $\pm R = \pm([s]P + [c]Q)$.

Special soundness. Let $(\pm R, c_0, s_0)$ and $(\pm R, c_1, s_1)$ be two valid transcripts such that $c_0 \neq c_1$. By verification, each $s_i \equiv \pm r \pm c_i d \bmod N$, so that $s_0 \pm s_1 \equiv (c_0 \pm c_1)\, d \bmod N$, where the signs are chosen to cancel r. Now $c_0 \pm c_1 \not\equiv 0 \bmod N$ because c_0 and c_1 are both in $\mathbb{Z}_N^+$, so we can extract a witness $d \equiv (s_0 \pm s_1)\,(c_0 \pm c_1)^{-1} \bmod N$.

Honest-verifier zero-knowledge. A simulator $\mathcal{S}$ generates $c \leftarrow_R \mathbb{Z}_N^+$ and sets $s \leftarrow_R \mathbb{Z}_N$ and[3] $R \leftarrow [s]P + [c]Q$. If $R = \mathcal{O}$, it restarts. It outputs $(\pm R, c, s)$. As in [PS00, Lemma 5], we let

$$\delta = \left\{(\pm R, c, s) : c \in_R \mathbb{Z}_N^+, r \in_R \mathbb{Z}_N^*, \pm R = \pm[r]P, s = r - cd\right\},$$
$$\delta' = \left\{(\pm R, c, s) : c \in_R \mathbb{Z}_N^+, s \in_R \mathbb{Z}_N, R = [s]P + [c]Q, R \neq \mathcal{O}\right\}$$

be the distributions of honest and simulated signatures, respectively. The elements of δ and δ' are the same. First, consider δ. There are exactly $N - 1$ choices for r, and exactly $(N + 1)/2$ for c; all of them lead to distinct tuples. There are thus $(N^2 - 1)/2$ possible tuples, all of which have probability $2/(N^2 - 1)$ of occurring. Now consider δ'. Again, there are $(N + 1)/2$ choices for c. We have

[3] As we only know Q up to sign, we may need two attempts to construct $\mathcal{S}$.

N choices for s, exactly one of which leads to $R = \mathcal{O}$. Thus, given c, there are $N - 1$ choices for s. We conclude that δ' also contains $(N^2 - 1)/2$ possible tuples, which all have probability $2/(N^2 - 1)$ of occurring.

This concludes the proof. □

2.4 Applying Fiat–Shamir

Applying the Fiat–Shamir transform [FS87] to qID yields a signature scheme qSIG. We will need a hash function $\overline{H} : \{0,1\}^* \to \mathbb{Z}_N^+$, which we define by taking a hash function $H : \{0,1\}^* \to \mathbb{Z}_N$ and then setting $\overline{H}$ by

$$\overline{H}(M) \longmapsto \begin{cases} H(M) & \text{if } \mathtt{LSB}(H(M)) = 0 \\ -H(M) & \text{if } \mathtt{LSB}(H(M)) = 1 \end{cases}.$$

The qSIG signature scheme is defined as follows:

1. To sign a message $M \in \{0,1\}^*$ with private key $d \in \mathbb{Z}_N$ and public key $\pm Q \in \mathcal{K}$, the prover sets $r \leftarrow_R \mathbb{Z}_N^*$, $\pm R \leftarrow \pm[r]R$, $h \leftarrow \overline{H}(\pm R \mid\mid M)$, and $s \leftarrow (r - hd) \bmod N$, and sends $(\pm R \mid\mid s)$ to the verifier.

2. To verify a signature $(\pm R \mid\mid s) \in \mathcal{K} \times \mathbb{Z}_N$ on a message $M \in \{0,1\}^*$ with public key $\pm Q \in \mathcal{K}$, the verifier sets $h \leftarrow \overline{H}(\pm R \mid\mid M)$, $\pm T_0 \leftarrow \pm[s]P$, and $\pm T_1 \leftarrow \pm[h]Q$, and accepts only if $\pm R \in \{\pm(T_0 + T_1), \pm(T_0 - T_1)\}$.

Proposition 2 asserts that the security properties of qID carry over to qSIG.

Proposition 2. *In the random oracle model, if an existential forgery of the qSIG signature scheme under an adaptive chosen message attack has non-negligible probability of success, then the DLP in $\mathcal{J}$ can be solved in polynomial time.*

Proof. This is the standard proof of applying the Fiat–Shamir transform to a sigma protocol: see [PS96, Theorem 13] or [PS00, §3.2]. □

2.5 The qDSA Signature Scheme

Moving towards the real world, we slightly alter the qSIG protocol with some pragmatic choices, following Bernstein et al. [Ber+12].

1. We replace the randomness r by the output of a pseudo-random function, which makes the signatures deterministic.

2. We include the public key $\pm Q$ in the generation of the challenge, to prevent attackers from attacking multiple public keys at the same time.

3. We compress and decompress points on $\mathcal{K}$ where necessary.

The resulting signature scheme, qDSA, is summarized in Algorithm 1.

Unified keys. Signatures are entirely computed and verified on $\mathcal{K}$, which is also the natural setting for Diffie–Hellman key exchange. We can therefore use identical key pairs for Diffie–Hellman and for qDSA signatures. This significantly simplifies the implementation of cryptographic libraries, as we no longer need arithmetic for the two distinct objects $\mathcal{J}$ and $\mathcal{K}$. Technically, there is no reason not to use a single key pair for both key exchange and signing; but one should be very careful in doing so, as using one key across multiple protocols could potentially lead to attacks. The primary interest of this aspect of qDSA is not necessarily in reducing the number of keys, but in unifying key formats and reducing the size of the trusted code base.

Security level. The security reduction to the discrete logarithm problem is almost identical to the case of Schnorr signatures [PS96]. Notably, the challenge space has about half the size ($\mathbb{Z}_N^+$ versus $\mathbb{Z}_N$) while the proof of soundness computes either $s_0 + s_1$ or $s_0 - s_1$. This results in a slightly weaker reduction, as should be expected by moving from $\mathcal{J}$ to $\mathcal{K}$ and by weakening verification. By choosing $\log_2 N \approx 256$ we obtain a scheme with about the same security level as state-of-the-art schemes (e. g. EdDSA combined with Ed25519). This could be made more precise (c. f. [PS00]), but we do not provide this analysis here.

Key and signature sizes. Public keys fit into 32 bytes in both the genus 1 and genus 2 settings. This is standard for Montgomery curves; for Kummer surfaces it requires a new compression technique, which we present in §6. In both cases $\log_2 N < 256$, which means that signatures $(\pm R \,||\, s)$ fit in 64 bytes.

Twist security. Rational points on $\mathcal{K}$ correspond to pairs of points on either $\mathcal{J}$ or its quadratic twist. As opposed to Diffie–Hellman, in qDSA scalar multiplications with secret scalars are *only* performed on the public parameter $\pm P$, which is chosen as the image of large prime order element of $\mathcal{J}$. Therefore $\mathcal{J}$ is not technically required to have a secure twist, unlike in the modern Diffie–Hellman setting. But if $\mathcal{K}$ is also used for key exchange (which is the whole point!), then twist security is crucial. We therefore strongly recommend twist-secure parameters for qDSA implementations.

Algorithm 1. The qDSA signature scheme

Function: KEYPAIR

Input: ()

Output: $(\overline{\pm Q} \mid\mid (d' \mid\mid d''))$: a compressed public key $\overline{\pm Q} \in \{0,1\}^{256}$ where $\pm Q \in \mathcal{K}$, and a private key $(d' \mid\mid d'') \in \left(\{0,1\}^{256}\right)^2$

1 $d \leftarrow \texttt{Random}(\{0,1\}^{256})$
2 $(d' \mid\mid d'') \leftarrow H(d)$
3 $\pm Q \leftarrow \text{LADDER}(d', \pm P)$ ▷ $\pm Q = \pm[d']P$
4 $\overline{\pm Q} \leftarrow \text{COMPRESS}(\pm Q)$
5 **return** $(\overline{\pm Q} \mid\mid (d' \mid\mid d''))$

– –

Function: SIGN

Input: $d', d'' \in \{0,1\}^{256}$, $\overline{\pm Q} \in \{0,1\}^{256}$, $M \in \{0,1\}^*$

Output: $(\overline{\pm R} \mid\mid s) \in \left(\{0,1\}^{256}\right)^2$

6 $r \leftarrow H(d'' \mid\mid M)$
7 $\pm R \leftarrow \text{LADDER}(r, \pm P)$ ▷ $\pm R = \pm[r]P$
8 $\overline{\pm R} \leftarrow \text{COMPRESS}(\pm R)$
9 $h \leftarrow \overline{H}(\overline{\pm R} \mid\mid \overline{\pm Q} \mid\mid M)$
10 $s \leftarrow (r - hd') \bmod N$
11 **return** $(\overline{\pm R} \mid\mid s)$

– –

Function: VERIFY

Input: $M \in \{0,1\}^*$, the compressed public key $\overline{\pm Q} \in \{0,1\}^{256}$, and a putative signature $(\overline{\pm R} \mid\mid s) \in \left(\{0,1\}^{256}\right)^2$

Output: `True` if $(\overline{\pm R} \mid\mid s)$ is a valid signature on M under $\overline{\pm Q}$, `False` otherwise

12 $\pm Q \leftarrow \text{DECOMPRESS}(\overline{\pm Q})$
13 **if** $\pm Q = \perp$ **then return** `False`
14 $h \leftarrow \overline{H}(\overline{\pm R} \mid\mid \overline{\pm Q} \mid\mid M)$
15 $\pm T_0 \leftarrow \text{LADDER}(s, \pm P)$ ▷ $\pm T_0 = \pm[s]P$
16 $\pm T_1 \leftarrow \text{LADDER}(h, \pm Q)$ ▷ $\pm T_1 = \pm[h]Q$
17 $\pm R \leftarrow \text{DECOMPRESS}(\overline{\pm R})$
18 **if** $\pm R = \perp$ **then return** `False`
19 $v \leftarrow \text{CHECK}(\pm T_0, \pm T_1, \pm R)$ ▷ is $\pm R = \pm\,(T_0 \pm T_1)$?
20 **return** v

Hash function. The function H can be any hash function with a security level of at least $\log_2 \sqrt{N}$ bits and at least $2\log_2 N$-bit output. Throughout this chapter we take H to be the extendable output function SHAKE128 [Dwo15] with fixed 512-bit output. This enables us to implicitly use H as a function mapping into either $\mathbb{Z}_N \times \{0,1\}^{256}$ (e. g. Line 2 of Algorithm 1), $\mathbb{Z}_N$ (e. g. Line 6 of Algorithm 1), or $\mathbb{Z}_N^+$ (e. g. Line 9 of Algorithm 1, by combining it with a conditional negation) by appropriately reducing (part of) the output modulo N.

Signature compression. Schnorr already mentions in [Sch90] that signatures $(R \mid\mid s)$ may be compressed to $(H(R \mid\mid Q \mid\mid M) \mid\mid s)$, taking only the first 128 bits of the hash, thus reducing signature size from 64 to 48 bytes. This is possible because we can recompute R from P, Q, s, and $H(R \mid\mid Q \mid\mid M)$. However, on $\mathcal{K}$ we cannot recover $\pm R$ from $\pm P$, $\pm Q$, s, and $H(\pm R \mid\mid \pm Q \mid\mid M)$, so Schnorr's compression technique is not an option for us.

Batching. Proposals for batch signature verification commonly rely on the group structure, verifying random linear combinations of points [Nac+95; Ber+12]. Since $\mathcal{K}$ has no group structure, these batching algorithms are not possible.

Scalar multiplication for verification. Instead of computing the full point $[s]P + [c]Q$ with a two-dimensional multiscalar multiplication operation, we have to compute $\pm[s]P$ and $\pm[c]Q$ separately. As a result we are unable to use standard tricks for speeding up two-dimensional scalar multiplications (e. g. [ElG85]), resulting in increased run-time. On the other hand, it has the benefit of relying on the already implemented LADDER function, mitigating the need for a separate algorithm, and is more memory-friendly. Our implementations show a significant decrease in stack usage, at the cost of a small loss of speed (see §7).

3 Implementing qDSA with Elliptic Curves

Our first concrete instantiation of qDSA uses the Kummer variety of an elliptic curve, which is just the x-line $\mathbb{P}^1$.

3.1 Montgomery Curves

Consider the elliptic curve in Montgomery form $E_{AB}/\mathbb{F}_p : By^2 = x(x^2 + Ax + 1)$, where $A^2 \neq 4$ and $B \neq 0$. The map $E_{AB} \to \mathcal{K} = \mathbb{P}^1$ defined by $P = (X : Y :$

$Z) \mapsto (X : Z)$ gives rise to efficient x-only arithmetic on $\mathbb{P}^1$ (see [Mon87]). We use the LADDER specified in [Dül+15, Alg. 1]. Compression uses the map COMPRESS : $\mathbb{P}^1(\mathbb{F}_p) \to \mathbb{F}_p$ such that $(X : Z) \mapsto XZ^{p-2}$ defined by Bernstein, while decompression is the near-trivial function DECOMPRESS : $\mathbb{F}_p \to \mathbb{P}^1(\mathbb{F}_p)$ given by $x \mapsto (x : 1)$. Note that DECOMPRESS never returns $\perp$, and that

$$\text{DECOMPRESS}(\text{COMPRESS}((X : Z))) = (X : Z)$$

whenever $Z \neq 0$. However, the points $(0 : 1)$ and $(1 : 0)$ should never appear as public keys or signatures.

3.2 Signature Verification

It remains to define the CHECK operation for Montgomery curves. In the final step of verification we are given $\pm R$, $\pm P$, and $\pm Q$ in $\mathbb{P}^1$, and we need to check whether $\pm R \in \{\pm(P+Q), \pm(P-Q)\}$. Proposition 3 reduces this to checking a quadratic relation in the coordinates of $\pm R$, $\pm P$, and $\pm Q$.

Proposition 3. *Write $(X^T : Z^T) = \pm T$ for any T in E_{AB}. If P, Q, and R are points on E_{AB}, then $\pm R \in \{\pm(P+Q), \pm(P-Q)\}$ if and only if*

$$B_{ZZ}(X^R)^2 - 2B_{XZ}X^R Z^R + B_{XX}(Z^R)^2 = 0\,, \tag{1}$$

where

$$\begin{aligned} B_{XX} &= \left(X^P X^Q - Z^P Z^Q\right)^2, \\ B_{XZ} &= \left(X^P X^Q + Z^P Z^Q\right)\left(X^P Z^Q + Z^P X^Q\right) + 2AX^P Z^P X^Q Z^Q, \\ B_{ZZ} &= \left(X^P Z^Q - Z^P X^Q\right)^2. \end{aligned}$$

Proof. Let $S = (X^S : Z^S) = \pm(P+Q)$ and $D = (X^D : Z^D) = \pm(P-Q)$. If we temporarily assume $\pm 0 \neq \pm P \neq \pm Q \neq \pm 0$ and put $x_P = X^P/Z^P$, etc., then the group law on E_{AB} gives us $x_S x_D = (x_P x_Q - 1)^2/(x_P - x_Q)^2$ and $x_S + x_D = 2((x_P x_Q + 1)(x_P + x_Q) + 2Ax_P x_Q)$. Homogenizing, we obtain

$$\left(X^S X^D : X^S Z^D + Z^S X^D : Z^S Z^D\right) = (\lambda B_{XX} : \lambda 2B_{XZ} : \lambda B_{ZZ})\,. \tag{2}$$

One readily verifies that Equation (2) still holds even when the temporary assumption does not (that is, when $\pm P = \pm Q$ or $\pm P = \pm 0$ or $\pm Q = \pm 0$). Having degree 2, the homogeneous polynomial $B_{ZZ}X^2 - B_{XZ}XZ + B_{XX}Z^2$ cuts out two points in

$\mathbb{P}^1$ (which may coincide); by Equation (2), they are $\pm(P+Q)$ and $\pm(P-Q)$, so if $(X^R : Z^R)$ satisfies Equation (1) then it must be one of them. □

Algorithm 2. Checking the verification relation for $\mathbb{P}^1$

Function: CHECK
Input: $\pm P, \pm Q, \pm R = (x : 1)$ in $\mathbb{P}^1$ images of points of $E_{AB}(\mathbb{F}_p)$
Output: `True` if $\pm R \in \{\pm(P+Q), \pm(P-Q)\}$, `False` otherwise
Cost: $8\mathbf{M} + 3\mathbf{S} + 1\mathbf{m_c} + 8\mathbf{a} + 4\mathbf{s}$

```
1 (B_XX, B_XZ, B_ZZ) ← BVALUES(±P, ±Q)
2 if B_XX x^2 − B_XZ x + B_ZZ = 0 then return True
3 else return False
```

Function: BVALUES
Input: $\pm P = (X^P : Z^P)$, $\pm Q = (X^Q : Z^Q)$ in $\mathcal{K}(\mathbb{F}_p)$
Output: $(B_{XX}(\pm P, \pm Q), B_{XZ}(\pm P, \pm Q), B_{ZZ}(\pm P, \pm Q))$ in $\mathbb{F}_p^3$
Cost: $6\mathbf{M} + 2\mathbf{S} + 1\mathbf{m_c} + 7\mathbf{a} + 3\mathbf{s}$

```
4 // See Algorithm 8 and Proposition 3
```

3.3 Using Cryptographic Parameters

We use the elliptic curve $E/\mathbb{F}_p : y^2 = x^3 + 486662x^2 + x$ where $p = 2^{255} - 19$, which is commonly referred to as Curve25519 [Ber06a]. Let $P \in E(\mathbb{F}_p)$ be such that $\pm P = (9 : 1)$. Then P has order $8N$, where

$$N = 2^{252} + 27742317777372353535851937790883648493$$

is prime. The XDBLADD operation requires us to store $(A+2)/4 = 121666$, and we implement optimized multiplication by this constant. In [Ber06a, §3] Bernstein sets and clears some bits of the private key, also referred to as "clamping". This is not necessary in qDSA, but we do it anyway in KEYPAIR for compatibility.

4 Implementing qDSA with Kummer Surfaces

A number of cryptographic protocols that have been successfully implemented with Montgomery curves have seen substantial practical improvements when the curves are replaced with *Kummer surfaces*. From a general point of view, a Kummer surface is the quotient of some genus-2 Jacobian $\mathcal{J}$ by ± 1; geometrically it is a surface in

$\mathbb{P}^3$ with sixteen point singularities, called *nodes*, which are the images in $\mathcal{K}$ of the 2-torsion points of $\mathcal{J}$ (since these are precisely the points fixed by -1). From a cryptographic point of view, a Kummer surface is just a 2-dimensional analogue of the x-coordinate used in Montgomery curve arithmetic. The algorithmic and software aspects of efficient Kummer surface arithmetic have already been covered in great detail elsewhere (see e. g. [Gau07; Ber+14] and Chapter IV). Indeed, the Kummer scalar multiplication algorithms and software that we use in our signature implementation are identical to those described in Chapter IV, and use the cryptographic parameters proposed by Gaudry and Schost [GS12].

This chapter includes two entirely new Kummer algorithms that are essential for our signature scheme: verification relation testing (CHECK, see Algorithm 3) and compression/decompression (COMPRESS and DECOMPRESS, see Algorithms 4 and 5). Both of these new techniques require a fair amount of technical development, which we begin in this section by recalling the basic Kummer equation and constants, and deconstructing the pseudo-doubling operation into a sequence of surfaces and maps that will play important roles later. Once the scene has been set, we will describe our signature verification algorithm in §5 and our point compression scheme in §6. The reader primarily interested in the resulting performance improvements may wish to skip directly to §7 on first reading. The CHECK, COMPRESS, and DECOMPRESS algorithms defined below require the following subroutines:

- The function $\mathcal{M} : \mathbb{F}_p^4 \times \mathbb{F}_p^4 \to \mathbb{F}_p^4$ implements a 4-way parallel multiplication. It takes a pair of vectors (x_1, x_2, x_3, x_4) and (y_1, y_2, y_3, y_4) and returns the coordinate-wise product vector $(x_1x_2 : y_1y_2 : z_1z_2 : t_1t_2)$.
- The function $\mathcal{S} : \mathbb{F}_p^4 \to \mathbb{F}_p^4$ implements a 4-way parallel squaring. It maps (x_1, x_2, x_3, x_4) to $(x_1^2, x_2^2, x_3^2, x_4^2)$.
- The function $\mathcal{H} : \mathbb{F}_p^4 \to \mathbb{F}_p^4$ is a Hadamard transform. It maps (x_1, x_2, x_3, x_4) to $(x_1 + x_2 + x_3 + x_4, x_1 + x_2 - x_3 - x_4, x_1 - x_2 + x_3 - x_4, x_1 - x_2 - x_3 + x_4)$.
- The function $\texttt{Dot} : \mathbb{F}_p^4 \times \mathbb{F}_p^4 \to \mathbb{F}_p$ computes the sum of a 4-way multiplication. Given (x_1, x_2, x_3, x_4) and (y_1, y_2, y_3, y_4), it returns $x_1y_1 + x_2y_2 + x_3y_3 + x_4y_4$.

4.1 Constants

Our Kummer surfaces are defined by four fundamental constants α_1, α_2, α_3, α_4 and four dual constants $\widehat{\alpha}_1$, $\widehat{\alpha}_2$, $\widehat{\alpha}_3$, and $\widehat{\alpha}_4$, which are related by

$$2\widehat{\alpha}_1^2 = \alpha_1^2 + \alpha_2^2 + \alpha_3^2 + \alpha_4^2, \qquad 2\widehat{\alpha}_2^2 = \alpha_1^2 + \alpha_2^2 - \alpha_3^2 - \alpha_4^2,$$
$$2\widehat{\alpha}_3^2 = \alpha_1^2 - \alpha_2^2 + \alpha_3^2 - \alpha_4^2, \qquad 2\widehat{\alpha}_4^2 = \alpha_1^2 - \alpha_2^2 - \alpha_3^2 + \alpha_4^2.$$

Table 1. Relations between our theta constants and others in selected related work.

Source	Fundamental constants	Dual constants
[Gau07; Ber+14]	$(\alpha_1 : \alpha_2 : \alpha_3 : \alpha_4)$	$(\widehat{\alpha}_1 : \widehat{\alpha}_2 : \widehat{\alpha}_3 : \widehat{\alpha}_4)$
[Bos+13]	$(\alpha_1 : \alpha_2 : \alpha_3 : \alpha_4)$	$(\widehat{\mu}_1 : \widehat{\mu}_2 : \widehat{\mu}_3 : \widehat{\mu}_4)$
[Cos11]	$(\mu_1 : \mu_2 : \mu_3 : \mu_4)$	$(\widehat{\mu}_1 : \widehat{\mu}_2 : \widehat{\mu}_3 : \widehat{\mu}_4)$
Chapter IV	$(\mu_1 : \mu_2 : \mu_3 : \mu_4)$	$(\widehat{\mu}_1 : \widehat{\mu}_2 : \widehat{\mu}_3 : \widehat{\mu}_4)$

We require all of the α_i and $\widehat{\alpha}_i$ to be nonzero. The fundamental constants determine the dual constants up to sign, and vice versa. These relations remain true when we exchange the α_i with the $\widehat{\alpha}_i$; we call this "swapping x with $\widehat{x}$" operation "dualizing". To make the symmetry in what follows clear, we define

$$\begin{aligned}
\mu_1 &:= \alpha_1^2, & \epsilon_1 &:= \mu_2\mu_3\mu_4, & \kappa_1 &:= \epsilon_1 + \epsilon_2 + \epsilon_3 + \epsilon_4, \\
\mu_2 &:= \alpha_2^2, & \epsilon_2 &:= \mu_1\mu_3\mu_4, & \kappa_2 &:= \epsilon_1 + \epsilon_2 - \epsilon_3 - \epsilon_4, \\
\mu_3 &:= \alpha_3^2, & \epsilon_3 &:= \mu_1\mu_2\mu_4, & \kappa_3 &:= \epsilon_1 - \epsilon_2 + \epsilon_3 - \epsilon_4, \\
\mu_4 &:= \alpha_4^2, & \epsilon_4 &:= \mu_1\mu_2\mu_3, & \kappa_4 &:= \epsilon_1 - \epsilon_2 - \epsilon_3 + \epsilon_4,
\end{aligned}$$

along with their respective duals $\widehat{\mu}_i$, $\widehat{\epsilon}_i$, and $\widehat{\kappa}_i$. Note that

$$(\epsilon_1 : \epsilon_2 : \epsilon_3 : \epsilon_4) = (1/\mu_1 : 1/\mu_2 : 1/\mu_3 : 1/\mu_4)$$

and $\mu_i\mu_j - \mu_k\mu_l = \widehat{\mu}_i\widehat{\mu}_j - \widehat{\mu}_k\widehat{\mu}_l$ for $\{i,j,k,l\} = \{1,2,3,4\}$. There are many clashing notational conventions for theta constants in the cryptographic Kummer literature; Table 1 provides a dictionary for converting between them.

Our applications use only the squared constants μ_i and $\widehat{\mu}_i$, so only they need be in $\mathbb{F}_p$. In practice we want them to be as "small" as possible, both to reduce the cost of multiplying by them and to reduce the cost of storing them. In fact, it follows from their definition that it is much easier to find simultaneously small μ_i and $\widehat{\mu}_i$ than it is to find simultaneously small α_i and $\widehat{\alpha}_i$ (or a mixture of the two); this is ultimately why we prefer the squared surface for scalar multiplication. We note that if the μ_i are very small, then the ϵ_i and κ_i are also small, and the same goes for their duals. While we will never actually compute with the unsquared constants, we need them to explain what is happening in the background below. Finally, the Kummer surface equations involve some derived constants

$$E := \frac{16\alpha_1\alpha_2\alpha_3\alpha_4\widehat{\mu}_1\widehat{\mu}_2\widehat{\mu}_3\widehat{\mu}_4}{(\mu_1\mu_4 - \mu_2\mu_3)(\mu_1\mu_3 - \mu_2\mu_4)(\mu_1\mu_2 - \mu_3\mu_4)},$$

and

$$F := 2\frac{\mu_1\mu_4 + \mu_2\mu_3}{\mu_1\mu_4 - \mu_2\mu_3}, \qquad G := 2\frac{\mu_1\mu_3 + \mu_2\mu_4}{\mu_1\mu_3 - \mu_2\mu_4}, \qquad H := 2\frac{\mu_1\mu_2 + \mu_3\mu_4}{\mu_1\mu_2 - \mu_3\mu_4},$$

and their duals $\widehat{E}$, $\widehat{F}$, $\widehat{G}$ and $\widehat{H}$. We observe that $E^2 = F^2 + G^2 + H^2 + FGH - 4$ and that $\widehat{E}^2 = \widehat{F}^2 + \widehat{G}^2 + \widehat{H}^2 + \widehat{F}\widehat{G}\widehat{H} - 4$.

4.2 Fast Kummer Surfaces

We compute all of the pseudoscalar multiplications in qDSA on the so-called *squared* Kummer surface

$$\mathcal{K}^{\mathrm{Sqr}} : 4E^2 \cdot X_1X_2X_3X_4 = \begin{pmatrix} X_1^2 + X_2^2 + X_3^2 + X_4^2 - F(X_1X_4 + X_2X_3) \\ - G(X_1X_3 + X_2X_4) - H(X_1X_2 + X_3X_4) \end{pmatrix}^2,$$

which was proposed for factorization algorithms by the Chudnovskys [CC86], then later for Diffie–Hellman by Bernstein [Ber06c]. Since E only appears as a square, $\mathcal{K}^{\mathrm{Sqr}}$ is defined over $\mathbb{F}_p$. The zero point on $\mathcal{K}^{\mathrm{Sqr}}$ is $\pm 0 = (\mu_1 : \mu_2 : \mu_3 : \mu_4)$. In our implementations we used the XDBLADD and Montgomery ladder exactly as they were presented in Algorithm IV.3 and IV.4 (see also Algorithm 9). The doubling XDBL on $\mathcal{K}^{\mathrm{Sqr}}$ is $\pm P = (X_1^P : X_2^P : X_3^P : X_4^P) \mapsto (X_1^{[2]P} : X_2^{[2]P} : X_3^{[2]P} : X_4^{[2]P})$, where

$$\begin{aligned} X_1^{[2]P} &= \epsilon_1(U_1 + U_2 + U_3 + U_4)^2, & X_2^{[2]P} &= \epsilon_2(U_1 + U_2 - U_3 - U_4)^2, \\ X_3^{[2]P} &= \epsilon_3(U_1 - U_2 + U_3 - U_4)^2, & X_4^{[2]P} &= \epsilon_4(U_1 - U_2 - U_3 + U_4)^2, \end{aligned} \tag{3}$$

with

$$\begin{aligned} U_1 &= \widehat{\epsilon}_1(X_1^P + X_2^P + X_3^P + X_4^P)^2, & U_2 &= \widehat{\epsilon}_2(X_1^P + X_2^P - X_3^P - X_4^P)^2, \\ U_3 &= \widehat{\epsilon}_3(X_1^P - X_2^P + X_3^P - X_4^P)^2, & U_4 &= \widehat{\epsilon}_4(X_1^P - X_2^P - X_3^P + X_4^P)^2. \end{aligned}$$

for $\pm P$ with all $X_i^P \neq 0$; more complicated formulæ exist for other $\pm P$ (c. f. §5.1).

4.3 Deconstructing Pseudo-doubling

In Figure 1 we deconstruct the pseudo-doubling on $\mathcal{K}^{\mathrm{Sqr}}$ from §4.2 into a cycle of atomic maps between different Kummer surfaces, which form a sort of hexagon. Starting at any one of the Kummers and doing a complete cycle of these maps carries out pseudo-doubling on that Kummer. Doing a half-cycle from a given Kummer around to its dual computes a $(2, 2)$-isogeny, splitting pseudo-doubling. Six different

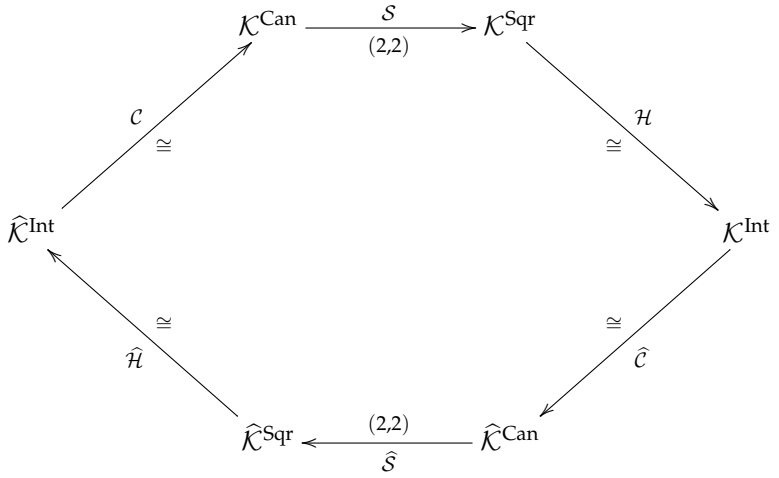


Figure 1. Decomposition of pseudo-doubling on fast Kummer surfaces into a cycle of morphisms. Here, $\mathcal{K}^{\mathrm{Sqr}}$ is the "squared" surface we mostly compute with; $\mathcal{K}^{\mathrm{Can}}$ is the related "canonical" surface; and $\mathcal{K}^{\mathrm{Int}}$ is a new "intermediate" surface which we use in signature verification. (The surfaces $\widehat{\mathcal{K}}^{\mathrm{Sqr}}$, $\widehat{\mathcal{K}}^{\mathrm{Can}}$, and $\widehat{\mathcal{K}}^{\mathrm{Int}}$ are their duals.)

Kummer surfaces may seem like a lot to keep track of (even if there are really only three, together with their duals). However, the new surfaces are important, because they are crucial in deriving our CHECK routine (of course, once the algorithm has been written down, the reader is free to forget about the existence of these other surfaces).

The cycle actually begins one step before $\mathcal{K}^{\mathrm{Sqr}}$, with the *canonical* surface

$$\mathcal{K}^{\mathrm{Can}} : 2E \cdot T_1T_2T_3T_4 = \begin{array}{c} T_1^4 + T_2^4 + T_3^4 + T_4^4 - F(T_1^2T_4^2 + T_2^2T_3^2) \\ - G(T_1^2T_3^2 + T_2^2T_4^2) - H(T_1^2T_2^2 + T_3^2T_4^2)\,. \end{array}$$

This was the model proposed for cryptographic applications by Gaudry in [Gau07]; we call it "canonical" because it is the model arising from a canonical basis of theta functions of level $(2,2)$. Now we can begin our tour around the hexagon, moving from $\mathcal{K}^{\mathrm{Can}}$ to $\mathcal{K}^{\mathrm{Sqr}}$ via the *squaring* map $\mathcal{S}$ (as defined above) which corresponds to a $(2,2)$-isogeny of Jacobians. Moving on from $\mathcal{K}^{\mathrm{Sqr}}$, the *Hadamard transform* isomorphism $\mathcal{H}$ maps $(X_1 : X_2 : X_3 : X_4) \mapsto (Y_1 : Y_2 : Y_3 : Y_4)$, where

$$\begin{aligned} Y_1 &= X_1 + X_2 + X_3 + X_4\,, & Y_2 &= X_1 + X_2 - X_3 - X_4\,, \\ Y_3 &= X_1 - X_2 + X_3 - X_4\,, & Y_4 &= X_1 - X_2 - X_3 + X_4\,. \end{aligned}$$

It takes us into a third kind of Kummer, which we call the *intermediate* surface

$$\mathcal{K}^{\mathrm{Int}} : \frac{2\widehat{E}}{\alpha_1\alpha_2\alpha_3\alpha_4} \cdot Y_1Y_2Y_3Y_4 = \begin{array}{l} \frac{Y_1^4}{\mu_1^2} + \frac{Y_2^4}{\mu_2^2} + \frac{Y_3^4}{\mu_3^2} + \frac{Y_4^4}{\mu_4^2} - \widehat{F}\left(\frac{Y_1^2}{\mu_1}\frac{Y_4^2}{\mu_4} + \frac{Y_2^2}{\mu_2}\frac{Y_3^2}{\mu_3}\right) \\ -\widehat{G}\left(\frac{Y_1^2}{\mu_1}\frac{Y_3^2}{\mu_3} + \frac{Y_2^2}{\mu_2}\frac{Y_4^2}{\mu_4}\right) - \widehat{H}\left(\frac{Y_1^2}{\mu_1}\frac{Y_2^2}{\mu_2} + \frac{Y_3^2}{\mu_3}\frac{Y_4^2}{\mu_4}\right). \end{array}$$

We will use $\mathcal{K}^{\mathrm{Int}}$ for signature verification. Now the *dual scaling* isomorphism

$$\widehat{\mathcal{C}} : (Y_1 : Y_2 : Y_3 : Y_4) \mapsto (\widehat{T}_1 : \widehat{T}_2 : \widehat{T}_3 : \widehat{T}_4) = (Y_1/\widehat{\alpha}_1 : Y_2/\widehat{\alpha}_2 : Y_3/\widehat{\alpha}_3 : Y_4/\widehat{\alpha}_4)$$

takes us into the *dual canonical* surface

$$\widehat{\mathcal{K}}^{\mathrm{Can}} : 2\widehat{E} \cdot \widehat{T}_1\widehat{T}_2\widehat{T}_3\widehat{T}_4 = \begin{array}{l} \widehat{T}_1^4 + \widehat{T}_2^4 + \widehat{T}_3^4 + \widehat{T}_4^4 - \widehat{F}(\widehat{T}_1^2\widehat{T}_4^2 + \widehat{T}_2^2\widehat{T}_3^2) \\ -\widehat{G}(\widehat{T}_1^2\widehat{T}_3^2 + \widehat{T}_2^2\widehat{T}_4^2) - \widehat{H}(\widehat{T}_1^2\widehat{T}_2^2 + \widehat{T}_3^2\widehat{T}_4^2). \end{array}$$

We are now halfway around the hexagon; the return journey is simply the dual of the outbound trip. The *dual squaring* map $\widehat{\mathcal{S}} : \widehat{\mathcal{K}}^{\mathrm{Can}} \to \widehat{\mathcal{K}}^{\mathrm{Sqr}}$ that maps $(\widehat{T}_1 : \widehat{T}_2 : \widehat{T}_3 : \widehat{T}_4) \mapsto (\widehat{T}_1^2 : \widehat{T}_2^2 : \widehat{T}_3^2 : \widehat{T}_4^2) = (\widehat{X}_1 : \widehat{X}_2 : \widehat{X}_3 : \widehat{X}_4)$ is another $(2,2)$-isogeny and carries us into the *dual squared* surface

$$\widehat{\mathcal{K}}^{\mathrm{Sqr}} : 4\widehat{E}^2 \cdot \widehat{X}_1\widehat{X}_2\widehat{X}_3\widehat{X}_4 = \left(\begin{array}{l} \widehat{X}_1^2 + \widehat{X}_2^2 + \widehat{X}_3^2 + \widehat{X}_4^2 - \widehat{F}(\widehat{X}_1\widehat{X}_4 + \widehat{X}_2\widehat{X}_3) \\ -\widehat{G}(\widehat{X}_1\widehat{X}_3 + \widehat{X}_2\widehat{X}_4) - \widehat{H}(\widehat{X}_1\widehat{X}_2 + \widehat{X}_3\widehat{X}_4) \end{array} \right)^2,$$

and the *dual Hadamard transform* $\widehat{\mathcal{H}} : (\widehat{X}_1 : \widehat{X}_2 : \widehat{X}_3 : \widehat{X}_4) \mapsto (\widehat{Y}_1 : \widehat{Y}_2 : \widehat{Y}_3 : \widehat{Y}_4)$, where

$$\begin{aligned} \widehat{Y}_1 &= \widehat{X}_1 + \widehat{X}_2 + \widehat{X}_3 + \widehat{X}_4, & \widehat{Y}_2 &= \widehat{X}_1 + \widehat{X}_2 - \widehat{X}_3 - \widehat{X}_4, \\ \widehat{Y}_3 &= \widehat{X}_1 - \widehat{X}_2 + \widehat{X}_3 - \widehat{X}_4, & \widehat{Y}_4 &= \widehat{X}_1 - \widehat{X}_2 - \widehat{X}_3 + \widehat{X}_4 \end{aligned}$$

takes us into the *dual intermediate* surface

$$\widehat{\mathcal{K}}^{\mathrm{Int}} : \frac{2E}{\alpha_1\alpha_2\alpha_3\alpha_4} \cdot \widehat{Y}_1\widehat{Y}_2\widehat{Y}_3\widehat{Y}_4 = \begin{array}{l} \frac{\widehat{Y}_1^4}{\mu_1^2} + \frac{\widehat{Y}_2^4}{\mu_2^2} + \frac{\widehat{Y}_3^4}{\mu_3^2} + \frac{\widehat{Y}_4^4}{\mu_4^2} - \widehat{F}\left(\frac{\widehat{Y}_1^2}{\mu_1}\frac{\widehat{Y}_4^2}{\mu_4} - \frac{\widehat{Y}_2^2}{\mu_2}\frac{\widehat{Y}_3^2}{\mu_3}\right) \\ -\widehat{G}\left(\frac{\widehat{Y}_1^2}{\mu_1}\frac{\widehat{Y}_3^2}{\mu_3} - \frac{\widehat{Y}_2^2}{\mu_2}\frac{\widehat{Y}_4^2}{\mu_4}\right) - \widehat{H}\left(\frac{\widehat{Y}_1^2}{\mu_1}\frac{\widehat{Y}_2^2}{\mu_2} - \frac{\widehat{Y}_3^2}{\mu_3}\frac{\widehat{Y}_4^2}{\mu_4}\right). \end{array}$$

A final *scaling* isomorphism

$$\mathcal{C} : (\widehat{Y}_1 : \widehat{Y}_2 : \widehat{Y}_3 : \widehat{Y}_4) \mapsto (T_1 : T_2 : T_3 : T_4) = (\widehat{Y}_1/\alpha_1 : \widehat{Y}_2/\alpha_2 : \widehat{Y}_3/\alpha_3 : \widehat{Y}_4/\alpha_4)$$

takes us from $\widehat{\mathcal{K}}^{\mathrm{Int}}$ back to $\mathcal{K}^{\mathrm{Can}}$, where we started.

The canonical surfaces $\mathcal{K}^{\mathrm{Can}}$ resp. $\widehat{\mathcal{K}}^{\mathrm{Can}}$ are only defined over $\mathbb{F}_p(\alpha_1\alpha_2\alpha_3\alpha_4)$ resp. $\mathbb{F}_p(\widehat{\alpha}_1\widehat{\alpha}_2\widehat{\alpha}_3\widehat{\alpha}_4)$, while the scaling isomorphisms $\widehat{\mathcal{C}}$ resp. $\mathcal{C}$ are defined over $\mathbb{F}_p(\widehat{\alpha}_1, \widehat{\alpha}_2, \widehat{\alpha}_3, \widehat{\alpha}_4)$ resp. $\mathbb{F}_p(\alpha_1, \alpha_2, \alpha_3, \alpha_4)$. Everything else is defined over $\mathbb{F}_p$. We confirm that one cycle around the hexagon starting and ending on $\mathcal{K}^{\mathrm{Sqr}}$ computes the pseudo-doubling of Equation (3). Similarly, one cycle around the hexagon starting and ending on $\mathcal{K}^{\mathrm{Can}}$ computes Gaudry's pseudo-doubling from [Gau07, §3.2].

5 Signature Verification on Kummer Surfaces

To verify signatures in the Kummer surface implementation, we need to supply a CHECK algorithm which, given $\pm P$, $\pm Q$, and $\pm R$ on $\mathcal{K}^{\mathrm{Sqr}}$, decides whether $\pm R \in \{\pm(P+Q), \pm(P-Q)\}$. For the elliptic version of qDSA described in §3, we saw that this came down to checking that $\pm R$ satisfied one quadratic relation whose three coefficients were biquadratic forms in $\pm P$ and $\pm Q$. The same principle extends to Kummer surfaces, where the pseudo-group law is similarly defined by biquadratic forms; but since Kummer surfaces are defined in terms of four coordinates (as opposed to the two coordinates of the x-line), this time there are six simple quadratic relations to verify, with a total of ten coefficient forms.

5.1 Biquadratic Forms and Pseudo-addition

Let $\mathcal{K}$ be a Kummer surface. If $\pm P$ is a point on $\mathcal{K}$, then we write $(Z_1^P : Z_2^P : Z_3^P : Z_4^P)$ for its projective coordinates. The classical theory of abelian varieties tells us that there exist biquadratic forms B_{ij} for $1 \le i, j \le 4$ such that for all $\pm P$ and $\pm Q$, if $\pm S = \pm(P+Q)$ and $\pm D = \pm(P-Q)$ then

$$\left(Z_i^S Z_j^D + Z_j^S Z_i^D\right)_{i,j=1}^4 = \lambda \left(B_{ij}(Z_1^P, Z_2^P, Z_3^P, Z_4^P, Z_1^Q, Z_2^Q, Z_3^Q, Z_4^Q)\right)_{i,j=1}^4 \tag{4}$$

where $\lambda \in k^*$ is some common projective factor depending only on the affine representatives chosen for $\pm P$, $\pm Q$, $\pm(P+Q)$ and $\pm(P-Q)$. These biquadratic forms are the foundation of pseudo-addition and doubling laws on $\mathcal{K}$: if the "difference" $\pm D$ is known, then we can use the B_{ij} to compute $\pm S$.

Proposition 4. *Let $\{B_{ij} : 1 \le i, j \le 4\}$ be a set of biquadratic forms on $\mathcal{K} \times \mathcal{K}$ satisfying Equation (4) for all $\pm P$, $\pm Q$, $\pm(P+Q)$, and $\pm(P-Q)$. Then*

$$\pm R = (Z_1^R : Z_2^R : Z_3^R : Z_4^R) \in \{\pm(P+Q), \pm(P-Q)\}$$

if and only if (writing B_{ij} for $B_{ij}(Z_1^P, \dots, Z_4^Q)$) we have

$$B_{jj} \cdot (Z_i^R)^2 - 2B_{ij} \cdot Z_i^R Z_j^R + B_{ii} \cdot (Z_j^R)^2 = 0 \quad \textit{for all } 1 \le i < j \le 4\,. \tag{5}$$

Proof. Looking at Equation (4), we see that the system of six quadratics from Equation (5) cuts out a zero-dimensional degree-2 subscheme of $\mathcal{K}$. That is, the pair of points $\{\pm(P+Q), \pm(P-Q)\}$ (which may coincide). Hence, if $(Z_1^R : Z_2^R : Z_3^R : Z_4^R) = \pm R$ satisfies all of the equations, then it must be one of them. □

5.2 Deriving Efficiently Computable Forms

Proposition 4 is the exact analogue of Proposition 3 for Kummer surfaces. All that we need to turn it into a CHECK algorithm for `qDSA` is an explicit and efficiently computable representation of the B_{ij}. These forms depend on the projective model of the Kummer surface. Hence we write B_{ij}^{Can}, B_{ij}^{Sqr} and B_{ij}^{Int} for the forms on the canonical, squared, and intermediate surfaces. On the canonical surface, the forms B_{ij}^{Can} are classical (see e. g. [Bai62, §2.2]). The on-diagonal forms B_{ii}^{Can} are

$$\begin{aligned} B_{11}^{\mathrm{Can}} &= \frac{1}{4}\Big(\frac{V_1}{\widehat{\mu}_1} + \frac{V_2}{\widehat{\mu}_2} + \frac{V_3}{\widehat{\mu}_3} + \frac{V_4}{\widehat{\mu}_4}\Big)\,, \quad B_{22}^{\mathrm{Can}} = \frac{1}{4}\Big(\frac{V_1}{\widehat{\mu}_1} + \frac{V_2}{\widehat{\mu}_2} - \frac{V_3}{\widehat{\mu}_3} - \frac{V_4}{\widehat{\mu}_4}\Big)\,, \\ B_{33}^{\mathrm{Can}} &= \frac{1}{4}\Big(\frac{V_1}{\widehat{\mu}_1} - \frac{V_2}{\widehat{\mu}_2} + \frac{V_3}{\widehat{\mu}_3} - \frac{V_4}{\widehat{\mu}_4}\Big)\,, \quad B_{44}^{\mathrm{Can}} = \frac{1}{4}\Big(\frac{V_1}{\widehat{\mu}_1} - \frac{V_2}{\widehat{\mu}_2} - \frac{V_3}{\widehat{\mu}_3} + \frac{V_4}{\widehat{\mu}_4}\Big)\,, \end{aligned} \tag{6}$$

where

$$\begin{aligned} V_1 &= \big((T_1^P)^2 + (T_2^P)^2 + (T_3^P)^2 + (T_4^P)^2\big)\big((T_1^Q)^2 + (T_2^Q)^2 + (T_3^Q)^2 + (T_4^Q)^2\big)\,, \\ V_2 &= \big((T_1^P)^2 + (T_2^P)^2 - (T_3^P)^2 - (T_4^P)^2\big)\big((T_1^Q)^2 + (T_2^Q)^2 - (T_3^Q)^2 - (T_4^Q)^2\big)\,, \\ V_3 &= \big((T_1^P)^2 - (T_2^P)^2 + (T_3^P)^2 - (T_4^P)^2\big)\big((T_1^Q)^2 - (T_2^Q)^2 + (T_3^Q)^2 - (T_4^Q)^2\big)\,, \\ V_4 &= \big((T_1^P)^2 - (T_2^P)^2 - (T_3^P)^2 + (T_4^P)^2\big)\big((T_1^Q)^2 - (T_2^Q)^2 - (T_3^Q)^2 + (T_4^Q)^2\big)\,, \end{aligned}$$

while the off-diagonal forms B_{ij} with $\{i,j,k,l\} = \{1,2,3,4\}$ and $i \neq j$ are

$$B_{ij}^{\mathrm{Can}} = \frac{2}{\widehat{\mu}_i\widehat{\mu}_j - \widehat{\mu}_k\widehat{\mu}_l} \begin{pmatrix} \alpha_i\alpha_j\big(T_i^P T_j^P T_i^Q T_j^Q + T_k^P T_l^P T_k^Q T_l^Q\big) \\ -\alpha_k\alpha_l\big(T_i^P T_j^P T_k^Q T_l^Q + T_k^P T_l^P T_i^Q T_j^Q\big) \end{pmatrix}. \tag{7}$$

All of these forms can be efficiently evaluated. The off-diagonal B_{ij}^{Can} have a particularly compact shape, while the symmetry of the on-diagonal B_{ii}^{Can} makes them particularly easy to compute simultaneously. Indeed, that is exactly what we do in

Gaudry's fast pseudo-addition algorithm for $\mathcal{K}^{\mathrm{Can}}$ [Gau07, §3.2].

Ideally, we would like to evaluate the B_{ij}^{Sqr} on $\mathcal{K}^{\mathrm{Sqr}}$, since that is where our inputs $\pm P$, $\pm Q$, and $\pm R$ live. We can compute the B_{ij}^{Sqr} by dualizing the B_{ij}^{Can}, then pulling the $\widehat{B}_{ij}^{\mathrm{Can}}$ on $\widehat{\mathcal{K}}^{\mathrm{Can}}$ back to $\mathcal{K}^{\mathrm{Sqr}}$ via $\widehat{\mathcal{C}} \circ \mathcal{H}$. But while the resulting on-diagonal B_{ii}^{Sqr} maintain the symmetry and efficiency[4] of the B_{ii}^{Can}, the off-diagonal B_{ij}^{Sqr} turn out to be much less pleasant, with less apparent exploitable symmetry. For our applications, this means that evaluating B_{ij}^{Sqr} for $i \neq j$ implies taking a significant hit in terms of stack and code size, not to mention time. We could avoid this difficulty by mapping the inputs of CHECK from $\mathcal{K}^{\mathrm{Sqr}}$ into $\widehat{\mathcal{K}}^{\mathrm{Can}}$, and then evaluating the $\widehat{B}_{ij}^{\mathrm{Can}}$. But this would involve using (and therefore storing) the four large unsquared $\widehat{\alpha}_i$, which is an important drawback.

One can wonder why the nice $\widehat{B}_{ij}^{\mathrm{Can}}$ become so ugly when pulled back to $\mathcal{K}^{\mathrm{Sqr}}$. The map $\widehat{\mathcal{C}} : \mathcal{K}^{\mathrm{Int}} \to \widehat{\mathcal{K}}^{\mathrm{Can}}$ has no impact on the shape or number of monomials, so most of the ugliness is due to the Hadamard transform $\mathcal{H} : \mathcal{K}^{\mathrm{Sqr}} \to \mathcal{K}^{\mathrm{Int}}$. In particular, if we only pull back the $\widehat{B}_{ij}^{\mathrm{Can}}$ as far as $\mathcal{K}^{\mathrm{Int}}$, then the resulting B_{ij}^{Int} retain the nice form of the B_{ij}^{Can} but do not involve the $\widehat{\alpha}_i$. This fact prompts our solution: we map $\pm P$, $\pm Q$, and $\pm R$ through $\mathcal{H}$ onto $\mathcal{K}^{\mathrm{Int}}$, and verify using the forms B_{ij}^{Int}.

Theorem 5. *Up to a common projective factor, the on-diagonal biquadratic forms on the intermediate surface $\mathcal{K}^{\mathrm{Int}}$ are*

$$\begin{aligned}
B_{11}^{\mathrm{Int}} &= \widehat{\mu}_1 \left(\kappa_1 F_1 + \kappa_2 F_2 + \kappa_3 F_3 + \kappa_4 F_4\right) , \\
B_{22}^{\mathrm{Int}} &= \widehat{\mu}_2 \left(\kappa_2 F_1 + \kappa_1 F_2 + \kappa_4 F_3 + \kappa_3 F_4\right) , \\
B_{33}^{\mathrm{Int}} &= \widehat{\mu}_3 \left(\kappa_3 F_1 + \kappa_4 F_2 + \kappa_1 F_3 + \kappa_2 F_4\right) , \\
B_{44}^{\mathrm{Int}} &= \widehat{\mu}_4 \left(\kappa_4 F_1 + \kappa_3 F_2 + \kappa_2 F_3 + \kappa_1 F_4\right) ,
\end{aligned} \tag{8}$$

where

$$\begin{aligned}
F_1 &= P_1Q_1 + P_2Q_2 + P_3Q_3 + P_4Q_4 , & F_2 &= P_1Q_2 + P_2Q_1 + P_3Q_4 + P_4Q_3 , \\
F_3 &= P_1Q_3 + P_3Q_1 + P_2Q_4 + P_4Q_2 , & F_4 &= P_1Q_4 + P_4Q_1 + P_2Q_3 + P_3Q_2 ,
\end{aligned}$$

and where $P_i = \widehat{\epsilon}_i(Y_i^P)^2$ and $Q_i = \widehat{\epsilon}_i(Y_i^Q)^2$ for $1 \leq i \leq 4$. Up to the same common projective factor, the off-diagonal forms for $\{i,j,k,l\} = \{1,2,3,4\}$ are

$$B_{ij}^{\mathrm{Int}} = C \cdot C_{ij} \cdot \left(\widehat{\mu}_k\widehat{\mu}_l\left(Y_{ij}^P - Y_{kl}^P\right)\left(Y_{ij}^Q - Y_{kl}^Q\right) + \left(\widehat{\mu}_i\widehat{\mu}_j - \widehat{\mu}_k\widehat{\mu}_l\right)Y_{kl}^P Y_{kl}^Q\right) , \tag{9}$$

[4] As they should, since they are the basis of the efficient pseudo-addition on $\mathcal{K}^{\mathrm{Sqr}}$!

where $C_{ij} = \widehat{\mu}_i\widehat{\mu}_j(\widehat{\mu}_i\widehat{\mu}_k - \widehat{\mu}_j\widehat{\mu}_l)(\widehat{\mu}_i\widehat{\mu}_l - \widehat{\mu}_j\widehat{\mu}_k)$, $Y_{ij}^P = Y_i^P Y_j^P$, $Y_{ij}^Q = Y_i^Q Y_j^Q$, *and*

$$C = \frac{8(\mu_1\mu_2\mu_3\mu_4)(\widehat{\mu}_1\widehat{\mu}_2\widehat{\mu}_3\widehat{\mu}_4)}{(\widehat{\mu}_1\widehat{\mu}_2 - \widehat{\mu}_3\widehat{\mu}_4)(\widehat{\mu}_1\widehat{\mu}_3 - \widehat{\mu}_2\widehat{\mu}_4)(\widehat{\mu}_1\widehat{\mu}_4 - \widehat{\mu}_2\widehat{\mu}_3)}\,.$$

Proof. By definition, $\widehat{T}_i^S\widehat{T}_j^D + \widehat{T}_j^S\widehat{T}_i^D = \widehat{B}_{ij}^{\mathrm{Can}}(\widehat{T}_1^P, \ldots, \widehat{T}_4^Q)$. Pulling back via $\widehat{\mathcal{C}}$ using $\widehat{T}_i = Y_i/\widehat{\alpha}_i$ yields

$$\begin{aligned} B_{ij}^{\mathrm{Int}}(Y_1^P, \ldots, Y_4^Q) = Y_i^S Y_j^D + Y_j^S Y_i^D &= \widehat{\alpha}_i\widehat{\alpha}_j(\widehat{T}_i^S\widehat{T}_j^D + \widehat{T}_j^S\widehat{T}_i^D) \\ &= \widehat{\alpha}_i\widehat{\alpha}_j \cdot \widehat{B}_{ij}^{\mathrm{Can}}(\widehat{T}_1^P, \ldots, \widehat{T}_4^Q) \\ &= \widehat{\alpha}_i\widehat{\alpha}_j \cdot \widehat{B}_{ij}^{\mathrm{Can}}(Y_1^P/\widehat{\alpha}_1, \ldots, Y_4^Q/\widehat{\alpha}_4)\,. \end{aligned}$$

Dualizing the B_{ij}^{Can} from Equations (6), and (7), we find

$$\begin{aligned} B_{11}^{\mathrm{Int}} &= \widehat{\mu}_1/\left(4\mu_1\mu_2\mu_3\mu_4(\widehat{\mu}_1\widehat{\mu}_2\widehat{\mu}_3\widehat{\mu}_4)^2\right) \cdot \left(\kappa_1F_1 + \kappa_2F_2 + \kappa_3F_3 + \kappa_4F_4\right), \\ B_{22}^{\mathrm{Int}} &= \widehat{\mu}_2/\left(4\mu_1\mu_2\mu_3\mu_4(\widehat{\mu}_1\widehat{\mu}_2\widehat{\mu}_3\widehat{\mu}_4)^2\right) \cdot \left(\kappa_2F_1 + \kappa_1F_2 + \kappa_4F_3 + \kappa_3F_4\right), \\ B_{33}^{\mathrm{Int}} &= \widehat{\mu}_3/\left(4\mu_1\mu_2\mu_3\mu_4(\widehat{\mu}_1\widehat{\mu}_2\widehat{\mu}_3\widehat{\mu}_4)^2\right) \cdot \left(\kappa_3F_1 + \kappa_4F_2 + \kappa_1F_3 + \kappa_2F_4\right), \\ B_{44}^{\mathrm{Int}} &= \widehat{\mu}_4/\left(4\mu_1\mu_2\mu_3\mu_4(\widehat{\mu}_1\widehat{\mu}_2\widehat{\mu}_3\widehat{\mu}_4)^2\right) \cdot \left(\kappa_4F_1 + \kappa_3F_2 + \kappa_2F_3 + \kappa_1F_4\right), \end{aligned}$$

while the off-diagonal forms B_{ij} with $i \neq j$ are

$$B_{ij}^{\mathrm{Int}} = \frac{2}{\widehat{\mu}_k\widehat{\mu}_l(\widehat{\mu}_i\widehat{\mu}_j - \widehat{\mu}_k\widehat{\mu}_l)} \begin{pmatrix} \widehat{\mu}_k\widehat{\mu}_l(Y_{ij}^P - Y_{kl}^P)(Y_{ij}^Q - Y_{kl}^Q) \\ +\,(\widehat{\mu}_i\widehat{\mu}_j - \widehat{\mu}_k\widehat{\mu}_l)Y_{kl}^P Y_{kl}^Q \end{pmatrix}$$

for $\{i,j,k,l\} = \{1,2,3,4\}$. Multiplying all of these forms by a common projective factor of $4(\mu_1\mu_2\mu_3\mu_4)(\widehat{\mu}_1\widehat{\mu}_2\widehat{\mu}_3\widehat{\mu}_4)^2$ eliminates the denominators in the coefficients, and yields the forms of the theorem. □

5.3 Signature Verification

We are now finally ready to implement the CHECK algorithm for $\mathcal{K}^{\mathrm{Sqr}}$. Algorithm 3 does this by applying $\mathcal{H}$ to its inputs, then using the biquadratic forms of Theorem 5. Its correctness is implied by Proposition 4.

5.4 Using Cryptographic Parameters

The surface defined by Gaudry and Schost [GS12] uses $p = 2^{127} - 1$ and $(\mu_1 : \mu_2 : \mu_3 : \mu_4) = (-11 : 22 : 19 : 3)$. We also need the constants $(\widehat{\mu}_1 : \widehat{\mu}_2 : \widehat{\mu}_3 : \widehat{\mu}_4) =$

Algorithm 3. Checking the verification relation for points on $\mathcal{K}^{\mathrm{Sqr}}$

Function: CHECK
Input: $\pm P, \pm Q, \pm R$ in $\mathcal{K}^{\mathrm{Sqr}}(\mathbb{F}_p)$
Output: `True` if $\pm R \in \{\pm(P+Q), \pm(P-Q)\}$, `False` otherwise
Cost: $76\mathbf{M} + 8\mathbf{S} + 88\mathbf{m_c} + 42\mathbf{a} + 42\mathbf{s}$

$(Y^P, Y^Q) \leftarrow (\mathcal{H}(\pm P), \mathcal{H}(\pm Q))$
$(B_{11}, B_{22}, B_{33}, B_{44}) \leftarrow$ BIIVALUES(Y^P, Y^Q)
$Y^R \leftarrow \mathcal{H}(\pm R)$
for (i,j) **in** $\{(1,2),(1,3),(1,4),(2,3),(2,4),(3,4)\}$ **do**
    `LHS` $\leftarrow B_{ii} \cdot (Y_j^R)^2 + B_{jj} \cdot (Y_i^R)^2$
    $B_{ij} \leftarrow$ BIJVALUE$(Y^P, Y^Q, (i,j))$
    `RHS` $\leftarrow 2B_{ij} \cdot Y_i^R \cdot Y_j^R$
    if `LHS` $\neq$ `RHS` **then return** `False`
return `True`

Function: BIIVALUES
Input: $\pm P, \pm Q$ in $\mathcal{K}^{\mathrm{Int}}(\mathbb{F}_p)$
Output: $(B_{ii}^{\mathrm{Int}}(\pm P, \pm Q))_{i=1}^4$ in $\mathbb{F}_p^4$
Cost: $16\mathbf{M} + 8\mathbf{S} + 28\mathbf{m_c} + 24\mathbf{a}$

`// See Algorithm 13 and Theorem 5`

Function: BIJVALUE
Input: $\pm P, \pm Q$ in $\mathcal{K}^{\mathrm{Int}}(\mathbb{F}_p)$ and (i,j) with $1 \le i, j \le 4$ and $i \neq j$
Output: $B_{ij}^{\mathrm{Int}}(\pm P, \pm Q)$ in $\mathbb{F}_p$
Cost: $10\mathbf{M} + 10\mathbf{m_c} + 1\mathbf{a} + 5\mathbf{s}$

`// See Algorithm 12 and Theorem 5`

$(-33 : 11 : 17 : 49)$, $(\kappa_1 : \kappa_2 : \kappa_3 : \kappa_4) = (-4697 : 5951 : 5753 : -1991)$, and $(\widehat{\epsilon}_1 : \widehat{\epsilon}_2 : \widehat{\epsilon}_3 : \widehat{\epsilon}_4) = (-833 : 2499 : 1617 : 561)$.[5] In practice, where these constants are "negative", we reverse their sign and amend the formulæ above accordingly. All of these constants are small, and fit into one or two bytes each (and the $\widehat{\epsilon}_i$ are already stored for use in LADDER). We store one large constant

$$C = \texttt{0x40F50EEFA320A2DD46F7E3D8CDDDA843}$$

and recompute the C_{ij} on the fly.

6 Kummer Point Compression

Our public keys are points on $\mathcal{K}^{\text{Sqr}}$, and each signature includes one point on $\mathcal{K}^{\text{Sqr}}$. Minimizing the space required by Kummer points is therefore essential. A projective Kummer point is composed of four field elements. Normalizing by dividing through by a nonzero coordinate reduces us to three field elements (this can also be achieved using Bernstein's "wrapping" technique [Ber06c], as in [Ber+14] and Chapter IV). But we are talking about Kummer *surfaces* (i. e. two-dimensional objects) so we might hope to compress to two field elements, plus a few bits to enable us to correctly recover the whole Kummer point. This is analogous to elliptic curve point compression, where we compress projective points $(X : Y : Z)$ by normalizing to $(x, y) = (X/Z, Y/Z)$, then storing (x, σ), where σ is a bit indicating the "sign" of y. Decompressing the datum (x, σ) to $(X : Y : Z) = (x : y : 1)$ then requires solving a simple quadratic to recover the correct y-coordinate.

For some reason, no such Kummer point compression method has explicitly appeared in the literature. Bernstein remarked in 2006 that if we compress a Kummer point to two coordinates, then decompression appears to require solving a complicated quartic equation [Ber06c]. This would be much more expensive than computing the single square root required for elliptic decompression. This has perhaps discouraged implementers from attempting to compress Kummer points. But while it may not always be obvious from their defining equations, the classical theory tells us that every Kummer surface is in fact a double cover of $\mathbb{P}^2$, just as elliptic curves are double covers of $\mathbb{P}^1$. We use this principle below to show that we can always compress any Kummer point to two field elements plus two auxiliary bits, and then decompress by solving a quadratic. In our applications, this gives us a convenient

[5] Following the definitions of §4.1, the $\widehat{\mu}_i$ are scaled by -2, the $\widehat{\epsilon}_i$ by $1/11$, and C by $2/11^2$. These changes influence the B^{Int}_{ij}, but only up to the same projective factor.

packaging of Kummer points in exactly 256 bits.

6.1 The General Principle

First, we sketch a general method for Kummer point compression that works for any Kummer presented as a singular quartic surface in $\mathbb{P}^3$. Recall that if N is any point in $\mathbb{P}^3$, then projection away from N defines a map $\pi_N : \mathbb{P}^3 \to \mathbb{P}^2$ sending points in $\mathbb{P}^3$ on the same line through N to the same point in $\mathbb{P}^2$. (The map π_N is only a rational map, and not a morphism; the image of N itself is not well-defined.) Now, let N be a node of a Kummer surface $\mathcal{K}$. That is, N is one of the 16 singular points of $\mathcal{K}$. The restriction of π_N to $\mathcal{K}$ forms a double cover of $\mathbb{P}^2$. By definition, π_N maps the points on $\mathcal{K}$ that lie on the same line through N to the same point of $\mathbb{P}^2$. Now $\mathcal{K}$ has degree 4, so each line in $\mathbb{P}^3$ intersects $\mathcal{K}$ in four points. But since N is a double point of $\mathcal{K}$, every line through N intersects $\mathcal{K}$ at N *twice*, and then in two other points. These two remaining points may be "compressed" to their common image in $\mathbb{P}^2$ under π_N, plus a single bit to distinguish the appropriate preimage.

To make this more concrete, let L_1, L_2, and L_3 be linearly independent linear forms on $\mathbb{P}^3$ vanishing on N. Then N is the intersection of the three planes in $\mathbb{P}^3$ cut out by the L_i. We can now realize the projection $\pi_N \colon \mathcal{K} \to \mathbb{P}^2$ as

$$\pi_N \colon (P_1 : \cdots : P_4) \longmapsto \big(L_1(P_1, \ldots, P_4) : L_2(P_1, \ldots, P_4) : L_3(P_1, \ldots, P_4)\big)\,.$$

Replacing (L_1, L_2, L_3) with another basis of $\langle L_1, L_2, L_3 \rangle$ yields another projection, which corresponds to composing π_N with a linear automorphism of $\mathbb{P}^2$.

If L_1, L_2, and L_3 are chosen as above to vanish on N, and L_4 is any linear form not in $\langle L_1, L_2, L_3 \rangle$, then the fact that π_N is a double cover of the (L_1, L_2, L_3)-plane implies that the defining equation of $\mathcal{K}$ can be rewritten in the form

$$\mathcal{K} : K_2(L_1, L_2, L_3)L_4^2 - 2K_3(L_1, L_2, L_3)L_4 + K_4(L_1, L_2, L_3) = 0$$

where each K_i is a homogeneous polynomial of degree i in L_1, L_2, and L_3. This form, quadratic in L_4, allows us to replace the L_4-coordinate with a single bit indicating the "sign" in the corresponding root of this quadratic. The remaining three coordinates can be normalized to an affine plane point. The net result is a compression to two field elements, plus one bit indicating the normalization, plus another bit to indicate the correct value of L_4.

Remark 6. Stahlke gives a compression algorithm in [Sta04] for points on genus-2 Jacobians in the usual Mumford representation. The first step can be seen as a pro-

jection to the most general model of the Kummer (as in [CF96, Chapter 3]), and then the second is an implicit implementation of the principle above.

6.2 From Squared Kummers to Tetragonal Kummers

We want to define an efficient point compression scheme for $\mathcal{K}^{\mathrm{Sqr}}$. The general principle above makes this possible, but it leaves open the choice of node N and the choice of forms L_i. These choices determine the complexity of the resulting K_i, and hence the cost of evaluating them. This in turn has a non-negligible impact on the time and space required to compress and decompress points, as well as the number of new auxiliary constants that must be stored.

In this section we define a choice of L_i reflecting the special symmetry of $\mathcal{K}^{\mathrm{Sqr}}$. A similar procedure for $\mathcal{K}^{\mathrm{Can}}$ appears in more classical language[6] in [Hud05, §54]. The trick is to distinguish not one node of $\mathcal{K}^{\mathrm{Sqr}}$, but rather the four nodes forming the kernel of the $(2,2)$-isogeny $\widehat{\mathcal{S}} \circ \widehat{\mathcal{C}} \circ \mathcal{H} : \mathcal{K}^{\mathrm{Sqr}} \to \widehat{\mathcal{K}}^{\mathrm{Sqr}}$, namely

$$\begin{aligned} \pm 0 = N_0 &= (\mu_1 : \mu_2 : \mu_3 : \mu_4)\,, & N_1 &= (\mu_2 : \mu_1 : \mu_4 : \mu_3)\,, \\ N_2 &= (\mu_3 : \mu_4 : \mu_1 : \mu_2)\,, & N_3 &= (\mu_4 : \mu_3 : \mu_2 : \mu_1)\,. \end{aligned}$$

We are going to define a coordinate system where these four nodes become the vertices of a coordinate tetrahedron; then, projection onto any three of the four coordinates will represent a projection away from one of these four nodes. The result will be an isomorphic Kummer $\mathcal{K}^{\mathrm{Tet}}$ whose defining equation is quadratic in *all four* of its variables. This might seem like overkill for point compression (quadratic in just one variable would suffice) but it has the agreeable effect of dramatically reducing the overall complexity of the defining equation, saving time and memory in our compression and decompression algorithms.

The key is the matrix identity

$$\begin{pmatrix} \widehat{\kappa}_4 & \widehat{\kappa}_3 & \widehat{\kappa}_2 & \widehat{\kappa}_1 \\ \widehat{\kappa}_3 & \widehat{\kappa}_4 & \widehat{\kappa}_1 & \widehat{\kappa}_2 \\ \widehat{\kappa}_2 & \widehat{\kappa}_1 & \widehat{\kappa}_4 & \widehat{\kappa}_3 \\ \widehat{\kappa}_1 & \widehat{\kappa}_2 & \widehat{\kappa}_3 & \widehat{\kappa}_4 \end{pmatrix} \begin{pmatrix} \mu_1 & \mu_2 & \mu_3 & \mu_4 \\ \mu_2 & \mu_1 & \mu_4 & \mu_3 \\ \mu_3 & \mu_4 & \mu_1 & \mu_2 \\ \mu_4 & \mu_3 & \mu_2 & \mu_1 \end{pmatrix} = 8\widehat{\mu}_1\widehat{\mu}_2\widehat{\mu}_3\widehat{\mu}_4 \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix},$$

[6] The analogous model of $\mathcal{K}^{\mathrm{Can}}$ in [Hud05, §54] is called "the equation referred to a Rosenhain tetrad", whose defining equation "...may be deduced from the fact that Kummer's surface is the focal surface of the congruence of rays common to a tetrahedral complex and a linear complex." Modern cryptographers will understand why we have chosen to give a little more algebraic detail here.

which tells us that the projective isomorphism $\mathcal{T}\colon \mathbb{P}^3 \to \mathbb{P}^3$ defined by

$$\mathcal{T}\colon \begin{pmatrix} X_1 \\ : X_2 \\ : X_3 \\ : X_4 \end{pmatrix} \mapsto \begin{pmatrix} L_1 \\ : L_2 \\ : L_3 \\ : L_4 \end{pmatrix} = \begin{pmatrix} \widehat{\kappa}_4 X_1 + \widehat{\kappa}_3 X_2 + \widehat{\kappa}_2 X_3 + \widehat{\kappa}_1 X_4 \\ : \widehat{\kappa}_3 X_1 + \widehat{\kappa}_4 X_2 + \widehat{\kappa}_1 X_3 + \widehat{\kappa}_2 X_4 \\ : \widehat{\kappa}_2 X_1 + \widehat{\kappa}_1 X_2 + \widehat{\kappa}_4 X_3 + \widehat{\kappa}_3 X_4 \\ : \widehat{\kappa}_1 X_1 + \widehat{\kappa}_2 X_2 + \widehat{\kappa}_3 X_3 + \widehat{\kappa}_4 X_4 \end{pmatrix}$$

maps the four "kernel" nodes to the corners of a coordinate tetrahedron. That is,

$$\begin{aligned} \mathcal{T}(N_0) &= (0:0:0:1)\,, & \mathcal{T}(N_2) &= (0:1:0:0)\,, \\ \mathcal{T}(N_1) &= (0:0:1:0)\,, & \mathcal{T}(N_3) &= (1:0:0:0)\,. \end{aligned}$$

The image of $\mathcal{K}^{\mathrm{Sqr}}$ under $\mathcal{T}$ is the **tetragonal surface**

$$\begin{aligned} \mathcal{K}^{\mathrm{Tet}}\colon 4tL_1L_2L_3L_4 = \quad & r_1^2(L_1L_2 + L_3L_4)^2 + r_2^2(L_1L_3 + L_2L_4)^2 + r_3^2(L_1L_4 + L_2L_3)^2 \\ & - 2r_1s_1((L_1^2 + L_2^2)L_3L_4 + L_1L_2(L_3^2 + L_4^2)) \\ & - 2r_2s_2((L_1^2 + L_3^2)L_2L_4 + L_1L_3(L_2^2 + L_4^2)) \\ & - 2r_3s_3((L_1^2 + L_4^2)L_2L_3 + L_1L_4(L_2^2 + L_3^2)) \end{aligned}$$

where $t = 16\mu_1\mu_2\mu_3\mu_4\widehat{\mu}_1\widehat{\mu}_2\widehat{\mu}_3\widehat{\mu}_4$ and

$$\begin{aligned} r_1 &= (\mu_1\mu_3 - \mu_2\mu_4)(\mu_1\mu_4 - \mu_2\mu_3)\,, & s_1 &= (\mu_1\mu_2 - \mu_3\mu_4)(\mu_1\mu_2 + \mu_3\mu_4)\,, \\ r_2 &= (\mu_1\mu_2 - \mu_3\mu_4)(\mu_1\mu_4 - \mu_2\mu_3)\,, & s_2 &= (\mu_1\mu_3 - \mu_2\mu_4)(\mu_1\mu_3 + \mu_2\mu_4)\,, \\ r_3 &= (\mu_1\mu_2 - \mu_3\mu_4)(\mu_1\mu_3 - \mu_2\mu_4)\,, & s_3 &= (\mu_1\mu_4 - \mu_2\mu_3)(\mu_1\mu_4 + \mu_2\mu_3)\,. \end{aligned}$$

As promised, the defining equation of $\mathcal{K}^{\mathrm{Tet}}$ is quadratic in all four of its variables. For compression, we project away from $\mathcal{T}(\pm 0) = (0:0:0:1)$ onto the $(L_1 : L_2 : L_3)$-plane. Rewriting the defining equation as a quadratic in L_4 gives

$$\mathcal{K}^{\mathrm{Tet}}\colon K_4(L_1, L_2, L_3) - 2K_3(L_1, L_2, L_3)L_4 + K_2(L_1, L_2, L_3)L_4^2 = 0$$

where

$$\begin{aligned} K_2 &= r_3^2L_1^2 + r_2^2L_2^2 + r_1^2L_3^2 - 2\left(r_3s_3L_2L_3 + r_2s_2L_1L_3 + r_1s_1L_1L_2\right)\,, \\ K_3 &= r_1s_1(L_1^2 + L_2^2)L_3 + r_2s_2(L_1^2 + L_3^2)L_2 + r_3s_3(L_2^2 + L_3^2)L_1 \\ &\qquad + (2t - (r_1^2 + r_2^2 + r_3^2))L_1L_2L_3\,, \\ K_4 &= r_3^2L_2^2L_3^2 + r_2^2L_1^2L_3^2 + r_1^2L_1^2L_2^2 - 2\left(r_3s_3L_1 + r_2s_2L_2 + r_1s_1L_3\right)L_1L_2L_3\,. \end{aligned}$$

Lemma 7. *If $(l_1 : l_2 : l_3 : l_4)$ is a point on $\mathcal{K}^{\mathrm{Tet}}$, then*

$$K_2(l_1, l_2, l_3) = K_3(l_1, l_2, l_3) = K_4(l_1, l_2, l_3) = 0 \iff l_1 = l_2 = l_3 = 0\,.$$

Proof. Write k_i for $K_i(l_1, l_2, l_3)$. If $(l_1, l_2, l_3) = 0$ then $(k_2, k_3, k_4) = 0$, because each K_i is non-constant and homogeneous. Conversely, if $(k_2, k_3, k_4) = 0$ and $(l_1, l_2, l_3) \neq 0$ then we could embed a line in $\mathcal{K}^{\mathrm{Tet}}$ via $\lambda \mapsto (l_1 : l_2 : l_3 : \lambda)$; but this is a contradiction, because $\mathcal{K}^{\mathrm{Tet}}$ contains no lines. □

6.3 Compression and Decompression for $\mathcal{K}^{\mathrm{Sqr}}$

In practice, we compress points on $\mathcal{K}^{\mathrm{Sqr}}$ to tuples (l_1, l_2, τ, σ), where l_1 and l_2 are field elements and τ and σ are bits. The recipe is

1. Map $(X_1 : X_2 : X_3 : X_4)$ through $\mathcal{T}$ to a point $(L_1 : L_2 : L_3 : L_4)$ on $\mathcal{K}^{\mathrm{Tet}}$.
2. Compute the unique (l_1, l_2, l_3, l_4) in a form $(*, *, 1, *)$, $(*, 1, 0, *)$, $(1, 0, 0, *)$, or $(0, 0, 0, 1)$ such that $(l_1 : l_2 : l_3 : l_4) = (L_1 : L_2 : L_3 : L_4)$.
3. Compute $k_2 = K_2(l_1, l_2, l_3)$, $k_3 = K_3(l_1, l_2, l_3)$ and $k_4 = K_4(l_1, l_2, l_3)$.
4. Define the bit $\sigma = \mathtt{Sign}(k_2 l_4 - k_3)$, so that (l_1, l_2, l_3, σ) determines l_4. Indeed, $q(l_4) = 0$, where $q(X) = k_2 X^2 - 2k_3 X + k_4$ and Lemma 7 tells us that $q(X)$ is either quadratic, linear, or identically zero.
   - If q is a nonsingular quadratic, then l_4 is determined by (l_1, l_2, l_3) and σ, because $\sigma = \mathtt{Sign}(R)$ where R is the correct square root in the quadratic formula $l_4 = (k_3 \pm \sqrt{k_3^2 - k_2 k_4})/k_2$.
   - If q is singular or linear, then (l_1, l_2, l_3) determines l_4, and σ is redundant.
   - If $q = 0$ then $(l_1, l_2, l_3) = (0, 0, 0)$, so $l_4 = 1$. Again, σ is redundant.

   Setting $\sigma = \mathtt{Sign}(k_2 l_4 - k_3)$ in every case, regardless of whether or not we need it to determine l_4, avoids ambiguity and simplifies code.
5. The normalization in Step 2 forces $l_3 \in \{0, 1\}$, so encode l_3 as a single bit τ.

The datum (l_1, l_2, τ, σ) completely determines (l_1, l_2, l_3, l_4), and thus determines $(X_1 : X_2 : X_3 : X_4) = \mathcal{T}^{-1}((l_1 : l_2 : l_2 : l_4))$. Conversely, the normalization in Step 2 ensures that (l_1, l_2, τ, σ) is uniquely determined by $(X_1 : X_2 : X_3 : X_4)$, and is independent of the representative values of the X_i. Algorithm 4 carries out the compression

process above. The most expensive step is the computation of an inverse in $\mathbb{F}_p$. Algorithm 5 is the corresponding decompression algorithm, and its cost is dominated by computing a square root in $\mathbb{F}_p$.

Algorithm 4. Kummer point compression for $\mathcal{K}^{\mathrm{Sqr}}$

Function: COMPRESS
Input: $\pm P$ in $\mathcal{K}^{\mathrm{Sqr}}(\mathbb{F}_p)$
Output: (l_1, l_2, τ, σ) with $l_1, l_2 \in \mathbb{F}_p$ and $\sigma, \tau \in \{0, 1\}$
Cost: $8\mathbf{M} + 5\mathbf{S} + 12\mathbf{m_c} + 8\mathbf{a} + 5\mathbf{s} + 1\mathbf{I}$

$(L_1, L_2) \leftarrow (\mathtt{Dot}(\pm P, (\widehat{\kappa}_4, \widehat{\kappa}_3, \widehat{\kappa}_2, \widehat{\kappa}_1)), \mathtt{Dot}(\pm P, (\widehat{\kappa}_3, \widehat{\kappa}_4, \widehat{\kappa}_1, \widehat{\kappa}_2)))$
$(L_3, L_4) \leftarrow (\mathtt{Dot}(\pm P, (\widehat{\kappa}_2, \widehat{\kappa}_1, \widehat{\kappa}_4, \widehat{\kappa}_3)), \mathtt{Dot}(\pm P, (\widehat{\kappa}_1, \widehat{\kappa}_2, \widehat{\kappa}_3, \widehat{\kappa}_4)))$
if $L_3 \neq 0$ **then** $(\tau, \lambda) \leftarrow (1, L_3^{-1})$ ▷ Normalize to $(* : * : 1 : *)$
else if $L_2 \neq 0$ **then** $(\tau, \lambda) \leftarrow (0, L_2^{-1})$ ▷ Normalize to $(* : 1 : 0 : *)$
else if $L_1 \neq 0$ **then** $(\tau, \lambda) \leftarrow (0, L_1^{-1})$ ▷ Normalize to $(1 : 0 : 0 : *)$
else $(\tau, \lambda) \leftarrow (0, L_4^{-1})$ ▷ Normalize to $(0 : 0 : 0 : 1)$
$(l_1, l_2, l_4) \leftarrow (L_1 \cdot \lambda, L_2 \cdot \lambda, L_4 \cdot \lambda)$ ▷ $(l_1 : l_2 : \tau : l_4) = (L_1 : L_2 : L_3 : L_4)$
$(k_2, k_3) \leftarrow (K_2(l_1, l_2, \tau), K_3(l_1, l_2, \tau))$ ▷ See Algorithm 14 and 15
$R \leftarrow k_2 \cdot l_4 - k_3$
$\sigma \leftarrow \mathtt{Sign}(R)$
return (l_1, l_2, τ, σ)

Proposition 8. *Algorithms 4 (*COMPRESS*) and 5 (*DECOMPRESS*) satisfy the following properties: given (l_1, l_2, τ, σ) in $\mathbb{F}_p^2 \times \{0, 1\}^2$,* DECOMPRESS *always returns either a valid point in $\mathcal{K}^{\mathrm{Sqr}}(\mathbb{F}_p)$ or $\perp$, and for every $\pm P$ in $\mathcal{K}^{\mathrm{Sqr}}(\mathbb{F}_p)$ we have*

$$\mathrm{DECOMPRESS}(\mathrm{COMPRESS}(\pm P)) = \pm P\,.$$

Proof. In Algorithm 5 we are given (l_1, l_2, τ, σ). We can immediately set $l_3 = \tau$, viewed as an element of $\mathbb{F}_p$. We want to compute an l_4 in $\mathbb{F}_p$, if it exists, such that $k_2 l_4^2 - 2k_3 l_4 + k_4 = 0$ and $\mathtt{Sign}(k_2 l_4 - l_3) = \sigma$ where $k_i = K_i(l_1, l_2, l_3)$. If such an l_4 exists, then we will have a preimage $(l_1 : l_2 : l_3 : l_4)$ in $\mathcal{K}^{\mathrm{Tet}}(\mathbb{F}_p)$, and we can return the decompressed $\mathcal{T}^{-1}((l_1 : l_2 : l_3 : l_4))$ in $\mathcal{K}^{\mathrm{Sqr}}$.

- If $(k_2, k_3) = (0, 0)$, then $k_4 = 2k_3 l_4 - k_2 l_4^2 = 0$, so $l_1 = l_2 = \tau = 0$ by Lemma 7. The only legitimate datum in this form is $(l_1 : l_2 : \tau : \sigma) = (0 : 0 : 0 : \mathtt{Sign}(0))$. If this was the input, then the preimage is $(0 : 0 : 0 : 1)$. Otherwise, we return $\perp$.

- If $k_2 = 0$ but $k_3 \neq 0$, then $k_4 = 2k_3 l_4$ and $(l_1 : l_2 : \tau : l_4) = (2k_3 l_1 : 2k_3 l_2 : 2k_3 \tau :$

Algorithm 5. Kummer point decompression to $\mathcal{K}^{\text{Sqr}}$

Function: DECOMPRESS

Input: (l_1, l_2, τ, σ) with $l_1, l_2 \in \mathbb{F}_p$ and $\tau, \sigma \in \{0, 1\}$

Output: The point $\pm P$ in $\mathcal{K}^{\text{Sqr}}(\mathbb{F}_p)$ such that $\text{COMPRESS}(\pm P) = (l_1, l_2, \tau, \sigma)$, or $\perp$ if no such $\pm P$ exists

Cost: $10\mathbf{M} + 9\mathbf{S} + 18\mathbf{m_c} + 13\mathbf{a} + 8\mathbf{s} + 1\mathbf{E}$

$(k_2, k_3, k_4) \leftarrow (K_2(l_1, l_2, \tau), K_3(l_1, l_2, \tau), K_4(l_1, l_2, \tau))$ ▷ Alg. 14, 15, 16
if $k_2 = 0$ **and** $k_3 = 0$ **then**
  if $(l_1, l_2, \tau, \sigma) \neq (0, 0, 0, \texttt{Sign}(0))$ **then**
    return $\perp$ ▷ Invalid compression
  $L \leftarrow (0, 0, 0, 1)$
else if $k_2 = 0$ **and** $k_3 \neq 0$ **then**
  if $\sigma \neq \texttt{Sign}(-k_3)$ **then**
    return $\perp$ ▷ Invalid compression
  $L \leftarrow (2 \cdot l_1 \cdot k_3, 2 \cdot l_2 \cdot k_3, 2 \cdot \tau \cdot k_3, k_4)$ ▷ $k_4 = 2k_3l_4$
else
  $\Delta \leftarrow k_3^2 - k_2k_4$
  $R \leftarrow \texttt{HasSquareRoot}(\Delta, \sigma)$ ▷ $R = \perp$ or $R^2 = \Delta$, $\texttt{Sign}(R) = \sigma$
  if $R = \perp$ **then**
    return $\perp$ ▷ No preimage in $\mathcal{K}^{\text{Tet}}(\mathbb{F}_p)$
  $L \leftarrow (k_2 \cdot l_1, k_2 \cdot l_2, k_2 \cdot \tau, k_3 + R)$ ▷ $k_3 + R = k_2l_4$
$(X_1, X_2) \leftarrow (\texttt{Dot}(L, (\mu_4, \mu_3, \mu_2, \mu_1)), \texttt{Dot}(L, (\mu_3, \mu_4, \mu_1, \mu_2)))$
$(X_3, X_4) \leftarrow (\texttt{Dot}(L, (\mu_2, \mu_1, \mu_4, \mu_3)), \texttt{Dot}(L, (\mu_1, \mu_2, \mu_3, \mu_4)))$
return $(X_1 : X_2 : X_3 : X_4)$

k_4). The datum is a valid compression unless $\sigma \neq \texttt{Sign}(-k_3)$, in which case we return $\perp$. Otherwise, the preimage is $(2k_3l_1 : 2k_3l_2 : 2k_3\tau : k_4)$.

– If $k_2 \neq 0$, then the quadratic formula tells us that any preimage satisfies $k_2l_4 = k_3 \pm \sqrt{k_3^2 - k_2k_4}$, with the sign determined by $\texttt{Sign}(k_2l_4 - k_3)$. If $k_3^2 - k_2k_4$ is not a square in $\mathbb{F}_p$ then there is no such l_4 in $\mathbb{F}_p$; the input is illegitimate, so we return $\perp$. Otherwise, we have a preimage $(k_2l_1 : k_2l_2 : k_2l_3 : l_3 \pm \sqrt{k_3^2 - k_2k_4})$.

Lines 16 and 17 map the preimage $(l_1 : l_2 : l_3 : l_4)$ in $\mathcal{K}^{\text{Tet}}(\mathbb{F}_p)$ back to $\mathcal{K}^{\text{Sqr}}(\mathbb{F}_p)$ via $\mathcal{T}^{-1}$, yielding the decompressed point $(X_1 : X_2 : X_3 : X_4)$. □

6.4 Using Cryptographic Parameters

Our compression scheme works out particularly nicely for the Gaudry–Schost Kummer over $\mathbb{F}_{2^{127}-1}$. First, since every field element fits into 127 bits, every compressed point fits into exactly 256 bits. Second, the auxiliary constants are small. We have $(\widehat{\kappa}_1 : \widehat{\kappa}_2 : \widehat{\kappa}_3 : \widehat{\kappa}_4) = (-961 : 128 : 569 : 1097)$, each of which fits into well under 16 bits. Computing the polynomials K_2, K_3, K_4 and dividing them all through by 11^2 (which does not change the roots of the quadratic) gives

$$K_2(l_1, l_2, \tau) = (q_5l_1)^2 + (q_3l_2)^2 + (q_4\tau)^2 - 2q_3\big(q_2l_1l_2 + \tau(q_0l_1 - q_1l_2)\big)\,, \tag{10}$$

$$K_3(l_1, l_2, \tau) = q_3\big(q_0(l_1^2 + \tau)l_2 - q_1l_1(l_2^2 + \tau) + q_2(l_1^2 + l_2^2)\tau\big) - q_6q_7l_1l_2\tau\,, \tag{11}$$

$$K_4(l_1, l_2, \tau) = \big((q_3l_1)^2 + (q_5l_2)^2 - 2q_3l_1l_2\big(q_0l_2 - q_1l_1 + q_2\big)\big)\tau + (q_4l_1l_2)^2\,, \tag{12}$$

where $(q_0, \ldots, q_7) = (3575, 9625, 4625, 12259, 11275, 7475, 6009, 43991)$. Each of the q_i fits into 16 bits. In total, the twelve new constants we need for COMPRESS and DECOMPRESS together fit into less than two field elements' worth of space.

7 Implementation

In this section we present the results of the implementation of the scheme on the AVR ATmega and ARM Cortex M0 platforms. We have a total of four implementations. On both platforms we implemented both the Curve25519-based scheme and the scheme based on a fast Kummer surface in genus 2. The benchmarks for the AVR software are obtained from the Arduino MEGA development board containing an ATmega2560 MCU, compiled with GCC v4.8.1. For the Cortex M0, they are measured on the STM32F051R8 MCU on the STMF0Discovery board, compiled with

Table 2. Cycle counts for the four key functions of qDSA at the 128-bit security level on the AVR ATmega and ARM Cortex M0 architectures.

Genus	Function	Ref.	AVR	ARM
1	LADDER	Alg. 6	12 539 098	3 338 554
	CHECK	Alg. 2	46 546	17 044
	COMPRESS	§3.1	1 067 004	270 867
	DECOMPRESS	§3.1	694	102
2	LADDER	Alg. 9	9 624 637	2 683 371
	CHECK[7]	Alg. 3	84 424	24 249
	COMPRESS	Alg. 4	212 374	62 165
	DECOMPRESS	Alg. 5	211 428	62 471

Clang v3.5.0. We refer to the (publicly available) code for more detailed compiler settings. For both Diffie–Hellman and signatures we follow the eBACS API [BLa].

7.1 Core Functionality

The arithmetic of the underlying finite fields is well-studied and optimized, and we do not reinvent the wheel. For field arithmetic in $\mathbb{F}_{2^{255}-19}$ we use the highly optimized functions presented by Hutter and Schwabe [HS13] for the AVR ATmega, and the code from Düll et al. [Dül+15] for the Cortex M0. For arithmetic in $\mathbb{F}_{2^{127}-1}$ we use the functions from Chapter IV, which in turn rely on [HS13] for the AVR ATmega, and on [Dül+15] for the Cortex M0.

The SHAKE128 functions for the ATmega are taken from [Ber+16], while on the Cortex M0 we use a modified version from [AJS16]. Cycle counts for the main functions defined in the rest of this chapter are presented in Table 2. Notably, the LADDER routine is by far the most expensive function. In genus 1 the COMPRESS function is relatively costly (it is essentially an inversion), while in genus 2 CHECK, COMPRESS and DECOMPRESS have only minor impact on the total run-time. More interestingly, as seen in Table 3 and Table 4, the simplicity of operating only on the Kummer variety allows smaller code and less stack usage.

7.2 Comparison to Previous Work

There are not many implementations of full signature and key exchange schemes on microcontrollers. On the other hand, there are implementations of scalar multi-

[7] The implementation decompresses $\pm R$ within CHECK, while Algorithm 3 assumes $\pm R$ to be decompressed. We have subtracted the cost of the DECOMPRESS function once.

Table 3. Performance comparison of the `qDSA` signature scheme against the best implementations, on the AVR ATmega architecture. The code size and stack size are measured in bytes.

Ref.	Object	Func.	Cycles	Stack	Code size[8]
[NLD15]	Ed25519	SIGN	19 047 706	1 473 B	—
		VERIFY	30 776 942	1 226 B	
[Liu+17]	FourQ	SIGN	5 174 800	1 572 B	25 354 B
		VERIFY	11 003 800	4 957 B	33 372 B
This	Curve25519	SIGN	14 067 995	512 B	21 347 B
		VERIFY	25 355 140	644 B	
Chapter IV	Gaudry–Schost $\mathcal{J}$	SIGN	10 404 033	926 B	20 242 B
		VERIFY	16 240 510	992 B	
This	Gaudry–Schost $\mathcal{K}$	SIGN	10 477 347	417 B	17 880 B
		VERIFY	20 423 937	609 B	

plication on elliptic curves. The fastest on our platforms are presented by Düll et al. [Dül+15], and since we are relying on exactly the same arithmetic, we have essentially the same results. Similarly, the records for scalar multiplication on Kummer surfaces are presented in Chapter IV. Since we use the same underlying functions, we have similar results.

More interestingly, we compare the speed and memory usage of signing and verification to best known results of implementations of complete signature schemes. To the best of our knowledge, the only other works are the Ed25519-based scheme by Nascimento et al [NLD15], the FourQ-based scheme (obtaining fast scalar multiplication by relying on easily computable endomorphisms) by Liu et al [Liu+17], and the genus-2 implementation from Chapter IV.

AVR ATmega. As we see in Table 3, our implementation of the scheme based on Curve25519 outperforms the Ed25519-based scheme from [NLD15] in every way. It reduces the number of clock cycles needed for SIGN resp. VERIFY by more than 26% resp. 17%, while reducing stack usage by more than 65% resp. 47%. Code size is not reported in [NLD15]. Comparing against the FourQ implementation of [Liu+17], we see a clear trade-off between speed and size: FourQ has a clear speed advantage, but `qDSA` on Curve25519 requires only a fraction of the stack space.

The implementation based on the Kummer surface of the genus-2 Gaudry–Schost Jacobian does better than the Curve25519-based implementation across the board. Compared to Chapter IV the stack usage of SIGN resp. VERIFY decreases by more

[8] All reported code sizes except those from [Liu+17, Table 6] include support for both signatures and key exchange.

Table 4. Performance comparison of the qDSA signature scheme against the current best implementations, on the ARM Cortex M0 platform. The code size and stack size are measured in bytes.

Ref.	Object	Func.	Cycles	Stack	Code size[9]
This	Curve25519	SIGN	3 889 116	660 B	18 443 B
		VERIFY	6 793 695	788 B	
Chapter IV	Gaudry–Schost $\mathcal{J}$	SIGN	2 865 351	1 360 B	19 606 B
		VERIFY	4 453 978	1 432 B	
This	Gaudry–Schost $\mathcal{K}$	SIGN	2 908 215	580 B	18 064 B
		VERIFY	5 694 414	808 B	

than 54% resp. 38%, while decreasing code size by about 11%. On the other hand, verification is about 26% slower. This is explained by the fact that in Chapter IV the signature is compressed to 48 bytes (following Schnorr's suggestion), which means that one of the scalar multiplications in verification is only half length. Comparing to the FourQ implementation of [Liu+17], again we see a clear trade-off between speed and size, but this time the loss of speed is less pronounced than in the comparison with Curve25519-based qDSA.

ARM Cortex M0. In this case there is no elliptic-curve-based signature scheme to compare to, so we present the first. As we see in Table 4, it is significantly slower than its genus-2 counterpart in this chapter (as should be expected), while using a similar amount of stack and code. The genus-2 signature scheme has similar trade-offs on this platform when compared to the implementation of Chapter IV. The stack usage for SIGN resp. VERIFY is reduced by about 57% resp. 43%, while code size is reduced by about 8%. For the same reasons as above, verification is about 28% slower.

[9] In this chapter 8 448 bytes come from the SHAKE128 implementation, while in Chapter IV we use 6 938 bytes. One could probably reduce this significantly by optimizing the implementation, or by using a more memory-friendly hash function.

A Elliptic Implementation Details

The algorithms in this section complete the description of elliptic qDSA in §3.

A.1 Pseudoscalar Multiplication

The KEYPAIR, SIGN, and VERIFY functions all require LADDER, which we define below. Algorithm 6 describes the scalar pseudomultiplication that we implemented for Montgomery curves, closely following our C reference implementation. To make our LADDER constant-time, we use a *conditional swap* procedure CSWAP. This takes a single bit and a pair of items as arguments, and swaps those items if and only if the bit is 1.

Algorithm 6. The Montgomery ladder for elliptic pseudo-multiplication on $\mathbb{P}^1$, using a combined differential double-and-add (Algorithm 7)

Function: LADDER
Input: $m = \sum_{i=0}^{255} m_i 2^i \in \mathbb{Z}$ and $\pm P = (x : 1) \in \mathbb{P}^1(\mathbb{F}_p)$, $x \neq 0$
Output: $\pm[m]P$
Cost: $1280\mathbf{M} + 1024\mathbf{S} + 256\mathbf{m_c} + 1024\mathbf{a} + 1024\mathbf{s}$

1 $\texttt{prevbit} \leftarrow 0$
2 $(V_0, V_1) \leftarrow ((1:0), \pm P)$
3 **for** $i = 255$ **down to** 0 **do**
4 $\quad(\texttt{bit}, \texttt{prevbit}, \texttt{swap}) \leftarrow (m_i, \texttt{bit}, \texttt{bit} \oplus \texttt{prevbit})$
5 $\quad\texttt{CSWAP}(\texttt{swap}, (V_0, V_1))$
6 $\quad\text{XDBLADD}(V_0, V_1, x)$
7 $\texttt{CSWAP}(\texttt{bit}, (V_0, V_1))$
8 **return** V_0

Algorithm 7 implements XDBLADD for Montgomery curves in the usual way. Note that the assumption that $\pm(P - Q) \notin \{(1:0), (0:1)\}$ implies that XDBLADD will always return the correct result.

A.2 The BVALUES Subroutine for Signature Verification

The elliptic version of the crucial CHECK subroutine of VERIFY (Algorithm 2) used a function BVALUES to calculate the values of the biquadratic forms B_{XX}, B_{XZ}, and B_{ZZ}. This function can be implemented in a number of ways, with different optimizations for speed or stack usage. Algorithm 8 illustrates the approach we used for BVALUES, motivated by simplicity and stack minimization.

Algorithm 7. Combined pseudo-addition and doubling on $\mathbb{P}^1$

Function: XDBLADD
Input: $\pm P = (X^P : Z^P)$ and $\pm Q = (X^Q : Z^Q)$ in $\mathbb{P}^1(\mathbb{F}_q)$, and $x \in \mathbb{F}_q^*$ such that $(x : 1) = \pm(P - Q)$
Output: $(\pm[2]P, \pm(P + Q))$
Cost: $5\mathbf{M} + 4\mathbf{S} + 1\mathbf{m_c} + 4\mathbf{a} + 4\mathbf{s}$

1 $(U_0, U_1) \leftarrow (X^P, Z^P)$
2 $(V_0, V_1) \leftarrow (X^Q, Z^Q)$
3 $(W_0, W_1) \leftarrow (U_0 + U_1, U_0 - U_1)$
4 $(U_0, U_1) \leftarrow (V_0 + V_1, V_0 - V_1)$
5 $(V_0, U_1) \leftarrow (W_0 \cdot U_1, W_1 \cdot U_0)$
6 $(U_0, V_1) \leftarrow (V_0 + U_1, V_0 - U_1)$
7 $(U_0, V_0, V_1) \leftarrow (U_0^2, V_0^2, x \cdot U_0)$
8 $(W_0, U_0) \leftarrow (W_1^2, W_0^2)$
9 $U_1 \leftarrow U_0 - W_0$
10 $U_0 \leftarrow W_0 \cdot U_0$
11 $W_1 \leftarrow \frac{A+2}{4} \cdot U_1$
12 $W_1 \leftarrow W_0 \cdot W_1$
13 $U_1 \leftarrow W_1 \cdot U_1$
14 **return** $\big((U_0, U_1), (V_0, V_1)\big)$

Algorithm 8. Evaluates B_{XX}, B_{XZ}, and B_{ZZ} on $\mathbb{P}^1$

Function: BVALUES
Input: $\pm P = (X^P : Z^P)$, $\pm Q = (X^Q : Z^Q)$ in $\mathcal{K}(\mathbb{F}_p)$
Output: $(B_{XX}(\pm P, \pm Q), B_{XZ}(\pm P, \pm Q), B_{ZZ}(\pm P, \pm Q))$ in $\mathbb{F}_p^3$
Cost: $6\mathbf{M} + 2\mathbf{S} + 1\mathbf{m_c} + 7\mathbf{a} + 3\mathbf{s}$

1 $(T_0, T_1) \leftarrow (X^P \cdot X^Q, Z^P \cdot Z^Q)$
2 $U \leftarrow (T_0 - T_1)^2$
3 $T_0 \leftarrow T_0 + T_1$
4 $(T_1, T_2) \leftarrow (X^P \cdot Z^Q, X^Q \cdot Z^P)$
5 $W \leftarrow (T_1 - T_2)^2$
6 $V \leftarrow T_0 \cdot (T_1 + T_2)$
7 $T_0 \leftarrow 4 \cdot T_1 \cdot T_2$
8 $T_1 \leftarrow 2 \cdot T_0$
9 $T_1 \leftarrow \frac{A+2}{4} \cdot T_1$
10 $V \leftarrow V + T_1 - T_0$
11 **return** (U, V, W)

B Kummer Surface Implementation Details

The algorithms in this section complete the description of Kummer qDSA in §§4-6. They follow our C reference implementation very closely. Recall that we have the following subroutines:

- The function $\mathcal{M} : \mathbb{F}_p^4 \times \mathbb{F}_p^4 \to \mathbb{F}_p^4$ implements a 4-way parallel multiplication. It takes a pair of vectors (x_1, x_2, x_3, x_4) and (y_1, y_2, y_3, y_4) and returns the coordinate-wise product vector $(x_1x_2 : y_1y_2 : z_1z_2 : t_1t_2)$.
- The function $\mathcal{S} : \mathbb{F}_p^4 \to \mathbb{F}_p^4$ implements a 4-way parallel squaring. It maps (x_1, x_2, x_3, x_4) to $(x_1^2, x_2^2, x_3^2, x_4^2)$.
- The function $\mathcal{H} : \mathbb{F}_p^4 \to \mathbb{F}_p^4$ is a Hadamard transform. It maps (x_1, x_2, x_3, x_4) to $(x_1 + x_2 + x_3 + x_4, x_1 + x_2 - x_3 - x_4, x_1 - x_2 + x_3 - x_4, x_1 - x_2 - x_3 + x_4)$.
- The function `Dot` : $\mathbb{F}_p^4 \times \mathbb{F}_p^4 \to \mathbb{F}_p$ computes the sum of a 4-way multiplication. Given (x_1, x_2, x_3, x_4) and (y_1, y_2, y_3, y_4), it returns $x_1y_1 + x_2y_2 + x_3y_3 + x_4y_4$.

B.1 Scalar Pseudomultiplication

The Montgomery LADDER for scalar pseudomultiplication on $\mathcal{K}^{\mathrm{Sqr}}$ is implemented in Algorithm 9, replicating the approach in Chapter IV. It relies on the XWRAP and XDBLADD functions, implemented in Algorithm 10 respectively 11. The function XWRAP takes a Kummer point $\pm P$ in $\mathcal{K}^{\mathrm{Sqr}}(\mathbb{F}_p)$ and returns w_2, w_3, and w_4 in $\mathbb{F}_p$ such that $(1 : w_2 : w_3 : w_4) = (1/X_1^P : 1/X_2^P : 1/X_3^P : 1/X_4^P)$. The resulting values are required in every XDBLADD within LADDER; the idea is to compute them once with a single inversion at the start of the procedure, thus avoiding further expensive inversions. We note that this "wrapped" form of the point $\pm P$ was previously used as a compressed form for Kummer point transmission, but since it requires three full field values it is far from an optimal compression.

B.2 Subroutines for Signature Verification

The crucial CHECK function for $\mathcal{K}^{\mathrm{Sqr}}$ (Algorithm 3) calls subroutines BIIVALUES and BIJVALUE to compute the values of the biquadratic forms on $\mathcal{K}^{\mathrm{Int}}$. Algorithm 12 and 13 are our simple implementations of these functions. We choose to only store the four constants $\widehat{\mu}_1$, $\widehat{\mu}_2$, $\widehat{\mu}_3$ and $\widehat{\mu}_4$, but clearly one can gain some efficiency by pre-computing more constants (e. g. $\widehat{\mu}_1\widehat{\mu}_2$, $\widehat{\mu}_1\widehat{\mu}_4 - \widehat{\mu}_2\widehat{\mu}_3$, etc.). As the speed of this operation is not critical, it allows us to reduce the number of necessary constants.

Algorithm 9. The Montgomery ladder for pseudomultiplication on $\mathcal{K}^{\mathrm{Sqr}}$, based on a combined differential double-and-add (Algorithm 11)

Function: LADDER
Input: $m = \sum_{i=0}^{255} m_i 2^i \in \mathbb{Z}$ and $\pm P \in \mathcal{K}^{\mathrm{Sqr}}(\mathbb{F}_p)$
Output: $\pm[m]P$
Cost: $1799\mathbf{M} + 3072\mathbf{S} + 3072\mathbf{m_c} + 4096\mathbf{a} + 4096\mathbf{s} + \mathbf{I}$

1 $\texttt{prevbit} \leftarrow 0$
2 $W \leftarrow \mathrm{XWRAP}(\pm P)$
3 $(V_0, V_1) \leftarrow ((\mu_1 : \mu_2 : \mu_3 : \mu_4), \pm P)$
4 **for** $i = 255$ **down to** 0 **do**
5 $\quad(\texttt{bit}, \texttt{prevbit}, \texttt{swap}) \leftarrow (m_i, \texttt{bit}, \texttt{bit} \oplus \texttt{prevbit})$
6 $\quad\texttt{CSWAP}(\texttt{swap}, (V_0, V_1))$
7 $\quad\mathrm{XDBLADD}(V_0, V_1, W)$
8 $\texttt{CSWAP}(\texttt{bit}, (V_0, V_1))$
9 **return** V_0

Algorithm 10. Precomputes inverted Kummer point coordinates

Function: XWRAP
Input: $\pm P \in \mathcal{K}^{\mathrm{Sqr}}(\mathbb{F}_p)$
Output: $(w_2, w_3, w_4) \in \mathbb{F}_p^3$ such that $(1 : w_2 : w_3 : w_4) = (1/X_1^P : 1/X_2^P : 1/X_3^P : 1/X_4^P)$
Cost: $7\mathbf{M} + \mathbf{I}$

1 $V_1 \leftarrow X_2^P \cdot X_3^P$
2 $V_2 \leftarrow X_1^P / (V_1 \cdot X_4^P)$
3 $V_3 \leftarrow V_2 \cdot X_4^P$
4 **return** $(V_3 \cdot X_3, V_3 \cdot X_2, V_1 \cdot V_2)$

Algorithm 11. Combined pseudo-addition and doubling on $\mathcal{K}^{\mathrm{Sqr}}$

Function: XDBLADD
Input: $\pm P, \pm Q$ in $\mathcal{K}^{\mathrm{Sqr}}(\mathbb{F}_p)$, and $(w_2, w_3, w_4) = \textsc{xwrap}(\pm(P-Q))$ in $\mathbb{F}_p^3$
Output: $(\pm[2]P, \pm(P+Q)) \in \mathcal{K}^{\mathrm{Sqr}}(\mathbb{F}_p)^2$
Cost: $7\mathbf{M} + 12\mathbf{S} + 12\mathbf{m_c} + 16\mathbf{a} + 16\mathbf{s}$

$(V_1, V_2) \leftarrow (\mathcal{H}(V_1), \mathcal{H}(V_2))$
$(V_1, V_2) \leftarrow (\mathcal{S}(V_1), \mathcal{M}(V_1, V_2))$
$(V_1, V_2) \leftarrow (\mathcal{M}(V_1, (\widehat{\epsilon}_1, \widehat{\epsilon}_2, \widehat{\epsilon}_3, \widehat{\epsilon}_4)), \mathcal{M}(V_2, (\widehat{\epsilon}_1, \widehat{\epsilon}_2, \widehat{\epsilon}_3, \widehat{\epsilon}_4)))$
$(V_1, V_2) \leftarrow (\mathcal{H}(V_1), \mathcal{H}(V_2))$
$(V_1, V_2) \leftarrow (\mathcal{S}(V_1), \mathcal{S}(V_2))$
$(V_1, V_2) \leftarrow (\mathcal{M}(V_1, (\epsilon_1, \epsilon_2, \epsilon_3, \epsilon_4))), \mathcal{M}(V_2, (1, w_2, w_3, w_4))))$
return (V_1, V_2)

The four values of B_{11}, B_{22}, B_{33}, and B_{44} are computed simultaneously, since many of the intermediate operands are shared (as is clear from Equation (8)).

Algorithm 12. Evaluates *one* of the off-diagonal B_{ij} on $\mathcal{K}^{\mathrm{Int}}$

Function: BIJVALUE
Input: $\pm P, \pm Q$ in $\mathcal{K}^{\mathrm{Int}}(\mathbb{F}_p)$ and (i, j) such that $\{i, j, k, l\} = \{1, 2, 3, 4\}$
Output: $B_{ij}^{\mathrm{Int}}(\pm P, \pm Q)$ in $\mathbb{F}_p$
Cost: $10\mathbf{M} + 10\mathbf{m_c} + 1\mathbf{a} + 5\mathbf{s}$

$(V_0, V_1, V_2, V_3) \leftarrow (Y_i^P \cdot Y_j^P, Y_k^P \cdot Y_l^P, Y_i^Q \cdot Y_j^Q, Y_k^Q \cdot Y_l^Q)$
$(V_0, V_2) \leftarrow (V_0 - V_1, V_2 - V_3)$
$(V_0, V_1) \leftarrow (V_0 \cdot V_2, V_1 \cdot V_3)$
$(V_0, V_1) \leftarrow (V_0 \cdot \widehat{\mu}_k \widehat{\mu}_l, V_1 \cdot (\widehat{\mu}_i \widehat{\mu}_j - \widehat{\mu}_k \widehat{\mu}_l))$
$V_0 \leftarrow V_0 + V_1$
$V_0 \leftarrow V_0 \cdot \widehat{\mu}_i \widehat{\mu}_j (\widehat{\mu}_i \widehat{\mu}_k - \widehat{\mu}_j \widehat{\mu}_l)(\widehat{\mu}_i \widehat{\mu}_l - \widehat{\mu}_j \widehat{\mu}_k)$
$V_0 \leftarrow V_0 \cdot C$
return V_0

B.3 Subroutines for Compression and Decompression

The COMPRESS and DECOMPRESS functions in Algorithms 4 and 5 require the evaluation of the polynomials K_2, K_3, and K_4. We used the simple strategy in Algorithms 14, 15, and 16 (GET_K2, GET_K3, and GET_K4 respectively), which prioritizes low stack usage over speed (which is again not critical here).

Algorithm 13. Evaluates B_{11}, B_{22}, B_{33}, and B_{44} on $\mathcal{K}^{\text{Int}}$

Function: BIIVALUES
Input: $\pm P, \pm Q$ in $\mathcal{K}^{\text{Int}}(\mathbb{F}_p)$
Output: $(B_{ii}^{\text{Int}}(\pm P, \pm Q))_{i=1}^{4}$ in $\mathbb{F}_p^4$
Cost: $16\mathbf{M} + 8\mathbf{S} + 28\mathbf{m_c} + 24\mathbf{a}$

1 $(V, W) \leftarrow (\pm P, \pm Q)$
2 $(V, W) \leftarrow (\mathcal{S}(V), \mathcal{S}(W))$
3 $(V, W) \leftarrow (\mathcal{M}(V, (\widehat{\epsilon}_1, \widehat{\epsilon}_2, \widehat{\epsilon}_3, \widehat{\epsilon}_4)), \mathcal{M}(W, (\widehat{\epsilon}_1, \widehat{\epsilon}_2, \widehat{\epsilon}_3, \widehat{\epsilon}_4)))$
4 $U \leftarrow \begin{pmatrix} \texttt{Dot}(V, (W_1, W_2, W_3, W_4)), \texttt{Dot}(V, (W_2, W_1, W_4, W_3)), \\ \texttt{Dot}(V, (W_3, W_4, W_1, W_2)), \texttt{Dot}(V, (W_4, W_3, W_2, W_1)) \end{pmatrix}$
5 $V \leftarrow \begin{pmatrix} \texttt{Dot}(U, (\widehat{\kappa}_1, \widehat{\kappa}_2, \widehat{\kappa}_3, \widehat{\kappa}_4)), \texttt{Dot}(U, (\widehat{\kappa}_2, \widehat{\kappa}_1, \widehat{\kappa}_4, \widehat{\kappa}_3)), \\ \texttt{Dot}(U, (\widehat{\kappa}_3, \widehat{\kappa}_4, \widehat{\kappa}_1, \widehat{\kappa}_2)), \texttt{Dot}(U, (\widehat{\kappa}_4, \widehat{\kappa}_3, \widehat{\kappa}_2, \widehat{\kappa}_1)) \end{pmatrix}$
6 $V \leftarrow \mathcal{M}(V, (\widehat{\mu}_1, \widehat{\mu}_2, \widehat{\mu}_3, \widehat{\mu}_4))$
7 **return** V

Algorithm 14. Evaluates the polynomial K_2 at (l_1, l_2, τ)

Function: GET_K2
Input: (l_1, l_2, τ) with $l_1, l_2 \in \mathbb{F}_p$ and $\tau \in \{0, 1\}$
Output: $K_2(l_1, l_2, \tau)$ in $\mathbb{F}_p$ as in Equation (10)
Cost: $1\mathbf{M} + 3\mathbf{S} + 6\mathbf{m_c} + 4\mathbf{a} + 2\mathbf{s}$

1 $V \leftarrow l_1 \cdot q_2$
2 $V \leftarrow l_2 \cdot V$
3 **if** $\tau = 1$ **then**
4 $\quad W \leftarrow l_1 \cdot q_0$
5 $\quad V \leftarrow V + W$
6 $\quad W \leftarrow l_2 \cdot q_1$
7 $\quad V \leftarrow V - W$
8 **end if**
9 $V \leftarrow V \cdot q_3$
10 $V \leftarrow V + V$
11 $W \leftarrow l_1 + q_5$
12 $W \leftarrow W^2$
13 $V \leftarrow W - V$
14 $W \leftarrow l_2 \cdot q_3$
15 $W \leftarrow W^2$
16 $V \leftarrow W + V$
17 **if** $\tau = 1$ **then**
18 $\quad W \leftarrow q_4^2$
19 $\quad V \leftarrow W + V$
20 **end if**
21 **return** V

Algorithm 15. Evaluates the polynomial K_3 at (l_1, l_2, τ)

Function: GET_K3
Input: (l_1, l_2, τ) with $l_1, l_2 \in \mathbb{F}_p$ and $\tau \in \{0, 1\}$
Output: $K_3(l_1, l_2, \tau)$ in $\mathbb{F}_p$ as in Equation (11)
Cost: $3\mathbf{M} + 2\mathbf{S} + 6\mathbf{m_c} + 4\mathbf{a} + 2\mathbf{s}$

$U \leftarrow l_2^2$
$V \leftarrow l_1^2$
if $\tau = 1$ **then**
  $W \leftarrow U + V$
  $W \leftarrow W \cdot q_2$
  $U \leftarrow U + 1$
  $V \leftarrow V + 1$
end if
$U \leftarrow U \cdot l_1$
$V \leftarrow V \cdot l_2$
$U \leftarrow U \cdot q_1$
$V \leftarrow V \cdot q_0$
$V \leftarrow V - U$
if $\tau = 1$ **then**
  $V \leftarrow V + W$
end if
$V \leftarrow V \cdot q_3$
if $\tau = 1$ **then**
  $U \leftarrow l_1 \cdot l_2$
  $U \leftarrow U \cdot q_6$
  $U \leftarrow U \cdot q_7$
  $V \leftarrow V - U$
end if
return V

Algorithm 16. Evaluates the polynomial K_4 at (l_1, l_2, τ)

Function: GET_K4
Input: (l_1, l_2, τ) with $l_1, l_2 \in \mathbb{F}_p$ and $\tau \in \{0, 1\}$
Output: $K_4(l_1, l_2, \tau)$ in $\mathbb{F}_p$ as in Equation (12)
Cost: $3\mathbf{M} + 3\mathbf{S} + 6\mathbf{m_c} + 4\mathbf{a} + 2\mathbf{s}$

if $\tau = 1$ **then**
  $W \leftarrow l_2 \cdot q_0$
  $V \leftarrow l_1 \cdot q_1$
  $W \leftarrow W - V$
  $W \leftarrow W + q_2$
  $W \leftarrow W \cdot l_1$
  $W \leftarrow W \cdot l_2$
  $W \leftarrow W \cdot q_3$
  $W \leftarrow W + W$
  $V \leftarrow l_1 \cdot q_3$
  $V \leftarrow V^2$
  $W \leftarrow V - W$
  $V \leftarrow l_2 \cdot q_5$
  $V \leftarrow V^2$
  $W \leftarrow V + W$
end if
$V \leftarrow l_1 \cdot q_4$
$V \leftarrow V \cdot l_2$
$V \leftarrow V^2$
if $\tau = 1$ **then**
  $V \leftarrow V + W$
end if
return V

Chapter VI

On Kummer Lines with Full Rational 2-torsion and Their Usage in Cryptography

A paper by Karati and Sarkar [KS17] has pointed out the potential for Kummer lines in genus one, by observing that its SIMD-friendly arithmetic is competitive with the status quo. A more recent preprint explores the connection with (twisted) Edwards curves. In this chapter we extend this work and significantly simplify their treatment. We show that their Kummer line is the x-line of a Montgomery curve translated by a point of order two, and exhibit a natural isomorphism to a twisted Edwards curve. Moreover, we show that the Kummer line presented by Gaudry and Lubicz can be obtained via the action of a point of order two on the y-line of an Edwards curve. The maps connecting these curves and lines are all very simple. As a result, a cryptographic implementation can use the arithmetic that is optimal for its instruction set at negligible cost.

1 Introduction

A decade after the introduction of public-key cryptography by Diffie and Hellman [DH76] it was observed (independently) by Miller [Mil86] and Koblitz [Kob87] that one can instantiate protocols based on the hardness of the discrete logarithm problem with the group of rational points of an elliptic curve E defined over a finite field.

Moreover, it was immediately noted by Miller that one can do a full key exchange by solely relying on the line of x-coordinates of points. That is, one can identify points with their inverses and as a result only work with points up to sign. In other words, one can work on the corresponding Kummer line $\mathcal{K} = E/\{\pm 1\}$, possibly simplifying the arithmetic. One can also directly use $\mathcal{K}$ for digital signatures very efficiently with the qDSA scheme (c. f. §V.2). In short, Kummer lines are a very interesting topic of study from a cryptographic perspective.

Because a reduction in the number of field operations needed for a scalar multiplication directly affects the efficiency of the cryptographic scheme, there have been multiple proposals for Kummer lines. Probably the most available example is Curve25519 [Ber06a], which is the Kummer line of a Montgomery curve. One can show that every Montgomery curve is birationally equivalent to a twisted Edwards curve [Ber+08, Theorem 3.2], which currently needs the least number of field operations to perform group operations [His+08] and underlies the very efficient FourQ curve [CL15]. As a result, the Kummer lines of Montgomery and twisted Edwards curves are strongly related, and one can move easily from one to the other [Ber+08; CGF08]. Through the usage of theta functions Gaudry and Lubicz [GL09, §6] derived yet another Kummer line. We shall refer to this as the *canonical* Kummer line, following the terminology of the genus-2 analogue presented in §V.4. By squaring its coefficients we arrive on a different variety, which we refer to as the *squared* Kummer line (again c. f. the genus-2 analogue [CC86; Ber06c]). Although Gaudry and Lubicz only presented arithmetic on the canonical line, the differential addition formulae on the squared Kummer line are well-known [BLb]. The squared Kummer line has the advantage that it is easier to find suitable small parameters, and it was shown by Karati and Sarkar [KS17] that its arithmetic leads to very efficient implementations when single-instruction multiple-data (SIMD) instructions are available. In a follow-up paper [KS19] the same authors present connections to twisted Edwards curves. This requires the associated Legendre curve to be put in Montgomery form or have a rational point of order 4, or otherwise relies on the usage of a 2-isogeny. Consequently, there are case distinctions and one must deal with the doubling induced by moving through a 2-isogeny and its dual. In [KS19, Table 7] they present the possibility of birational maps and isogenies between the Legendre form for certain choices of small constants.

In this chapter we significantly simplify the connections between the various Kummer lines. Since the field of definition of the canonical and squared Kummer lines corresponds to their rational 2-torsion, we shall assume all points of order 2 to be rational. In that case, we show that the squared (resp. canonical) Kummer

arises as the x-line (resp. y-line) of a Montgomery (resp. Edwards) curve translated by a suitable point of order 2. Moreover, a third Kummer line (referred to as the *intermediate* Kummer) appears as the y-line of a *twisted* Edwards curve via a translation by a point of order 2. These observations induce very simple isomorphisms between them. Furthermore, the respective translations by a point of order 2 lead to fast isomorphisms (in fact, involutions) with the well-known x-lines (or y-lines) of Montgomery, Edwards and twisted Edwards curves. As a result, we unify the most popular Kummer lines in the literature and conclude that their usage is completely interchangeable on an implementation level. For example, we can directly use the squared Kummer line in the qDSA scheme through its connection with a Montgomery curve (see §V.3). Moreover, although there exist efficient implementations of Montgomery curves based on 4-way SIMD parallelization by optimizing the field arithmetic [FHL15], it is unclear how to optimally parallelize instructions 4-way on the level of the x-line [Cho15]. This is straightforward on the squared Kummer line, and therefore by extension becomes trivial on Montgomery curves with full rational 2-torsion by moving through the isomorphism. Of course, if desired, one can also do arithmetic on the full group of points of the twisted Edwards curve (as also noted by Karati and Sarkar [KS19]). In particular, we provide isomorphic Montgomery and twisted Edwards models for all the Kummer lines present in [KS19, Table 7] (see Table 1 in §3.2).

Organization. We establish notation in §2, while the main contribution appears as Theorem 5 in §3. In §4 we count the number of Kummer lines up to isomorphism.

2 Notation

We begin by fixing notation, referring back for preliminaries to §I.2.1. For simplicity and readability of this chapter we recall some notions.

As usual, we let k be a field. Throughout the whole document we assume that $\mathrm{char}(k) \neq 2$ unless mentioned otherwise. An elliptic curve is a smooth projective curve $\mathcal{C}$ of genus 1 with specified base point $\mathcal{O}_{\mathcal{C}}$. We denote it by $(\mathcal{C}, \mathcal{O}_{\mathcal{C}})$, or simply by $\mathcal{C}$ if the base point is implicit. For example, the *Weierstrass form*[1] $y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \subset \mathbb{P}^2$ has implicit base point $\mathcal{O} = (0 : 1 : 0)$, but the curves we

[1] We shall in many cases talk about *affine* curves and maps for simplicity, but always mean their projective counterparts. This depends on the particular embedding of the affine curve into projective space, but the embedding should be clear from context. In particular, we always embed Montgomery curves into $\mathbb{P}^2$ while (twisted) Edwards curves are embedded into $\mathbb{P}^3$ (as opposed to $\mathbb{P}^1 \times \mathbb{P}^1$, which is also commonplace).

consider in this chapter are not necessarily in this standard model.

Montgomery form. For $A, B \in k$ such that $B(A^2 - 4) \neq 0$, we denote by

$$M/k : By^2 = x^3 + Ax^2 + x$$

an elliptic curve in Montgomery form, with base point $\mathcal{O}_M = (0 : 1 : 0)$. We have the usual projection map $\mathbf{x} : M \to \mathbb{P}^1$ mapping $(X : Y : Z)$ onto $(X : Z)$ on its Kummer line denoted by $\mathcal{K}_M^{\mathcal{O}_M}$, inheriting a pseudo-group structure from $(M, \mathcal{O}_M)$.

Suppose that $T \in (M, \mathcal{O}_M)$ is a point such that $[2]T = \mathcal{O}_M$. Then the translation-by-T map $\tau_T : (M, \mathcal{O}_M) \to (M, T)$ that maps $P \mapsto P + T$ is an isomorphism of elliptic curves. Moreover, the map $\mathbf{x}$ is again well-defined on M and we denote its Kummer line by $\mathcal{K}_M^T$. Note that $\mathcal{K}_M^{\mathcal{O}_M} \cong \mathcal{K}_M^T$ as *algebraic varieties* (i. e. they are the projective line $\mathbb{P}^1$), but that they have a different pseudo-group structure. That is, we have a commutative diagram

$$\begin{array}{ccc} (M, \mathcal{O}_M) & \xleftrightarrow{\tau_T} & (M, T) \\ \downarrow{\scriptstyle \mathbf{x}} & & \downarrow{\scriptstyle \mathbf{x}} \\ \mathcal{K}_M^{\mathcal{O}_M} & \xleftrightarrow{\overline{\tau}_T} & \mathcal{K}_M^T \end{array}$$

where $\overline{\tau}_T$ is the induced isomorphism (again, involution) between the corresponding Kummer lines. For example, we obtain the map $\overline{\tau}_{(0,0)} : (X : Z) \mapsto (Z : X)$. Since $\#(M, \mathcal{O}_M)[2] = 4$, there are at most two other points of order 2. This gives rise to only a single non-trivial action on the Kummer line $\mathcal{K}_M^{\mathcal{O}_M}$, since the other is simply the composition with $\overline{\tau}_{(0,0)}$.

(Twisted) Edwards form. Given $c \in \bar{k}$ such that $c^5 \neq c$, the elliptic curve defined by the equation $\mathcal{E} : x^2 + y^2 = c^2(1 + x^2y^2)$ with base point $\mathcal{O}_\mathcal{E} = (0, c)$ is said to be in Edwards form. The projection map $\mathbf{y} : \mathcal{E} \to \mathbb{P}^1$ onto the y-coordinate gives rise to the Kummer line denoted $\mathcal{K}_\mathcal{E}^{\mathcal{O}_\mathcal{E}}$. Any point $R \in (\mathcal{E}, \mathcal{O}_\mathcal{E})[2]$ induces a commutative diagram

$$\begin{array}{ccc} (\mathcal{E}, \mathcal{O}_\mathcal{E}) & \xleftrightarrow{\tau_R} & (\mathcal{E}, R) \\ \downarrow{\scriptstyle \mathbf{y}} & & \downarrow{\scriptstyle \mathbf{y}} \\ \mathcal{K}_\mathcal{E}^{\mathcal{O}_\mathcal{E}} & \xleftrightarrow{\overline{\tau}_R} & \mathcal{K}_\mathcal{E}^R \end{array} . \tag{1}$$

In particular, for $R = (0 : 0 : -c : 1)$ the induced map $\overline{\tau}_R$ is simply given by $\overline{\tau}_R : (Y : Z) \mapsto (-Y : Z)$. The two other non-trivial 2-torsion points induce one other non-trivial translation (similar to the Montgomery model).

Completely analogously, for $\alpha, \delta \in k$ such that $\alpha\delta(\alpha - \delta) \neq 0$, we denote by $E : \alpha x^2 + y^2 = 1 + \delta x^2 y^2$ an elliptic curve in twisted Edwards form with base point $\mathcal{O}_E = (0, 1)$. For any $S \in (E, \mathcal{O}_E)[2]$ we obtain a commutative diagram as in (1). In particular, the point $S = (0 : 0 : -1 : 1)$ induces the action $\overline{\tau}_S : \mathcal{K}_E^{\mathcal{O}_E} \to \mathcal{K}_E^S$ that maps $(Y : Z) \mapsto (-Y : Z)$.

Rationality and quadratic twists. Suppose that q is a prime power and $k = \mathbb{F}_q$ is a finite field. Then any elliptic curve $\mathcal{C}$ defined over $\mathbb{F}_q$ will have a quadratic twist, i. e. an elliptic curve $\mathcal{C}^t$ which is $\mathbb{F}_{q^2}$-isomorphic but not $\mathbb{F}_q$-isomorphic to $\mathcal{C}$. This is unique up to $\mathbb{F}_q$-isomorphism (hence why we talk about *the* quadratic twist).

In all the curve models (c. f. the above) that we consider there is an immediate connection between $\mathbb{F}_q$-rational points on the Kummer line $\mathcal{K}_\mathcal{C}$ of $\mathcal{C}$, and $\mathbb{F}_q$-rational points of $\mathcal{C}$ and $\mathcal{C}^t$. As such, when thinking about Kummer lines it is natural not to distinguish these (i. e. to consider everything up to $\mathbb{F}_{q^2}$-isomorphism). As a result, although some maps may only be defined over $\mathbb{F}_{q^2}$, this will at most induce a twist. Since we are only concerned with the $\mathbb{F}_q$-rational points of the Kummer line, this is not an issue. In all that follows we *could* easily make everything defined over $\mathbb{F}_q$, but as we shall see in §4 this may limit us when finding instantiations.

3 Maps between Kummer Lines

In this section we present the theoretical basis. We observe first that many Kummer lines have appeared in the literature; the work of Gaudry and Lubicz [GL09] present the so-called *canonical* Kummer line, while Karati and Sarkar use[2] the *squared* Kummer line [KS17]. Moreover, there is the x-line of Montgomery curve (e. g. Curve25519 [Ber06a] by Bernstein) and the y-line of a (twisted) Edwards curve [CGF08; FH17]. It is not immediately clear how these are all connected, in particular the relation between the (canonical and squared) Kummer lines and Montgomery and (twisted) Edwards curves is not clear. Though a recent paper by Karati and Sarkar [KS19] provides some connections, this is not completely satisfying. For instance, it relies on having rational points or using 2-isogeny, and does not give a unique connection.

In this section we settle this and, in essence, show that they are all the same up to isomorphism. These isomorphisms are natural and simple (including computationally) and lead to natural connections between all the above Kummer lines. The core of the section is summarized in Theorem 5.

[2] The formulas for this model had already appeared in the Explicit-Formulas Database [BLb] referring to a discussion between Bernstein, Kohel and Lange and contributing the main idea to Gaudry [Gau06].

3.1 Models with Rational 2-torsion

It is immediate (through their description via theta functions) that the canonical and squared Kummer lines are projections of curves that have full rational 2-torsion. As such, we shall always assume to have this. We begin by showing that this allows a nice parametrization of Montgomery curves.

Proposition 1. *Let k be a field such that* $\mathrm{char}(k) \neq 2$ *and let $(M_{A,B}, \mathcal{O}_M)$ be a Montgomery curve such that $M_{A,B}[2] \subset M_{A,B}(k)$. Then there exist $a, b \in \bar{k}^*$ such that $ab(a^4 - b^4) \neq 0$ and $a^2/b^2 \in k$ such that*

$$A = -\frac{a^4 + b^4}{a^2 b^2}, \quad \Delta_M = 16B^6 \cdot \frac{(a^4 - b^4)^2}{a^4 b^4}.$$

Moreover, its points of order 2 are $(0 : 0 : 1)$, $(a^2 : 0 : b^2)$ and $(b^2 : 0 : a^2)$.

Proof. As $M_{A,B}[2] \subset M_{A,B}(k)$, the polynomial $x^2 + Ax + 1$ splits over k and thus $\sqrt{A^2 - 4} \in k$. Now fix any $b \in \bar{k}^*$ and take $a \in \bar{k}^*$ such that $a^2/b^2 = (\sqrt{A^2 - 4} - A)/2$. Note that $\sqrt{A^2 - 4} - A \neq 0, \pm 2$ because $\mathrm{char}(k) \neq 2$. Moreover $a^4 - b^4 = 0 \iff a^4/b^4 - 1 = 0 \iff a^2/b^2 = \pm 1$. Again, this is not possible since $\mathrm{char}(k) \neq 2$. The statements for A, Δ_M and the 2-torsion points are simple calculations, recalling that M has discriminant $\Delta_M = 16B^6(A^2 - 4)$. □

For simplicity we would like to have $B = 1$. Note that the curve $M_{A,B}$ is isomorphic to the curve $M_{A,1} : y^2 = x^3 + Ax^2 + x$ over $\bar{k}$, but not necessarily over k. Therefore, by making the assumption that $B = 1$ we are working *up to twist*. In what follows this shall not give rise to any issues, and as remarked earlier it does not impact the k-rational points of the Kummer line (even though it does change the k-rational point of the curve itself). So from this point on we consider

$$M/k : y^2 = x^3 - \frac{a^4 + b^4}{a^2 b^2} x^2 + x,$$

where $a, b \in \bar{k}^*$ such that $ab(a^4 - b^4) \neq 0$ and $a^2/b^2 \in k$.

Given this model we can define a *dual* curve. For this purpose, we define $\hat{a}, \hat{b} \in \bar{k}^*$ such that

$$2\hat{a}^2 = a^2 + b^2, \quad 2\hat{b}^2 = a^2 - b^2.$$

It is easily checked that $\hat{a}^2/\hat{b}^2 \in k^*$ and that $\hat{a}\hat{b}(\hat{a}^4 - \hat{b}^4) \neq 0$. Therefore

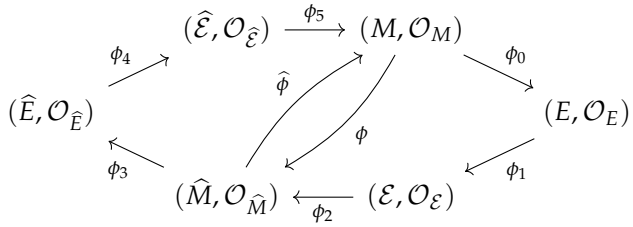
$$\widehat{M} : y^2 = x^3 - \frac{\hat{a}^4 + \hat{b}^4}{\hat{a}^2 \hat{b}^2} x^2 + x, \quad \Delta_{\widehat{M}} = 16 \cdot \frac{(\hat{a}^4 - \hat{b}^4)^2}{\hat{a}^4 \hat{b}^4}$$

is a Montgomery curve whose elements of order 2 are $(0:0:1)$, $(\hat{a}^2:0:\hat{b}^2)$ and $(\hat{b}^2:0:\hat{a}^2)$. We call $\widehat{M}$ the *dual* of M. More generally, for any curve model we call the action of swapping a resp. b by $\hat{a}$ resp. $\hat{b}$ (and vice versa) *dualizing* (c. f. §V.4.1). The curves M and $\widehat{M}$ are 2-isogenous via a 2-isogeny $\phi : M \to \widehat{M}$, and the kernel of both ϕ and $\widehat{\phi}$ is generated by the point $(0:0:1)$ on the respective curves (see Remark VIII.11). This leads to a decomposition of the doubling map [2], which we use to construct the following sequence of maps.

Proposition 2. *Let $a, b \in \bar{k}^*$ with $ab(a^4 - b^4) \neq 0$ and $a^2/b^2 \in k^*$ and*

$$M/k : y^2 = x^3 - \frac{a^4 + b^4}{a^2 b^2} x^2 + x\,.$$

Then there exists a commutative diagram[3] of isogenies (over $\bar{k}$)



(2)

where

$$E/k : -x^2 + y^2 = 1 - \frac{(a^2 - b^2)^2}{(a^2 + b^2)^2} x^2 y^2\,, \quad \mathcal{E}/k : x^2 + y^2 = \frac{a^2 - b^2}{a^2 + b^2}\left(1 + x^2 y^2\right)$$

and $\widehat{E}$ and $\widehat{\mathcal{E}}$ are their respective duals. The maps ϕ_2 and ϕ_5 are 2-isogenies with

$$\ker(\phi_2) = \langle (0:0:-\hat{b}:\hat{a}) \rangle\,, \quad \ker(\phi_5) = \langle (0:0:-b:a) \rangle\,,$$

while the maps ϕ_0, ϕ_1, ϕ_3 and ϕ_4 are isomorphisms.

Proof. We define

$$\phi_0 : (x, y) \mapsto \left(\frac{2\hat{a}^2 x}{aby}, \frac{x+1}{x-1}\right)\,, \quad \phi_0^{-1} : (x, y) \mapsto \left(\frac{y+1}{y-1}, \frac{2\hat{a}^2(y+1)}{abx(y-1)}\right)\,.$$

Note that this is a priori only a birational map, but naturally becomes an isomorphism when (canonically) extended to the smooth $\mathbb{P}^3$ model, see e. g. [Sil09, Propo-

[3] The diagram is drawn in the shape of a hexagon because its induced diagram on the Kummer lines after translations by points of order 2 is the genus-1 analogue of the hexagon in genus 2 in Figure V.1.

sition II.2.1]. In particular, $\phi_0 : \mathcal{O}_M \mapsto \mathcal{O}_E, (0:0:1) \mapsto (0:0:-1:1)$. It is similar to the maps used by Bernstein et al. [Ber+08, Theorem 3.2(i)] and by Castryck et al [CGF08], but composed with the map by Hisil et al. [His+08, §3.1] to ensure a twisted Edwards curve $E_{\alpha,\delta}$ with $\alpha = -1$ that is well-defined everywhere. Moreover, we tweak it such that it acts as an involution (i. e. a Hadamard transformation) on the Kummer line. We define the isomorphism ϕ_1 as

$$\phi_1 : (x,y) \mapsto \Big(-\frac{i\hat{b}}{\hat{a}}x, \frac{\hat{b}}{\hat{a}}y \Big), \quad \phi_1^{-1} : (x,y) \mapsto \Big(\frac{i\hat{a}}{\hat{b}}x, \frac{\hat{a}}{\hat{b}}y \Big),$$

where $i \in \bar{k}$ is such that $i^2 = -1$. Then we set $\phi_2 = \phi \circ \phi_0^{-1} \circ \phi_1^{-1}$. It follows that

$$\ker(\phi_2) = \langle \phi_1\phi_0(0,0) \rangle = \langle (0:0:-\hat{b}:\hat{a}) \rangle .$$

A completely analogous construction can be made for ϕ_3, ϕ_4 and ϕ_5. □

Remark 3. Note that one can argue that the above construction can be done for any sequence of isomorphisms starting at M. Indeed this is the case, but the above choice is a natural one and gives rise to nice arithmetic on the Kummer lines. Moreover, it is a choice that allows to explain the connection between Montgomery curves and the genus-1 Kummer lines arising from theta functions (i. e. [GL09, §6.2] and [KS17, §2.4]).

Corollary 4. *There is an induced commutative diagram of Kummer lines*

$$\begin{array}{ccccc} & & \mathcal{K}_{\hat{\mathcal{E}}}^{\mathcal{O}_{\hat{\mathcal{E}}}} & \xrightarrow{\bar{\phi}_5} & \mathcal{K}_{M}^{\mathcal{O}_{M}} & & \\ & \overset{\bar{\phi}_4}{\nearrow} & & & & \overset{\bar{\phi}_0}{\searrow} & \\ \mathcal{K}_{\hat{E}}^{\mathcal{O}_{\hat{E}}} & & & & & & \mathcal{K}_{E}^{\mathcal{O}_{E}} \\ & \underset{\bar{\phi}_3}{\nwarrow} & & & & \underset{\bar{\phi}_1}{\swarrow} & \\ & & \mathcal{K}_{\hat{M}}^{\mathcal{O}_{\hat{M}}} & \xleftarrow[\bar{\phi}_2]{} & \mathcal{K}_{\mathcal{E}}^{\mathcal{O}_{\mathcal{E}}} & & \end{array} \tag{3}$$

such that

$$\bar{\phi}_0 : (X:Z) \mapsto (X+Z : X-Z), \quad \bar{\phi}_1 : (X:Z) \mapsto (\hat{b}X : \hat{a}Z),$$
$$\bar{\phi}_2 : (X:Z) \mapsto (\hat{b}^2X^2 - \hat{a}^2Z^2 : \hat{a}^2X^2 - \hat{b}^2Z^2),$$

while $\overline{\phi}_3 = \overline{\phi}_0$ and $\overline{\phi}_4$ resp. $\overline{\phi}_5$ are obtained from $\overline{\phi}_1$ resp. $\overline{\phi}_2$ by dualizing.

Proof. Apply the respective **x** and **y** projection maps to the curves in (2). □

This provides clear connections between the x- and y- lines of Montgomery and (twisted) Edwards curves with full rational 2-torsion. We now show that we can use these 2-torsion points to obtain simple isomorphisms to the canonical and squared Kummer lines.

3.2 Actions of Points of Order 2

First recall from §I.2.1 that we have points

$$T = (a^2 : 0 : b^2) \in M\,, \quad \Omega_1 = (\hat{a}^2 : \hat{b}^2 : 0 : 0) \in E\,, \quad \Theta_1 = (\hat{a} : \hat{b} : 0 : 0) \in \mathcal{E}\,,$$
$$\widehat{T} = (\hat{a}^2 : 0 : \hat{b}^2) \in \widehat{M}\,, \quad \widehat{\Omega}_1 = (a^2 : b^2 : 0 : 0) \in \widehat{E}\,, \quad \widehat{\Theta}_1 = (a : b : 0 : 0) \in \widehat{\mathcal{E}}\,.$$

of order 2 (with the base point $\mathcal{O}$ on the respective curves). One can check that these are all respective images of one another under the ϕ_i and $\widehat{\phi}_i$. They correspond to translations[4] τ by the respective points which commute with the projection maps to $\mathbb{P}^1$. As a result, we obtain induced involutions $\overline{\tau}$ on the Kummer lines. More concretely, we can show that for any point $P = (X : Z) \in \mathbb{P}^1$ we have

$$\overline{\tau}_T : P \mapsto (a^2X - b^2Z : b^2X - a^2Z)\,, \quad \overline{\tau}_{\Omega_1} : P \mapsto (\hat{a}^2Z : \hat{b}^2X)\,, \quad \overline{\tau}_{\Theta_1} : P \mapsto (Z : X)\,,$$
$$\overline{\tau}_{\widehat{T}} : P \mapsto (\hat{a}^2X - \hat{b}^2Z : \hat{b}^2X - \hat{a}^2Z)\,, \quad \overline{\tau}_{\widehat{\Omega}_1} : P \mapsto (a^2Z : b^2X)\,, \quad \overline{\tau}_{\widehat{\Theta}_1} : P \mapsto (Z : X)\,.$$

Note that we could apply the maps τ to the diagram (2), but that requires keeping track of multiple coordinates and is somewhat tedious. Instead, for simplicity, we will focus on the Kummer lines. Applying the maps $\overline{\tau}$ to (3), we obtain the following result.

Theorem 5. *For any $P = (X : Z) \in \mathbb{P}^1$, we denote by*

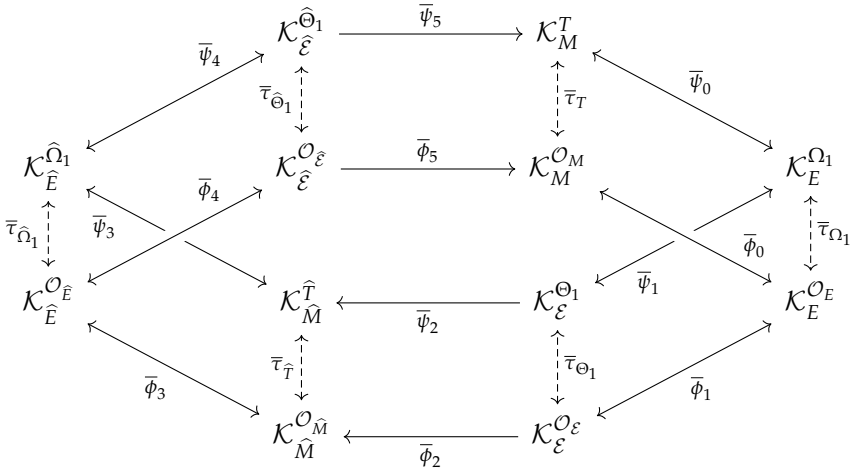
$$\overline{\psi}_0 : P \mapsto (X + Z : X - Z)\,, \quad \overline{\psi}_1 : P \mapsto (\hat{b}X : \hat{a}Z)\,, \quad \overline{\psi}_2 : P \mapsto (X^2 : Z^2)\,,$$
$$\overline{\psi}_3 : P \mapsto (X + Z : X - Z)\,, \quad \overline{\psi}_4 : P \mapsto (bX : aZ)\,, \quad \overline{\psi}_5 : P \mapsto (X^2 : Z^2)\,,$$

maps $\mathbb{P}^1 \to \mathbb{P}^1$. The diagram in (4) is commutative and every $\leftrightarrow$ is an isomorphism.

Proof. This is the diagram from (2) translated by corresponding points of order 2 through the different τ, projected to their respective Kummer lines. We construct

$$\overline{\psi}_0 = \overline{\tau}_{\Omega_1} \circ \overline{\phi}_0 \circ \overline{\tau}_T$$

[4] Translations are morphisms [Sil09, Theorem 3.6] and are therefore isogenies if and only if they send the base point of the domain curve to the base point of the co-domain curve. For example, $\tau_T : (M, \mathcal{O}_M) \to (M, T)$ is an isogeny. As such, it is a group homomorphism.



(4)

and proceed similarly for the other $\overline{\psi}_i$. □

Recall that (the duals of) $\mathcal{K}^{\mathcal{O}_M}_{M}$, $\mathcal{K}^{\mathcal{O}_E}_{E}$ resp. $\mathcal{K}^{\mathcal{O}_{\mathcal{E}}}_{\mathcal{E}}$ are the Kummer lines of (the duals of) a Montgomery, twisted Edwards resp. Edwards curve. Hence it remains to identify $\mathcal{K}^{T}_{M}$, $\mathcal{K}^{\Omega_1}_{E}$ and $\mathcal{K}^{\Theta_1}_{\mathcal{E}}$ (and their duals). Since they are all simply $\mathbb{P}^1$ as an algebraic variety, we analyze their (pseudo-)addition formulae.

First note that Proposition 1 says that moving through the sequence $\overline{\phi}_0, \ldots, \overline{\phi}_5$ corresponds to the [2] map (starting at any of the $\overline{\phi}_i$). Since the $\overline{\tau}$ are isomorphisms, the same is true for $\overline{\psi}_0, \ldots, \overline{\psi}_5$. In other words, for example

$$[2] = \overline{\psi}_5 \circ \cdots \circ \overline{\psi}_0 \text{ on } \mathcal{K}^{T}_{M}, \quad [2] = \overline{\psi}_4 \circ \cdots \circ \overline{\psi}_0 \circ \overline{\psi}_5 \text{ on } \mathcal{K}^{\widehat{\Theta}_1}_{\widehat{\mathcal{E}}}.$$

Comparing these with the algorithm from Gaudry and Lubicz [GL09, §6.2] (and the formulas also appear in [BLb]) reveals that these are the doubling formulae for the squared and canonical Kummer lines. One readily[5] verifies that the same is true for the differential addition formulae. The third Kummer line $\mathcal{K}^{\Omega_1}_{E}$ has not appeared to our knowledge, and has similar arithmetic to the squared Kummer line. We refer to it as the *intermediate* Kummer (c. f. §V.4.3). Interestingly, it appears as the y-line of a twisted Edwards curve where the coefficient of x^2 is -1, in which case the optimal formulas by Hisil et al. [His+08] are available. For completeness, we summarize the associated curve constants for the instances provided by Karati and Sarkar in

[5] This can be done by using the known addition formulae on the elliptic curves whose identities are at infinity, and composing with the translation and projection maps. This is somewhat tedious, but is relatively straightforward by using a computer algebra package [BCP97; Sag18].

Table 1. Kummer lines over a finite field $\mathbb{F}_q$ and their associated (i) squared Kummer $(a^2 : b^2)$ (ii) Montgomery A (iii) twisted Edwards δ and (iv) Edwards c^2 constants.

q	$(a^2 : b^2)$	$(A : 1)$	$(\delta : 1)$	$(c^2 : 1)$
$2^{251} - 9$	$(81 : 20)$	$(-6961 : 1620)$	$(-3721 : 10201)$	$(61 : 101)$
$2^{251} - 9$	$(186 : 175)$	$(-65221 : 130200)$	$(-121 : 130221)$	$(11 : 361)$
$2^{255} - 19$	$(82 : 77)$	$(-12653 : 6314)$	$(-25 : 25281)$	$(5 : 159)$
$2^{266} - 3$	$(260 : 139)$	$(-86921 : 36140)$	$(-14641 : 159201)$	$(121 : 399)$

Table 1, connecting the squared Kummer line to the Kummer lines of Montgomery and twisted Edwards models via isomorphisms (as opposed to birational maps or isogenies).

Remark 6. We reiterate that only the intermediate Kummer line is new, while all the others have already appeared in the literature and are well-known. However, there had been little work in providing explicit maps between them, and this is exactly what we provide.

3.3 Hybrid Kummer Lines

Since the arithmetic on these Kummer lines is generally well-studied, the (cryptographic) value of this study does not come from improved operation counts. Beside its theoretical contribution, we ease the problem of selecting which curves to use for best performance (e. g. for standardization). That is, the simplicity of the isomorphisms gives quasi-cost-free transformations that allow interchangeable usage of any of the models. This is similar to the usage of a birational map to move between the Montgomery and twisted Edwards model, but we extend it with the squared Kummer line. We summarize this in Figure 1. In particular, Karati and Sarkar [KS17] show the benefits of the squared Kummer line on platforms where SIMD instructions are available.

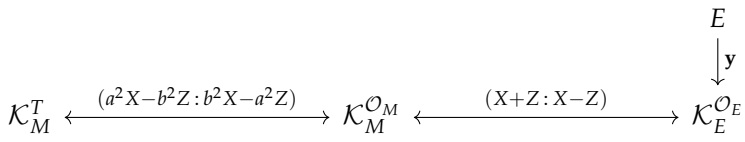


Figure 1. The squared Kummer line, the x-line of a Montgomery curve and the y-line of a twisted Edwards curve E, connected by involutions.

Remark 7. Recall that all the above works under the assumption of having full rational 2-torsion. Although Montgomery and (twisted) Edwards curves always have a group order divisible by 4, it does not necessarily mean that they have full 2-torsion (i. e. they could have a point of order 4). Note that standardized curves such as Curve25519 and Curve448 do not have full 2-torsion, so this theory does not directly apply.

Moreover, results from the well-studied Montgomery model immediately carry over to the squared Kummer line. For example, we can straightforwardly fit a (squared) Kummer line into the qDSA signature scheme. For signature verification, given $\mathbf{x}(P), \mathbf{x}(Q), \mathbf{x}(R) \in \mathcal{K}_M^T$ we must be able to check whether $\mathbf{x}(R) = \mathbf{x}(P \pm Q)$. Although this can certainly be directly defined on $\mathcal{K}_M^T$, we note that it is equivalent to checking whether

$$\overline{\tau}_T(\mathbf{x}(R)) = \overline{\tau}_T(\mathbf{x}(P \pm Q)) .$$

This is simply the function $\texttt{Check}(\overline{\tau}_T(\mathbf{x}(P)), \overline{\tau}_T(\mathbf{x}(Q)), \overline{\tau}_T(\mathbf{x}(R)))$, where `Check` is defined in Algorithm V.2.

To demonstrate feasibility of this approach, we extend the (publicly available) Curve25519-based instantiation of qDSA from Chapter V on the ARM Cortex M0 architecture. For this purpose we choose a squared Kummer line over $2^{255}-19$, allowing field arithmetic to remain essentially unchanged. A notable exception to this is an efficient assembly implementation of 16×256-bit field multiplication, which is used for the multiplications by the line constants. This replaces the highly optimized multiplication by 121666 from Düll et al. [Dül+15]. We select $(a^2, b^2) = (159, 5)$, so that the squared Kummer line $\mathcal{K}_M^T$ corresponds to the *dual*[6] of `KL25519(82,77)` presented and implemented by Karati and Sarkar [KS17]. This implies the Montgomery constant of the curve above the line $\mathcal{K}_M^{\mathcal{O}_M}$ to be $(A+2:4) = (-5929:795)$. We summarize the implementation results in Table 2. We emphasize that the point of this work is not to provide the most efficient implementation for this given platform, but rather to show the close connection between the different Kummer lines. Although on this platform results differ only by a minimal margin, the difference can be much larger on other devices (in particular, when SIMD instructions are available). Our isomorphisms allow an implementer to select the model that is most appropriate for a given architecture.

Remark 8. The implementations that we present are constant-time, and all standard countermeasures (e. g. projective blinding, scalar blinding [Cor99, §5]) against more

[6] The constants $(a^2, b^2) = (88, 77)$ lead to $(A+2:4) = (-25:25256)$ which has slightly larger constants on $\mathcal{K}_M^{\mathcal{O}_M}$ than its dual. However, results should be very similar.

Table 2. Comparison of an implementation of the qDSA signature scheme based on Curve25519 and the Montgomery model of the squared Kummer line defined by $(a^2, b^2) = (159, 5)$, where the memory is measured in bytes (B).

Ref.	Object	Constant	Clock cycles	Stack	Code
Chap. V	Curve25519	$(A+2:4) =$ $(121666:1)$	3 889 116 (`sign`) 6 793 695 (`verify`)	660 B 788 B	18 443 B
This	$\mathcal{K}_M^{\mathcal{O}_M}$	$(A+2:4) =$ $(-5929:795)$	3 916 879 (`sign`) 6 857 007 (`verify`)	660 B 788 B	18 391 B
This	$\mathcal{K}_M^T$	$(a^2:b^2) =$ $(159:5)$	3 824 857 (`sign`) 6 673 039 (`verify`)	660 B 788 B	18 557 B

advanced side-channel and fault attacks can be applied if required. In particular, as mentioned by the authors, the recent fault attack by Takahashi, Tibouchi and Abe [TTA18] can (cheaply) be thwarted by requiring nonces to be multiples of the cofactor (i. e. by "clamping"). However, such countermeasures are only necessary when an implementation is used in a context where fault attacks are considered part of the attacker model. We emphasize that our implementation is intended as a reference and *not* for production use.

4 Isomorphism Classes over Finite Fields

For cryptographic purposes, we are mostly concerned with the case that $k = \mathbb{F}_q$, for some prime (power) q. As using extension fields is generally expensive, we would like to set things up such that all computation is performed in $\mathbb{F}_q$. Whether or not we can do this in a way such that constants remain small, depends on the number of Kummer lines that exist. Following earlier studies on the number of isomorphism classes for certain curve models [Ber+08; FS09; FMW12], we provide counts for the canonical, squared and intermediate Kummer lines.

4.1 Identifying Kummer Lines

For this purpose it is interesting to ask when two Kummer lines should be considered to be the same. Given two Kummer lines $\mathcal{K}_1 = E_1/\{\pm 1\}$ and $\mathcal{K}_2 = E_2/\{\pm 1\}$ of elliptic curves E_1, E_2 defined over $\mathbb{F}_q$, it could be natural to identify $\mathcal{K}_1$ with $\mathcal{K}_2$ whenever E_1 is $\mathbb{F}_q$-isomorphic to E_2. However, as noted in §I.2.1, the arithmetic on the $\mathbb{F}_q$-rational points of the Kummer lines will be identical whenever E_1 is $\mathbb{F}_{q^2}$-

isomorphic to E_2 (i. e. E_2 is the quadratic twist of E_1). Since the curves are defined over $\mathbb{F}_q$, this will happen if and only if $j(E_1) = j(E_2)$. As such, we equate the number of Kummer lines with the number of elliptic curves defined over $\mathbb{F}_q$ up to $\overline{\mathbb{F}}_q$-isomorphism.

Recall that we parametrize Kummer lines by $a, b \in \overline{\mathbb{F}}_q$ such that $ab(a^4 - b^4) \neq 0$ and $a^2/b^2 \in \mathbb{F}_q$. Since $b \neq 0$, a Kummer line is defined by the fraction a/b or, equivalently, by the point $(a : b) \in \mathbb{P}^1$. Again, since $b \neq 0$ we can therefore simply assume $b = 1$. As such, we can consider $a \in \overline{\mathbb{F}}_q$ such that $a^2 \in \mathbb{F}_q$ and $a^5 - a \neq 0$.

4.2 Canonical Kummer Lines

We begin by considering the canonical Kummer line from Gaudry and Lubicz [GL09] defined by some a as above. Recall that it corresponds to the y-line of the curve $\widehat{\mathcal{E}}/\mathbb{F}_q : x^2 + y^2 = \frac{1}{a^2}\left(1 + x^2y^2\right)$, with identity $\widehat{\Omega}_1 = (a : 1 : 0 : 0)$ whose image in $\mathbb{P}^1$ is $(a : 1)$. Therefore, we certainly require that $a \in \mathbb{F}_q$. It is easily seen that $\hat{a}^2, \hat{b}^2 \in \mathbb{F}_q$ and that this is enough to perform all arithmetic with $\mathbb{F}_q$-operations.

Now note that $(\widehat{\mathcal{E}}, \widehat{\Omega}_1)$ is $\mathbb{F}_q$-isomorphic to $(\widehat{\mathcal{E}}, \mathcal{O}_{\widehat{\mathcal{E}}})$ via $\tau_{\widehat{\Omega}_1}$, which is an Edwards curve if and only if $a \in \mathbb{F}_q$ and $1/a^5 \neq 1/a$. The first is true by assumption, while the latter follows from $a^5 \neq a$. Therefore we simply count the number of Edwards curves defined over $\mathbb{F}_q$ up to $\overline{\mathbb{F}}_q$-isomorphism. A result by Farashahi and Shparlinski [FS09, Theorem 5] shows that there are exactly

$$\begin{cases} \left\lfloor \dfrac{q+23}{24} \right\rfloor & \text{if } q \equiv 1, 9, 13, 17 \pmod{24}, \\ \left\lfloor \dfrac{q-5}{24} \right\rfloor & \text{if } q \equiv 5 \pmod{24}, \\ \left\lfloor \dfrac{q+1}{8} \right\rfloor & \text{if } q \equiv 3 \pmod{4}. \end{cases}$$

Thus, in general there will be no problem to find Kummer lines with the desired security properties. However, it may not be easy to find them such that its constants are small. For that reason, we look towards the squared and intermediate Kummer lines.

4.3 Squared and Intermediate Kummer Lines

If we use canonical Kummer lines, we restrict ourselves to $a \in \mathbb{F}_q$ for all of the arithmetic to be in $\mathbb{F}_q$. This (seemingly) limits the number of Kummer lines that we can use. This is no longer the case on squared and intermediate Kummer lines; it

suffices to only have $a^2 \in \mathbb{F}_q$. Note that this implies that $a \in \mathbb{F}_{q^2}$.

Since the j-invariants of M, E and $\mathcal{E}$ and their duals are all equal, we can count the number of curves up to isomorphism of the form $\widehat{\mathcal{E}} : x^2 + y^2 = \frac{1}{a^2}\left(1 + x^2y^2\right)$ such that $a^5 \neq a$ (but note that $\widehat{\mathcal{E}}$ is not necessarily an Edwards curve over $\mathbb{F}_q$). There are exactly $q - 3$ such curves, so it remains to determine how many are in the same $\overline{\mathbb{F}}_q$-isomorphism class. This question has already been considered by Edwards [Edw07, Proposition 6.1], whose statement implies that two Edwards curves determined by $a^2, \bar{a}^2 \in \mathbb{F}_q$ have the same j-invariant whenever $\bar{a}^2$ is one of the following:

$$\pm a^2, \pm\frac{1}{a^2}, \pm\left(\frac{a-1}{a+1}\right)^2, \pm\left(\frac{a+1}{a-1}\right)^2, \pm\left(\frac{a-i}{a+i}\right)^2, \pm\left(\frac{a+i}{a-i}\right)^2. \tag{5}$$

If $q \equiv 1 \pmod 4$, then $i^q = i$ and a straightforward computation show that

$$\pm\left(\frac{a-1}{a+1}\right)^2, \pm\left(\frac{a+1}{a-1}\right)^2, \pm\left(\frac{a-i}{a+i}\right)^2, \pm\left(\frac{a+i}{a-i}\right)^2 \in \mathbb{F}_q \iff a \in \mathbb{F}_q.$$

If $q \equiv 3 \pmod 4$, then $i^q = -i$ and a similar computation shows that

$$\begin{aligned}\pm\left(\frac{a-1}{a+1}\right)^2, \pm\left(\frac{a+1}{a-1}\right)^2 \in \mathbb{F}_q &\iff a \in \mathbb{F}_q,\\ \pm\left(\frac{a-i}{a+i}\right)^2, \pm\left(\frac{a+i}{a-i}\right)^2 \in \mathbb{F}_q &\iff i \cdot a \in \mathbb{F}_q.\end{aligned}$$

Given that either $a \in \mathbb{F}_q$ or $i \cdot a \in \mathbb{F}_q$, while half the elements of $\mathbb{F}_q$ have square roots in $\mathbb{F}_q$, we closely approximate[7] that the number of isomorphism classes is

$$\approx \begin{cases} \left\lfloor \left(\frac{1}{4} + \frac{1}{12}\right)\frac{q}{2} \right\rfloor = \left\lfloor \frac{q}{6} \right\rfloor & \text{if } q \equiv 1 \pmod 4, \\ \left\lfloor \left(\frac{1}{8} + \frac{1}{8}\right)\frac{q}{2} \right\rfloor = \left\lfloor \frac{q}{8} \right\rfloor & \text{if } q \equiv 3 \pmod 4. \end{cases}$$

A more careful analysis c. f. [FS09] could be done, but such a close estimate suffices for our purposes. Interestingly, for $q \equiv 3 \pmod 4$ the number of canonical and squared Kummer lines is about the same. Thus although $a^2 \in \mathbb{F}_q$ is a weaker restriction than $a \in \mathbb{F}_q$, it does not actually lead to more Kummer lines (up to isomorphism). This is explained by the fact that -1 is a non-square since $q \equiv 3 \pmod 4$, hence exactly one of a^2 or $-a^2$ must be a square in $\mathbb{F}_q$, while their corresponding

[7] This statement is exact up to the observation that some of the elements in (5) can be the same, which happens only exceptionally.

Edwards curves are isomorphic. For $q \equiv 1 \pmod 4$ there is a clear difference in the number of Kummer lines, so in that case there is a significant advantage in finding small parameters for a squared or intermediate Kummer line over a canonical Kummer line.

Part 3

Post-Quantum Cryptography

Chapter VII

Efficient Compression of SIDH Public Keys

Supersingular isogeny Diffie-Hellman (SIDH) is an attractive candidate for post-quantum key exchange, in large part due to its relatively small public key sizes. A paper by Azarderakhsh, Jao, Kalach, Koziel and Leonardi [Aza+16] showed that the public keys defined in Jao and De Feo's original SIDH scheme can be further compressed by around a factor of two, but reported that the performance penalty in utilizing this compression blew the overall SIDH runtime out by more than an order of magnitude. Given that the runtime of SIDH key exchange is currently its main drawback in relation to its lattice- and code-based post-quantum alternatives, an order of magnitude performance penalty for a factor of two improvement in bandwidth presents a trade-off that is unlikely to favor public-key compression in many scenarios.

In this chapter, we propose a range of new algorithms and techniques that accelerate SIDH public-key compression by more than an order of magnitude, making it roughly as fast as a round of standalone SIDH key exchange, while further reducing the size of the compressed public keys by approximately 12.5%. These improvements enable the practical use of compression, achieving public keys of only 330 bytes for the concrete parameters used to target 128 bits of quantum security and further strengthens SIDH as a promising post-quantum primitive.

1 Introduction

In their February 2016 report on post-quantum cryptography [Che+16], the United States National Institute of Standards and Technology (NIST) stated that *"It seems improbable that any of the currently known [public-key] algorithms can serve as a drop-in replacement for what is in use today"*, citing that one major challenge is that quantum resistant algorithms have larger key sizes than the algorithms they will replace. While this statement is certainly applicable to many of the lattice- and code-based schemes (e. g. LWE encryption [Reg05] and the McEliece cryptosystem [McE78]), Jao and De Feo's 2011 supersingular isogeny Diffie-Hellman (SIDH) proposal [JDF11] is one post-quantum candidate that could serve as a drop-in replacement to existing internet protocols. Not only are high-security SIDH public keys smaller than their lattice- and code-based counterparts, they are even smaller than some of the traditional (i. e. finite field) Diffie-Hellman public keys.

SIDH public-key compression. The public keys defined in the original SIDH papers [JDF11; DFJP14] take the form $\texttt{PK} = (E, P, Q)$, where $E/\mathbb{F}_{p^2} : y^2 = x^3 + ax + b$ is a supersingular elliptic curve, $p = n_A n_B \pm 1$ is a large prime, the cardinality of E is $\#E(\mathbb{F}_{p^2}) = (p \mp 1) = (n_A n_B)^2$, and depending on whether the public key corresponds to Alice or Bob, the points P and Q either both lie in $E(\mathbb{F}_{p^2})[n_A]$, or both lie in $E(\mathbb{F}_{p^2})[n_B]$. Since P and Q can both be transmitted via their x-coordinates (together with a sign bit that determines the correct y-coordinate), and the curve can be transmitted by sending the two $\mathbb{F}_{p^2}$ elements a and b, the original SIDH public keys essentially consist of four $\mathbb{F}_{p^2}$ elements, and so are around $8 \log p$ bits in size.

A recent paper by Azarderakhsh, Jao, Kalach, Koziel and Leonardi [Aza+16] showed that it is possible to compress the size of SIDH public keys to around $4 \log p$ bits as follows. Firstly, to send the supersingular curve E, they pointed out that one can send the j-invariant $j(E) \in \mathbb{F}_{p^2}$ rather than $(a, b) \in \mathbb{F}_{p^2}^2$, and showed how to recover a and b (uniquely, up to isomorphism) from $j(E)$ on the other side. Secondly, for $n \in \{n_A, n_B\}$, they showed that since $E(\mathbb{F}_{p^2})[n] \cong \mathbb{Z}/n\mathbb{Z} \times \mathbb{Z}/n\mathbb{Z}$, an element in $E(\mathbb{F}_{p^2})[n]$ can instead be transmitted by sending two scalars $(\alpha, \beta) \in \mathbb{Z}/n\mathbb{Z} \times \mathbb{Z}/n\mathbb{Z}$ that determine its representation with respect to a basis of the torsion subgroup. This requires that Alice and Bob have a way of arriving at the same basis for $E(\mathbb{F}_{p^2})[n]$. Following [Aza+16], we note that it is possible to decompose points into their $\mathbb{Z}/n\mathbb{Z} \times \mathbb{Z}/n\mathbb{Z}$ representation since for well-chosen SIDH parameters, $n = \ell^e$ is always smooth, which means that discrete logarithms in order n groups can be solved in polynomial time using the Pohlig-Hellman algorithm [PH78]. Given

that such SIDH parameters have $n_A \approx n_B$ (see [JDF11]), it follows that $n \approx \sqrt{p}$ and that sending elements of $E(\mathbb{F}_{p^2})[n]$ as two elements of $\mathbb{Z}/n\mathbb{Z}$ (instead of an element in $\mathbb{F}_{p^2}$) cuts the bandwidth required to send torsion points in half.

Although passing back and forth between (a, b) and $j(E)$ to (de)compress the curve is relatively inexpensive, the compression of the points P and Q requires three computationally intensive steps:

- *Step 1 – Constructing the n-torsion basis.* During both compression and decompression, Alice and Bob must, on input of the curve E, use a deterministic method to generate the same two-dimensional basis $\{R_1, R_2\} \in E(\mathbb{F}_{p^2})[n]$. The method used in [Aza+16] involves systematically sampling candidate points $R \in E(\mathbb{F}_{p^2})$, performing cofactor multiplication by h to move into $E(\mathbb{F}_{p^2})[n]$, and then testing whether or not $[h]R$ has "full" order n (and, if not, restarting).

- *Step 2 – Pairing computations.* After computing a basis $\{R_1, R_2\}$ of the group $E(\mathbb{F}_{p^2})[n]$, the task is to decompose the point P (and identically, Q) as $P = [\alpha_P]R_1 + [\beta_P]R_2$ and determine (α_P, β_P). While this could be done by solving a two-dimensional discrete logarithm problem (DLP) directly on the curve, Azarderakhsh et al. [Aza+16] use a number of Weil pairing computations to transform these instances into one-dimensional finite field DLPs in $\mu_n \subset \mathbb{F}_{p^2}^*$.

- *Step 3 – Solving discrete logarithms in μ_n.* The last step is to repeatedly use the Pohlig-Hellman algorithm [PH78] to solve DLPs in μ_n, and to output the four scalars α_P, β_P, α_Q and β_Q in $\mathbb{Z}/n\mathbb{Z}$.

Each of these steps presents a significant performance drawback for SIDH public-key compression. Subsequently, Azarderakhsh et al. report that, at interesting levels of security, each party's individual compression latency is more than a factor of ten times the latency of a full round of uncompressed key exchange [Aza+16, §5].

Our contributions. We present a range of new algorithmic improvements that decrease the total runtime of SIDH compression and decompression by an order of magnitude, bringing its performance close to that of a single round of SIDH key exchange. We believe that this makes it possible to consider public-key compression a default choice for SIDH, and it can further widen the gap between the key sizes resulting from practical SIDH key exchange implementations and their code- and lattice-based counterparts.

We provide a brief overview of our main improvements with respect to the three compression steps described above. All known implementations of SIDH (see e. g.

[DFJP14; AFJ14; CLN16a]) currently choose $n_A = \ell_A^{e_A} = 2^{e_A}$ and $n_B = \ell_B^{e_B} = 3^{e_B}$ for simplicity and efficiency reasons, so we focus on $\ell \in \{2, 3\}$ below; however, unless specified otherwise, we note that all of our improvements will readily apply to other values of ℓ.

- *Step 1 – Constructing the n-torsion basis.* We make use of some results arising from explicit 2- and 3-descent of elliptic curves to avoid the need for the expensive cofactor multiplication that tests the order of points. These results characterize the images of the multiplication-by-2 and multiplication-by-3 maps on E, and allow us to quickly generate points that are elements of $E(\mathbb{F}_{p^2}) \setminus [2]E(\mathbb{F}_{p^2})$ and $E(\mathbb{F}_{p^2}) \setminus [3]E(\mathbb{F}_{p^2})$. Therefore, we no longer need to check the order of (possibly multiple!) points using a full-length scalar multiplication by $n_A n_B$, but instead are *guaranteed* that one half-length cofactor multiplication produces a point of the correct order. For our purposes, producing points in $E \setminus [2]E$ is as easy as generating elliptic curve points whose x-coordinates are non-square (this is classical, e. g. [Hus04, Ch. 1(§4), Thm 4.1]). On the other hand, to efficiently produce points in $E \setminus [3]E$, we make use of the analogous characteristic described in more recent work on explicit 3-descent by Schaefer and Stoll [SS04]. Combined with a tailored version of the Elligator 2 encoding [Ber+13] for efficiently generating points on E, this approach gives rise to highly efficient n-torsion basis generation. This is described in detail in §2.

- *Step 2 – Pairing computations.* We apply a number of optimizations from the literature on elliptic curve pairings in order to significantly speed up the runtime of all pairing computations. Rather than using the Weil pairing (as was done in [Aza+16]), we use the more efficient Tate pairing [GHS02; Bar+02]. We organize the five pairing computations that are required during compression in such a way that only two Miller functions are necessary. Unlike all of the prior work done on optimized pairing computation, the pairings used in SIDH compression cannot take advantage of torsion subgroups that lie in subfields, which means that fast explicit formulas for point operations and Miller line computations are crucial to achieving a fast implementation. Subsequently, we derive new and fast inversion-free explicit formulas for computing pairings on supersingular curves, specific to the scenario of SIDH compression. Following the Miller loops, we compute all five final exponentiations by exploiting a fast combination of Frobenius operations together with either fast repeated cyclotomic squarings (from [SL03]) or our new formulas for enhanced cyclotomic cubing operations. The pairing optimizations are described in §3.

– *Step 3 – Solving discrete logarithms in μ_n.* All computations during the Pohlig-Hellman phase take place in the subgroup μ_n of the multiplicative group G_{p+1} of order $p+1$ in $\mathbb{F}^*_{p^2}$, where we take advantage of the fast cyclotomic squarings and cubings mentioned above, as well as the fact that $\mathbb{F}_{p^2}$ inversions are simply conjugations, so come almost for free (see §4.1). On top of this fast arithmetic, we build an improved version of the Pohlig-Hellman algorithm that exploits windowing methods to solve the discrete logarithm instances with lower asymptotic complexity than the original algorithm. For the concrete parameters, the new algorithm is approximately $14\times$ (resp. $10\times$) faster in $\mu_{2^{372}}$ (resp. $\mu_{3^{239}}$), while having very low memory requirements (see Table 1 and 2). This is all described in more detail in §4.

– *Improved compression.* Through normalizing the representation of P and Q in $(\mathbb{Z}/n\mathbb{Z})^4$, we are able to further compress this part of the public key representation into $(\mathbb{Z}/n\mathbb{Z})^3$. Subsequently, our public keys are around $\frac{7}{2}\log p$ bits, rather than the $4\log p$ bits achieved in [Aza+16]. To the best of our knowledge, this is as far as SIDH public keys can be compressed in practice. This is explained in §5.1.

– *Decompression.* The decompression algorithm – which involves only the first of the three steps above and a double-scalar multiplication – is also accelerated in this chapter. In particular, on top of the faster torsion basis generation, we show that the double-scalar multiplications can be absorbed into the shared secret computation. This makes them essentially free of cost. This is described in §5.2.

The combination of the three main improvements mentioned above, along with a number of further optimizations described in the rest of this chapter, yields enhanced compression software that is an order of magnitude faster than the initial software benchmarked in [Aza+16].

The compression software. We wrote the new suite of algorithms in plain C and incorporated the compression software into the SIDH library recently made available by Costello, Longa and Naehrig [CLN16a]; their software uses a curve with $\log p = 751$ that currently offers around 192 bits of classical security and 128 bits of quantum security. The public keys in their uncompressed software were $6\log p = 564$ bytes, while the compressed public keys resulting from our software are $\frac{7}{2}\log p = 330$ bytes. The software described in this chapter can be found in the version 2.0 release of the SIDH library at

`https://www.microsoft.com/en-us/research/project/sidh-library/`.

Although our software is significantly faster than the previously given compression benchmarks by Azarderakhsh et al. [Aza+16], we believe that the most meaningful benchmarks we can present are those that compare the latency of our optimized SIDH compression to the latency of the state-of-the-art key generation and shared secret computations in [CLN16a]. This gives the reader (and the PQ audience at large) an idea of the cost of public-key compression when both the raw SIDH key exchange and the optional compression are optimized to a similar level. We emphasize that although the SIDH key exchange software from [CLN16a] targeted one isogeny class at one particular security level, and therefore so does our compression software, all of our improvements apply identically to curves used for SIDH at other security levels, especially if the chosen isogeny degrees remain (powers of) 2 and 3. Moreover, we expect that the relative cost of compressed SIDH to uncompressed SIDH will stay roughly consistent across different security levels, and that our targeted benchmarks therefore give a good gauge on the current state-of-the-art.

It is important to note that, unlike the SIDH software from [CLN16a] that uses private keys and computes shared secrets, by definition our public-key compression software only operates on public data.[1] Thus, while we call several of their constant-time functions when appropriate, none of our functions need to run in constant-time.

Remark 1 (Ephemeral SIDH). A paper by Galbraith, Petit, Shani and Ti [Gal+16] gives, among other results, a realistic and serious attack on instantiations of SIDH that reuse static private/public key pairs. Although direct public-key validation in the context of isogeny-based cryptography is currently non-trivial, there are methods of indirect public-key validation (see e. g. [Kir+15; Gal+16]) that mirror the same technique proposed by Peikert [Pei14, §5-6] in the context of lattice-based cryptography, which is itself a slight modification of the well-known Fujisaki-Okamoto transform [FO99]. At present, the software from [CLN16a] only supports secure *ephemeral* SIDH key exchange, and does not yet include sufficient (direct or indirect) validation that allows the secure use of static keys. Thus, since our software was written around that of [CLN16a], we note that it too is only written for the target application of ephemeral SIDH key exchange. In this case attackers are not incentivized to tamper with public keys, so we can safely assume throughout this chapter that all public keys are well-formed. Nevertheless, we note that the updated key exchange protocols in [FO99; Pei14; Kir+15; Gal+16] still send values that can be compressed using

[1] There is a minor caveat here in that we absorb part of the decompression into the shared secret computation, which uses the constant-time software from [CLN16a] – see §5.

our algorithms. On a related note, we also point out that our algorithms readily apply to the other isogeny-based cryptosystems described in [DFJP14] for which the compression techniques were detailed in [Aza+16]. In all of these other scenarios, however, the overall performance ratios and relative bandwidth savings offered by our compression algorithms are likely to differ from those we report for ephemeral SIDH.

Remark 2 (Trading speed for simplicity and space). Since the compression code in our software library only runs on public data, and therefore need not run in constant-time, we use a variable-time algorithm for field inversions (a variant of the extended binary GCD algorithm [Kal95]) that runs faster than usual exponentiation methods (via Fermat's little theorem). Although inversions are used sparingly in our code and are not the bottleneck of the overall compression runtime, we opted to add a single variable-time algorithm in this case. However, during the design of our software library, we made several decisions in the name of simplicity that inevitably hampered the performance of the compression algorithms.

One such performance sacrifice is made during the computation of the torsion basis points in §2, where tests of quadratic and cubic residuosity are performed using field exponentiations. Here we could use significantly faster, but more complicated algorithms that take advantage of the classic quadratic and cubic reciprocity identities. Such algorithms require intermediate reductions modulo many variable integers, and a reasonably optimized generic reduction routine would increase the code complexity significantly. These tests are also used sparingly and are not the bottleneck of public-key compression, and in this case, we deemed the benefits of optimizing them to be outweighed by their complexity. A second and perhaps the most significant performance sacrifice made in our software is during the Pohlig-Hellman computations, where our windowed version of the algorithm currently fixes small window sizes in the name of choosing moderate space requirements. If larger storage is permitted, then Sutherland's analysis of an optimized version of the Pohlig-Hellman algorithm [Sut11] shows that this phase could be sped up significantly (see §4). But again, the motivation to push the limits of the Pohlig-Hellman phase is stunted by the prior (pairing computation) phase being the bottleneck of the overall compression routine. Finally, we note that the probabilistic components of the torsion basis generation phase (see §2) lend themselves to an amended definition of the compressed public keys, where the compressor can send a few extra bits or bytes in their public key to make for a faster and deterministic decompression. For simplicity (and again due to this phase not being the bottleneck of compression), we leave this more complicated adaptation to future consideration.

Organization. In §2 we present alternative algorithms that deterministically generate a basis, while in §3 we exploit the fact that the Weil pairings can be replaced by reduced Tate pairings and give an optimized algorithm that computes them all simultaneously. In §4 we present an efficient version of the Pohlig-Hellman algorithm that exploits windowing methods to solve the discrete logarithm instances with lower complexity than the original algorithm. In §5 we show that one of the four scalar in $\mathbb{Z}/n\mathbb{Z}$ need not be transmitted by normalizing the tuple $(\alpha_P, \beta_P, \alpha_Q, \beta_Q)$.

2 Constructing Torsion Bases

For a given $A \in \mathbb{F}_{p^2}$ corresponding to a supersingular curve

$$E/\mathbb{F}_{p^2} : y^2 = x^3 + Ax^2 + x$$

with $\#E(\mathbb{F}_{p^2}) = (n_A n_B)^2$, the goal of this section is to produce a basis for $E(\mathbb{F}_{p^2})[n]$ (with $n \in \{n_A, n_B\}$) as efficiently as possible. This amounts to computing two order n points R_1 and R_2 whose Weil pairing $w_n(R_1, R_2)$ has exact order n. Checking the order of the Weil pairing either comes for free during subsequent computations, or requires the amendments discussed in Remark 3 at the end of this section. Thus, for now our goal is simplified to efficiently computing points of order $n \in \{n_A, n_B\}$ in $E(\mathbb{F}_{p^2})$.

Let $\{n, \widetilde{n}\} = \{n_A, n_B\}$, write $n = \ell^e$ and let $\mathcal{O}$ be the identity in $E(\mathbb{F}_{p^2})$. The typical process of computing a point of exact order n is to start by computing $R \in E(\mathbb{F}_{p^2})$ and multiplying by the cofactor $\widetilde{n}$ to compute the candidate output $\widetilde{R} = [\widetilde{n}]R$. Note that the order of $\widetilde{R}$ divides n, but might not be n. Thus, we multiply $\widetilde{R}$ by ℓ^{e-1}, and if $[\ell^{e-1}]\widetilde{R} \neq \mathcal{O}$, we output $\widetilde{R}$, otherwise we must pick a new R and restart.

In this section we use explicit results arising from 2- and 3-descent to show that the cofactor multiplications by $\widetilde{n}$ and by ℓ^{e-1} can be omitted by making use of elementary functions involving points of order 2 and 3 to check whether points are (respectively) in $E \setminus [2]E$ or $E \setminus [3]E$. In both cases this guarantees that the subsequent multiplication by $\widetilde{n}$ produces a point of exact order n, avoiding the need to perform full cofactor multiplications to check order prior to the pairing computation, and avoiding the need to restart the process if the full cofactor multiplication process above fails to output a point of the correct order (which happens regularly in practice). This yields much faster algorithms for basis generation than those that are used in [Aza+16].

We discuss the 2^e-torsion basis generation in §2.2 and the 3^e-torsion basis gener-

ation in §2.3. We start in §2.1 by describing some arithmetic ingredients.

2.1 Square Roots, Cube Roots, and Elligator 2

In this section we briefly describe the computation of square roots and that of testing cubic residuosity in $\mathbb{F}_{p^2}$, as well as our tailoring of the Elligator 2 method [Ber+13] for efficiently producing points in $E(\mathbb{F}_{p^2})$.

Computing square roots in $\mathbb{F}_{p^2}$. Square roots in $\mathbb{F}_{p^2}$ are most efficiently computed via two square roots in the base field $\mathbb{F}_p$. Since $p \equiv 3 \pmod 4$, write $\mathbb{F}_{p^2} = \mathbb{F}_p(i)$ with $i^2 + 1 = 0$. Following [Sco07, §3.3], we use the simple identity

$$\sqrt{a + b \cdot i} = \pm\,(\alpha + \beta \cdot i)\ ,\ \text{where } \alpha = \sqrt{(a \pm \sqrt{a^2 + b^2})/2}\,,\ \beta = b/(2\alpha)\,, \qquad (1)$$

for $a, b, \alpha, \beta \in \mathbb{F}_p$. Both of $(a + \sqrt{a^2 + b^2})/2$ and $(a - \sqrt{a^2 + b^2})/2$ will not necessarily be square, so we make the correct choice by assuming that $z = (a + \sqrt{a^2 + b^2})/2$ is square and setting $\alpha = z^{(p+1)/4}$; if $\alpha^2 = z$, we output a square root as $\pm(\alpha + \beta i)$, otherwise we can output a square root as $\pm(\beta - \alpha i)$.

Checking cubic residuosity in $\mathbb{F}_{p^2}$. In §2.3 we will need to efficiently test whether elements $v \in \mathbb{F}_{p^2}$ are cubic residues or not. This amounts to checking whether $v^{(p^2-1)/3} = 1$ or not, which we do by first computing $v' = v^{p-1} = v^p/v$ via one application of Frobenius (i. e. $\mathbb{F}_{p^2}$ conjugation) and one $\mathbb{F}_{p^2}$ inversion. We then compute $v'^{(p+1)/3}$ as a sequence of $e_A = 372$ repeated squarings followed by $e_B - 1 = 238$ repeated cubings. Both of these squaring and cubing operations are in the order $p + 1$ cyclotomic subgroup of $\mathbb{F}_{p^2}^*$, so can take advantage of the fast operations described in §4.1.

Elligator 2. The naïve approach to obtaining points in $E(\mathbb{F}_{p^2})$ is to sequentially test candidate x-coordinates in $\mathbb{F}_{p^2}$ until $f(x) = x^3 + Ax^2 + x$ is square. Each of these tests requires at least one exponentiation in $\mathbb{F}_p$, and a further one (to obtain the corresponding y) if $f(x)$ is a square. The Elligator 2 construction deterministically produces points in $E(\mathbb{F}_{p^2})$ using essentially the same operations, so given that the naïve method can fail (and waste exponentiations), Elligator 2 performs significantly faster on average. The idea behind Elligator 2 is to let u be any non-square in $\mathbb{F}_{p^2}$, and for

any $r \in \mathbb{F}_{p^2}$, write

$$v = -\frac{A}{1+ur^2} \qquad \text{and} \qquad v' = \frac{A}{1+ur^2} - A\,. \tag{2}$$

Then either v is an x-coordinate of a point in $E(\mathbb{F}_{p^2})$, or else v' is [Ber+13]; this is because $f(v)$ and $f(v')$ differ by the non-square factor ur^2.

In our implementation we fix $u = i + 4$ as a system parameter and precompute a public table consisting of the values $-1/(1+ur^2) \in \mathbb{F}_{p^2}$ where r^2 ranges from 1 to 10. This table is fixed once-and-for-all and can be used (by any party) to efficiently generate torsion bases as A varies over the isogeny class. Note that the size of the table here is overkill, we very rarely need to use more than 3 or 4 table values to produce basis points of the correct exact order.

The key to optimizing the Elligator 2 construction (see [Ber+13, §5.5]) is to be able to efficiently modify the square root computation in the case that $f(v)$ is non-square, to produce $\sqrt{f(v')}$. This update is less obvious for our field than in the case of prime fields, but nevertheless achieves the same result. Referring back to (1), we note that whether or not $a + b \cdot i$ is a square in $\mathbb{F}_{p^2}$ is determined solely by whether or not $a^2 + b^2$ is a square in $\mathbb{F}_p$ [Sco07, §3.3]. Thus, if this check deems that $a + bi$ is non-square, we multiply it by $ur^2 = (i+4)r^2$ to yield a square, and this is equivalent to updating $(a, b) = (r(4a - b), r(a + 4b))$, which is trivial in the implementation.

2.2 Generating a Torsion Basis for $E(\mathbb{F}_{p^2})[2^{e_A}]$

The above discussion showed how to efficiently generate candidate points R inside $E(\mathbb{F}_{p^2})$. In this subsection we show how to efficiently check that R is in $E \setminus [2]E$, which guarantees that $[3^{e_B}]R$ is a point of exact order 2^{e_A}, and can subsequently be used as a basis element.

Since the supersingular curves $E/\mathbb{F}_{p^2} : y^2 = x(x^2 + Ax + 1)$ in our isogeny class have a full rational 2-torsion, we can always write them as $E/\mathbb{F}_{p^2} : y^2 = x(x - \gamma)(x - \delta)$. A classic result (c. f. [Hus04, Ch. 1(§4), Thm 4.1]) says that, in our case, any point $R = (x_R, y_R)$ in $E(\mathbb{F}_{p^2})$ is in $[2]E(\mathbb{F}_{p^2})$, i. e. is the result of doubling another point, if and only if x_R, $x_R - \gamma$ and $x_R - \delta$ are all squares in $\mathbb{F}_{p^2}$. This means that we do not need to find the roots δ and γ of $x^2 + Ax + 1$ to test for squareness, since we want the x_R such that at least one of x_R, $x_R - \gamma$ and $x_R - \delta$ are a non-square. We found it most efficient to simply ignore the latter two terms and reject any x_R that is square, since the first non-square x_R we find corresponds to a point R such that $[3^{e_B}]R$ has exact order 2^{e_A}, and further testing square values of x_R is both expensive

and often results in the rejection of R anyway.

In light of the above, we note that for the 2-torsion basis generation, the Elligator approach is not as useful as it is in the next subsection. The reason here is that we want to only try points with a non-square x-coordinate, and there is no exploitable relationship between the squareness of v and v' in (2) (such a relation only exists between $f(v)$ and $f(v')$). Thus, the best approach here is to simply proceed by trying candidate v's as consecutive elements of a list $L = [u, 2u, 3u, \dots]$ of non-squares in $\mathbb{F}_{p^2}$ until $(v^3 + Av^2 + v)$ is square; recall from above that this check is performed efficiently using one exponentiation in $\mathbb{F}_p$.

To summarize the computation of a basis $\{R_1, R_2\}$ for $E(\mathbb{F}_{p^2})[2^{e_A}]$, we compute R_1 by letting v be the first element in L where $(v^3 + Av^2 + v)$ is square. We do not compute the square root of $(v^3 + Av^2 + v)$, but rather use e_B repeated x-only tripling operations starting from v to compute x_{R_1}. We then compute y_{R_1} as the square root of $x_{R_1}^3 + Ax_{R_1}^2 + x_{R_1}$. Note that either choice of square root is fine, so long as Alice and Bob take the same one. The point R_2 is found identically, i. e. using the second element in L that corresponds to an x-coordinate of a point on $E(\mathbb{F}_{p^2})$, followed by e_B x-only tripling operations to arrive at x_{R_2}, and the square root computation to recover y_{R_2}. Note that the points R_1 and R_2 need not be normalized before their input into the pairing computation; as we will see in §3, the doubling-only and tripling-only pairings do not ever perform additions with the original input points, so the input points are essentially forgotten after the first iteration.

2.3 Generating a Torsion Basis for $E(\mathbb{F}_{p^2})[3^{e_B}]$

The theorem used in the previous subsection was a special case of more general theory that characterizes the action of multiplication-by-m on E. We refer to Silverman's chapter [Sil09, Ch. X] and to [SS04] for the deeper discussion in the general case, but in this chapter we make use of the explicit results derived in the case of $m = 3$ by Schaefer and Stoll [SS04], stating only what is needed for our purposes.

Let $P_3 = (x_{P_3}, y_{P_3})$ be any point of order 3 in $E(\mathbb{F}_{p^2})$ (recall that the entire 3-torsion is rational here), and let $g_{P_3}(x, y) = y - (\lambda x + \mu)$ be the usual tangent function to E at P_3. For any other point $R \in E(\mathbb{F}_{p^2})$, the result we use from [SS04] states that $R \in [3]E$ if and only if $g_{P_3}(R)$ is in $(\mathbb{F}_{p^2})^3$ (i. e. is a cube) for all of the 3-torsion points[2] P_3. Again, since we do not want points in $[3]E$, but rather points in $E \setminus [3]E$, we do not need to test that R gives a cube for all of the $g_{P_3}(R)$, we simply want to compute an R where *any* one of the $g_{P_3}(R)$ is not a cube. In this case the test involves both

[2] The astute reader can return to §2.2 and see that this is indeed a natural analogue of [Hus04, Ch. 1 (§4), Thm 4.1].

coordinates of R, so we make use of Elligator 2 as it is described in §2.1 to produce candidate points $R \in E(\mathbb{F}_{p^2})$.

Unlike the previous case, where the 2-torsion point $(0, 0)$ is common to all curves in the isogeny class, in this case it is computing a 3-torsion point P_3 that is the most difficult computation. We attempted to derive an efficient algorithm that finds x_{P_3} as any root of the (quartic) 3-division polynomial $\psi_3(A, x)$, but this solution involved several exponentiations in both $\mathbb{F}_{p^2}$ and $\mathbb{F}_p$, and was also hampered by the lack of an efficient enough analogue of (1) in the case of cube roots.[3] We found that a much faster solution was to compute the initial 3-torsion point via an x-only cofactor multiplication: we use the first step of Elligator 2 to produce an x-coordinate x_R, compute $x_{\widetilde{R}} = x_{[2^{e_A}]R}$ via e_A repeated doublings, and then apply k repeated triplings until the result of a tripling is $(X : Z) \in \mathbb{P}^1$ with $Z = 0$, which corresponds to the point $\mathcal{O}$, at which point we can set out x_{P_3}, the x-coordinate of a 3-torsion point P_3, as the last input to the tripling function. Moreover, if the number of triplings required to produce $Z = 0$ was $k = e_B$, then it must be that $\widetilde{R}$ is a point of exact order 3^{e_B}. If this is the case, we can use a square root to recover $y_{\widetilde{R}}$ from $x_{\widetilde{R}}$, and we already have one of our two basis points.

At this stage we either need to find one more point of order 3^{e_B}, or two. In either case we use the full Elligator routine to obtain candidate points R exactly as described in §2.1, use our point P_3 (together with our efficient test of cubic residuosity above) to test whether $g_{P_3}(R) = y_R - (\lambda x_R + \mu)$ is a cube, and if it is not, we output $\pm[2^{e_A}]R$ as a basis point; this is computed via a sequence of x-only doublings and one square root to recover $y_{[2^{e_A}]R}$ at the end. On the other hand, if $g_{P_3}(R)$ is a cube, then $R \in [3]E$, so we discard it and proceed to generate the next R via the tailored version of Elligator 2 above.

We highlight the significant speed advantage that is obtained by the use of the result of Schaefer and Stoll [SS04]. Testing that points are in $E \setminus [3]E$ by cofactor multiplication requires e_A point doubling operations and e_B point tripling operations, while the same test using the explicit results from 3-descent require one field exponentiation that tests cubic residuosity. Moreover, this exponentiation only involves almost-for-free Frobenius operations and fast cyclotomic squaring and cubing operations (again, see §4.1).

Remark 3. As mentioned at the beginning of this section, until now we have simplified the discussion to focus on generating two points R_1 and R_2 of exact order n. However, this does not mean that $\{R_1, R_2\}$ is a basis for $E(\mathbb{F}_{p^2})[n]$; this is the case if and only if the Weil pairing $w_n(R_1, R_2)$ has full order n. Although the Weil pairing

[3] A common subroutine when finding roots of quartics involve solving the so-called *depressed cubic*.

will have order n with high probability for random R_1 and R_2, the probability is not so high that we do not encounter it in practice. Thus, baked into our software is a check that this is indeed the case, and if not, an appropriate backtracking mechanism that generates a new R_2. We note that, following [CLN16a, §9] and [Gal+16, Section 2.5], checking whether or not the Weil pairing $w_n(R_1, R_2)$ has full order is much easier than computing it, and can be done by comparing the values $[n/\ell]R_1$ and $[n/\ell]R_2$.

3 The Tate Pairing Computation

Given the basis points R_1 and R_2 resulting from the previous section, and the two points P and Q in the (otherwise uncompressed) public key, we now have four points of exact order n. As outlined in §II.2.1, the next step is to compute the following five pairings to transfer the discrete logarithms to the multiplicative group $\mu_n \subset \mathbb{F}_{p^2}^*$:

$$
\begin{aligned}
e_0 &:= e_n(R_1, R_2) = f_{n,R_1}(R_2)^{(p^2-1)/n}, & e_1 &:= e_n(R_1, P) = f_{n,R_1}(P)^{(p^2-1)/n},\\
e_2 &:= e_n(R_1, Q) = f_{n,R_1}(Q)^{(p^2-1)/n}, & e_3 &:= e_n(R_2, P) = f_{n,R_2}(P)^{(p^2-1)/n},\\
e_4 &:= e_n(R_2, Q) = f_{n,R_2}(Q)^{(p^2-1)/n}.
\end{aligned}
$$

As promised in §1, the above pairings are already defined by the order n reduced Tate pairing $e_n : E(\mathbb{F}_{p^2})[n] \times E(\mathbb{F}_{p^2})/nE(\mathbb{F}_{p^2}) \mapsto \mu_n$, rather than the Weil pairing that was used in [Aza+16]. The rationale behind this choice is clear: the lack of special (subfield) groups inside the n-torsion means that many of the tricks used in the pairing literature cannot be exploited in the traditional sense. For example, there does not seem to be a straight-forward way to shorten the Miller loop by using efficiently computable maps arising from Frobenius (see e. g. [Bar+07], [HSV06], [Hes08]), our denominators lie in $\mathbb{F}_{p^2}$ so cannot be eliminated [Bar+02], and, while distortion maps exist on all supersingular curves [Ver04], finding efficiently computable and therefore useful maps seems hard for random curves in the isogeny class. The upshot is that the Miller loop is far more expensive than the final exponentiation in our case, and organizing the Tate pairings in the above manner allows us to get away with the computation of only two Miller functions, rather than the four that were needed in the case of the Weil pairing [Aza+16].

In the case of ordinary pairings over curves with a larger embedding degree,[4] the elliptic curve operations during the Miller loop take place in a much smaller

[4] This has long been the preferred choice of curve in the pairing-based cryptography literature.

field than the extension field; in the Tate pairing the point operations take place in the base field, while in the loop-shortened ate pairing [HSV06] (and its variants) they take place in a small-degree subfield. Thus, in those cases the elliptic curve arithmetic has only a minor influence on the overall runtime of the pairing.

In our scenario, however, we are stuck with elliptic curve points that have both coordinates in the full extension field. This means that the Miller line function computations are the bottleneck of the pairing computations (and, as it turns out, this is the main bottleneck of the whole compression routine). The main point of this section is to present optimized explicit formulas in this case; this is done in §3.1. In §3.2 we discuss how to compute the five pairings in parallel and detail how to compute the final exponentiations efficiently.

3.1 Optimized Miller Functions

We now present explicit formulas for the point operations and line computations in Miller's algorithm [Mil04]. In the case of the order-2^{e_A} Tate pairings in $E(\mathbb{F}_{p^2})[2^{e_A}]$, we only need the doubling-and-line computations, since no additions are needed. In the case of the order-3^{e_B} Tate pairings inside $E(\mathbb{F}_{p^2})[3^{e_B}]$, we investigated two options: the first option computes the pairing in the usual "double-and-add" fashion, reusing the doubling-and-line formulas with addition-and-line formulas, while the second uses a simple sequence of e_B tripling-and-parabola operations. The latter option proved to offer much better performance and is arguably more simple than the double-and-add approach.

We tried several coordinate systems in order to lower the overall number of field operations in both pairings, and after a close inspection of the explicit formulas in both the doubling-and-line and tripling-and-parabola operations, we opted to use the coordinate tuple $(X^2 : XZ : Z^2 : YZ)$ to represent intermediate projective points $P = (X : Y : Z) \in \mathbb{P}^2$ in $E(\mathbb{F}_{p^2})$. Note that all points in our routines for which we use this representation satisfy $XYZ \neq 0$, as their orders are strictly larger than 2. This ensures that the formulas presented below do not contain exceptional cases.[5]

Doubling-and-line operations. The doubling step in Miller's algorithm takes as input the tuple $(U_1 : U_2 : U_3 : U_4) = (X^2 : XZ : Z^2 : YZ)$ corresponding to the point $P = (X : Y : Z) \in \mathbb{P}^2$ in $E(\mathbb{F}_{p^2})$, and outputs the tuple $(V_1 : V_2 : V_3 : V_4) = (X_2^2 : X_2Z_2 : Z_2^2 : Y_2Z_2)$ corresponding to the point $[2]P = (X_2 : Y_2 : Z_2) \in \mathbb{P}^2$, as well

[5] The input into the final iteration in the doubling-only pairing is a point of order 2, but (as is well-known in the pairing literature) this last iterate is handled differently than all of the prior ones.

as the 5 coefficients in the Miller line function $l/v = (l_x \cdot x + l_y \cdot y + l_0)/(v_x x + v_0)$ with divisor $2(P) - (2P) - (\mathcal{O})$. The explicit formulas are given as

$$\begin{aligned} l_x &= 4U_4^3 + 2U_2U_4\left(U_1 - U_3\right), \quad l_y = 4U_2U_4^2, \qquad l_0 = 2U_1U_4\left(U_1 - U_3\right), \\ v_x &= 4U_2U_4^2, \qquad v_0 = U_2\left(U_1 - U_3\right)^2, \end{aligned}$$

together with

$$\begin{aligned} V_1 &= \left(U_1 - U_3\right)^4, \quad V_2 = 4U_4^2\left(U_1 - U_3\right)^2, \quad V_3 = 16U_4^4, \\ V_4 &= 2U_4\left(U_1 - U_3\right)\left(\left(U_1 - U_3\right)^2 + 2U_2\left(4U_2 + A\left(U_1 + U_3\right)\right)\right). \end{aligned}$$

The above point doubling-and-line function computation can be computed in $9\mathbf{M} + 5\mathbf{S} + 7\mathbf{a} + 1\mathbf{s}$. The subsequent evaluation of the line function at the second argument of a pairing, the squaring that follows, and the absorption of the squared line function into the running paired value costs $5\mathbf{M} + 2\mathbf{S} + 1\mathbf{a} + 2\mathbf{s}$.

Tripling-and-parabola operations. The tripling-and-parabola operation has as input the tuple $(U_1 : U_2 : U_3 : U_4) = (X^2 : XZ : Z^2 : YZ)$ corresponding to the point $P = (X : Y : Z) \in \mathbb{P}^2$ in $E(\mathbb{F}_{p^2})$, and outputs the tuple $(V_1 : V_2 : V_3 : V_4) = (X_3^2 : X_3Z_3 : Z_3^2 : Y_3Z_3)$ corresponding to the point $[3]P = (X_3 : Y_3 : Z_3) \in \mathbb{P}^2$, as well as the 6 coefficients in the Miller parabola function $l/v = (l_y \cdot y + l_{x,2} \cdot x^2 + l_{x,1}x + l_{x,0})/(v_x x + v_0)$ with divisor $3(P) - (3P) - 2(\mathcal{O})$. The explicit formulas are given as

$$\begin{aligned} l_y &= 8U_4^3, \\ l_{x,2} &= U_3(3U_1^2 + 4U_1AU_2 + 6U_1U_3 - U_3^2), \\ l_{x,1} &= 2U_2(3U_1^2 + 2U_1U_3 + 3U_3^2 + 6U_1AU_2 + 4A^2U_2^2 + 6AU_2U_3), \\ l_{x,0} &= U_1(-U_1^2 + 6U_1U_3 + 3U_3^2 + 4AU_2U_3), \\ v_x &= 8U_3U_4^3(3U_1^2 + 4U_1AU_2 + 6U_1U_3 - U_3^2)^4, \\ v_0 &= -8U_2U_4^3(3U_1^2 + 4U_1AU_2 + 6U_1U_3 - U_3^2)^2(U_1^2 - 6U_1U_3 - 3U_3^2 - 4AU_2U_3)^2, \end{aligned}$$

together with

$$\begin{aligned} V_1 &= 8U_4^3U_1(-U_1^2 + 6U_1U_3 + 3U_3^2 + 4AU_2U_3)^4, \\ V_2 &= 8U_2U_4^3(3U_1^2 + 4U_1AU_2 + 6U_1U_3 - U_3^2)^2(U_1^2 - 6U_1U_3 - 3U_3^2 - 4AU_2U_3)^2, \\ V_3 &= 8U_3U_4^3(3U_1^2 + 4U_1AU_2 + 6U_1U_3 - U_3^2)^4, \\ V_4 &= -8U_1U_3(3U_1^2 + 4U_1AU_2 + 6U_1U_3 - U_3^2)(-U_1^2 + 6U_1U_3 + 3U_3^2 + 4AU_2U_3) \end{aligned}$$

$$\cdot \left(16U_1U_3A^2U_2^2 + 28U_1^2AU_2U_3 + 28U_1U_3^2AU_2 + 4U_3^3AU_2 + 4U_1^3AU_2 \right.$$
$$\left. + 6U_1^2U_3^2 + 28U_1^3U_3 + U_3^4 + 28U_1U_3^3 + U_1^4\right)(U_3 + U_1 + AU_2)^2 .$$

The above point tripling-and-parabola function computation can be computed in $19\mathbf{M} + 6\mathbf{S} + 15\mathbf{a} + 6\mathbf{s}$. The subsequent evaluation of the line function at the second argument of a pairing, the cubing that follows, and the absorption of the cubed line function into the running paired value costs $10\mathbf{M} + 2\mathbf{S} + 4\mathbf{a}$.

Remark 4 (No irrelevant factors). It is common in the pairing literature to abuse notation and define the order-n Tate pairing as $e_n(P,Q) = f_P(Q)^{(p^k-1)/n}$, where k is the embedding degree (in our case $k = 2$), and f_P has divisor $\operatorname{div}(f_P) = n(P) - n(\mathcal{O})$. This is due to an early result of Barreto, Kim, Lynn and Scott [Bar+02, Theorem 1], who showed that the actual definition of the Tate pairing, i. e. $e_n(P,Q) = f_P(D_Q)^{(p^k-1)/n}$ where D_Q is a divisor equivalent to $(Q) - (\mathcal{O})$, could be relaxed in practical cases of interest by replacing the divisor D_Q with the point Q. This is due to the fact that the evaluation of f_P at $\mathcal{O}$ in such scenarios typically lies in a proper subfield of $\mathbb{F}_{p^k}^*$, so becomes an *irrelevant factor* under exponentiation by $(p^k - 1)/n$. In our case, however, this is generally not the case because the coefficients in our Miller functions lie in the full extension field $\mathbb{F}_{p^2}^*$. Subsequently, our derivation of explicit formulas replaces Q with the divisor $D_Q = (Q) - (\mathcal{O})$, and if the evaluation of the Miller functions at $\mathcal{O}$ is ill-defined, we instead evaluate them at the divisor $(Q + T) - (T)$ that is linearly equivalent to D_Q, where we fixed $T = (0,0)$ as the (universal) point of order 2. If $Q = (x_Q, y_Q)$, then $Q + T = (1/x_Q, -y_Q/x_Q^2)$, so evaluating the Miller functions at the translated point amounts to a permutation of the coefficients, and evaluating the Miller functions at $T = (0,0)$ simply leaves a quotient of the constant terms. These modifications are already reflected in the operation counts quoted above.

Remark 5. In the same vein as Remark 2, there is another possible speed improvement within the pairing computation that is not currently exploited in our library. Recall that during the generation of the torsion bases described in §2, the candidate basis point R is multiplied by the cofactor $n \in \{n_A, n_B\}$ to check whether it has the correct (full) order, and if so, R is kept and stored as one of the two basis points. Following the idea of Scott [Sco07, §9], the intermediate multiples of R (and partial information about the corresponding Miller line functions) that are computed in this cofactor multiplication could be stored in anticipation for the subsequent pairing computation, should R indeed be one of the two basis points. Another alternative here would be to immediately compute the pairings using the first two candidate ba-

sis points and to absorb the point order checks inside the pairing computations, but given the overhead incurred if either or both of these order checks fails, this could end up being too wasteful (on average).

3.2 Parallel Pairing Computation and the Final Exponentiation

In order to solve the discrete logarithms in the subgroup μ_n of n-th roots of unity in $\mathbb{F}_{p^2}^*$, we compute the five pairings $e_0 := e(R_1, R_2)$, $e_1 := e(R_1, P)$, $e_2 := e(R_1, Q)$, $e_3 := e(R_2, P)$, and $e_4 := e(R_2, Q)$. The first argument of all these pairings is either R_1 or R_2, i. e. all are of the form $f_{n,R_i}(S)^{(p^2-1)/n}$ for $i \in \{1, 2\}$ and $S \in \{R_2, P, Q\}$. This means that the only Miller functions we need are f_{n,R_1} and f_{n,R_2}, and we get away with computing only those two functions for the five pairing values. The two functions are evaluated at the points R_2, P, Q during the Miller loop to obtain the desired combinations. It therefore makes sense to accumulate all five Miller values simultaneously.

Computing the pairings simultaneously also becomes advantageous when it is time to perform inversions. Since we cannot eliminate denominators due to the lack of a subfield group, we employ the classic way of storing numerators and denominators separately to delay all inversions until the very end of the Miller loop. At this point, we have ten values (five numerators and five denominators), all of which we invert using Montgomery's inversion sharing trick [Mon87] at the total cost of one inversion and 30 $\mathbb{F}_{p^2}$ multiplications. The five inverted denominators are then multiplied by the corresponding numerators to give the five unreduced paired values. The reason we not only invert the denominators, but also the numerators, is because these inverses are needed in the easy part of the final exponentiation.

The final exponentiation is an exponentiation to the power $(p^2 - 1)/n = (p - 1)\frac{p+1}{n}$. The so-called *easy part*, i. e. raising to the power $p - 1$, is done by one application of the Frobenius automorphism and one inversion. The Frobenius is simply a conjugation in $\mathbb{F}_{p^2}$, and the inversion is actually a multiplication since we had already computed all required inverses as above. The so-called *hard part* of the final exponentiation has exponent $(p + 1)/n$ and needs to be done with regular exponentiation techniques. A nice advantage that makes the hard part quite a little easier is the fact that after a field element $a = a_0 + a_1 \cdot i \in \mathbb{F}_{p^2}$ has been raised to the power $p - 1$, it has order $p + 1$, which means it satisfies $1 = a^p \cdot a = a_0^2 + a_1^2$. This equation can be used to deduce more efficient squaring and cubing formulas that speed up this final part of the pairing computation (see §4.1 for further details).

Finally, in the specific setting of SIDH, where $p = n_A n_B - 1$, we have that $(p +$

$1)/n_A = n_B$ and $(p+1)/n_B = n_A$. When n_A and n_B are powers of 2 and 3, respectively, the hard part of the final exponentiation consists of only squarings or only cubings, respectively. These are done with the particularly efficient formulas described in §4.1 below.

4 Efficient Pohlig-Hellman in μ_{ℓ^e}

In this section, we describe how we optimize the Pohlig-Hellman [PH78] algorithm to compute discrete logarithms in the context of public-key compression for supersingular isogeny-based cryptosystems, and we show that we are able to improve on the quadratic complexity described in [PH78]. A similar result has already been presented in the more general context of finite abelian p-groups by Sutherland [Sut11]. However, our software employs a different optimization of the Pohlig-Hellman algorithm, by choosing small memory consumption over more efficient computation, which affects parameter choices. We emphasize that there are different time-memory trade-offs that could be chosen, possibly speeding up the Pohlig-Hellman computation by another factor of two (see Remark 2).

Following the previous sections, the two-dimensional discrete logarithm problems have been reduced to four discrete logarithm problems in the multiplicative group $\mu_{\ell^e} \subset \mathbb{F}_{p^2}^*$ of ℓ^e-th roots of unity, where $\ell, e \in \mathbb{Z}$ are positive integers and ℓ is a (small) prime. Before elaborating on the details of the windowed Pohlig-Hellman algorithm, we note that the condition $\ell^e \mid p+1$ makes various operations in μ_{ℓ^e} more efficient than their generic counterpart in $\mathbb{F}_{p^2}^*$.

4.1 Arithmetic in the Cyclotomic Subgroup

Efficient arithmetic in μ_{ℓ^e} can make use of the fact that μ_{ℓ^e} is a subgroup of the multiplicative group $G_{p+1} \subset \mathbb{F}_{p^2}^*$ of order $p+1$. Various subgroup cryptosystems based on the hardness of the discrete logarithm problem have been proposed in the literature [SS95; LV00], which can be interpreted in the general framework of torus-based cryptography [RS03]. The following observations for speeding up divisions and squarings in G_{p+1} have been described by Stam and Lenstra [SL03, §3.23 and Lemma 3.24].

Division in μ_{ℓ^e}. Let $p \equiv 3 \pmod 4$ and $\mathbb{F}_{p^2} = \mathbb{F}_p(i)$, $i^2 = -1$. For any $a = a_0 + a_1 \cdot i \in G_{p+1}$, where $a_0, a_1 \in \mathbb{F}_p$, we have that $a \cdot a^p = a^{p+1} = 1$, and therefore, the inverse $a^{-1} = a^p = a_0 - a_1 \cdot i$. This means that inversion in μ_{ℓ^e} can be

computed almost for free by conjugation, i. e. a single negation in $\mathbb{F}_p$, and thus divisions become as efficient as multiplications in μ_{ℓ^e}.

Squaring in μ_{ℓ^e}. The square of $a = a_0 + a_1 \cdot i$ can be computed as $a^2 = (2a_0^2 - 1) + ((a_0 + a_1)^2 - 1) \cdot i$ by essentially two base field squarings. In the case where such squarings are faster than multiplications, this yields a speed-up over generic squaring in $\mathbb{F}_{p^2}$.

Cubing in μ_{ℓ^e}. As far as we know, a cubing formula in G_{p+1} has not been considered in the literature before. We make the simple observation that a^3 can be computed as $a^3 = (a_0 + a_1 \cdot i)^3 = a_0(4a_0^2 - 3) + a_1(4a_0^2 - 1) \cdot i$, which needs only one squaring and two multiplications in $\mathbb{F}_p$, and is significantly faster than a naïve computation via a squaring and a multiplication in μ_{ℓ^e}.

4.2 Pohlig-Hellman

We now discuss the Pohlig-Hellman algorithm as presented in [PH78] for the group μ_{ℓ^e}. Let $r, g \in \mu_{\ell^e}$ be (non-trivial elements) such that $r = g^\alpha$ for some $\alpha \in \mathbb{Z}$. Given r and g, the goal is to determine the unknown scalar α. Denote α as

$$\alpha = \sum_{i=0}^{e-1} \alpha_i \ell^i \quad (\alpha_i \in \{0, \ldots, \ell - 1\}) .$$

Now define $s = g^{\ell^{e-1}}$, which is an element of order ℓ, and let $r_0 = r$. Finally, define

$$g_i = g^{\ell^i} \quad (0 \leq i \leq e - 1)$$

and

$$r_i = \frac{r_{i-1}}{g_{i-1}^{\alpha_{i-1}}} \quad (1 \leq i \leq e - 1) .$$

A straightforward computation then shows that for all $0 \leq i \leq e - 1$,

$$r_i^{\ell^{e-(i+1)}} = s^{\alpha_i} . \tag{3}$$

As proven in [PH78], this allows to inductively recover all α_i, by solving the discrete logarithms of Equation (3) in the group $\langle s \rangle$ of order ℓ. This can be done by precomputing a table containing all elements of $\langle s \rangle$. Alternatively, if ℓ is not small enough, one can improve the efficiency by applying the Baby-Step Giant-Step algorithm [Sha71], at the cost of some more precomputation. For small ℓ the computation

has complexity $O(e^2)$, while precomputing and storing the g_i requires $O(e)$ memory.

4.3 Windowed Pohlig-Hellman

The original version of the Pohlig-Hellman algorithm reduces a single discrete logarithm in the large group μ_{ℓ^e} to e discrete logarithms in the small group μ_ℓ. In this section we consider an intermediate version, by reducing the discrete logarithm in μ_{ℓ^e} to $\frac{e}{w}$ discrete logarithms in μ_{ℓ^w}. Let r, g, α as in the previous section, and let $w \in \mathbb{Z}$ be such that $w \mid e$. Note that it is not necessary for e to be divisible by w. If it is not, we replace e by $e - (e \pmod w)$, and compute the discrete logarithm for the final $e \pmod w$ bits at the end. However the assumption $w \mid e$ improves the readability of the arguments with little impact on the results, so we focus on this case here. Write

$$\alpha = \sum_{i=0}^{\frac{e}{w}-1} \alpha_i \ell^{wi} \quad (\alpha_i \in \{0, \ldots, \ell^w - 1\}),$$

define $s = g^{\ell^{e-w}}$, which is an element of order ℓ^w, and let $r_0 = r$. Let

$$g_i = g^{\ell^{wi}} \quad (0 \le i \le \frac{e}{w} - 1)$$

and

$$r_i = \frac{r_{i-1}}{g_{i-1}^{\alpha_{i-1}}} \quad (1 \le i \le \frac{e}{w} - 1). \tag{4}$$

A analogous computation to the one in [PH78] proves that

$$r_i^{\ell^{e-w(i+1)}} = s^{\alpha_i} \quad (0 \le i \le \frac{e}{w} - 1). \tag{5}$$

Hence we inductively obtain α_i for all $0 \le i \le \frac{e}{w} - 1$, and thereby α. To solve the discrete logarithm in the smaller subgroup μ_{ℓ^w}, we consider two strategies as follows.

Baby-step Giant-step in $\langle s \rangle$. As before, for small ℓ and w we can compute a table containing all ℓ^w elements of $\langle s \rangle$, making the discrete logarithms in (5) trivial to solve. As explained in [Sha71], the Baby-Step Giant-Step algorithm allows us to make a trade-off between the size of the precomputed table and the computational cost. That is, given some $v \le w$, we can compute discrete logarithms in $\langle s \rangle$ with computational complexity $O(\ell^v)$ and $O(\ell^{w-v})$ memory. Note that the computational complexity grows exponentially with v, whereas the memory requirement grows exponentially with $w - v$. This means that if we want to make w larger, we need to grow v as well,

as otherwise the table-size will increase. Therefore in order to obtain an efficient and compact algorithm, we must seemingly limit ourselves to small w. We overcome this limitation in the next section.

Pohlig-Hellman in $\langle s \rangle$. We observe that $\langle s \rangle$ has order ℓ^w, which is again smooth. This allows us to solve the discrete logarithms in $\langle s \rangle$ by using the original Pohlig-Hellman algorithm of §4.2. However, we can also choose to solve the discrete logarithm in $\langle s \rangle$ with a second windowed Pohlig-Hellman algorithm. Note the recursion that occurs, and we can ask what the optimal depth of this recursion is. We further investigate this question in §4.4.

4.4 The Complexity of Nested Pohlig-Hellman

We estimate the cost of an execution of the nested Pohlig-Hellman algorithm by looking at the cost of doing the computations in (4) and (5). Let F_n ($n \geq 0$) denote the cost of an n-times nested Pohlig-Hellman algorithm, and set $F_{-1} = 0$. Let $w_0, w_1, \ldots, w_n, w_{n+1}$ be the window sizes, and set $w_0 = e$, $w_{n+1} = 1$ (note that $n = 0$ corresponds to the original Pohlig-Hellman algorithm). Again, assume that $w_n \mid w_{n-1} \mid \cdots \mid w_1 \mid e$, which is merely for ease of exposition. The first iteration has window size w_1, which means that the cost of the exponentiations in (5) is

$$\left(\sum_{i=0}^{\frac{e}{w_1}-1} w_1 i \right) \mathbf{L} = \frac{1}{2} w_1 \left(\frac{e}{w_1} - 1 \right) \frac{e}{w_1} \mathbf{L} = \frac{1}{2} e \left(\frac{e}{w_1} - 1 \right) \mathbf{L} ,$$

where $\mathbf{L}$ denotes the cost of an exponentiation by ℓ. The exponentiations in (4) are performed with a scalar of size $\log \alpha_i \approx w_1 \log \ell$, which costs $\frac{1}{2} w_1 \log \ell \, \mathbf{M} + w_1 \log \ell \, \mathbf{S}$ on average. To do all $\frac{e}{w_1}$ of them then costs on average $\frac{1}{2} e \log \ell \, \mathbf{M} + e \log \ell \, \mathbf{S}$. We emphasize that for small w_i and ℓ this is a somewhat crude estimation, yet it is enough to get a good feeling for how to choose our parameters (i. e. window sizes). We choose to ignore the divisions, since there are only a few (see Remark 6) and, as we showed in §4.1, they can essentially be done at the small cost of a multiplication. We also ignore the cost of the precomputation for the $g^{\ell^{wi}}$, which is small as well (see Remark 7). To complete the algorithm, we have to finish the remaining $\frac{e}{w_1}$ $(n-1)$-times nested Pohlig-Hellman routines. In other words, we have shown that

$$F_n \approx \frac{1}{2} e \left(\frac{e}{w_1} - 1 \right) \mathbf{L} + \frac{1}{2} e \log \ell \, \mathbf{M} + e \log \ell \, \mathbf{S} + \frac{e}{w_1} F_{n-1} .$$

Now, by using analogous arguments on F_{n-1}, and induction on n, we can show that

$$F_n \approx \frac{1}{2}e\left(\frac{e}{w_1} + \ldots + \frac{w_{n-1}}{w_n} + w_n - n\right)\mathbf{L} + \frac{n+1}{2}e\log\ell\,\mathbf{M} + (n+1)e\log\ell\,\mathbf{S}\,. \tag{6}$$

To compute the optimal choice of $(w_1, \ldots, w_n)$, we compute the derivatives,

$$\frac{\partial F_n}{\partial w_i} = \frac{1}{2}e\left(\frac{1}{w_{i+1}} - \frac{w_{i-1}}{w_i^2}\right)\mathbf{L} \quad (1 \le i \le n)$$

and simultaneously equate them to zero to obtain the equations

$$w_i = \sqrt{w_{i-1}w_{i+1}} \quad (1 \le i \le n)\,.$$

From this we can straightforwardly compute that the optimal choice is

$$(w_1, \ldots, w_n) = \left(e^{\frac{n}{n+1}}, e^{\frac{n-1}{n+1}}, \ldots, e^{\frac{2}{n+1}}, e^{\frac{1}{n+1}}\right)\,. \tag{7}$$

Plugging this back into the Equation (6), we conclude that

$$F_n \approx \frac{1}{2}e\,(n+1)\left(e^{\frac{1}{n+1}} - 1\right)\mathbf{L} + \frac{n+1}{2}e\log\ell\,\mathbf{M} + (n+1)e\log\ell\,\mathbf{S}\,.$$

Observe that $F_0 \approx \frac{1}{2}e^2$, agreeing with the complexity described in [PH78]. However, as n grows, the complexity of the nested Pohlig-Hellman algorithm goes from quadratic to linear in e, giving a significant improvement.

Remark 6. We observe that for every two consecutive windows w_i and w_{i+1}, we need less than $\frac{w_i}{w_{i+1}}$ divisions for (4). Breaking the full computation down, it is easy to show that the total number of divisions is less than

$$\frac{e}{w_1} + \frac{e}{w_1}\left(\frac{w_1}{w_2} + \frac{w_1}{w_2}\left(\cdots + \frac{w_{n-2}}{w_{n-1}}\left(\frac{w_{n-1}}{w_n} + \frac{w_{n-1}}{w_n}w_n\right)\right)\right)\,,$$

which can be rewritten as $e\left(\frac{1}{w_1} + \frac{1}{w_2} + \ldots + \frac{1}{w_n} + \frac{w_n}{w_{n-1}}\right)$. Now we note that $w_{i+1} \mid w_i$, while $w_{i+1} \neq w_i$, for all $0 \le i \le n$. As $w_{n+1} = 1$, it follows that $w_{n+1-i} \ge 2^i$ for all $0 \le i \le n$. Therefore

$$e\left(\frac{1}{w_1} + \frac{1}{w_2} + \ldots + \frac{1}{w_n} + \frac{w_n}{w_{n-1}}\right) \le e\left(\frac{1}{2^n} + \frac{1}{2^{n-1}} + \ldots + \frac{1}{2} + 1\right) < 2e\,.$$

Table 1. Estimations of F_n in $\mu_{2^{372}}$ via a Magma implementation. Here **m** and **s** are the cost of multiplications and squarings in $\mathbb{F}_p$, while $\mathbf{M} = 3 \cdot \mathbf{m}$ and $\mathbf{S} = 2 \cdot \mathbf{s}$ are the cost of multiplications and squarings in $\mathbb{F}_{p^2}$. The costs are averaged over 100 executions of the algorithm. The $\mathbb{F}_p$ operation counts are generated from the $\mathbb{F}_{p^2}$ operation count estimations (c. f. §4.1), while the $\mathbb{F}_{p^2}$ are rounded down after averaging.

#	Windows				$\mathbb{F}_{p^2}$		$\mathbb{F}_p$		Table
n	w_1	w_2	w_3	w_4	**M**	**S**	**m**	**s**	$\mathbb{F}_{p^2}$
0	–	–	–	–	372	69 378	1 116	138 756	375
1	19	–	–	–	375	7 445	1 125	14 890	43
2	51	7	–	–	643	4 437	1 929	8 874	25
3	84	21	5	–	716	3 826	2 148	7 652	25
4	114	35	11	3	1 065	3 917	3 195	7 834	27

Remark 7. As every table element is of the form g^{ℓ^i}, where i is an integer such that $0 \leq i \leq e-1$, we conclude that we need at most $(e-1)\mathbf{L}$ to pre-compute all tables.

4.5 Discrete Logarithms in $\mu_{2^{372}}$

For this section we fix $\ell = 2$ and $e = 372$. In this case **L** is the cost of a squaring, i. e. $\mathbf{L} = \mathbf{S}$. To validate the approach, we present estimates for the costs of the discrete logarithm computations in $\mu_{2^{372}}$ through a Magma implementation. In this implementation we count every multiplication, squaring and division operation; on the other hand, some of these were ignored for the estimation of F_n above. The results are shown in Table 1 for $0 \leq n \leq 4$ choosing the window sizes as computed in (7). The improved efficiency as well as the significantly smaller table sizes are clear, and we observe that in the group $\mu_{2^{372}}$ it is optimal to choose $n = 3$.

4.6 Discrete Logarithms in $\mu_{3^{239}}$

We now fix $\ell = 3$ and $e = 239$ and present estimates for the costs of the discrete logarithm computations in $\mu_{3^{239}}$. Here **L** is now the cost of a cubing in $\mu_{3^{239}}$. As explained in §4.1, this is done at the cost of two multiplications and one squaring in $\mathbb{F}_p$. As shown in Table 2, the optimal case in $\mu_{3^{239}}$ is also $n = 3$.

5 Final Compression and Decompression

In this section we explain how to further compress a public key PK from $\mathbb{F}_{p^2} \times (\mathbb{Z}/n\mathbb{Z})^4$ to $\mathbb{F}_{p^2} \times \{0,1\} \times (\mathbb{Z}/n\mathbb{Z})^3$. Moreover, we also show how to merge the

Table 2. Estimations of F_n in $\mu_{3^{239}}$ via a Magma implementation. Here **m** and **s** are the cost of multiplications and squarings in $\mathbb{F}_p$, while $\mathbf{M} = 3 \cdot \mathbf{m}$, $\mathbf{S} = 2 \cdot \mathbf{s}$ and $\mathbf{C} = 2 \cdot \mathbf{m} + \mathbf{s}$ are the cost of multiplications, squarings and cubings in $\mathbb{F}_{p^2}$ respectively. The costs are averaged over 100 executions of the algorithm. The $\mathbb{F}_p$ operation counts are generated from the $\mathbb{F}_{p^2}$ operation count estimations (c. f. §4.1), while the $\mathbb{F}_{p^2}$ are rounded down after averaging.

#	Windows				$\mathbb{F}_{p^2}$			$\mathbb{F}_p$		Table
n	w_1	w_2	w_3	w_4	**M**	**S**	**C**	**m**	**s**	$\mathbb{F}_{p^2}$
0	–	–	–	–	239	78	28 680	58 077	28 836	242
1	19	–	–	–	349	341	3 646	8 339	4 328	35
2	51	7	–	–	612	660	2 192	6 220	3 512	22
3	84	21	5	–	656	836	1 676	5 320	3 348	17
4	114	35	11	3	942	1 252	1 427	5 716	3 931	16

key decompression with one of the operations of the SIDH scheme, making much of the decompression essentially free of cost. For ease of notation we follow the scheme described in [CLN16a], but everything that follows in this section generalizes naturally to the theory as originally described in [DFJP14].

5.1 Compression

Using the techniques explained in all previous sections, we can compress a triple $(E_A, P, Q) \in \mathbb{F}_{p^2}^3$ to a tuple $\left(A, \alpha_P, \beta_P, \alpha_Q, \beta_Q\right) \in \mathbb{F}_{p^2} \times (\mathbb{Z}/n\mathbb{Z})^4$ such that

$$(P, Q) = \left(\alpha_P R_1 + \beta_P R_2, \alpha_Q R_1 + \beta_Q R_2\right) ,$$

where $\{R_1, R_2\}$ is a basis of $E_A[n]$. As described in [CLN16a], the goal is to compute $\langle P + \ell m Q\rangle$ for $\ell \in \{2, 3\}$ and a secret key m. Again, we note that the original proposal expects to compute $\langle n_1 P + n_2 Q\rangle$ for secret key (n_1, n_2), but we emphasize that all that follows can be generalized to this case.

Since P is an element of order n, one of α_P or β_P lies in $(\mathbb{Z}/n\mathbb{Z})^*$, and

$$\langle P + \ell m Q\rangle = \begin{cases} \langle \alpha_P^{-1} P + \ell m \alpha_P^{-1} Q\rangle & \text{if} \quad \alpha_P \in (\mathbb{Z}/n\mathbb{Z})^* \\ \langle \beta_P^{-1} P + \ell m \beta_P^{-1} Q\rangle & \text{if} \quad \beta_P \in (\mathbb{Z}/n\mathbb{Z})^* \end{cases} .$$

Hence, to compute $\langle P + \ell m Q\rangle$, we do not necessarily have to recompute (P, Q). Instead, we can compute

$$\left(\alpha_P^{-1} P, \alpha_P^{-1} Q\right) = \left(R_1 + \alpha_P^{-1}\beta_P R_2, \alpha_P^{-1}\alpha_Q R_1 + \alpha_P^{-1}\beta_Q R_2\right)$$

or

$$\left(\beta_P^{-1}P, \beta_P^{-1}Q\right) = \left(\beta_P^{-1}\alpha_P R_1 + R_2, \beta_P^{-1}\alpha_Q R_1 + \beta_P^{-1}\beta_Q R_2\right).$$

Note that in both cases we have normalized one of the scalars. We conclude that we can compress the public key to $\texttt{PK} \in \mathbb{F}_{p^2} \times \{0,1\} \times (\mathbb{Z}/n\mathbb{Z})^3$, where

$$\texttt{PK} = \begin{cases} \left(A, 0, \alpha_P^{-1}\beta_P, \alpha_P^{-1}\alpha_Q, \alpha_P^{-1}\beta_Q\right) & \text{if} \quad \alpha_P \in (\mathbb{Z}/n\mathbb{Z})^* \\ \left(A, 1, \beta_P^{-1}\alpha_P, \beta_P^{-1}\alpha_Q, \beta_P^{-1}\beta_Q\right) & \text{if} \quad \beta_P \in (\mathbb{Z}/n\mathbb{Z})^* \end{cases}.$$

5.2 Decompression

Let $\left(A, b, \widetilde{\alpha}_P, \widetilde{\alpha}_Q, \widetilde{\beta}_Q\right) \in \mathbb{F}_{p^2} \times \{0,1\} \times (\mathbb{Z}/n\mathbb{Z})^3$ be a compressed public key. Note that, by the construction of the compression, there exists a $\gamma \in (\mathbb{Z}/n\mathbb{Z})^*$ such that

$$\left(\gamma^{-1}P, \gamma^{-1}Q\right) = \begin{cases} \left(R_1 + \widetilde{\alpha}_P R_2, \widetilde{\alpha}_Q R_1 + \widetilde{\beta}_Q R_2\right) & \text{if} \quad b = 0 \\ \left(\widetilde{\alpha}_P R_1 + R_2, \widetilde{\alpha}_Q R_1 + \widetilde{\beta}_Q R_2\right) & \text{if} \quad b = 1 \end{cases}. \tag{8}$$

The naïve strategy, analogous to the one originally explained in [Aza+16], would be to generate the basis $\{R_1, R_2\}$ of $E_A[n]$, use the public key to compute $\left(\gamma^{-1}P, \gamma^{-1}Q\right)$ via (8), and finally compute

$$\langle P + \ell m Q\rangle = \langle \gamma^{-1}P + \ell m \gamma^{-1}Q\rangle,$$

where $m \in \mathbb{Z}/n\mathbb{Z}$ is the secret key. The cost is approximately a 1-dimensional and a 2-dimensional scalar multiplication on E_A, while the final 1-dimensional scalar multiplication is part of the SIDH scheme.

Instead, we use (8) to observe that

$$\begin{aligned} \langle P + \ell m Q\rangle &= \langle \gamma^{-1}P + \ell m \gamma^{-1}Q\rangle \\ &= \begin{cases} \langle \left(1 + \ell m \widetilde{\alpha}_Q\right) R_1 + \left(\widetilde{\alpha}_P + \ell m \widetilde{\beta}_Q\right) R_2\rangle & \text{if} \quad b = 0 \\ \langle \left(\widetilde{\alpha}_P + \ell m \widetilde{\alpha}_Q\right) R_1 + \left(1 + \ell m \widetilde{\beta}_Q\right) R_2\rangle & \text{if} \quad b = 1 \end{cases}. \end{aligned}$$

Thus, since $1 + \ell m \widetilde{\alpha}_Q, 1 + \ell m \widetilde{\beta}_Q \in (\mathbb{Z}/n\mathbb{Z})^*$ (recall that $n = \ell^e$), we conclude that

$$\langle P + \ell m Q\rangle = \begin{cases} \langle R_1 + \left(1 + \ell m \widetilde{\alpha}_Q\right)^{-1} \left(\widetilde{\alpha}_P + \ell m \widetilde{\beta}_Q\right) R_2\rangle & \text{if} \quad b = 0 \\ \langle \left(1 + \ell m \widetilde{\beta}_Q\right)^{-1} \left(\widetilde{\alpha}_P + \ell m \widetilde{\alpha}_Q\right) R_1 + R_2\rangle & \text{if} \quad b = 1 \end{cases}.$$

Table 3. Comparison of SIDH key exchange and public key compression implementations targeting the 128-bit post-quantum and 192-bit classical security level. Benchmarks for our implementation were done on a 3.4GHz Intel Core i7-4770 Haswell processor running Ubuntu 14.04 LTS with TurboBoost disabled. Results for [Aza+16], obtained on a 4.0GHz Intel Core i7-4790K Haswell processor, were scaled from seconds to cycles using the CPU frequency; the use of TurboBoost is not specified in [Aza+16]. The performance results, expressed in millions of clock cycles, were rounded to the nearest 10^6 cycles.

	Function	**Party**	**This**	**[Aza+16]**
Public key (bytes)	—	Alice	328	385
		Bob	330	
Cycles (cc$\times 10^6$)	Keygen + Shared key	Alice	80	—
		Bob	92	
	Compression	Alice	109	6 081
		Bob	112	7 747
	Decompression	Alice	42	539
		Bob	34	493
	Total (no compression)	—	192	535
	Total (compression)	—	469	15 395

Decompressing in this way costs only a handful of field operations in $\mathbb{F}_{p^2}$ in addition to a 1-dimensional scalar multiplication on E_A. Since the scalar multiplication is already part of the SIDH scheme, this makes the cost of decompression essentially the cost of generating $\{R_1, R_2\}$. This is done exactly as explained in §2.

6 Implementation Details

To evaluate the performance of the new compression and decompression, we implemented the proposed algorithms in plain C and wrapped them around the SIDH software from [CLN16a]. This library supports a supersingular isogeny class defined over $p = 2^{372} \cdot 3^{239} - 1$, which contains curves of order $(2^{372} \cdot 3^{239})^2$. These parameters target 128 bits of post-quantum security.

Table 3 summarizes the results after benchmarking the software with the clang compiler v3.8.0 on a 3.4GHz Intel Core i7-4770 Haswell processor running Ubuntu 14.04 LTS with TurboBoost turned off. The details in the table include the size of compressed and uncompressed public keys, the performance of Alice's and Bob's key exchange computations using compression, the performance of the proposed compression and decompression routines, and the total costs of SIDH key exchange with and without the use of compression. These results are compared with those from the prior work by Azarderakhsh et al. [Aza+16], which uses a supersingular

isogeny class defined over $p = 2^{387} \cdot 3^{242} - 1$.

As can be seen in Table 3, the new algorithms for compression and decompression are significantly faster than those from [Aza+16]: compression is up to 66 times faster, while decompression is up to 15 times faster. Similarly, the full key exchange with compressed public keys can be performed about 30 times faster. Even though part of these speedups can indeed be attributed to the efficiency of the original SIDH library, this only represents a very small fraction of the performance difference (note that the original key exchange from the SIDH library is only 2.8 times faster than the corresponding result from [Aza+16]).

Our experimental results show that the use of compressed public keys introduces a factor-2.4 slowdown to SIDH. However, the use of compact keys (in this case, of 330 bytes) can now be considered practical; e. g. one round of SIDH key exchange is computed in only 150 milliseconds on the targeted platform.

Chapter VIII

Computing Isogenies between Montgomery Curves Using the Action of $(0,0)$

A recent paper by Costello and Hisil [CH17] presents efficient formulas for computing isogenies with odd-degree cyclic kernels on Montgomery curves. We provide a constructive proof of a generalization of this theorem which shows the connection between the shape of the isogeny and the simple action of the point $(0,0)$. This generalization removes the restriction of a cyclic kernel and allows for any separable isogeny whose kernel does not contain $(0,0)$. As a particular case, we provide efficient formulas for 2-isogenies between Montgomery curves and show that these formulas can be used in isogeny-based cryptosystems without expensive square root computations and without knowledge of a special point of order 8. We also consider elliptic curves in triangular form containing an explicit point of order 3.

1 Introduction

Ever since the introduction of elliptic curves to public-key cryptography by Miller [Mil86] and Koblitz [Kob87], they have been of interest to the cryptographic community. By using the group of points on an appropriately chosen elliptic curve where the discrete logarithm problem is assumed to be hard, many standard protocols can be instantiated. Notably, the Diffie–Hellman key exchange [DH76] and the Schnorr

signature scheme [Sch90] and its variants [Acc99a; Ber+12] allow for efficient implementations with high security and small keys. The efficiency of these curve-based algorithms is largely determined by the scalar multiplication routine, and as a result a lot of research has gone into optimizing this operation.

However, the threat of large-scale quantum computers has initiated the search for alternative algorithms that also resist quantum adversaries (which the classical curve-based systems do not [Sho94]). Building on the work of Couveignes [Cou06] and Rostovsev and Stolbunov [RS06], in 2011 Jao and De Feo [JDF11] proposed supersingular isogeny Diffie–Hellman (SIDH) as a key exchange protocol offering post-quantum security. Being based on the theory of elliptic curves, SIDH inherits several operations from traditional curve-based cryptography. As such, it has immediately benefited from decades of prior research into optimizing their operations. In particular, the Montgomery form of an elliptic curve has resulted in great performance. Initially proposed by Montgomery to speed up factoring using ECM [Mon87; Len87] and having been used for very efficient Diffie–Hellman key exchange (e. g. Bernstein's Curve25519 [Ber06a]), the current fastest instantiations of SIDH also employ Montgomery curves [CLN16b; KAMK16]. But, although the optimizations for scalar multiplication immediately carry over, the work on computing explicit isogenies on Montgomery curves is more limited.

For isogeny computations one commonly uses Vélu's formulas [Vél71]. However, if the elliptic curve has a form which is less general than (or different from) Weierstrass form, the formulas from Vélu are not guaranteed to preserve this. As isogenies are only well-defined up to isomorphism, one can post-compose with an appropriate isomorphism to return to the required form, but it may not be obvious with which isomorphism, or the isomorphism may be expensive to compute. A more elegant approach is to observe some extra structure on the curve model and require the isogenies to preserve this. For example, Moody and Shumow [MS16] apply this idea to Edwards and Huff curves by fixing certain points. Moreover, since the isogeny is invariant under addition by kernel points, there is a close connection between the isogeny and the action (by translation) of some chosen point. We make this more explicit in Theorem 1 for curves in Weierstrass form.

So far the approaches for obtaining formulas for isogenies on curves in Montgomery form have been rather ad hoc. In [DFJP14], De Feo, Jao and Plût apply Vélu's formulas and compose with the appropriate isomorphisms to return to Montgomery form. As noted in [DFJP14, §4.3.2], this approach fails to produce efficient results for 2-isogenies. That is, either one has to compute expensive square roots in a finite field (see e. g. §VII.2.1), or one relies on having an appropriate point of order 8. However,

this point of order 8 is not readily available for the final two 2-isogenies. As one suggested workaround in [DFJP14] they derive formulas for 4-isogenies between two curves in Montgomery form and propose to compute 2^e-isogenies as a chain of 4-isogenies. As a result, optimized SIDH implementations [CLN16a; KAMK16] have employed curves where e is even so that 2^e-isogenies can be comprised entirely of 4-isogenies. In [CH17], Costello and Hisil present elegant formulas for isogenies between Montgomery curves, but their theorem covers only the case of odd cyclic kernels and subsequently also does not address the case of 2-isogenies. Moreover, there is no justification for the derivation of these isogenies (except for showing that they work).

We bridge this gap by providing a more thorough analysis on isogenies between Montgomery curves. We show that the isogenies arising in [CH17] are exactly those fixing $(0,0)$. Since we enforce the isogeny to fix $(0,0)$, this point cannot be in the kernel. We show in Proposition 5 that this is the only restriction, and as a result present a generalization of [CH17, Theorem 1]. As a special case, we obtain formulas for 2-isogenies for 2-torsion points other than $(0,0)$. We then show that this point can be naturally avoided in well-designed isogeny-based cryptosystems (see §3.3), and discuss the application of the 2-isogeny formulas to isogeny-based cryptosystems.

Finally, although currently it does not give rise to faster isogeny formulas, we consider it worthwhile to point out that the same techniques immediately apply to other models. In particular, models derived from the *Tate Normal Form* [Hus04, §4.4], where one could expect to get simple ℓ-isogenies for $\ell \geq 3$. We work out the case $\ell = 3$, also known as the *triangular form* [Ber+15b], and derive isogenies by again fixing the action of the special point $(0,0)$.

Organization. We state a theorem in §2 that allows to describe an isogeny in terms of the abscissas of its kernel points and their translations by a chosen point Q. We apply this to Montgomery curves in §3 and to curves in triangular form in §4, in both cases using $Q = (0,0)$.

2 Isogenies on Weierstrass Curves

We begin by stating a straightforward, but rather useful theorem. By assuming to have knowledge on the action of an isogeny on a single point Q, we can translate this point by elements of the kernel to obtain a simple description of the isogeny. Many curve models have a natural choice for this point (e. g. $Q = (0,0)$ in Montgomery form, see §3).

Theorem 1. *Let k be a field and E/k an elliptic curve in Weierstrass form. Let $G \subset E(\bar{k})$ be a finite subgroup defined over k and*

$$\varphi : (x, y) \mapsto \big(f(x), c_0 y f'(x) + g(x)\big) \,, \quad c_0 \in \bar{k}^* \,, \tag{1}$$

a separable isogeny with $\ker(\varphi) = G$. Let $Q \in E(\bar{k})$ such that $Q \notin G$. Then

$$f(x) = c_1(x - x_Q) \prod_{T \in G \setminus \{\mathcal{O}\}} \frac{(x - x_{Q+T})}{(x - x_T)} + f(x_Q) \,, \quad \text{for } c_1 \in \bar{k}^* \,.$$

Proof. First note that the existence of φ follows from Vélu's formulas [Vél71], while a standard result [Gal12, Theorem 9.7.5] shows that it can be written in the form of (1) (where $f'(x)$ is the formal derivative df/dx of $f(x)$). More explicitly, following the notation of [Gal12, Theorem 25.1.6], there exist functions $u, t : G \setminus \{\mathcal{O}\} \to \bar{k}$ such that

$$f(x) = x + \sum_{T \in G_1 \cup G_2} \left(\frac{t(T)}{x - x_T} + \frac{u(T)}{(x - x_T)^2} \right) \,,$$

where $G_2 \subset G$ is the set of points of order 2 and $G_1 \subset E(\bar{k})$ is such that $G = \{\mathcal{O}\} \cup G_2 \cup G_1 \cup \{-T : T \in G_1\}$. Moreover, $u(T) = 0$ if and only if T has order 2. Collecting denominators, it is then immediate that there exists a function $w \in \bar{k}[x]$ such that $\deg(w) = |G|$ and

$$f(x) = \frac{w(x)}{v(x)} \,, \quad \text{where } v(x) = \prod_{T \in G \setminus \{\mathcal{O}\}} (x - x_T) \,.$$

Now define

$$h(x) = w(x)v(x_Q) - w(x_Q)v(x) \,.$$

Note that clearly $h(x_Q) = 0$. Since the value of f is invariant under the action of points in G, we in fact have that $h(x_{Q+T}) = 0$ for all $T \in G$. Therefore it follows that $(x - x_{Q+T}) \mid h(x)$ for all $T \in G$. If for all $T_1, T_2 \in G$ such that $T_1 \neq T_2$ we have that $x_{Q+T_1} \neq x_{Q+T_2}$, then it immediately follows that

$$\prod_{T \in G} (x - x_{Q+T}) \mid h(x) \,.$$

Otherwise[1], assume we have $T_1, T_2 \in G$ such that $T_1 \neq T_2$ and $x_{Q+T_1} = x_{Q+T_2}$.

[1] This proof is quite elementary. An alternative method (which is perhaps more illuminating) is to consider the divisor of $x - f(x_Q)$ on E/G and to pull it back via φ. We can then use the fact that $\operatorname{div}(f(x) - f(x_Q)) = \varphi^* \operatorname{div}(x - f(x_Q))$.

Since any x-coordinate corresponds to at most 2 points on E, it follows that $Q + T_2 = \pm(Q + T_1)$. However, $Q + T_2 = Q + T_1$ implies that $T_1 = T_2$, which contradicts our assumption. Therefore $Q + T_2 = -(Q + T_1)$ and

$$\begin{aligned}[2]\varphi(Q + T_1) &= \varphi(Q + T_1) + \varphi(Q + T_1) \\ &= \varphi(Q + T_1) + \varphi(Q + T_2) \\ &= \varphi(Q + T_1) - \varphi(Q + T_1) = \mathcal{O}\,.\end{aligned}$$

Moreover,

$$\begin{aligned}[2](Q + T_1) = \mathcal{O} &\iff Q + T_1 + Q + T_1 = \mathcal{O} \\ &\iff Q + T_1 - (Q + T_2) = \mathcal{O} \\ &\iff T_1 = T_2\,,\end{aligned}$$

which contradicts the assumption that $T_1 \neq T_2$. Thus $\psi_2(Q + T_1) \neq 0$ and by Lemma 2 we can conclude that $f'(x_{Q+T_1}) = 0$. Since away from the zeros of v we have

$$h(x) = \big(f(x) - f(x_Q)\big)v(x)v(x_Q)\,,$$

it follows from the fact that $f'(x_{Q+T_1}) = f(x_{Q+T_1}) - f(x_Q) = 0$ that $h'(x_{Q+T_1}) = 0$. Thus h has (at least) a double root at x_{Q+T_1}. That is, $(x - x_{Q+T_1})(x - x_{Q+T_2}) \mid h(x)$. It is then clear that indeed

$$\prod_{T \in G}(x - x_{Q+T}) \mid h(x)\,.$$

As $\deg(h) \leq \max(\deg(w), \deg(v)) = |G|$, there exists a constant $c \in \bar{k}^*$ such that

$$h(x) = c \prod_{T \in G}(x - x_{Q+T})\,.$$

Thus,

$$f(x) = \frac{w(x)}{v(x)} = \frac{h(x)}{v(x)v(x_Q)} + f(x_Q)\,.$$

The result follows by setting $c_1 = c/v(x_Q)$. □

Lemma 2. *Let the setup be as in Theorem 1 and let $R \in E(\bar{k}) \setminus G$. Then*

$$[2]\varphi(R) = \mathcal{O} \iff \psi_2(R)f'(x_R) = 0\,,$$

where ψ_2 is the 2-division polynomial.

Proof. Firstly note that $R \notin G$ and thus $\varphi(R) \neq \mathcal{O}$. Therefore, by definition of the 2-division polynomial on $\widetilde{E} = E/G$ it follows that

$$[2]\varphi(R) = \mathcal{O} \iff 2y_{\varphi(R)} + \widetilde{a}_1 x_{\varphi(R)} + \widetilde{a}_3 = 0\,,$$

where $\widetilde{a}_1$ and $\widetilde{a}_3$ are Weierstrass constants of $\widetilde{E}$ c. f. (1). Using the definition of φ and by recalling that (see e. g. [Gal12, Theorem 9.7.5])

$$2g(x_R) = c_0(a_1 x_R + a_3)f'(x_R) - \widetilde{a}_1 f(x_R) - \widetilde{a}_3\,,$$

a straightforward computation shows that

$$2y_{\varphi(R)} + \widetilde{a}_1 x_{\varphi(R)} + \widetilde{a}_3 = 0 \iff c_0\,(2y_R + a_1 x_R + a_3)\,f'(x_R) = 0\,.$$

Finally observe that $\psi_2(R) = 2y_R + a_1 x_R + a_3$ and $c_0 \neq 0$. □

Remark 3. Theorem 1 shows the connection between φ and the action of the point Q on abscissas of kernel elements, as φ is given by a product of functions $(x - x_{Q+T})/(x - x_T)$. If this action is simple (e. g. in Montgomery form where $x_{(0,0)+T} = 1/x_T$) then we can expect simple formulas for isogenies.

Remark 4. By relying on Theorem 1 we simplify the proof compared to earlier works [CH17; MS16]. Whereas those works present rational maps and prove them to be isogenies, we turn this argument around. We use the existence of the isogeny (by Vélu's formulas) and apply appropriate isomorphisms to enforce some structure to be maintained (e. g. $(0,0) \mapsto (0,0)$ in Montgomery form). We can then apply Theorem 1 to get formulas for the isogeny up to some constants. Finally we also use the formal group law. However, as opposed to proving that the rational functions defining the isogeny satisfy the curve relation of the co-domain curve, we can assume them to vanish and therefore extract the constants and the coefficients of the co-domain curve. This significantly simplifies the proof compared to earlier works (e. g. [MS16, Theorem 2] and [CH17, Theorem 1]).

3 Montgomery Form and 2-isogenies

In [CH17, Theorem 1] Costello and Hisil present rational maps which they prove to be isogenies between Montgomery curves. These isogenies are not unique, and are for example different from the formulas directly derived using Vélu's formulas. It is immediate that the isogenies in [CH17] have the property of fixing $(0,0)$. In §3.1

we show that this fact, together with the co-domain curve being in Montgomery form, characterizes their formulas (up to some sign choices). This generalizes the theorem by Costello and Hisil, by removing the restriction of kernels being cyclic and having odd order. In particular, in §3.2 we present formulas for 2-isogenies determined by points of order 2 other than $(0,0)$. Until now these had not appeared, and were considered to require the computation of a square root. In §3.3 we show how one could apply these formulas in an implementation. Although it requires only a modest change to the parameters, this does require care and can simplify the implementation. Finally in §3.4 we comment on a comparison to the state-of-the-art.

3.1 The General Formula

We begin by stating Proposition 5, which is the analogue of [CH17, Theorem 1].

Proposition 5. *Let k be a field with $\mathrm{char}(k) \neq 2$. Let $a \in k$ such that $a^2 \neq 4$ and $E/k : y^2 = x^3 + ax^2 + x$ is a Montgomery curve. Let $G \subset E(\bar{k})$ be a finite subgroup defined over k such that $(0,0) \notin G$ and let φ be a separable isogeny such that $\ker(\varphi) = G$. Then there exists a curve $\widetilde{E}/k : y^2 = x^3 + Ax^2 + x$ such that, up to post-composition by an isomorphism,*

$$\begin{aligned} \varphi : E &\to \widetilde{E} \\ (x,y) &\mapsto (f(x), c_0 y f'(x)) \end{aligned}$$

where

$$f(x) = x \prod_{T \in G \setminus \{\mathcal{O}\}} \frac{x x_T - 1}{x - x_T}.$$

Moreover, writing

$$\pi = \prod_{T \in G \setminus \{\mathcal{O}\}} x_T, \qquad \sigma = \sum_{T \in G \setminus \{\mathcal{O}\}} \left(x_T - \frac{1}{x_T} \right),$$

we have that $A = \pi(a - 3\sigma)$ and $c_0^2 = \pi$.

Proof. Over $\bar{k}$ we can always move E/G to Montgomery form. Let $P \in E(\bar{k})$ such that $x_P = 1$. Then $[2]P = (0,0)$, hence $[2]\varphi(P) = \varphi([2]P) \neq \mathcal{O}$ while $[4]\varphi(P) = [2](0,0) = \mathcal{O}$. Thus $\varphi(P)$ is a point of exact order 4, and we apply an isomorphism such that $x_{\varphi(P)} = (-1)^{|G|-1}$ (see e. g. [DFJP14, §4.3.2]). In particular this assures that $\varphi : (0,0) \mapsto (0,0)$. We then twist the y-coordinate via another isomorphism to set

the coefficient of y^2 to 1 and have

$$\widetilde{E} = E/G : y^2 = x^3 + Ax^2 + x\,.$$

Now apply Theorem 1 with $Q = (0,0)$. We obtain that

$$\begin{aligned} f(x) &= c_1(x - x_{(0,0)}) \prod_{T \in G\setminus\{\mathcal{O}\}} \frac{(x - x_{(0,0)+T})}{(x - x_T)} + f(x_{(0,0)}) \\ &= c_1 x \prod_{T \in G\setminus\{\mathcal{O}\}} \frac{\left(x - \frac{1}{x_T}\right)}{(x - x_T)}\,. \end{aligned}$$

As we set up $\widetilde{E}$ such that $f(1) = (-1)^{|G|-1}$, we find that

$$c_1 = \prod_{T \in G\setminus\{\mathcal{O}\}} x_T\,.$$

Feeding c_1 back into the equation for f puts it in the right form. At this point it only remains to find A and c_0 (observe that $g = 0$ in Montgomery form [Gal12, Theorem 9.7.5]). To this end we utilize the formal group law, similar to [CH17; MS16].

Let $t = x/y$ be a uniformizer at $\mathcal{O}$ and write $s = 1/y$. By observing that $s = t^3 + at^2 s + ts^2$ we can recursively substitute s into itself to get an expression $s(t) \in \mathbb{Z}[a][\![t]\!]$ as a power series[2]

$$s(t) = t^3 + at^5 + (a^2 + 1)t^7 + O(t^9)$$

This is well-defined, see for example [Sil09, §IV.1]. As a result we can write

$$\begin{aligned} 1/s(t) = y(t) &= t^{-3} - at^{-1} + O(t)\,, \\ ty(t) = x(t) &= t^{-2} - a + O(t^2)\,. \end{aligned}$$

Let $X(t) = f(x(t))$. Then

$$\begin{aligned} X(t) &= \pi t^{-2} + \pi(\sigma - a) + O(t^2)\,, & dX/dt &= -2\pi t^{-3} + O(t)\,, \\ dx/dt &= -2t^{-3} + O(t)\,, & (dx/dt)^{-1} &= -t^3/2 + O(t^7)\,. \end{aligned}$$

Now define

$$Y(t) = c_0 y(t) \cdot (df/dx) = c_0 y(t) \cdot (dX/dt) \cdot (dx/dt)^{-1}\,,$$

[2] We denote by $O(t^n)$ a series whose coefficients of t^m are zero for all $m < n$.

so that

$$Y(t) = c_0 \pi t^{-3} - c_0 a \pi t^{-1} + O(t)\,.$$

Writing

$$F(t) = Y(t)^2 - \left(X(t)^3 + AX(t)^2 + X(t)\right)$$

it follows that

$$F(t) = F_{-6} \cdot t^{-6} + F_{-4} \cdot t^{-4} + O(t^{-2})\,,$$

with

$$F_{-6} = \pi^2(c_0^2 - \pi)\,, \quad F_{-4} = \pi^2\left(3\pi(a-\sigma) - 2ac_0^2 - A\right)\,.$$

Now since by assumption φ is an isogeny with co-domain curve $\widetilde{E}$, and since F is precisely the equation defining $\widetilde{E}$, we must have $F = 0$. Solving $F_{-6} = 0$ and $F_{-4} = 0$ simultaneously leads to the desired equations for c_0^2 and A. Note that this way we have only defined c_0 up to sign. However, the sign choice merely induces a composition with $[-1]$ and therefore does not affect φ up to isomorphism. □

Remark 6. It is perhaps not immediately obvious that Proposition 5 is a generalization of the result by Costello and Hisil [CH17, Theorem 1]. Our result assumes the domain curve E to be of the form $y^2 = x^3 + ax^2 + x$, while their theorem also accounts for curves $E_0 : by^2 = x^3 + ax^2 + x$. Moreover, the map itself looks slightly different. However, it is straightforward to check that if one pre-composes with the isomorphism

$$\psi_0 : E_0 \to E\,, \quad (x, y) \mapsto (x, y\sqrt{b})$$

and post-composes with the isomorphism

$$\begin{aligned}\psi_1 : \widetilde{E} \to E_1 : By^2 + x^3 + Ax^2 + x\,,\\ (x, y) \mapsto \left(x, \frac{y}{\sqrt{\pi b}}\right)\end{aligned}$$

then one recovers the theorem from Costello and Hisil in the case of odd-degree cyclic kernels. Ignoring these twists in Proposition 5 simplifies the proof. For example, see Proposition 9.

Remark 7. If $k = \mathbb{F}_q$ is a (large-characteristic) finite field, then possibly π is a non-square in $\mathbb{F}_q$. As a result φ is not defined over $\mathbb{F}_q$. However, in that case the map

$$(x, y) \mapsto (f(x), yf'(x))$$

is defined over $\mathbb{F}_q$ with co-domain curve $\widetilde{E}^{(t)} : \pi y^2 = x^3 + Ax^2 + x$. This is the

quadratic twist of $\widetilde{E}$. Since $\widetilde{E}$ and its twist have the same Kummer line, we eliminate this issue by projecting to $\mathbb{P}^1$ (i. e. by using x-only arithmetic).

Remark 8. If we set up an SIDH instance with $\ell_A = 2$ and $e_A \geq 2$ then the x-coordinates of points of order 2 are in fact squares. This follows from [Hus04, Ch. 1, Thm 4.1] combined with the doubling formulas for Montgomery curves, as noted in §VII.2.2. Since all x-coordinates of points with orders other than 2 appear twice in the equation for π, it follows that π is actually a square. Therefore φ is defined over $\mathbb{F}_{p^2}$, and in particular we always have $\#E(\mathbb{F}_{p^2}) = \#\widetilde{E}(\mathbb{F}_{p^2})$. This is (implicitly) used in formulas for public-key compression (see Chapter VII and [Zan+18]).

3.2 2-isogenies

As an immediate consequence of Proposition 5 we obtain formulas for 2-isogenies for 2-torsion points other than $(0,0)$.

Proposition 9. *Let k be a field with $\mathrm{char}(k) \neq 2$. Let $a, b \in k$ such that $b \neq 0$ and $a^2 \neq 4$, and $E/k : by^2 = x^3 + ax^2 + x$ is a Montgomery curve. Let $P \in E(k)$ such that $P \neq (0,0)$ and $[2]P = \mathcal{O}$. Then*

$$\begin{aligned} \varphi : E &\to \widetilde{E}/k : By^2 = x^3 + Ax^2 + x \\ (x, y) &\mapsto (f(x), yf'(x)) , \end{aligned}$$

with $B = x_P b$ and $A = 2(1 - 2x_P^2)$ is a 2-isogeny with $\ker(\varphi) = \langle P \rangle$, where

$$f(x) = x \cdot \frac{xx_P - 1}{x - x_P} .$$

Proof. This is exactly the statement in Proposition 5 composed with the isomorphisms ψ_0 and ψ_1 from Remark 6. The result follows by using the identity $ax_P = -(x_P^2 + 1)$ to derive A. □

We also compute the kernel of the dual of φ, which will be helpful in §3.3 for larger degree isogenies.

Corollary 10. *Let the setup be as in Proposition 9. Then $\ker(\widehat{\varphi}) = \langle (0,0) \rangle$.*

Proof. Let ψ be a separable isogeny with domain $\widetilde{E}$ and kernel $\langle (0,0) \rangle$. Then certainly $E[2] \subset \ker(\psi \circ \varphi)$, and since $\deg(\psi \circ \varphi) = 4$ we in fact have $E[2] = \ker(\psi \circ \varphi)$. Thus $\psi = \widehat{\varphi}$ up to isomorphism by uniqueness of the dual isogeny, and hence $\ker(\widehat{\varphi}) = \ker(\psi)$. □

The statement and proof of Proposition 9 does not explain why we are able to compute 2-isogenies without explicit square roots, while earlier works [CH17; DFJP14] could not. We provide a more direct computation in Remark 11 to show why this is the case.

Remark 11. In [DFJP14, §4.3.2] the authors describe a 2-isogeny with kernel (0,0) as

$$\begin{aligned} \phi : E \to F : by^2 &= x^3 + (a+6)x^2 + 4(2+a)x \\ (x,y) &\mapsto \left(\frac{(x-1)^2}{x}, y\left(1 - \frac{1}{x^2}\right)\right). \end{aligned}$$

The coefficient of x can be removed by computing $2\sqrt{a+2}$ and composing with the isomorphism

$$(x,y) \mapsto \left(\frac{x}{2\sqrt{a+2}}, \frac{y}{2\sqrt{a+2}}\right),$$

putting F in the desired form. This requires computing a square root, which could be avoided by having knowledge of a point $P_8 = \left(2\sqrt{a+2}, —\right)$ of order 8 above $(0,0)$. Instead, we observe that we can compose with the isomorphism

$$\begin{aligned} \psi : F \to G : \frac{b}{\sqrt{a^2-4}} y^2 &= x^3 - \frac{2a}{\sqrt{a^2-4}} \cdot x^2 + x \\ (x,y) &\mapsto \left(\frac{x+a+2}{\sqrt{a^2-4}}, \frac{y}{\sqrt{a^2-4}}\right), \end{aligned}$$

which moves the kernel of $\widetilde{\phi}$ to $(0,0)$. This requires computing $\sqrt{a^2-4}$ and therefore also relies on a square root. However, if $P_2 = (x_2, 0)$ is a point of order 2 on E with $x_2 \neq 0$, then $x_2^2 + ax_2 + 1 = 0$. Therefore it is immediate that $\sqrt{a^2-4} = 2x_2 + a$, allowing us to compute the isomorphism efficiently. We have such a point by assumption in Proposition 9. We can now compute φ as $\psi \circ \phi \circ \chi$, where χ is an isomorphism mapping P_2 to $(0,0)$ (e. g. [DFJP14, Equation (15)]).

To provide explicit operation counts we move to projective space and project to $\mathbb{P}^1$. Let $P = (X_P : 0 : Z_P)$ be a point of order 2 on $E : bY^2Z = X^3 + aX^2Z + XZ^2$ such that $X_P \neq 0$. Then by Proposition 9

$$\begin{aligned} \varphi : E \to \widetilde{E} : BY^2Z &= X^3 + AX^2Z + XZ^2 \\ (X : — : Z) &\mapsto (X(XX_P - ZZ_P) : — : Z(XZ_P - ZX_P)) \end{aligned}$$

is a 2-isogeny with kernel $\langle P \rangle$. We have $A = 2\left(Z_P^2 - 2X_P^2\right)/Z_P^2$, and to avoid inversions we represent it projectively as $(A : 1) = \left(2\left(Z_P^2 - 2X_P^2\right) : Z_P^2\right)$. However, the

doubling formulas on Montgomery curves use $(A+2)/4$ instead of A, and we see that $(A+2:4) = (Z_P^2 - X_P^2 : Z_P^2)$. This can be computed in $2\mathbf{S} + 1\mathbf{a}$. Moreover, we observe that

$$X(XX_P - ZZ_P) = X\Big[(X-Z)(X_P+Z_P) - (X+Z)(Z_P-X_P)\Big],$$
$$Z(XZ_P - ZX_P) = Z\Big[(X-Z)(X_P+Z_P) + (X+Z)(Z_P-X_P)\Big].$$

This can be computed in $4\mathbf{M} + 6\mathbf{a}$ via the sequence of operations

$$T_0 = X_P + Z_P\,,\ T_1 = X_P - Z_P\,,\ T_2 = X + Z\,,\ T_3 = X - Z\,,\ T_4 = T_3 \cdot T_0\,,$$
$$T_5 = T_2 \cdot T_1\,,\ T_6 = T_4 - T_5\,,\ T_7 = T_4 + T_5\,,\ T_8 = X \cdot T_6\,,\ T_9 = Z \cdot T_7\,.$$

If we assume $X_P + Z_P$ and $Z_P - X_P$ to be pre-computed, the cost reduces to $4\mathbf{M} + 4\mathbf{a}$. This would for example apply if we require multiple evaluations of the isogeny (e. g. in SIDH). Also note that $Z_P^2 - X_P^2 = (X_P + Z_P)(Z_P - X_P)$ which allows us to compute the curve coefficient in $\mathbf{M} + \mathbf{S}$. This may or may not be worth it, depending on the underlying architecture.

3.3 Application to Isogeny-based Cryptography

In the general setting it is not true that the kernels appearing in the computations cannot contain the point $(0,0)$, so it is not clear that the 2-isogenies can immediately be used. In a similar fashion, it is not true in general that kernels of 4-isogenies cannot contain $(1, \pm\sqrt{(a+2)/b})$ or $(-1, \pm\sqrt{(a-2)/b})$. In [CLN16a, §3] and [CH17] this assumption is used without justification (implicitly by replacing ψ_4 with $\widehat{\psi}_4$). This is dealt with by using a separate function `first_4_isog` for the first 4-isogeny, which is the only kernel that can contain such a point (a proof of which does not appear). However, Lemma 12 and Corollary 13 show that we can avoid these points with only a minor restriction on the keyspace. Applying this restriction to [CLN16a] makes the function `first_4_isog` redundant, simplifying the implementation.

Lemma 12. *Let $e, f \in \mathbb{Z}_{\geq 0}$ and let $p = 2^e 3^f - 1$ be prime. Let $E/\mathbb{F}_{p^2}$ be a supersingular elliptic curve in Montgomery form such that $\#E(\mathbb{F}_{p^2}) = (p+1)^2$. Let $P, Q \in E(\mathbb{F}_{p^2})$ such that $E[2^e] = \langle P, Q\rangle$ and $[2^{e-1}]Q = (0,0)$. Let $\alpha \in \mathbb{Z}/2^e\mathbb{Z}$. Then $(0,0) \notin \langle P + [\alpha]Q\rangle$.*

Proof. It is clear that $\langle P + [\alpha]Q\rangle$ can only contain a single point of order 2, namely $[2^{e-1}](P + [\alpha]Q)$. But by assumption on Q we know that $[2^{e-1}](P + [\alpha]Q) \neq (0,0)$, hence the result follows. □

By Lemma 12 we know that we can compute the 2^e-isogenies as defined in Proposition 5. However, as the degrees grow this will quickly be impractical. Instead, we do the computations as a sequence of 2-isogenies (i. e. as in Proposition 9) [DFJP14, §4]. Therefore we must show that none of these intermediate isogenies has a kernel generated by $(0,0)$.

Corollary 13. *Let the setup be as in Lemma 12 and write $R = P + [\alpha]Q$. Let φ be an isogeny such that $\ker(\varphi) = \langle R \rangle$ and suppose that we compute*

$$\begin{aligned} \varphi &= \varphi_{e-1} \circ \cdots \circ \varphi_0 \,, \\ \ker(\varphi_0) &= \langle [2^{e-1}]R \rangle \,, \\ \ker(\varphi_i) &= \langle \left[2^{e-i-1}\right] \varphi_{i-1} \cdots \varphi_0 \,(R) \rangle \,, \quad (\textit{for } 1 \leq i \leq e-1) \end{aligned}$$

as a sequence of 2-isogenies, each one computed as in Proposition 9. Then $(0,0) \notin \ker(\varphi_i)$ for all $0 \leq i \leq e-1$.

Proof. We apply induction on i. The statement for $i = 0$ follows from Lemma 12. Let $i > 0$. Then $\ker(\widehat{\varphi}_{i-1}) = \langle (0,0) \rangle$ by the inductive hypothesis and by Corollary 10. But since the walk determined by φ is non-backtracking, it follows that $\ker(\varphi_i) \neq \langle (0,0) \rangle$. As $\#\ker(\varphi_i) = 2$, we conclude that $(0,0) \notin \ker(\varphi_i)$. □

The keyspace is determined by tuples (γ, δ) which define kernels of the form $\langle [\gamma]P + [\delta]Q \rangle$, where not simultaneously $\gamma \equiv 0 \pmod 2$ and $\delta \equiv 0 \pmod 2$. We can divide the space into the three disjoint sets (of equal size)

$$\mathcal{K}_{(i,j)} = \{(\gamma, \delta) : \gamma \equiv i \pmod 2 \,, \delta \equiv j \pmod 2\} \,,$$

for $(i,j) \in \{(0,1),(1,0),(1,1)\}$. The restriction on the keyspace then corresponds exactly to disallowing $\mathcal{K}_{(0,1)}$, removing $1/3$ of the keyspace. It is easy to see that these keys define the isogeny walks for which the first 2-isogeny has kernel $\langle (0,0) \rangle$. Note that this depends on the choice of 2^e-torsion basis $\{P, Q\}$, where we choose Q to lie above $(0,0)$. A similar argument applies to the use of 4-isogenies in [CLN16a].

Remark 14. The initial proposal to use curves in Montgomery form [CLN16a, §4] suggested taking P as an $\mathbb{F}_p$-rational point on the curve $E_0/\mathbb{F}_p : y^2 = x^3 + x$ with $j(E_0) = 1728$ and Q as the image of P under the distortion map $(x, y) \mapsto (-x, iy)$. This allows a compressed representation of $\{P, Q\}$. Although this does not work for the basis as chosen in Lemma 12, it only results in a small increase in the size of public parameters (which never need to be transferred).

Table 1. Comparison of the costs of evaluating 2-isogenies and 4-isogenies.

Operation	2-isogeny	2×2-isogeny	4-isogeny [CH17]
Compute $(A+2:4)$	$2\mathbf{S}+1\mathbf{a}$	$4\mathbf{S}+2\mathbf{a}$	$4\mathbf{S}+5\mathbf{a}$[3]
First evaluation	$4\mathbf{M}+6\mathbf{a}$	$8\mathbf{M}+12\mathbf{a}$	$6\mathbf{M}+2\mathbf{S}+6\mathbf{a}$
Subsequent evaluations	$4\mathbf{M}+4\mathbf{a}$	$8\mathbf{M}+8\mathbf{a}$	$6\mathbf{M}+2\mathbf{S}+6\mathbf{a}$

3.4 Relating 2-isogenies and 4-isogenies

It is easy to see that the 4-isogenies from [CH17, Appendix A], which are currently the fastest formulas, can be derived by applying the 2-isogenies from §3.2 twice. That is, since they have equal kernel they are equal up to composition with an isomorphism. Both isogenies have a Montgomery curve as co-domain, of which there are at most six per isomorphism class (by looking at the formula for the j-invariant). Also, the dual is generated by a point $P \in \{(1, \pm\sqrt{(a+2)/b}), (-1, \pm\sqrt{(a-2)/b})\}$ in both cases. Therefore we can transform one into the other by possibly composing with the simple isomorphisms $(x, y) \mapsto (x, -y)$ and $(x, y) \mapsto (-x, iy)$, where $i \in \bar{k}$ such that $i^2 = -1$. As a result, applying the 2-isogenies twice will not have more efficient formulas than the 4-isogenies. Indeed, if this were the case we could use the above transformation to obtain equally fast 4-isogenies. We summarize the costs in Table 1.

Beside their theoretic value, there are some small upsides to using 2-isogenies in an implementation. Firstly, the computation leaks only a single bit as opposed to two [DFJP14, §4.3.2]. Instead of leaking the dual of the final 4-isogeny, it would only leak the dual of the last 2-isogeny. Also, in some cases one may be able to select smaller parameters for a certain given security level. Primes of the form $2^e3^f - 1$ where $e \approx \log_2(3^f)$ are somewhat sparse, and depending on one's requirements restricting e to be even could result in a (much) larger prime than hoped for. Alternatively, one could of course achieve this by doing a single 2-isogeny followed by a chain of 4-isogenies. However, this does come at the cost of having to implement more algorithms, increasing the size and complexity of an (already complex) implementation. Finally, having worked out formulas for isogenies of even degree and by showing how to avoid $(0, 0)$, we are able to straightforwardly write down formulas for 2^e-isogenies with $e \geq 3$. It remains to be seen if these can be made more efficient

[3] Many of these additions are not needed to compute $(A+2:4)$, but are used as pre-computation for the isogeny evaluation. We provide the counts as is to align with [CH17] since it does not affect our comparison of the costs of large degree isogeny evaluations.

than repeated applications of 4-isogenies.

4 Triangular Form and 3-isogenies

Given the generality of Theorem 1, an obvious question is whether there are other classes of curves which could possibly give rise to simple formulas for isogenies. In this section we analyze curves in triangular form $E/k : y^2 + axy + y = x^3$ containing a point $(0,0)$ of order 3. Most of the ideas from earlier sections apply and in particular we get analogous statements for computing 3-isogenies (see §4.2). Although these allow to compute the co-domain curve very efficiently, the evaluation of the isogeny is not as efficient as its Montgomery counterpart. Moreover, since tripling formulas are currently slower, at this point Montgomery form still performs better with respect to 3-isogenies.

4.1 The General Formula

We start by presenting formulas for triangular curves that work for any separable isogeny whose kernel is an odd order subgroup. It is possible to include groups of even order, but this creates a case distinction which makes the proof more tedious. Since having (enough) rational points of even order would enable us to go to Montgomery form and reduce to §3, we discard that case here.

There are a couple of (minor) complications compared to the proof of Proposition 5. Firstly, we cannot assume that $g = 0$. If we work on $\mathbb{P}^1$ this will not affect the efficiency, but we will have to take it into account in the proof. Secondly, the action of $(0,0)$ does not involve only x-coordinates. To eliminate the y-coordinates that arise, we group the kernel points into sets $\{T, -T\}$ (similar to [CH17, Theorem 1]).

Proposition 15. *Let k be a field with $\mathrm{char}(k) \neq 2$. Let $a \in k$ such that $a^3 \neq 27$ and $E/k : y^2 + axy + y = x^3$ in triangular form. Let $G \subset E(\bar{k})$ be a finite subgroup of odd order defined over k such that $(0,0) \notin G$ and let φ be a separable isogeny such that $\ker(\varphi) = G$. Let*

$$X = \left\{ x_P \;\middle|\; P \in G \setminus \{\mathcal{O}\} \right\}.$$

Then there exists a curve $\widetilde{E}/k : y^2 + Axy + y = x^3$ such that, up to post-composition by an isomorphism,

$$\begin{aligned} \varphi : E &\to \widetilde{E} \\ (x,y) &\mapsto (f(x), c_0 y f'(x) + g(x)) \end{aligned}$$

where

$$f(x) = x \prod_{z \in X} \frac{x^2z^2 - x(az+1) - z}{(x-z)^2}\,.$$

Moreover, writing

$$\pi = \prod_{z \in X} z\,, \qquad \sigma = \sum_{z \in X} \left(\frac{1}{z^2} + \frac{a}{z} - 2z \right)\,,$$

we have that $A^2 = \pi^2(a^2 + 12\sigma)$ *and* $c_0 = (-1)^{|X|}\pi$.

Proof. Let $P = (0,0)$. As $\varphi(P) \neq \mathcal{O}$, while $[3]\varphi(P) = \varphi([3]P) = \mathcal{O}$, it follows that $\varphi(P)$ must have exact order 3 on E/G. Therefore by moving $\varphi(P)$ to the origin we can put E/G in triangular form and therefore assume that $\widetilde{E} = E/G : y^2 + Axy + y = x^3$. Now apply Theorem 1 with $Q = P$. We find that

$$\begin{aligned} f(x) &= c_1(x - x_{(0,0)}) \prod_{T \in G \setminus \{\mathcal{O}\}} \frac{(x - x_{(0,0)+T})}{(x - x_T)} + f(x_{(0,0)}) \\ &= c_1 \prod_{T \in G \setminus \{\mathcal{O}\}} \frac{\left(x + \frac{y_T}{x_T^2}\right)}{(x - x_T)} \\ &= c_1 \prod_{x_T \in X} \frac{\left(x + \frac{y_T}{x_T^2}\right)\left(x + \frac{y_{-T}}{x_T^2}\right)}{(x - x_T)^2} \\ &= c_1 \prod_{x_T \in X} \frac{x^2 - \frac{x(ax_T+1)}{x_T^2} - \frac{1}{x_T}}{(x - x_T)^2}\,. \end{aligned}$$

Observe that we use the fact that there are no points of order 2, and that

$$y_T y_{-T} = -x_T^3\,, \quad y_T + y_{-T} = -ax_T - 1\,.$$

By [Gal12, Theorem 9.7.5] we can write

$$g(x) = \frac{1}{2}\left(-Af(x) - 1 + c_0 a x f'(x) + c_0 f'(x)\right)\,,$$

so that $g(0) = \left(-1 + c_0 f'(0)\right)/2$. Now we use the fact that $\varphi([2]P) = [2]\varphi(P)$, i. e. $\varphi : (0,-1) \mapsto (0,-1)$. Therefore

$$\begin{aligned} -1 = -c_0 f'(0) + g(0) &\iff -1 = \left(-1 - c_0 f'(0)\right)/2 \\ &\iff c_0 = 1/f'(0) \\ &\iff c_0 = (-1)^{|X|}\pi^3/c_1\,. \end{aligned} \tag{2}$$

It remains to find A and c_1 and for this we use the same strategy as earlier. Let $t = x/y$ be the uniformizer at $\mathcal{O}$ and write $s = 1/y$. Then as a power series

$$s(t) = t^3 - at^4 + a^2t^5 + O(t^6)\,.$$

As $y = 1/s$ and $x = ty$ we find that

$$x(t) = t^{-2} + at^{-1} + O(t)\,, \quad y(t) = t^{-3} + at^{-2} + O(1)\,.$$

Letting $X(t) = f(x(t))$ we get

$$\begin{aligned} X(t) &= c_1t^{-2} + ac_1t^{-1} - c_1\sigma + O(t)\,,\\ dX/dt &= -2c_1t^{-3} - ac_1t^{-2} + O(1)\,,\\ dx/dt &= -2t^{-3} - at^{-2} + O(t)\,,\\ (dx/dt)^{-1} &= -t^3/2 + at^4/4 - a^2t^5/8 + O(t^6)\,. \end{aligned}$$

It follows that

$$\begin{aligned} g(x(t)) &= \frac{1}{2}\left(-AX(t) - 1 + c_0 \cdot (dX/dt) \cdot (dx/dt)^{-1} \cdot (ax(t) + 1)\right)\\ &= \frac{1}{2}\left(ac_0c_1 - Ac_1\right)t^{-2} + \frac{1}{2}a\left(ac_0c_1 - Ac_1\right)t^{-1} + O(1)\,. \end{aligned}$$

Now define $Y(t) = c_0y(t)\,(dX/dt) \cdot (dx/dt)^{-1} + g(x(t))$ and

$$F(t) = Y(t)^2 + AX(t)Y(t) + Y(t) - X(t)^3\,.$$

We get that $F(t) = F_{-6} \cdot t^{-6} + F_{-5} \cdot t^{-5} + F_{-4} \cdot t^{-4} + O(t^{-2})$, where

$$\begin{aligned} F_{-6} &= c_1^2(c_0^2 - c_1)\,, \quad F_{-5} = 3ac_1^2(c_0^2 - c_1)\,,\\ F_{-4} &= c_1^2\big(13a^2c_0^2/4 - A^2/4 + 3c_1\sigma - 3a^2c_1\big)\,. \end{aligned}$$

Again, as F is precisely the equation defining $\widetilde{E}$, we have $F_{-6} = F_{-5} = F_{-4} = 0$. The first two identities lead to $c_1 = c_0^2$, which together with (2) gives $c_1^3 = \pi^6$. Therefore $c_1 = \zeta_3\pi^2$ where $\zeta_3 \in \bar{k}$ is such that $\zeta_3^3 = 1$. Inserting this into F_{-4} and equating to zero we find that

$$A^2 = \pi^2\left(a^2 + 12\sigma\right)/\zeta_3^2\,.$$

Therefore, by composing with the isomorphism $(x, y) \mapsto (\zeta_3^2x, y)$ we can assume that $\zeta_3 = 1$. From (2) we get that $c_0 = (-1)^{|X|}\pi$. The result is now clear. □

4.2 3-isogenies

We work out explicit formulas for 3-isogenies.

Proposition 16. *Let k be a field with* $\mathrm{char}(k) \neq 2$. *Let $a \in k$ such that $a^3 \neq 27$ and $E/k : y^2 + axy + y = x^3$ in triangular form. Let $P \in E(\bar{k})$ a point such that $[3]P = \mathcal{O}$ and $x_P \neq 0$. Then*

$$\begin{aligned} \varphi : E &\to \widetilde{E}/k : y^2 + Axy + y = x^3 \\ (x, y) &\mapsto \big(f(x), -x_P y f'(x) + g(x)\big) \end{aligned}$$

with $A = -3\,(2 + ax_P)$ is a 3-isogeny such that $\ker(\varphi) = \langle P \rangle$, *where*

$$f(x) = x \cdot \frac{x^2 x_P^2 - x(ax_P + 1) - x_P}{(x - x_P)^2}\,.$$

Proof. This is Proposition 15 with $X = \{x_P\}$. Using the division polynomial

$$\psi_3(x) = x\left(3x^3 + a^2x^2 + 3ax + 3\right)$$

it follows that $9\,(2 + ax_P)^2 = \pi^2\left(a^2 + 12\sigma\right)$. Hence $A = \pm 3\,(2 + ax_P)$ and the only remaining uncertainty is the choice of sign. However, setting $A = -3\,(2 + ax_P)$, a direct computation shows that

$$f'(x) = x_P^2 \cdot \frac{\big((x - x_P)^3 - (6x_P^2 + a^2 x_P + a)x + x_P^3 + 1\big)}{(x - x_P)^3}\,,$$

while

$$g(x) = x^3 \cdot \frac{\big((3 + ax_P)x_P^2 x + x_P^3 + 1\big)}{(x - x_P)^3}\,.$$

For $X = f(x)$ and $Y = -x_P y f'(x) + g(x)$, a straightforward calculation shows that $Y^2 + AXY + Y = X^3$. It is then clear that φ is an isogeny and that $\ker(\varphi) = \langle P \rangle$. □

Again, as a consequence of fixing $(0, 0)$ the dual will be generated by it.

Corollary 17. *Let the setup be as in Proposition 16. Then* $\ker(\widehat{\varphi}) = \langle (0, 0) \rangle$.

Proof. Since $(0, 0) \in E$ has order 3 and is not in $\ker(\varphi)$, it follows from $\widehat{\varphi} \circ \varphi = [3]$ that $\varphi\,((0, 0)) \neq \mathcal{O}$, while $(\widehat{\varphi} \circ \varphi)\,((0, 0)) = \mathcal{O}$. Hence $\varphi\,((0, 0)) \in \ker(\widehat{\varphi})$, and since $\deg(\widehat{\varphi}) = 3$ we have that $\ker(\widehat{\varphi}) = \langle \varphi\,((0, 0)) \rangle$. The result is now immediate by observing that $\varphi\,((0, 0)) = (0, 0)$. □

4.3 Application to Isogeny-based Cryptography

By doing an analogous analysis as in §3.3 it is straightforward to see that it is theoretically possible to use the triangular form as above in isogeny-based systems. More specifically, by choosing a basis $E(\mathbb{F}_{p^2})[3^e] = \langle P, Q \rangle$ such that $[3^{e-1}]Q = (0, 0)$ and by only allowing secret kernels of the form $\langle P + [\alpha]Q \rangle$, we can always apply the isogeny from Proposition 16. However, to be seriously considered for implementations the efficiency must be at least on par with those coming from the Montgomery form. Although the computation of A can be done with only two multiplications, we have not been able to reduce the cost of the 3-isogeny evaluation far enough to be considered as efficient as its Montgomery counterpart. Moreover, the x-only tripling formulas (which can for example be obtained by using the 3-isogenies from [Ber+15b, Theorem 5.4]) are significantly slower.

Chapter IX

CSIDH: An Efficient Post-Quantum Commutative Group Action

We propose an efficient commutative group action suitable for non-interactive key exchange in a post-quantum setting. Our construction follows the layout of the Couveignes–Rostovtsev–Stolbunov [Cou06; RS06] cryptosystem, but we apply it to supersingular elliptic curves defined over a large prime field $\mathbb{F}_p$ rather than to ordinary elliptic curves. The Diffie–Hellman scheme resulting from the group action allows for public-key validation at very little cost, runs reasonably fast in practice, and has public keys of only 64 bytes at a conjectured AES-128 security level conform post-quantum security category I of NIST [Nat16].

1 Introduction

During the past five to ten years, elliptic-curve cryptography (ECC) has taken over public-key cryptography on the internet and in security applications. Many protocols such as Signal [Sig] or TLS 1.3 [Res18] rely on the small key sizes and efficient computations to achieve forward secrecy, often meaning that keys are used only once. However, it is also important to notice that security does not break down if keys are reused. Indeed, some implementations of TLS such as Microsoft's Secure Channel (SChannel) reuse keys for some fixed amount of time rather than for

one connection [Ber+15a]. The QUIC protocol (see `https://chromium.org/quic`) by Google relies on servers maintaining their keys fixed for a while to achieve quick session resumption. Several more examples are given by Freire, Hofheinz, Kiltz, and Paterson [Fre+13], formalizing non-interactive key exchange. Some applications specifically require this functionality and for many it provides significant savings in terms of roundtrips or implementation complexity. Finding a post-quantum system that permits non-interactive key exchange while still offering decent performance is considered an open problem. This chapter presents a solution to this problem.

Recall from §II.2.2 that Couveignes' central observation was that the commutativity of $\mathrm{cl}(\mathcal{O})$ naturally allows for a key-exchange protocol in the style of Diffie and Hellman [DH76]. In 2010, Childs, Jao and Soukharev [CJS14] showed that breaking the Couveignes–Rostovtsev–Stolbunov scheme amounts to solving an instance of the abelian hidden-shift problem, for which quantum algorithms with a time complexity of $L_q[1/2]$ are known to exist; see [Kup05; Reg04]. While this may be tolerable (e. g. classical subexponential factorization methods have not ended the widespread use of RSA), a much bigger concern is that the scheme is unacceptably slow: despite recent clever speedups due to De Feo, Kieffer, and Smith [DFKS18; Kie17], several minutes are needed for a single key exchange at a presumed classical security level of 128 bits. Nevertheless, in view of its conceptual simplicity, compactness, and flexibility, it seems a shame to discard the Couveignes–Rostovtsev–Stolbunov scheme.

The attack due to Childs–Jao–Soukharev strongly relies on the fact that $\mathrm{cl}(\mathcal{O})$ is commutative, hence indirectly on the fact that $\mathcal{O}$ is commutative. This led Jao and De Feo [JDF11] to consider (see §II.2.1) the use of supersingular elliptic curves, whose full ring of endomorphisms is an order in a quaternion algebra; in particular, it is non-commutative. Their resulting (interactive) key-agreement scheme "Supersingular Isogeny Diffie–Hellman" (SIDH) has attracted almost the entire focus of isogeny-based cryptography over the past six years. The current state-of-the-art implementation is SIKE [Jao+16], which was recently submitted to the NIST competition on post-quantum cryptography [Nat16]. It should be stressed that SIDH is *not* the Couveignes–Rostovtsev–Stolbunov scheme in which one substitutes supersingular elliptic curves for ordinary elliptic curves; in fact, SIDH is much more reminiscent of a cryptographic hash function from 2006 due to Charles, Goren, and Lauter [CLG09]. The public keys of SIDH consist of the codomain of a secret isogeny and the image points of certain public points under that isogeny. Galbraith, Petit, Shani, and Ti [Gal+16] showed that SIDH keys succumb to active attacks and thus should not be reused, unless combined with a CCA transform such as the Fujisaki–Okamoto transform [FO99].

In this chapter we show that adapting the scheme by Couveignes–Rostovtsev–Stolbunov to supersingular elliptic curves is possible, provided that one restricts to supersingular elliptic curves defined over a prime field $\mathbb{F}_p$. Instead of the full ring of endomorphisms, which is non-commutative, one should consider the subring of $\mathbb{F}_p$-rational endomorphisms. This is again an order $\mathcal{O}$ in an imaginary quadratic field. As before, $\mathrm{cl}(\mathcal{O})$ acts via isogenies on the set of $\mathbb{F}_p$-isomorphism classes of elliptic curves whose $\mathbb{F}_p$-rational endomorphism ring is isomorphic to $\mathcal{O}$ and whose trace of Frobenius has a prescribed value; in fact if $p \geq 5$ then there is only one option for this value, namely 0, in contrast with the ordinary case. See for example [Wat69, Theorem 4.5], with further details to be found in [Brö08; DG16] and in §3. Starting from these observations, the desired adaptation of the Couveignes–Rostovtsev–Stolbunov scheme almost unrolls itself; the details can be found in §4. We call the resulting scheme CSIDH, where the C stands for "commutative".[1]

While this fails to address the initial motivation of Jao and De Feo for using supersingular elliptic curves, which was to avoid the $L_q[1/2]$ quantum attack due to Childs–Jao–Soukharev, we show that CSIDH eliminates the main problem of the scheme of Couveignes–Rostovtsev–Stolbunov, namely its inefficiency. Indeed, in §8 we will report on a proof-of-concept implementation which carries out a non-interactive key exchange at a presumed classical security level of 128 bits and a conjectured post-quantum security level of 64 bits in about 80 milliseconds, while using key sizes of only 64 bytes. This is over 2000 times faster[2] than the current state-of-the-art instantiation of the Couveignes–Rostovtsev–Stolbunov scheme by De Feo, Kieffer and Smith [DFKS18; Kie17], which itself presents many new ideas and speedups to even achieve that speed.

For comparison we remark that SIKE, which is the NIST submission with the smallest combined key and ciphertext length, uses public keys and ciphertexts of over 300 bytes each. More precisely, SIKE version `p503` uses uncompressed keys of 378 bytes long [Jao+16] for achieving CCA security. The optimized SIKE implementation is about ten times faster than our proof-of-concept C implementation, but even at 80 ms, CSIDH is practical. Another major advantage of CSIDH is that we can efficiently validate public keys, making it possible to reuse a key without the need for transformations to confirm that the other party's key was honestly generated. Finally we note that just like the original Couveignes–Rostovtsev–Stolbunov scheme, CSIDH relies purely on the isogeny-finding problem; no extra points are sent that

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

[2] This speedup is explained in part by comparing our own C implementation to the `sage` implementation of De Feo–Kieffer–Smith.

could potentially harm security, as argued in [Pet17].

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

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

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

resort to other primes with less beneficial properties, requiring to work in extensions of $\mathbb{F}_p$.

The use of supersingular elliptic curves over $\mathbb{F}_p$ has various other advantages. For instance, their trace of Frobenius t is 0, so that the absolute value of the discriminant $|t^2 - 4p| = 4p$ is as large as possible. As a consequence, generically the size of the class group $\mathrm{cl}(\mathcal{O})$ is close to its maximal possible value for a fixed choice of p. Conversely, this implies that for a fixed security level we can make a close-to-minimal choice for p, which directly affects the key size. Note that this contrasts with the CM construction from [BS07], which could in principle be used to construct ordinary elliptic curves having many points of small order, but whose endomorphism rings have very small class groups, ruling them out for the Couveignes–Rostovtsev–Stolbunov key exchange.

To explain why key validation works, note that we work over $\mathbb{F}_p$ with $p \equiv 3 \pmod 8$ and start from the curve $E_0 \colon y^2 = x^3 + x$ with $\mathbb{F}_p$-rational endomorphism ring $\mathcal{O} = \mathbb{Z}[\pi]$. As it turns out, all Montgomery curves $E_A \colon y^2 = x^3 + Ax^2 + x$ over $\mathbb{F}_p$ that are supersingular appear in the $\mathrm{cl}(\mathcal{O})$-orbit of E_0. Moreover their $\mathbb{F}_p$-isomorphism class is uniquely determined by A. So all one needs to do upon receiving a candidate public key $y^2 = x^3 + Ax^2 + x$ is check for supersingularity, which is an easy task; see §5. The combination of large size of $\mathrm{cl}(\mathcal{O})$ and representation by a single $\mathbb{F}_p$-element A explains the small key size of 64 bytes (for NIST category I).

One-way group actions. Although non-interactive key exchange is the main application of our primitive, it is actually more general. It is (conjecturally) an instance of Couveignes' *hard homogeneous spaces* [Cou06], ultimately nothing but a finite commutative group action for which some operations are easy to compute while others are hard. Such group actions were first formalized and studied by Brassard and Yung [BY91]. We summarize Couveignes' definition.

Definition 1. A *hard homogeneous space* consists of a finite commutative group G acting freely and transitively on some set X. The following tasks are required to be easy (e. g. polynomial-time):

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

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

– (*Vectorization*.) Given $x, y \in X$, find $g \in G$ such that $g * x = y$.
– (*Parallelization*.) Given $x, y, z \in X$ such that $y = g * x$, find w such that $w = g * z$.

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

Notation and terminology. We stress that throughout this chapter, we consider two elliptic curves defined over the same field identical whenever they are isomorphic *over that field*. Note that we do *not* identify curves that are only isomorphic over some extension field, as opposed to what is done in SIDH, for instance. In the same vein, for an elliptic curve E defined over a finite field $\mathbb{F}_p$ we let $\mathrm{End}_p(E)$ be the subring of the endomorphism ring $\mathrm{End}(E)$ consisting of endomorphisms defined over $\mathbb{F}_p$.[3] This subring is always isomorphic to an order in an imaginary quadratic number field. Conversely, for a given order $\mathcal{O}$ in an imaginary quadratic field and an element $\pi \in \mathcal{O}$, we let $\mathcal{E}\ell\ell_p(\mathcal{O}, \pi)$ denote the set of elliptic curves E defined over $\mathbb{F}_p$ with $\mathrm{End}_p(E) \cong \mathcal{O}$ such that π corresponds to the $\mathbb{F}_p$-Frobenius endomorphism of E. In particular, this implies that $\varphi \circ \beta = \beta \circ \varphi$ for all $\mathbb{F}_p$-isogenies φ between two curves in $\mathcal{E}\ell\ell_p(\mathcal{O}, \pi)$ and all $\beta \in \mathcal{O}$ interpreted as endomorphisms. Finally, we always assume ideals to be non-zero.

Organization. In §2 and §3 we recall the main properties of isogeny graphs of elliptic curves with endomorphism ring of $\mathbb{Z}$-rank 2. We treat the construction of our class group action in §4, while §5 discuss public-key validation. We summarize the key exchange in §6 and elaborate on its security in §7. Finally, we present an implementation in §8.

2 Isogeny Graphs

Good mixing properties of the underlying isogeny graph are relevant for the security of isogeny-based cryptosystems. Just as in the original Couveignes–Rostovtsev–

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

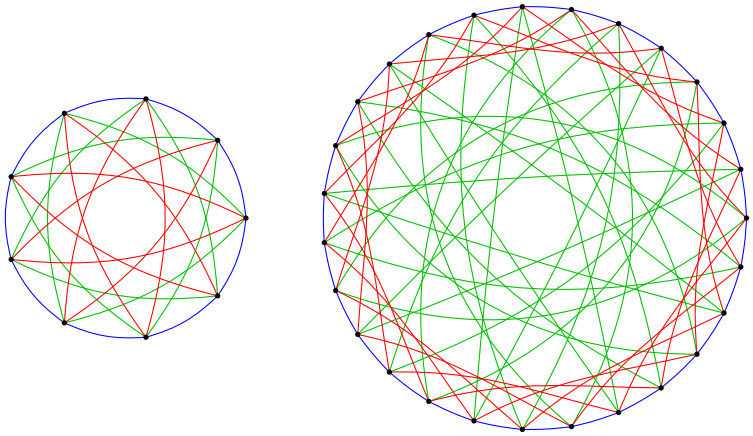


Figure 1. Union of the supersingular ℓ-isogeny graphs for $\ell \in \{3, 5, 7\}$ over $\mathbb{F}_{419}$. CSIDH makes use of the larger component, corresponding to curves whose ring of $\mathbb{F}_{419}$-rational endomorphisms is isomorphic to $\mathbb{Z}[\sqrt{-419}]$.

Stolbunov cryptosystem, in our case this graph is obtained by taking the union of several large subgraphs (each being a union of large isomorphic cycle graphs) on the same vertex set, one for each prime ℓ under consideration; see Figure 1 for a (small) example. Such a graph is the *Schreier graph* associated with our class-group action and the chosen generators. We refer to the lecture notes of De Feo [DF17, §14.1] for more background and to [JMV09] for a discussion of its rapid mixing properties. One point of view on this is that one can quickly move between distant nodes in the subgraph corresponding to one generator by switching to the subgraph corresponding to another generator. This thereby replaces the square-and-multiply algorithm in exponentiation-based cryptosystems (such as classical Diffie–Hellman). The goal of this section is to analyze the structure of the individual cycles.

Definition 2. For a field k and a prime $\ell \nmid \operatorname{char} k$, the *$k$-rational ℓ-isogeny graph* $G_{k,\ell}$ is defined as having all the elliptic curves defined over k as its vertices, and having a directed edge (E_1, E_2) for each k-rational ℓ-isogeny from E_1 to E_2.[4]

Remark 3. A priori $G_{k,\ell}$ is a directed graph, but given two elliptic curves E_1 and E_2 whose j-invariants are not in $\{0, 1728\}$, there are exactly as many edges (E_2, E_1) as (E_1, E_2) by taking dual isogenies. Annoyingly, the nodes with j-invariants 0 and 1728 are more complicated, since these are exactly the curves with extra automorphisms; an elliptic curve E in $G_{k,\ell}$ has fewer incoming than outgoing edges if and only if either $j(E) = 0$ and $\sqrt{-3} \in k$, or if $j(E) = 1728$ and $\sqrt{-1} \in k$. Throughout this

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

chapter, we will assume for simplicity that $\sqrt{-3}, \sqrt{-1} \notin k$, so that neither of these automorphisms are defined over k and we may view $G_{k,\ell}$ as an undirected graph. In the case of a finite prime field $k = \mathbb{F}_p$ it suffices to restrict to $p \equiv 11 \pmod{12}$, which will be satisfied in the class of instantiations we suggest.

If $k = \mathbb{F}_q$ is a finite field, then $G_{k,\ell}$ is a finite graph that is the disjoint union of ordinary connected components and supersingular connected components. The ordinary components were studied in Kohel's PhD thesis [Koh96]. Due to their regular structure, these components later became known as *isogeny volcanoes*. In general (e. g. over non-prime fields), the supersingular components may bear no similarity at all to the volcanoes of the ordinary case. Traditionally, following Pizer [Piz90], one instead studies the unique supersingular component of $G_{k,\ell}$ where $k = \overline{\mathbb{F}}_q$, which turns out to be a finite $(\ell+1)$-regular Ramanujan graph and forms the basis for the SIDH protocol. However, Delfs and Galbraith [DG16] showed that if $k = \mathbb{F}_p$ is a finite prime field, then all connected components are volcanoes, even in the supersingular case (where the depth is at most 1 at $\ell = 2$ and 0 otherwise). We present a special case of a unified statement, restricting our attention to the cases in which $G_{\mathbb{F}_p,\ell}$ is a cycle. Recall that $\mathrm{End}_p(E)$ is an order $\mathcal{O}$ in the imaginary quadratic field $\mathrm{End}_p(E) \otimes_{\mathbb{Z}} \mathbb{Q} \cong \mathbb{Q}(\sqrt{t^2 - 4p}) = K$, where $|t| \leq 2\sqrt{p}$ denotes the (absolute value of the) trace of the Frobenius endomorphism, and that two curves are isogenous over $\mathbb{F}_p$ if and only if their traces of Frobenius are equal [Tat66, Theorem 1].

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

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

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

In the ordinary case a curve and its quadratic twist can never appear in the same component because they have a different trace of Frobenius. This is the main difference with the supersingular case, where this possibility is not excluded. To avoid

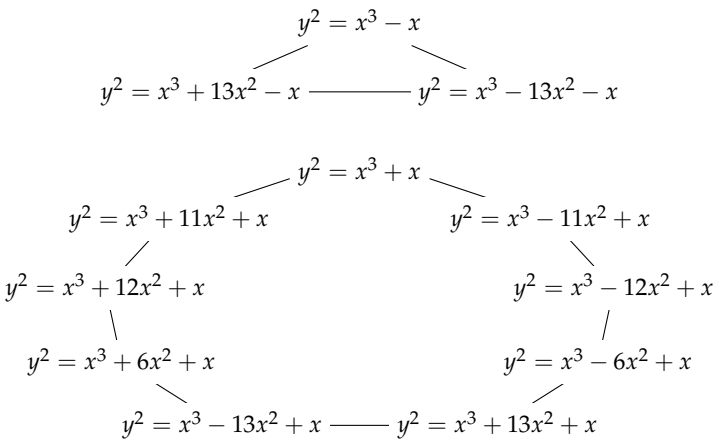


Figure 2. The two supersingular components of $G_{\mathbb{F}_{83},3}$. The curves in the top component have $\mathbb{F}_p$-rational endomorphism ring $\mathbb{Z}[(1+\sqrt{-83})/2]$, while those in the lower component correspond to $\mathbb{Z}[\sqrt{-83}]$. Running clockwise through these components corresponds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

confusion, we clarify that by the quadratic twist of a given elliptic curve $E\colon y^2 = f(x)$ over $\mathbb{F}_p$ we mean the curve $E^t\colon dy^2 = f(x)$, where $d \in \mathbb{F}_p^*$ is any non-square. If $p \equiv 3 \pmod 4$ and $j(E) = 1728$ then this may deviate from what some readers are used to, because in this case E^t and E are $\mathbb{F}_p$-isomorphic. Note that such a curve is necessarily supersingular.

Remark 5. In fact, if $p \equiv 3 \pmod 4$ then there are two non-isomorphic curves over $\mathbb{F}_p$ with j-invariant 1728, namely $y^2 = x^3 - x$ and $y^2 = x^3 + x$, whose endomorphism rings are the full ring of integers $\mathbb{Z}[(1+\sqrt{-p})/2]$ and the order $\mathbb{Z}[\sqrt{-p}]$ of conductor 2 respectively. The connected component of each curve is "symmetric"; if E is n steps along $G_{\mathbb{F}_p,\ell}$ in one direction from a curve of j-invariant 1728 then the curve that is n steps in the other direction is the quadratic twist of E. In the case of $G_{\mathbb{F}_{83},3}$ we can see this in Figure 2, which is taken from [DG16, Figure 8]. It is also interesting to observe that the symmetry around $j = 1728$ confirms the known fact that the class numbers of $\mathbb{Z}[(1+\sqrt{-p})/2]$ and $\mathbb{Z}[\sqrt{-p}]$ are odd, at least in the case that $p \equiv 3 \pmod 4$; see [Mor61].

3 The Class-group Action

It is well-known that the ideal-class group of an imaginary quadratic order $\mathcal{O}$ acts freely via isogenies on the set of elliptic curves with $\mathbb{F}_p$-rational endomorphism ring

$\mathcal{O}$. Using this group action on a set of ordinary elliptic curves for cryptographic purposes was first put forward by Couveignes [Cou06] and independently rediscovered later by Rostovtsev and Stolbunov [Sto04; RS06]. Our suggestion is to use the equivalent of their construction in the supersingular setting, thus the following discussion covers both cases at once. For concreteness, we focus on prime fields with $p \geq 5$ and point out that the ordinary (but not the supersingular) case generalizes to all finite fields. We recall the following standard lemma:

Lemma 6. *Let $E/\mathbb{F}_p$ be an elliptic curve and G a finite $\mathbb{F}_p$-rational (i. e. stable under the action of the $\mathbb{F}_p$-Frobenius) subgroup of E. Then there exists an elliptic curve $\widetilde{E}/\mathbb{F}_p$ and a separable isogeny $\varphi\colon E \to \widetilde{E}$ defined over $\mathbb{F}_p$ with kernel G. The codomain $\widetilde{E}$ and isogeny φ are unique up to $\mathbb{F}_p$-isomorphism.*[5]

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

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

A *fractional ideal* of $\mathcal{O}$ is an $\mathcal{O}$-submodule of K of the form $\alpha\mathfrak{a}$, where $\alpha \in K^*$ and $\mathfrak{a}$ is an $\mathcal{O}$-ideal.[6] Fractional ideals can be multiplied and conjugated in the evident way, and the norm extends multiplicatively to fractional ideals. A fractional $\mathcal{O}$-ideal $\mathfrak{a}$ is *invertible* if there exists a fractional $\mathcal{O}$-ideal $\mathfrak{b}$ such that $\mathfrak{ab} = \mathcal{O}$. If such a $\mathfrak{b}$ exists, we define $\mathfrak{a}^{-1} = \mathfrak{b}$. Clearly all *principal* fractional ideals $\alpha\mathcal{O}$, where $\alpha \in K^*$, are invertible. By construction, the set of invertible fractional ideals $I(\mathcal{O})$ forms an abelian group under ideal multiplication. This group contains the principal fractional ideals $P(\mathcal{O})$ as a (clearly normal) subgroup, hence we may define the *ideal-class group* of $\mathcal{O}$ as the quotient

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

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

There is a unique *maximal order* of K with respect to inclusion called the ring of integers and denoted $\mathcal{O}_K$. The *conductor* of $\mathcal{O}$ (in $\mathcal{O}_K$) is the index $f = [\mathcal{O}_K : \mathcal{O}]$.

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

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

Away from the conductor, ideals are well-behaved; every $\mathcal{O}$-ideal of norm coprime to the conductor is invertible and factors uniquely into prime ideals.

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

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

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

Any invertible ideal $\mathfrak{a}$ of $\mathcal{O}$ splits into a product of $\mathcal{O}$-ideals as $(\pi\mathcal{O})^r \mathfrak{a}_s$, where $\mathfrak{a}_s \nsubseteq \pi\mathcal{O}$. This defines an elliptic curve $E/\mathfrak{a}$ and an isogeny $\varphi_{\mathfrak{a}} \colon E \to E/\mathfrak{a}$ of degree $\mathrm{N}(\mathfrak{a})$ as follows (see e. g. [Wat69]). The separable part of $\varphi_{\mathfrak{a}}$ has kernel $\bigcap_{\alpha \in \mathfrak{a}_s} \ker \alpha$, and the purely inseparable part consists of r iterations of Frobenius. The isogeny $\varphi_{\mathfrak{a}}$ and codomain $E/\mathfrak{a}$ are both defined over $\mathbb{F}_p$ and are unique up to $\mathbb{F}_p$-isomorphism (by Lemma 6), justifying the notation $E/\mathfrak{a}$. Multiplication of ideals corresponds to the composition of isogenies. Since principal ideals correspond to endomorphisms, two ideals lead to the same codomain if and only if they are equal up to multiplication by a principal fractional ideal. Moreover, every $\mathbb{F}_p$-isogeny ψ between curves in $\mathcal{Ell}_p(\mathcal{O}, \pi)$ comes from an invertible $\mathcal{O}$-ideal in this way, and the ideal $\mathfrak{a}_s$ can be recovered from ψ as $\mathfrak{a}_s = \{\alpha \in \mathcal{O} \mid \ker \alpha \supseteq \ker \psi\}$. In other words:

Theorem 7. *Let $\mathcal{O}$ be an order in an imaginary quadratic field and $\pi \in \mathcal{O}$ such that $\mathcal{Ell}_p(\mathcal{O}, \pi)$ is non-empty. Then the ideal-class group $\mathrm{cl}(\mathcal{O})$ acts freely and transitively on the set $\mathcal{Ell}_p(\mathcal{O}, \pi)$ via the map*

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

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

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

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

The structure of the class group. The class group $\mathrm{cl}(\mathcal{O})$ is a finite abelian group whose cardinality is asymptotically [Sie35] $\#\,\mathrm{cl}(\mathcal{O}) \approx \sqrt{|\Delta|}$. More precise heuristics

actually predict that $\#\mathrm{cl}(\mathcal{O})$ grows a little bit faster than $\sqrt{|\Delta|}$, but the ratio is logarithmically bounded so we content ourselves with the above estimate. The exact structure of the class group can be computed in subexponential time $L_{|\Delta|}[1/2; \sqrt{2} + o(1)]$ using an algorithm of Hafner and McCurley [HM89]. Unfortunately, this requires too much computation for the sizes of Δ we are working with, but there are convincing heuristics concerning the properties of the class group we need. See §7.1 for these arguments. If the absolute value $|t|$ of the trace of Frobenius is "not too big", the discriminant Δ is about the size of p, hence by the above approximation we may assume $\#\mathrm{cl}(\mathcal{O}) \approx \sqrt{p}$. This holds in particular when E is supersingular, where $t = 0$, hence $|\Delta| = 4p$.

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

Computing the group action. Any element of the class group can be represented as a product of small prime ideals [BV07, Propositions 9.5.2 and 9.5.3], hence we describe how to compute $[\mathfrak{l}]E$ for a prime ideal $\mathfrak{l} = (\ell, \pi - \lambda)$. There are (at least) the following ways to proceed, which vary in efficiency depending on the circumstances [DFKS18; Kie17]:

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

- Factor the ℓ^{th} division polynomial $\psi_\ell(E)$ over $\mathbb{F}_p$; collect irreducible factors with the right Frobenius eigenvalues (as above); use Kohel's formulas [Koh96, §2.4] to compute the codomain.

- Find a basis of the ℓ-torsion — possibly over an extension field — and compute the eigenspaces of Frobenius; apply Vélu's formulas [Vél71] to a basis point of the correct eigenspace to compute the codomain.

As observed in [DFKS18; Kie17], the last method is the fastest if the necessary extension fields are small. The optimal case is $\lambda = 1$; in that case, the curve has a rational point defined over the base field $\mathbb{F}_p$. If in addition $p/\lambda = -1$, the other eigenspace of Frobenius modulo ℓ is defined over $\mathbb{F}_{p^2}$, so both codomains can easily be computed using Vélu's formulas over an at most quadratic extension (but in fact, a good choice of curve model allows for pure prime field computations, see §8; alternatively one could switch to the quadratic twist). Note that if $p \equiv -1 \pmod{\ell}$, then $\lambda = 1$ automatically implies $p/\lambda = -1$.

Much of De Feo–Kieffer–Smith's work [DFKS18; Kie17] is devoted to finding an ordinary elliptic curve E with many small Elkies primes ℓ such that both E and its quadratic twist E^t have an $\mathbb{F}_p$-rational ℓ-torsion point. Despite considerable effort leading to various improvements, the results are discouraging. With the best parameters found within 17 000 hours of CPU time, evaluating one class-group action still requires several minutes of computation to complete. This suggests that without new ideas, the original Couveignes–Rostovtsev–Stolbunov scheme will not become anything close to practical in the foreseeable future.

4 Construction and Design Choices

In this section, we discuss the construction of our proposed group action and justify our design decisions. For algorithmic details, see §8. Notice that the main obstacle to performance in the Couveignes–Rostovtsev–Stolbunov scheme — constructing a curve with highly composite order — becomes trivial when using supersingular curves instead of ordinary curves, since for $p \geq 5$ any supersingular elliptic curve over $\mathbb{F}_p$ has exactly $p + 1$ rational points.

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

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

More precisely, Proposition 8 (below) shows $\mathrm{End}_p(E_0) = \mathbb{Z}[\pi]$, which has conductor 2.

Rational Elkies primes. By Theorem 4, the choices made above imply that the ℓ_i-isogeny graph is a disjoint union of cycles. Moreover, since $\pi^2 - 1 \equiv 0 \pmod{\ell_i}$ the ideals $\ell_i\mathcal{O}$ split as $\ell_i\mathcal{O} = \mathfrak{l}_i\bar{\mathfrak{l}}_i$, where $\mathfrak{l}_i = (\ell_i, \pi - 1)$ and $\bar{\mathfrak{l}}_i = (\ell_i, \pi + 1)$. In other words, *all* the ℓ_i are Elkies primes. In particular, we can use any one of the three algorithms described at the end of §3 to walk along the cycles.

Furthermore, the kernel of $\varphi_{\mathfrak{l}_i}$ is the intersection of the kernels of the scalar multiplication $[\ell_i]$ and the endomorphism $\pi - 1$. That is, it is the subgroup generated by a point P of order ℓ_i which lies in the kernel of $\pi - 1$ or, in other words, is defined over $\mathbb{F}_p$. Similarly, the kernel of $\varphi_{\bar{\mathfrak{l}}_i}$ is generated by a point Q of order ℓ_i that is defined over $\mathbb{F}_{p^2}$ but not $\mathbb{F}_p$ and such that $\pi(Q) = -Q$. This greatly simplifies and accelerates the implementation, since it allows performing all computations over the base field (see §8 for details).

Sampling from the class group. Ideally, we would like to know the exact structure of the ideal-class group $\mathrm{cl}(\mathcal{O})$ to be able to sample elements uniformly at random. However, such a computation is currently not feasible for the size of discriminant we need, hence we resort to heuristic arguments. Assuming that the $\mathfrak{l}_i$ do not have very small order and are "evenly distributed" in the class group, we can expect ideals of the form $\mathfrak{l}_1^{e_1}\mathfrak{l}_2^{e_2}\cdots\mathfrak{l}_n^{e_n}$ for small e_i to lie in the same class only very occasionally. For efficiency reasons, it is desirable to sample the exponents e_i from a short range centered around zero, say $\{-m, \ldots, m\}$ for some integer m. We will argue in §7.1 that choosing m such that $2m + 1 \geq \sqrt[n]{\#\,\mathrm{cl}(\mathcal{O})}$ is sufficient. Since the prime ideals $\mathfrak{l}_i$ are fixed global parameters, the ideal $\prod_i \mathfrak{l}_i^{e_i}$ may simply be represented as a vector $(e_1, \ldots, e_n)$.

Evaluating the class-group action. Computing the action of an ideal class represented by $\prod_i \mathfrak{l}_i^{e_i}$ on an elliptic curve E proceeds as outlined in §3. Since $\pi^2 = -p$ which is equivalent to 1 modulo all ℓ_i, we are now in the favorable situation that the eigenvalues of Frobenius on *all* ℓ_i-torsion subgroups are 1 and -1. Hence we can efficiently compute the action of $\mathfrak{l}_i$ (resp. $\bar{\mathfrak{l}}_i$) by finding an $\mathbb{F}_p$-rational point (resp. $\mathbb{F}_{p^2}$-rational with Frobenius eigenvalue -1) of order ℓ_i and applying Vélu-type formulas. This step could simply be repeated for each ideal $\mathfrak{l}_i^{\pm 1}$ whose action is to be evaluated, but see §8 for a more efficient method.

5 Representing & Validating $\mathbb{F}_p$-isomorphism Classes

A major unsolved problem of SIDH is its lack of public-key validation, i. e. the inability to verify that a public key was honestly generated. This shortcoming leads to polynomial-time active attacks [Gal+16] on static variants for which countermeasures are expensive. For example, the actively secure variant SIKE [Jao+16] applies a transformation proposed by Hofheinz, Hövelmanns, and Kiltz [HHK17] which is similar to the Fujisaki–Okamoto transform [FO99], essentially doubling the running time on the recipient's side compared to an ephemeral key exchange.

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

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

Proof. First suppose that E is isomorphic over $\mathbb{F}_p$ to E_A for some $A \in \mathbb{F}_p$. If E_A has full $\mathbb{F}_p$-rational 2-torsion, then Table 1 of [CS17] shows that either E_A or its quadratic twist must have order divisible by 8. However, both have cardinality $p + 1 \equiv 4 \pmod 8$. Hence E_A can only have one $\mathbb{F}_p$-rational point of order 2. With Theorem 2.7 of [DG16], we can conclude $\operatorname{End}_p(E) = \operatorname{End}_p(E_A) = \mathbb{Z}[\pi]$.

Now assume that $\operatorname{End}_p(E) = \mathbb{Z}[\pi]$. By Theorem 7, the class group $\operatorname{cl}(\mathbb{Z}[\pi])$ acts transitively on $\mathcal{E}\ell\ell_p(\mathbb{Z}[\pi], \pi)$, so in particular there exists $[\mathfrak{a}] \in \operatorname{cl}(\mathbb{Z}[\pi])$ such that $[\mathfrak{a}]E_0 = E$, where $E_0 \colon y^2 = x^3 + x$. Choosing a representative $\mathfrak{a}$ that has norm coprime to $2p$ yields a separable $\mathbb{F}_p$-isogeny $\varphi_{\mathfrak{a}} \colon E_0 \to E$ of odd degree. Thus, by Proposition VIII.5 there exists an $A \in \mathbb{F}_p$ and a separable isogeny $\psi \colon E_0 \to E_A \colon y^2 = x^3 + Ax^2 + x$ defined over $\mathbb{F}_p$ such that $\ker \psi = \ker \varphi_{\mathfrak{a}}$. As isogenies defined over $\mathbb{F}_p$ with given kernel are unique up to post-composition with isomorphisms defined over $\mathbb{F}_p$ (Lemma 6), we conclude that E is $\mathbb{F}_p$-isomorphic to E_A.

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

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

Substituting this into the curve equation for E_A and subtracting the equation of E_B (scaled by u^6) equals zero in the function field and thus leads to a linear relation

over $\mathbb{F}_p$ between the functions 1, X, X^2, Y, and XY. Writing ∞ for the point at infinity of E_B, it follows from Riemann–Roch [Sil09, Theorem 5.4] that $\mathcal{L}(5(\infty))$ is a 5-dimensional $\mathbb{F}_p$-vector space with basis $\{1, X, Y, X^2, XY\}$. Hence the obtained linear relation must be trivial, and a straightforward computation yields the relations

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

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

Therefore, by choosing public keys to consist of a Montgomery coefficient $A \in \mathbb{F}_p$, Proposition 8 guarantees that A represents a curve in the correct isogeny class $\mathcal{E}\ell\ell_p(\mathcal{O}, \pi)$, where $\pi = \sqrt{-p}$ and $\mathcal{O} = \mathbb{Z}[\pi]$, under the assumption that it is smooth (i. e. $A \neq \pm 2$) and supersingular.

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

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

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

This product is about the same size as p, and it is easily seen that a random point will with overwhelming probability have order (much) greater than $4\sqrt{p}$. This observation leads to a straightforward verification method, see Algorithm 1.[7] If the condition $d > 4\sqrt{p}$ does not hold at the end of Algorithm 1, the point P had too

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

small order to prove $\#E(\mathbb{F}_p) = p + 1$. In this case one may retry with a new random point P (although this outcome has negligible probability and could just be ignored). There is no possibility of wrongly classifying an ordinary curve as supersingular.

Algorithm 1. Verifying supersingularity.

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

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

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

There are more optimized variants of this algorithm; the bulk of the work are the scalar multiplications required to compute the points $Q_i = [(p+1)/\ell_i]P$. Since they are all multiples of P with shared factors, one may more efficiently compute all Q_i at the same time using a divide-and-conquer strategy (at the expense of higher memory usage). See §8, and in particular Algorithm 3, for details.

6 Non-interactive Key Exchange

Starting from the class-group action on supersingular elliptic curves and the parameter choices outlined in Sections 3 and 4, one obtains the following non-interactive key-exchange protocol.

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

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

Key exchange. Suppose Alice and Bob have key pairs $([\mathfrak{a}], A)$ and $([\mathfrak{b}], B)$. Upon receiving Bob's public key $B \in \mathbb{F}_p \setminus \{\pm 2\}$, Alice verifies that the elliptic curve $E_B \colon y^2 = x^3 + Bx^2 + x$ is indeed in $\mathcal{E}\ell\ell_p(\mathcal{O}, \pi)$ using Algorithm 1. She then applies the action of her secret key $[\mathfrak{a}]$ to E_B to compute the curve $[\mathfrak{a}]E_B = [\mathfrak{a}][\mathfrak{b}]E_0$. Bob proceeds analogously with his own secret $[\mathfrak{b}]$ and Alice's public key A to compute the curve $[\mathfrak{b}]E_A = [\mathfrak{b}][\mathfrak{a}]E_0$. The shared secret is the Montgomery coefficient S of the common secret curve $[\mathfrak{a}][\mathfrak{b}]E_0 = [\mathfrak{b}][\mathfrak{a}]E_0$ written in the form $y^2 = x^3 + Sx^2 + x$, which is the same for Alice and Bob due to the commutativity of $\mathrm{cl}(\mathcal{O})$ and Proposition 8.

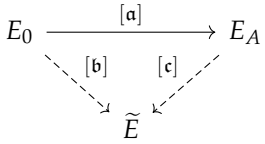


Figure 3. A 1-bit identification protocol.

Remark 9. Besides key exchange, we expect that our cryptographic group action will have several other applications, given the resemblance with traditional Diffie–Hellman and the ease of verifying the correctness of public keys. We refer to previous papers on group actions for a number of suggestions in this direction, in particular Brassard–Yung [BY91], Couveignes [Cou06, §4] and Stolbunov [Sto10]. We highlight the following 1-bit identification scheme, which in our case uses a key pair $([\mathfrak{a}], A)$ as above. One randomly samples an element $[\mathfrak{b}] \in \mathrm{cl}(\mathcal{O})$ and commits to a curve $\widetilde{E} = [\mathfrak{b}]E_0$. Depending on a challenge bit b one then releases either $[\mathfrak{b}]$ or $[\mathfrak{c}] := [\mathfrak{b}][\mathfrak{a}]^{-1}$, as depicted in Figure 3. As already pointed out in Stolbunov's PhD thesis [Sto11, §2.B], this can be turned into a signature scheme by repeated application of the 1-bit protocol and by applying the Fiat–Shamir [FS87] or Unruh [Unr12] transformation. However, we point out that it is not immediately clear how to represent $[\mathfrak{c}]$ in a way that is efficiently computable and leaks no information about the secret key $[\mathfrak{a}]$. We leave a resolution of this issue for future research, but mention

that a related problem was recently tackled by Galbraith, Petit and Silva [GPS17] who studied a similar triangular identification protocol in the context of SIDH.[8]

7 Security

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

Problem 10 (Key recovery). Given two supersingular elliptic curves $E, \widetilde{E}$ defined over $\mathbb{F}_p$ with the same $\mathbb{F}_p$-rational endomorphism ring $\mathcal{O}$, find an ideal $\mathfrak{a}$ of $\mathcal{O}$ such that $[\mathfrak{a}]E = \widetilde{E}$. This ideal must be represented in such a way that the action of $[\mathfrak{a}]$ on a curve can be evaluated efficiently, for instance $\mathfrak{a}$ could be given as a product of ideals of small norm.

Note that just like in the classical group-based scenario, security notions of Diffie–Hellman schemes built from our primitive rely on slightly different hardness assumptions (c. f. Definition 1) that are straightforward translations of the computational and decisional Diffie–Hellman problems. However, continuing the analogy with the classical case, and since we are not aware of any ideas to attack the key exchange without recovering one of the keys, we will assume in the following analysis that the best approach to breaking the key-exchange protocol is to solve Problem 10.

We point out that the "inverse Diffie-Hellman problem" is easy in the context of CSIDH: given $[\mathfrak{a}]E_0$ we can compute $[\mathfrak{a}]^{-1}E_0$ by mere quadratic twisting; see Remark 5. This contrasts with the classical group-based setting [Gal12, §21.1]. Note that just like identifying a point (x, y) with its inverse $(x, -y)$ in an ECDLP setting, this implies a security loss of one bit under some attacks: An attacker may consider the curves $[\mathfrak{a}]E$ and $[\mathfrak{a}]^{-1}E$ identical, which reduces the search space by half.

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

[8] The "square" SIDH counterparts of this protocol, as considered in [DFJP14; GPS17; Yoo+17], are not meaningful in the case of a commutative group action.

that a potential future attack against SIDH based on these torsion points would not apply to CSIDH.

Chosen-ciphertext attacks. As explained in §5, the CSIDH group action features efficient public-key validation. This implies it can be used without applying a CCA transform such as the Fujisaki–Okamoto transform [FO99], thus enabling efficient non-interactive key exchange and other applications in a post-quantum world.

7.1 Classical Security

We begin by considering classical attacks.

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

As explained in §4, a private key of our scheme consists of an exponent vector $(e_1, \dots, e_n)$ where each e_i is in the range $\{-m, \dots, m\}$, representing the ideal class $[\mathfrak{l}_1^{e_1}\mathfrak{l}_2^{e_2}\cdots\mathfrak{l}_n^{e_n}] \in \mathrm{cl}(\mathcal{O})$. There may (and typically will) be multiple such vectors that represent the same ideal class and thus form equivalent private keys. However, we argue (heuristically) that the number of *short* representations per ideal class is small. Here and in the following, "short" means that all e_i are in the range $\{-m, \dots, m\}$. The maximum number of such short representations immediately yields the min-entropy[9] of our sampling method, which measures the amount of work a brute-force attacker has to do while conducting an exhaustive search for the key.

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

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

is a surjective group homomorphism (which may be thought of as a projection to the large cyclic subgroup followed by an isomorphism) and define $\alpha_i = \rho([\mathfrak{l}_i])$. We may assume that $\alpha_1 = 1$; this can be done without loss of generality whenever at least

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

one of the $[\mathfrak{l}_i]$ has order N in the class group. For some fixed $[\mathfrak{a}] \in \mathrm{cl}(\mathcal{O})$, any short representation $[\mathfrak{l}_1^{e_1}\mathfrak{l}_2^{e_2}\cdots\mathfrak{l}_n^{e_n}] = [\mathfrak{a}]$ yields a short solution to the linear congruence

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

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

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

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

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

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

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

Meet-in-the-middle key search. Since a private key trivially decomposes into a product of two smooth ideals drawn from smaller sets (e. g. splitting $[\mathfrak{l}_1^{e_1}\mathfrak{l}_2^{e_2}\cdots\mathfrak{l}_n^{e_n}]$ as $[\mathfrak{l}_1^{e_1}\cdots\mathfrak{l}_\nu^{e_\nu}]\cdot[\mathfrak{l}_{\nu+1}^{e_{\nu+1}}\cdots\mathfrak{l}_n^{e_n}]$ for some $\nu \in \{1,\ldots,n\}$), the usual time-memory trade-offs à

la baby-step giant-step [Sha71] with an optimal time complexity of $O\left(\sqrt{\#\operatorname{cl}(\mathcal{O})}\right) \approx O(\sqrt[4]{p})$ apply.[10] Another interpretation of this algorithm is finding a path between two nodes in the underlying isogeny graph by constructing a breadth-first tree starting from each of them, each using a certain subset of the edges, and looking for a collision. Details, including a memoryless variation of this concept, can be found in Delfs and Galbraith's paper [DG16], and for the ordinary case in [Gal99].

Remark 11. The algorithms mentioned thus far scale exponentially in the size of the key space, hence they are asymptotically more expensive than the quantum attacks outlined below which is subexponential in the class-group size. This implies one could possibly balance the costs of the different attacks and use a key space smaller than $\#\operatorname{cl}(\mathcal{O})$ without any loss of security (unless the key space is chosen particularly badly, e. g. as a subgroup), which leads to improved performance. We leave a more thorough analysis of this idea for future work.

Pohlig–Hellman-style attacks. Notice that the set $\mathcal{E}\ell\ell_p(\mathcal{O}, \pi)$ we are acting on does not form a group with efficiently computable operations (that are compatible with the action of $\operatorname{cl}(\mathcal{O})$). Thus there seems to be no way to apply Pohlig–Hellman-style algorithms making use of the decomposition of finite abelian groups. In fact, the Pohlig–Hellman algorithm relies on efficiently computable homomorphisms to proper subgroups, which in the setting at hand would correspond to an efficient algorithm that "projects" a given curve to the orbit of E_0 under a *sub*group action. Therefore, we believe the structure of the class group to be largely irrelevant (assuming it is big enough). In particular, we do not require it to have a large prime-order subgroup.

7.2 Quantum Security

We now discuss the state of quantum algorithms to solve Problem 10.

Grover's algorithm and claw finding. Applying Grover search [Gro96] via claw finding as described in [JDF11] is fully applicable to CSIDH as well, leading to an attack on Problem 10 in $O(\sqrt[6]{p})$ calls to a quantum oracle that computes our group action. The idea is to split the search space for collisions into a classical $O(\sqrt[6]{p})$ target part and a $O(\sqrt[3]{p})$ search part on which a quantum search is applied. Our choices of p that lead to classical security are also immediately large enough to imply quantum

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

security against this attack (c. f. [Nat16, §4.A.5 in Call for Proposals]). That is, the number of queries to our quantum oracle necessary to solve Problem 10 is larger than the number of quantum queries to an AES oracle needed to retrieve the key of the corresponding AES instantiation via Grover's algorithm. For example, an AES-128 key can be recovered with approximately 2^{64} (quantum) oracle queries, which requires us to set $p > 2^{384}$. However, p is much larger than that (see Table 1) due to the existence of subexponential quantum attacks.

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

$$2^{O(\sqrt{\log N \log\log N})}.$$

A follow-up algorithm by Kuperberg [Kup13] uses $2^{O(\sqrt{\log N})}$ time, queries and classical space, but only $O(\log N)$ quantum space. All these algorithms have subexponential time and space complexity.

Attacking the isogeny problem. The relevance of these quantum algorithms to Problem 10 has been observed by Childs–Jao–Soukharev [CJS14] in the ordinary case and by Biasse–Jao–Sankar [BJS14] in the supersingular setting. By defining functions $f_0, f_1 \colon \mathrm{cl}(\mathcal{O}) \to \mathcal{E}\ell\ell_p(\mathcal{O}, \pi)$ as $f_0 \colon [\mathfrak{b}] \mapsto [\mathfrak{b}]E$ and $f_1 \colon [\mathfrak{b}] \mapsto [\mathfrak{b}]\widetilde{E} = [\mathfrak{b}][\mathfrak{a}]E$, the problem can be viewed as an abelian hidden-shift problem with respect to f_0 and f_1. We note that each query requires evaluating the functions f_i on arbitrary ideal classes (i. e. without being given a representative that is a product of ideals of small prime norm) which is non-trivial. However, Childs–Jao–Soukharev show this can be done in subexponential time and space [CJS14, §4].

Subexponential vs. practical. An important remark about all these quantum algorithms is that they do not immediately lead to estimates for runtime and memory requirements on concrete instantiations with $H = \mathrm{cl}(\mathcal{O})$. Although the algorithms by Kuperberg and Regev are shown to have subexponential complexity in the limit, this asymptotic behavior is not enough to understand the space and time complexity on actual (small) instances. For example, Kuperberg's first paper [Kup05, Theorem

3.1] mentions $O(2^{3\sqrt{\log N}})$ oracle queries to achieve a non-negligible success probability when N is a power of a small integer. It also presents a second algorithm that runs in $\tilde{O}(3^{\sqrt{2\log_3 N}}) = O(2^{1.8\sqrt{\log N}})$ [Kup05, Theorem 5.1]. His algorithms handle arbitrary group structures but he does not work out more exact counts for those. Of course, this does not contradict the time complexity of $2^{O(\sqrt{\log N})}$ as stated above, but for a concrete security analysis the hidden constants certainly matter a lot and ignoring the O typically underestimates the security. Childs–Jao–Soukharev [CJS14, Theorem 5.2] prove a query complexity of

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

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

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

on the running time of Bisson's algorithm.

Remark 12. Childs–Jao–Soukharev compute the total cost for computing the secret isogeny in [CJS14, Remark 5.5] to be $L_p[1/2, 3/\sqrt{2}]$ (using Regev and Bisson's algorithms, requiring only polynomial space). They appear to obtain this by setting $N = p$ when multiplying (1) and (2), but as $N \sim \sqrt{p}$ this is an overestimation and should be $L_p[1/2, 1 + 1/\sqrt{2}]$. Either way, this is the largest asymptotic complexity of the estimates. Also, Galbraith and Vercauteren [GV18] point out this algorithm actually has superpolynomial space complexity due to the high memory usage of the isogeny oracle in [CJS14], but see [Jao+18].

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

It is important to mention that asymptotically worse algorithms may provide practical improvements on our "small" instances over either of the algorithms studied by Childs–Jao–Soukharev. For example, Couveignes [Cou06, §5] provides arguments (albeit heuristic ones) that one can find smooth representatives of ideal classes

by computing the class-group structure (which can be done in polynomial time on a quantum computer [Hal05]) and applying a lattice-basis-reduction algorithm such as LLL [LLL82] to its lattice of relations. This might be more efficient than using the subexponential oracle of Childs–Jao–Soukharev. However, note that this method makes evaluating the oracle several times harder for the attacker than for legitimate users, thus immediately giving a few additional bits of security, since users only evaluate the action of very smooth ideals by construction. We believe further research in this direction is necessary and important, since it will directly impact the cost of an attack, but we consider a detailed analysis of all these algorithms and possible trade-offs to be beyond the scope of this chapter.

Remark 13. After we posted a first version of this chapter on the Cryptology ePrint Archive, there appeared several independent attempts at assessing the security of CSIDH. Biasse, Iezzi, and Jacobson [BIJ18] work out some more details of the attack ideas mentioned above for Regev's algorithm. They focus on the class-group-computation part of the oracle and they work out how to represent random elements of the class group as a product of small prime ideals. Their analysis is purely asymptotic and an assessment of the actual cost on specific instances is explicitly left for future work. Bonnetain and Schrottenloher [BS18] determine (quantum) query complexities for breaking CSIDH under the assumption that the quantum memory can be made very large, which implies that Kuperberg's faster algorithms would be applicable. They estimate the number of oracle queries as $(5\pi^2/4)2^{1.8\sqrt{\log N}}$. The 1.8 appears to approximate the $\sqrt{2\log 3}$ in Kuperberg [Kup05, Theorem 5.1]. They state $2^{1.8\sqrt{\log N}+2.3}$ for the number of qubits. While we ignored Kuperberg's algorithm due to the large memory costs, they take the stance that "the most time-efficient version is relevant", and so do not ignore this algorithm. For small N the number of qubits stated in [BS18] might be possible, which would indeed make Kuperberg's algorithm relevant for these sizes. However, in this case the total cost is dominated by the high cost of computing the oracle, which Childs–Jao–Soukharev placed at $L_p[1/2, 1/\sqrt{2}]$. Bonnetain and Schrottenloher instead make use of Couveignes' (exponential-time, but perhaps better for small parameters) LLL-based method for the oracle computation, but apply BKZ for more effective lattice-basis reduction. The current version of Bonnetain–Schrottenloher [BS18] also presents concrete estimates for the attack costs for our parameter sets, but unfortunately this version ignores most of the cost of evaluating isogenies. For example: (1) Algorithm 2 in this chapter makes heavy use of input-dependent branches, which is impossible in superposition [Jao+18, Section 4]; (2) [BS18] skips finding points of order ℓ_i which are needed as the kernel of the ℓ_i-isogeny; (3) [BS18] applies a result for multiplication costs in $\mathbb{F}_{2^n}$ to multiplications

in $\mathbb{F}_p$. We analyzed the (significantly higher) cost of a quantum oracle for isogeny evaluation and conclude that the current estimates of Bonnetain–Schrottenloher do *not* imply that CSIDH-512 (see §7.3) is broken under NIST level 1 (c. f. the reference to [Ber+19] below).

Jao, LeGrow, Leonardi, and Ruiz-Lopez recently made a preprint [Jao+18] of their MathCrypt paper available to us. They address the issue of superpolynomial space in the oracle computation identified by Galbraith and Vercauteren (stated above) and give a new algorithm for finding short representations of elements. Their paper focuses on the asymptotic analysis of the oracle step so that they achieve overall polynomial quantum space, but does not obtain any concrete cost estimates.

Bernstein, Lange, Martindale, and Panny analyze the cost of quantum evaluation of the CSIDH group action in [Ber+19]. Even after introducing several speedups to arithmetic in finite fields and computing isogenies in superposition, for CSIDH-512 it still takes 2^{40} quantum operations on a quantum computer of 2^{40} qubits to compute a single evaluation of the Kuperberg or Regev oracle for success probability 2^{-32} and reduced range of exponents. They also give a more detailed analysis of the shortcomings and errors in [BS18] mentioned above.

7.3 Instantiations

Finally we present estimates for some sizes of p.

Security estimates. As explained in §7.1, the best classical attack has query complexity $O(\sqrt[4]{p})$, and the number of queries has been worked out for different quantum attacks. We consider [CJS14] in combination with Regev and Kuperberg (i. e. $L_p\left[1/2, 3/\sqrt{2}\right]$ and $L_p\left[1/2, 1/\sqrt{2}\right]$, respectively) as well as the pure query complexity of Regev's and Kuperberg's algorithms (i. e. $L_N\left[1/2, \sqrt{2}\right]$, $O(2^{3\sqrt{\log N}})$ and $O(2^{1.8\sqrt{\log N}})$, respectively). We summarize the resulting attack complexities, ignoring the memory costs and without restricting the maximum depth of quantum circuits, for some sizes of p in Table 1. We note again that we expect these complexities to be subject to more careful analysis, taking into account the implicit constants,[11] the (in-)feasibility of long sequential quantum operations, and the large memory requirement. We also include the recent estimates on the query complexity and full

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

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

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

attack complexity by Bonnetain and Schrottenloher [BS18].

We point out a recent analysis [Adj+19] which shows that the classical attack on SIDH (which is the same for CSIDH) is likely slower in practice than current parameter estimates assumed, which is due to the huge memory requirements of the searches. Similarly, the cost of the quantum attacks is significantly higher than just the query complexity times the cost of the group action because evaluating the oracle in superposition is significantly more expensive than a regular group action.

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

Security levels. We approximate security levels as proposed by NIST for the post-quantum standardization effort [Nat16, §4.A.5]. That is, a k-bit security level means that the required effort for the best attacks is at least as large as that needed for a key-retrieval attack on a block cipher with a k-bit key (e. g. AES-k for $k \in \{128, 192, 256\}$). In other words, under the assumption that the attacks query an oracle on a circuit at least as costly as AES, we should have a query complexity of at least 2^{k-1} resp. $\sqrt{2^k}$ to a classical resp. quantum oracle. NIST further restricts the power of the quantum computation to circuits of maximum depth 2^{40} up to 2^{96}, meaning that theoretically optimal tradeoffs (such as the formulas in Table 1 above) might not be

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

Some other algorithms give lower estimates which makes it necessary to evaluate the exact cost of the oracle queries or compute the lower-order terms in the complexity. The analysis in [BS18, Table 8] states lower overall costs compared to AES. While this is a significant improvement, we believe that this does not affect our security claim when accounting precisely for the actual cost of oracle queries, as stated above. Our preliminary analysis shows costs of more than 2^{50} qubit operations for evaluating the oracle for $\log p = 512$, where [BS18] assumes 2^{37}. This means that the NIST levels are reached even with the low query numbers in [BS18]. More analysis is certainly needed and it is unclear whether that will result in larger or smaller choices of p. Note that adjusting parameters only involves changing the prime p (and a few numbers derived from it) and is therefore very simple, should it turn out that our initial estimates are insufficient.

8 Implementation

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

Montgomery curves. The condition $p + 1 \equiv 4 \pmod 8$ implies that all curves in $\mathcal{E}\ell\ell_p(\mathbb{Z}[\pi], \pi)$ can be put in the form $y^2 = x^3 + Ax^2 + x$ (c. f. Proposition 8) for some $A \in \mathbb{F}_p$ via an $\mathbb{F}_p$-isomorphism. This is commonly referred to as the Montgomery form [Mon87] of an elliptic curve and is popular due to the very efficient arithmetic on its x-line. This extends well to computations of isogenies on the x-line, as was first shown by Costello–Longa–Naehrig [CLN16a, §3]. Our implementation uses exactly the same formulas for operations on curves. For isogeny computations on Montgomery curves we use a projectivized variant (to avoid almost all inversions) of the formulas from Costello–Hisil [CH17] and Chapter VIII. This can be done as follows.

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

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

as polynomials in w, we observe that

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

where

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

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

Note that for a given prime ℓ, we could reduce the number of field operations by finding an appropriate representative of the isogeny formulas modulo (a factor of) the ℓ-division polynomial ψ_ℓ (as done in [CLN16a] for 3- and 4-isogenies). Although this would allow for a more efficient implementation, we do not pursue this now for the sake of simplicity.

Rational points. Recall that the goal is to evaluate the action of (the class of) an ideal $\mathfrak{l}_1^{e_1} \cdots \mathfrak{l}_n^{e_n}$ on a curve $E \in \mathcal{E}\ell\ell_p(\mathbb{Z}[\pi], \pi)$, where each $\mathfrak{l}_i = (\ell_i, \pi - 1)$ is a prime ideal of small odd norm ℓ_i and the e_i are integers in a short range $\{-m, \ldots, m\}$. We assume E is given in the form $E_A \colon y^2 = x^3 + Ax^2 + x$. The obvious way to do this is to consider each factor $\mathfrak{l}_i^{\pm 1}$ in this product and to find the abscissa of a point P of order ℓ_i on E, which (depending on the sign) is defined over $\mathbb{F}_p$ or $\mathbb{F}_{p^2} \setminus \mathbb{F}_p$. This exists by our choice of p and ℓ_i (c. f. §4). Finding such an abscissa amounts to sampling a random $\mathbb{F}_p$-rational x-coordinate, checking whether $x^3 + Ax^2 + x$ is a square or not (for $\mathfrak{l}_i^{+1}$ resp. $\mathfrak{l}_i^{-1}$) in $\mathbb{F}_p$ (and resampling if it was wrong), followed by a multiplication by $(p + 1)/\ell_i$ and repeating from the start if the result is ∞. The kernel of the isogeny given by $\mathfrak{l}_i^{\pm 1}$ is then $\langle P \rangle$, so the isogeny may be computed using VÃl'lu-type formulas. Repeating this procedure for all $\mathfrak{l}_i^{\pm 1}$ gives the result.

However, fixing a sign before sampling a random point effectively means wasting about half of all random points, including an ultimately useless square test. Moreover, deciding on a prime ℓ_i before sampling a point and doing the cofactor multiplication wastes another proportion of the points, including both an ultimately useless square test and a scalar multiplication. Both of these issues can be remedied by not fixing an ℓ_i before sampling a point, but instead taking *any* x-coordinate, determining the smallest field of definition (i. e. $\mathbb{F}_p$ or $\mathbb{F}_{p^2}$) of the corresponding point, and then performing whatever isogeny computations are possible using that point (based on its field of definition and order). The steps are detailed in Algorithm 2. Due to the commutativity of $\mathrm{cl}(\mathcal{O})$, and since we only decrease (the absolute value

Algorithm 2. Evaluating the class-group action.

Input: An element $A \in \mathbb{F}_p$ and a list of integers $(e_1, \ldots, e_n)$.
Output: An element B such that $[\mathfrak{l}_1^{e_1} \cdots \mathfrak{l}_n^{e_n}]E_A = E_B$ (where $E_B \colon y^2 = x^3 + Bx^2 + x$).

1 **while** some $e_i \neq 0$ **do**
2 Sample a random $x \in \mathbb{F}_p$
3 **if** $x^3 + Ax^2 + x$ is a square in $\mathbb{F}_p$ **then** $s \leftarrow 1$
4 **else** $s \leftarrow -1$
5 Set $S \leftarrow \{i \mid e_i \neq 0,\ \mathrm{sign}(e_i) = s\}$
6 **if** $S = \emptyset$ **then** goto Line 2
7 Set $k \leftarrow \prod_{i \in S} \ell_i$ and $Q \leftarrow [(p+1)/k]P$
8 **for** each $i \in S$ **do**
9 Set $R \leftarrow [k/\ell_i]Q$
10 **if** $R = \infty$ **then** skip this i
11 Compute an isogeny $\varphi \colon E_A \to E_B \colon y^2 = x^3 + Bx^2 + x$ with $\ker \varphi = R$
12 Set $A \leftarrow B$, $Q \leftarrow \varphi(Q)$, $k \leftarrow k/\ell_i$, and finally $e_i \leftarrow e_i - s$
13 **return** A

of) each e_i once we successfully applied the action of $\mathfrak{l}_i^{\pm 1}$ to the current curve, this algorithm indeed computes the action of $[\mathfrak{l}_1^{e_1} \mathfrak{l}_2^{e_2} \cdots \mathfrak{l}_n^{e_n}]$.

Remark 14. Since the probability that a random point has order divisible by ℓ_i (and hence leads to an isogeny step in Algorithm 2) grows with ℓ_i, the isogeny steps for big ℓ_i are typically completed before those for small ℓ_i. Hence it may make sense to sample the exponents e_i for ideals $\mathfrak{l}_i$ from different ranges depending on the size of ℓ_i, or to not include any very small ℓ_i in the factorization of $p + 1$ at all to reduce the expected number of repetitions of the loop above. Note moreover that doing so may also improve the performance of straightforward constant-time adaptations of our algorithms, since it yields stronger upper bounds on the maximum number of

required loop iterations (at the expense of slightly higher cost per isogeny computation). Varying the choice of the ℓ_i can also lead to performance improvements if the resulting prime p has lower Hamming weight. Finding such a p is a significant computational effort but needs to be done only once; all users can use the same finite field.

Remark 15. Algorithm 2 is obviously variable-time when implemented naïvely. Indeed, the number of points computed in the isogeny formulas is linear in the degree, hence the iteration counts of certain loops in our implementation are very directly related to the private key. We note that it would not be very hard to create a constant-time implementation based on this algorithm by always performing the maximal required number of iterations in each loop and only storing the results that were actually needed (using constant-time conditional instructions), although this incurs quite a bit of useless computation, leading to a doubling of the number of curve operations on average. We leave the design of optimized constant-time algorithms for future work.

Public-key validation. Recall that the public-key validation method outlined in §5 essentially consists of computing $[(p+1)/\ell_i]P$ for each i, where P is a random point on E. Performing this computation in the straightforward way is simple and effective. On the other hand, a divide-and-conquer approach, such as the following recursive algorithm, yields better speeds at the expense of slightly higher memory usage. Note that Algorithm 3 only operates on public data, hence need not be constant-time in a side-channel resistant implementation.

Algorithm 3. Batch cofactor multiplication [Sut07, Algorithm 7.3].

Input: An elliptic-curve point P and positive integers $(k_1, \ldots, k_n)$.
Output: The points $(Q_1, \ldots, Q_n)$, where $Q_i = \left[\prod_{j \neq i} k_j\right] P$.

1 **if** $n = 1$ **then return** (P) ▷ Base case
2 Set $m \leftarrow \lceil n/2 \rceil$ and let $u \leftarrow \prod_{i=1}^{m} k_i$, $v \leftarrow \prod_{i=m+1}^{n} k_i$
3 Set $L \leftarrow [v]P$ and $R \leftarrow [u]P$
4 **recurse** with input $(L, (k_1, \ldots, k_m))$ giving $(Q_1, \ldots, Q_m)$ ▷ Left half
5 **recurse** with input $(R, (k_{m+1}, \ldots, k_n))$ giving $(Q_{m+1}, \ldots, Q_n)$ ▷ Right half
6 **return** $(Q_1, \ldots, Q_n)$

This routine can be used for verifying that an elliptic curve $E/\mathbb{F}_p$ is supersingular as follows: Pick a random point $P \in E(\mathbb{F}_p)$ and run Algorithm 3 on input $[4]P$ and $(\ell_1, \ldots, \ell_n)$ to obtain the points $Q_i = [(p+1)/\ell_i]P$. Then continue like in Algo-

Table 2. Performance numbers for our proof-of-concept implementation (2018.08.26), averaged over 10 000 runs on an Intel Skylake i5 processor clocked at 3.5 GHz.

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

rithm 1 to verify that E is supersingular using these precomputed points.

In practice, it is not necessary to run Algorithm 3 as a black-box function until it returns all the points $Q_1, \ldots, Q_n$: The order checking in Algorithm 1 can be performed as soon as a new point Q_i becomes available, i. e. in the base case of Algorithm 3. This reduces the memory usage (since the points Q_i can be discarded immediately after use) and increases the speed (since the algorithm terminates as soon as enough information was obtained) of public-key validation using Algorithms 1 and 3. We note that the improved performance of this algorithm compared to Algorithm 1 alone essentially comes from a time-space trade-off, hence the memory usage is higher (c. f. §8.1). On severely memory-constrained devices one may instead opt for the naïve algorithm, which requires less space but is slower.

8.1 Performance Results

On top of a minimal implementation in the `sage` computer algebra system [Sag18] for demonstrative purposes, we created a somewhat optimized proof-of-concept implementation of the CSIDH group action for a particular 512-bit prime p. While this implementation features 512-bit field arithmetic written in assembly (for Intel Skylake processors), it also contains generic C code supporting other field sizes and can therefore easily be ported to other computer architectures or parameter sets if desired. The prime p is chosen as $p = 4 \cdot \ell_1 \cdots \ell_{74} - 1$ where ℓ_1 through ℓ_{73} are the smallest 73 odd primes and $\ell_{74} = 587$ is the smallest prime distinct from the other ℓ_i that renders p prime. This parameter choice implies that public keys have a size of 64 bytes. Private keys are stored in 37 bytes for simplicity, but an optimal encoding would reduce this to only 32 bytes. Table 2 summarizes performance numbers for our proof-of-concept implementation. Note that private-key generation is not listed as it only consists of sampling n random integers in a small range $\{-m, \ldots, m\}$, which has negligible cost.

We emphasize that both our implementations are intended as a proof of concept and unfit for production use; in particular, they are explicitly *not side-channel resistant*

and may contain any number of bugs. We leave the design of hardened and more optimized implementations for future work.

Discussion & Conclusions

Finally, we reflect on the material that has been discussed thus far. This thesis has been divided into two parts that deal with cryptographic protocols intended for security against classical and quantum adversaries, respectively. It is only natural that the separation of these two adversarial models plays an important role in the discussion as well. We emphasize that this division is not necessarily legitimate in practice; typically established systems relying on some cryptographic primitive will need to migrate to an alternative secure against a quantum adversary, enforcing design restrictions on the post-quantum primitive. Furthermore, the uncertainty of the advancements in quantum computing may warrant for an intermediate period of time where classical and post-quantum cryptographic primitives are used simultaneously (called *hybrid* systems). However, we limit ourselves to a discussion on the impact of our results and for that reason feel that a separation between classical and post-quantum primitives is justified.

Moreover, an interesting aspect of the field of cryptography is that it is exceptionally fast-paced. As all of the work in this thesis has been made public, the first paper appearing online on 30 October 2015 (Chapter III), much of the material has been followed up on by various people. Since the content of this thesis is based on published work, these follow-up articles have not (with a few exceptions) been discussed. On the other hand, they can significantly influence the conclusions to be drawn from our work. Wherever necessary, we comment on those of interest before presenting conclusions.

Classical Cryptography (Part 2)

We recall that the main goal of the second part (Chapter III–VI) was to improve the security and efficiency of curve-based primitives based on the discrete logarithm problem. The first step in this direction (Chapter III) provides complete addition

formulas for elliptic curves in short Weierstrass form whose group of rational points over the base (prime) field is odd. These curves are ubiquitous due to their appearance in standards (e. g. [Acc99b; Nat13] though many others exist). As the formulas do not negatively impact the protocols based on the curves themselves, these formulas can be integrated into existing systems and directly increase the confidence in their security. This becomes increasingly true in complex systems where exceptional cases are hard to avoid, in particular when one knows an adversary to have access to side-channels (e. g. for embedded systems). Although the formulas are not as efficient as established ones (see e. g. Table III.5), we believe the incurred slowdown is often worthwhile with respect to the ease of implementation. Again, this is especially true in systems where the overall efficiency is not determined by the cryptographic primitive, for which the overhead caused by the formulas will be negligible.

The remaining chapters in the second part (Chapter IV–VI) look towards primitives based on Kummer varieties. In the case of elliptic curves these have been popularized through the use of Curve25519 [Ber06a], which has been recommended for use by the Crypto Forum Research Group (CFRG) [LHT16] and is to be included in the NIST SP 800-186 standard. The attractiveness of Curve25519 does not only come from its speed, but also from its simplicity. Although initially aimed at the Diffie–Hellman key exchange, we have aimed to capture this simplicity and transfer it to the qDSA signature scheme (Chapter V). The main advantage is that one never needs to compute on the elliptic curve, reducing the code base and improving the ease of implementation. We have shown its feasibility in Chapter IV by implementing the scheme based on Curve25519, providing highly memory friendly implementations on the AVR ATmega (see Table V.3) and ARM Cortex M0 (see Table V.4). Moreover, its efficiency has since been demonstrated on the ARM Cortex M family [FA17] of microprocessors and on Intel's Haswell and Skylake processors [FH+17]. A study of the arithmetic of the Kummer lines of curves with rational 2-torsion (see Chapter VI) shows that one can easily extend the protocol to other forms of curves. This is particularly interesting for the case of squared Kummer lines [GL09] where single-instruction multiple-data (SIMD) instructions can prove beneficial [KS17]. We have provided an extension to the squared Kummer line on the ARM Cortex M0 (see Table VI.2), which has since been implemented on the Skylake platform as well [KS17]. We expect the rise of SIMD to increase their popularity.

The genus-2 counterpart has not received much attention with respect to standardization, as its computational advantage had not been clear until the work of Gaudry in 2007 [Gau07]. In Chapter IV we provided the first implementation of a signature scheme based on a genus-2 Jacobian on the AVR ATmega and ARM Cor-

tex M0 architectures, and significantly outperformed the state-of-the-art. There have since appeared several implementations on FPGAs [GCT17; GT17; Kop+18], for which the arithmetic is highly suitable due to its inherent parallelism. Although very efficient, its major downside is the complexity of operations on the Jacobian. This is precisely the problem that the qDSA scheme intends to solve; all operations are performed on the Kummer surface instead. Its benefits compared to elliptic curves (e. g. Curve25519) are clear, and they outperform them in terms of code size as well as efficiency. On the other hand, the use of genus-2 curve-based cryptography is much less standard and therefore not likely to be adopted on a large scale any time soon. We believe to have alleviated the *technical* restrictions on doing so, and hope confidence in its security will be maintained and increased.

Concluding, we believe our work has made a positive impact on the state of curve-based cryptography. The implementation of the most basic cryptographic protocols (i. e. key exchange and signatures) has been made more efficient while retaining security, both in the case of more established prime order Weierstrass curves as well as the more modern schemes based on Kummer varieties. We expect that these will not only be integrated in existing and upcoming curve-based *classical* schemes, but hope that the study of the arithmetic of Kummer varieties could also be valuable for isogeny-based protocols.

Post-Quantum Cryptography (Part 3)

The second part focuses on isogeny-based cryptography, which has seen a relatively recent surge in interest. The original had been proposed in 2011 [JDF11], yet it was not until the paper by Costello, Longa and Naehrig [CLN16a] that its significance was first demonstrated. The instantiation based on the x-line of a Montgomery curve computed an isogeny in approximately 15 milliseconds, while having public keys of 564 bytes at an estimated 128-bit security level. It has since gained much attention and improvements, culminating in the SIKE proposal [Jao+16] for standardization by NIST. It computes an isogeny in roughly 2.5 milliseconds with 378-byte public keys, for NIST security category I [Nat16, §4.A.5].[12] The main advantage of SIKE is the small size of its public keys; it is the key exchange proposal with the smallest public keys.

In Chapter VII we significantly improved the efficiency of compressing the public keys. The main idea was proposed by [Aza+16] and our improvements both made it practical, as well as decreased the size of the compressed keys. A follow-up paper

[12] This means that breaking the security is at least as hard as recovering a key from AES-128.

by Zanon et al. [Zan+18] includes additional optimizations. The main downside is that the techniques to obtain efficient key compression are complex, while rather cumbersome to implement. For that reason it had initially been excluded from the SIKE submission, though the advancement to Round 2 has led to the future inclusion of public-key compression into the proposal. An interesting line of research would be to improve this further, or alternatively to simplify its implementation.

The next chapter (Chapter VIII) aims to improve the understanding of isogenies between Montgomery curves, and related models. It builds upon the work by Costello and Hisil [CH17], and generalizes their statements while simultaneously simplifying their proofs. We believe this work can be extended; that is, it could lead to interesting isogeny formulas for alternative curve models. In particular, those that have a canonical choice of point (compared to $(0, 0)$ for Montgomery curves). We provide a first step in this direction in §VIII.4, but expect this could be improved.

Furthermore, an unfortunate property of SIDH is its insecurity when using static keys [Gal+16]. This ultimately boils down to a lack of public-key validation, whereas computation with respect to erroneous public keys can leak information about the private key. As a result, it is only secure in an ephemeral setting. Alternatively, the SIKE submission applies modifications to the protocol to provide active security (and is actually a *key encapsulation mechanism*), essentially doubling its runtime. Although not affecting its security, it prohibits using SIKE as a direct replacement for Diffie–Hellman. This led to the development of CSIDH in Chapter IX which clearly shares certain characteristics with SIKE. For example, they are based on isogeny graphs supersingular elliptic curves and both use the formulas present in Chapter VIII. However, on a protocol level there are significant differences. The straightforward public-key validation of CSIDH (see Proposition IX.8) makes the scheme *non-interactive*, i. e. allows for static–static key exchange. This gives it a unique flavor and makes it extremely interesting for practitioners. The main limitation for adoption of CSIDH has been a lack of concrete understanding of the (sub-exponential) quantum attacks, though recent papers [BS18; Ber+19] have provided more clarity in this area. We are convinced more work in this direction will be very beneficial for the use of CSIDH. Moreover, work has appeared that increases the efficiency of the scheme [MR18] or considers constant-time implementation [MCR18]. We believe improvements are still to be made and should be looked into.

Finally, we comment on an obviously desirable yet (seemingly) hard to attain goal of isogeny-based cryptography. That is, to achieve a full isogeny-based public-key infrastructure one would want to have a signature scheme. There have been multiple attempts; Yoo et al. [Yoo+17] present the first general purpose scheme by

applying the Fiat-Shamir (or Unruh) transformation to a 1-bit identification scheme of Jao and De Feo [JDF11], while Galbraith, Petit and Silva [GPS17] independently present a similar and a second more intricate scheme. Notably, both schemes are extremely inefficient and lead to large signatures. It is an interesting direction to look into, though significant changes to the identification scheme (or the transformation) need to be made to make it practical. Alternatively, a recent paper by Galbraith and De Feo [DFG19] puts forward a signature scheme based on a 1-bit identification protocol [RS06] related to the CSIDH class group action. Though clever tricks are applied (and further ones have since been proposed [DPV18]), the scheme remains slow and either the public key or the signature is big (trade-offs are possible). Given its relative novelty, worthwhile improvements could still be found. Again, for significant gains we expect drastic changes will be necessary. However, all in all we believe the search for isogeny-based signatures deserves attention and is a valuable research direction.

Bibliography

[Acc99a] Accredited Standards Committee X9. *American National Standard X9.62-1999, Public key cryptography for the financial services industry: the elliptic curve digital signature algorithm (ECDSA).* Draft at http://grouper.ieee.org/groups/1363/Research/Other.html. 1999.

[Acc99b] Accredited Standards Committee X9. *American National Standard X9.63-2001, Public key cryptography for the financial services industry: key agreement and key transport using elliptic curve cryptography.* Draft at http://grouper.ieee.org/groups/1363/Research/Other.html. 1999.

[Adj+19] Gora Adj, Daniel Cervantes-Vázquez, Jesús-Javier Chi-Domínguez, Alfred Menezes, and Francisco Rodríguez-Henríquez. "On the Cost of Computing Isogenies Between Supersingular Elliptic Curves". In: *Selected Areas in Cryptography – SAC 2018*. Ed. by Carlos Cid and Michael J. Jacobson Jr. Cham: Springer International Publishing, 2019, pp. 322–343. DOI: 10.1007/978-3-030-10970-7_15.

[AFJ14] Reza Azarderakhsh, Dieter Fishbein, and David Jao. *Efficient Implementations of A Quantum-Resistant Key-Exchange Protocol on Embedded systems*. Tech. rep. http://cacr.uwaterloo.ca/techreports/2014/cacr2014-20.pdf. 2014.

[Age14] Agence nationale de la sécurité des sysèmes d'information (ANSSI). *Mécanismes cryptographiques: Règles et recommandations concernant le choix et le dimensionnement des mécanismes cryptographiques*. http://www.ssi.gouv.fr/uploads/2015/01/RGS_v-2-0_B1.pdf. 2014.

[AJS16] Erdem Alkim, Philipp Jakubeit, and Peter Schwabe. "NewHope on ARM Cortex-M". In: *Security, Privacy, and Applied Cryptography Engineering*. Ed. by Claude Carlet, M. Anwar Hasan, and Vishal Saraswat. Cham: Springer International Publishing, 2016, pp. 332–349. DOI: 10.1007/978-3-319-49445-6_19.

[AKR12] Christophe Arène, David Kohel, and Christophe Ritzenthaler. "Complete addition laws on abelian varieties". In: *LMS Journal of Computation and Mathematics* 15 (2012), pp. 308–316. DOI: 10.1112/S1461157012001027.

[And] *Android Documentation for App Developers*. https://developer.android.com/guide/topics/security/cryptography.

[App] *Apple Documentation for App Developers*. https://developer.apple.com/documentation/security/seckeyalgorithm.

[Arè+11] Christophe Arène, Tanja Lange, Michael Naehrig, and Christophe Ritzenthaler. "Faster computation of the Tate pairing". In: *Journal of Number Theory* 131.5 (2011). Elliptic Curve Cryptography, pp. 842 –857. DOI: 10.1016/j.jnt.2010.05.013.

[Aza+16] Reza Azarderakhsh, David Jao, Kassem Kalach, Brian Koziel, and Christopher Leonardi. "Key Compression for Isogeny-Based Cryptosystems". In: *Proceedings of the 3rd ACM International Workshop on ASIA Public-Key Cryptography, AsiaPKC@AsiaCCS, Xi'an, China, May 30 - June 03, 2016*. 2016, pp. 1–10. DOI: 10.1145/2898420.2898421.

[Bai62] Walter L. Baily Jr. "On the theory of θ-functions, the moduli of abelian varieties, and the moduli of curves". In: *Annals of Mathematics (2)* 75 (1962), pp. 342–381. DOI: 10.2307/1970178.

[Bar+02] Paulo S. L. M. Barreto, Hae Y. Kim, Ben Lynn, and Michael Scott. "Efficient Algorithms for Pairing-Based Cryptosystems". In: *Advances in Cryptology — CRYPTO 2002*. Ed. by Moti Yung. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 354–369. DOI: 10.1007/3-540-45708-9_23.

[Bar+07] Paulo S. L. M. Barreto, Steven D. Galbraith, Colm Ó hÉigeartaigh, and Michael Scott. "Efficient pairing computation on supersingular Abelian varieties". In: *Designs, Codes and Cryptography* 42.3 (2007), pp. 239–271. DOI: 10.1007/s10623-006-9033-6.

[Bat+05] Lejla Batina, David Hwang, Alireza Hodjat, Bart Preneel, and Ingrid Verbauwhede. "Hardware/Software Co-design for Hyperelliptic Curve Cryptography (HECC) on the 8051 μP". In: *Cryptographic Hardware and Embedded Systems – CHES 2005*. Ed. by Josyula R. Rao and Berk Sunar. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 106–118. DOI: 10.1007/11545262_8.

[BBMÖ04] Lejla Batina, Geeke Bruin-Muurling, and Sıddıka Berna Örs. "Flexible Hardware Design for RSA and Elliptic Curve Cryptosystems". In: *Topics in Cryptology – CT-RSA 2004*. Ed. by Tatsuaki Okamoto. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 250–263. DOI: 10.1007/978-3-540-24660-2_20.

[BCL14] Daniel J. Bernstein, Chitchanok Chuengsatiansup, and Tanja Lange. "Curve41417: Karatsuba Revisited". In: *Cryptographic Hardware and Embedded Systems – CHES 2014*. Ed. by Lejla Batina and Matthew Robshaw. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 316–334. DOI: 10.1007/978-3-662-44709-3_18.

[BCP97] Wieb Bosma, John J. Cannon, and Catherine Playoust. "The Magma Algebra System I: The User Language". In: *Journal of Symbolic Computation* 24.3/4 (1997), pp. 235–265. DOI: 10.1006/jsco.1996.0125.

[BDL97] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. "On the Importance of Checking Cryptographic Protocols for Faults". In: *Advances in Cryptology — EUROCRYPT '97*. Ed. by Walter Fumy. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 37–51. DOI: 10.1007/3-540-69053-0_4.

[Ber+08] Daniel J. Bernstein, Peter Birkner, Marc Joye, Tanja Lange, and Christiane Peters. "Twisted Edwards Curves". In: *Progress in Cryptology – AFRICACRYPT 2008*. Ed. by Serge Vaudenay. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 389–405. DOI: 10.1007/978-3-540-68164-9_26.

[Ber+12] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. "High-speed high-security signatures". In: *Journal of Cryptographic Engineering* 2.2 (2012), pp. 77–89. DOI: 10.1007/s13389-012-0027-1.

[Ber+13] Daniel J. Bernstein, Mike Hamburg, Anna Krasnova, and Tanja Lange. "Elligator: elliptic-curve points indistinguishable from uniform random strings". In: *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. New York, NY, USA: ACM, 2013, pp. 967–980. DOI: 10.1145/2508859.2516734.

[Ber+14] Daniel J. Bernstein, Chitchanok Chuengsatiansup, Tanja Lange, and Peter Schwabe. "Kummer Strikes Back: New DH Speed Records". In: *Advances in Cryptology – ASIACRYPT 2014*. Ed. by Palash Sarkar and Tetsu Iwata. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 317–337. DOI: 10.1007/978-3-662-45611-8_17.

[Ber+15a] Daniel J. Bernstein, Bernard van Gastel, Wesley Janssen, Tanja Lange, Peter Schwabe, and Sjaak Smetsers. "TweetNaCl: A Crypto Library in 100 Tweets". In: *Progress in Cryptology - LATINCRYPT 2014*. Ed. by Diego F. Aranha and Alfred Menezes. Cham: Springer International Publishing, 2015, pp. 64–83. DOI: 10.1007/978-3-319-16295-9_4.

[Ber+15b] Daniel J. Bernstein, Chitchanok Chuengsatiansup, David Kohel, and Tanja Lange. "Twisted Hessian Curves". In: *Progress in Cryptology – LATINCRYPT 2015*. Ed. by Kristin Lauter and Francisco Rodríguez-Henríquez. Cham: Springer International Publishing, 2015, pp. 269–294. DOI: 10.1007/978-3-319-22174-8_15.

[Ber+16] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. *The* KECCAK *sponge function family*. http://keccak.noekeon.org/. 2016.

[Ber+19] Daniel J. Bernstein, Tanja Lange, Chloe Martindale, and Lorenz Panny. "Quantum Circuits for the CSIDH: Optimizing Quantum Evaluation of Isogenies". In: *Advances in Cryptology – EUROCRYPT 2019*. Ed. by Yuval Ishai and Vincent Rijmen. Cham: Springer International Publishing, 2019, pp. 409–441. DOI: 10.1007/978-3-030-17656-3_15.

[Ber06a] Daniel J. Bernstein. "Curve25519: New Diffie-Hellman Speed Records". In: *Public Key Cryptography - PKC 2006*. Ed. by Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 207–228. DOI: 10.1007/11745853_14.

[Ber06b] Daniel J. Bernstein. *Differential addition chains*. http://cr.yp.to/ecdh/diffchain-20060219.pdf. 2006.

[Ber06c] Daniel J. Bernstein. *Elliptic vs. hyperelliptic, part 1*. http://cr.yp.to/talks/2006.09.20/slides.pdf. 2006.

[BIJ18] Jean-François Biasse, Annamaria Iezzi, and Michael J. Jacobson. "A Note on the Security of CSIDH". In: *Progress in Cryptology – INDOCRYPT 2018*. Ed. by Debrup Chakraborty and Tetsu Iwata. Cham: Springer International Publishing, 2018, pp. 153–168. DOI: 10.1007/978-3-030-05378-9_9.

[Bis12] Gaetan Bisson. "Computing endomorphism rings of elliptic curves under the GRH". In: *Journal of Mathematical Cryptology* 5.2 (2012), pp. 101–114. DOI: 10.1515/JMC.2011.008.

[BJ02] Éric Brier and Marc Joye. "Weierstraß Elliptic Curves and Side-Channel Attacks". In: *Public Key Cryptography*. Ed. by David Naccache and Pascal Paillier. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 335–345. DOI: 10.1007/3-540-45664-3_24.

[BJ03] Eric Brier and Marc Joye. "Fast Point Multiplication on Elliptic Curves through Isogenies". In: *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*. Ed. by Marc Fossorier, Tom Høholdt, and Alain Poli. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 43–50. DOI: 10.1007/3-540-44828-4_6.

[BJS14] Jean-François Biasse, David Jao, and Anirudh Sankar. "A Quantum Algorithm for Computing Isogenies between Supersingular Elliptic Curves". In: *Progress in Cryptology – INDOCRYPT 2014*. Ed. by Willi Meier and Debdeep Mukhopadhyay. Cham: Springer International Publishing, 2014, pp. 428–442. DOI: 10.1007/978-3-319-13039-2_25.

[BLa] Daniel J. Bernstein and Tanja Lange. *eBACS: ECRYPT Benchmarking of Cryptographic Systems*. https://bench.cr.yp.to/index.html.

[BLb] Daniel J. Bernstein and Tanja Lange. *Explicit-Formulas Database*. https://hyperelliptic.org/EFD/index.html.

[BLc] Daniel J. Bernstein and Tanja Lange. *SafeCurves: choosing safe curves for elliptic-curve cryptography*. http://safecurves.cr.yp.to/.

[BL07] Daniel J. Bernstein and Tanja Lange. "Faster Addition and Doubling on Elliptic Curves". In: *Advances in Cryptology – ASIACRYPT 2007*. Ed. by Kaoru Kurosawa. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 29–50. DOI: 10.1007/978-3-540-76900-2_3.

[BL09] Daniel J. Bernstein and Tanja Lange. *Complete addition laws for elliptic curves*. http://cr.yp.to/talks/2009.04.17/slides.pdf. 2009.

[BL95] W. Bosma and H. W. Lenstra. "Complete Systems of Two Addition Laws for Elliptic Curves". In: *Journal of Number theory* 53.2 (1995), pp. 229–240. DOI: 10.1006/jnth.1995.1088.

[BLS12] Daniel J. Bernstein, Tanja Lange, and Peter Schwabe. "The Security Impact of a New Cryptographic Library". In: *Progress in Cryptology – LATINCRYPT 2012*. Ed. by Alejandro Hevia and Gregory Neven. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 159–176. DOI: 10.1007/978-3-642-33481-8_9.

[BN06] Paulo S. L. M. Barreto and Michael Naehrig. "Pairing-Friendly Elliptic Curves of Prime Order". In: *Selected Areas in Cryptography*. Ed. by Bart Preneel and Stafford Tavares. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 319–331. DOI: 10.1007/11693383_22.

[Boe90] Bert den Boer. "Diffie-Hellman is as Strong as Discrete Log for Certain Primes". In: *Advances in Cryptology — CRYPTO' 88*. Ed. by Shafi Goldwasser. New York, NY: Springer New York, 1990, pp. 530–539. DOI: 10.1007/0-387-34799-2_38.

[Bos+13] Joppe W. Bos, Craig Costello, Huseyin Hisil, and Kristin Lauter. "Fast Cryptography in Genus 2". In: *Advances in Cryptology – EUROCRYPT 2013*. Ed. by Thomas Johansson and Phong Q. Nguyen. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 194–210. DOI: 10.1007/978-3-642-38348-9_12.

[Bos+14] Joppe W. Bos, J. Alex Halderman, Nadia Heninger, Jonathan Moore, Michael Naehrig, and Eric Wustrow. "Elliptic Curve Cryptography in Practice". In: *Financial Cryptography and Data Security*. Ed. by Nicolas Christin and Reihaneh Safavi-Naini. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 157–175. DOI: 10.1007/978-3-662-45472-5_11.

[Bos+16] Joppe W. Bos, Craig Costello, Patrick Longa, and Michael Naehrig. "Selecting elliptic curves for cryptography: an efficiency and security analysis". In: *Journal of Cryptographic Engineering* 6.4 (2016), pp. 259–286. DOI: 10.1007/s13389-015-0097-y.

[BR93] Mihir Bellare and Phillip Rogaway. "Random Oracles Are Practical: A Paradigm for Designing Efficient Protocols". In: *Proceedings of the 1st ACM Conference on Computer and Communications Security*. New York, NY, USA: ACM, 1993, pp. 62–73. DOI: 10.1145/168588.168596.

[Brö08] Reinier Bröker. "A p-adic algorithm to compute the Hilbert class polynomial". In: *Mathematics of Computation* 77.264 (2008), pp. 2417–2435. DOI: 10.1090/S0025-5718-08-02091-7.

[BS07] Reinier Bröker and Peter Stevenhagen. "Efficient CM-constructions of elliptic curves over finite fields". In: *Mathematics of Computation* 76.260 (2007), pp. 2161–2179. DOI: 10.1090/S0025-5718-07-01980-1.

[BS18] Xavier Bonnetain and André Schrottenloher. *Quantum Security Analysis of CSIDH and Ordinary Isogeny-based Schemes*. IACR Cryptology ePrint Archive 2018/537, version 20180621:135910. https://eprint.iacr.org/2018/537/20180621:135910. 2018.

[BV07] Johannes Buchmann and Ulrich Vollmer. *Binary quadratic forms: an algorithmic approach*. Vol. 20. Algorithms and Computation in Mathematics. Springer, 2007. DOI: 10.1007/978-3-540-46368-9.

[BY91] Gilles Brassard and Moti Yung. "One-Way Group Actions". In: *Advances in Cryptology – CRYPTO' 90*. Ed. by Alfred J. Menezes and Scott A. Vanstone. Berlin, Heidelberg: Springer Berlin Heidelberg, 1991, pp. 94–107. DOI: 10.1007/3-540-38424-3_7.

[Can87] David G. Cantor. "Computing in the Jacobian of a Hyperelliptic Curve". In: *Mathematics of Computation* 48 (1987), pp. 95–101. DOI: 10.1090/S0025-5718-1987-0866101-0.

[CBC07] Gang Chen, Guoqiang Bai, and Hongyi Chen. "A High-Performance Elliptic Curve Cryptographic Processor for General Curves Over GF(p) Based on a Systolic Arithmetic Unit". In: *Circuits and Systems II: Express Briefs, IEEE Transactions on* 54.5 (2007), pp. 412–416. DOI: 10.1109/TCSII.2006.889459.

[CC86] David V. Chudnovsky and Gregory V. Chudnovsky. "Sequences of numbers generated by addition in formal groups and new primality and factorization tests". In: *Advances in Applied Mathematics* 7 (1986), pp. 385–434. DOI: 10.1016/0196-8858(86)90023-0.

[CCS15] Craig Costello, Ping-Ngai Chung, and Benjamin Smith. *Fast, uniform, and compact scalar multiplication for elliptic curves and genus 2 Jacobians with applications to signature schemes*. Cryptology ePrint Archive, Report 2015/983. https://eprint.iacr.org/2015/983. 2015.

[CCS17] Ping Ngai Chung, Craig Costello, and Benjamin Smith. "Fast, Uniform Scalar Multiplication for Genus 2 Jacobians with Fast Kummers". In: *Selected Areas in Cryptography – SAC 2016*. Ed. by Roberto Avanzi and Howard Heys. Cham: Springer International Publishing, 2017, pp. 465–481. DOI: 10.1007/978-3-319-69453-5_25.

[Cer] Ltd Certivox UK. *CertiVox Standard Curves*. http://docs.certivox.com/docs/miracl/certivox-standard-curves.

[Cer10] Certicom Research. *SEC 2: Recommended Elliptic Curve Domain Parameters, Version 2.0*. http://www.secg.org/sec2-v2.pdf. 2010.

[CF96] John W. S. Cassels and E. Victor Flynn. *Prolegomena to a middlebrow arithmetic of curves of genus 2*. Vol. 230. Cambridge University Press, 1996. DOI: 10.1017/CBO9780511526084.

[CGF08] Wouter Castryck, Steven Galbraith, and Reza Rezaeian Farashahi. *Efficient arithmetic on elliptic curves using a mixed Edwards-Montgomery representation*. Cryptology ePrint Archive, Report 2008/218. https://eprint.iacr.org/2008/218. 2008.

[CH17] Craig Costello and Huseyin Hisil. "A Simple and Compact Algorithm for SIDH with Arbitrary Degree Isogenies". In: *Advances in Cryptology – ASIACRYPT 2017*. Ed. by Tsuyoshi Takagi and Thomas Peyrin. Cham: Springer International Publishing, 2017, pp. 303–329. DOI: 10.1007/978-3-319-70697-9_11.

[Che+16] Lidong Chen, Stephen P. Jordan, Yi-Kai Liu, Dustin Moody, Rene C. Peralta, Ray A. Perlner, and Daniel C. Smith-Tone. *Report on Post-Quantum Cryptography*. NISTIR 8105, DRAFT. http://csrc.nist.gov/publications/drafts/nistir-8105/nistir_8105_draft.pdf. 2016.

[Chi10] Chinese Commerical Cryptography Administration Office. *SM2 Digital Signature Algorithm*. See http://www.oscca.gov.cn/UpFile/2010122214836668.pdf and http://tools.ietf.org/html/draft-shen-sm2-ecdsa-02. 2010.

[Cho15] Tung Chou. *Sandy2x*. Message on the Curves mailing list at `https://moderncrypto.org/mail-archive/curves/2015/000637.html`. 2015.

[CJS14] Andrew M. Childs, David Jao, and Vladimir Soukharev. "Constructing elliptic curve isogenies in quantum subexponential time". In: *Journal of Mathematical Cryptology* 8.1 (2014), pp. 1–29. DOI: 10.1515/jmc-2012-0016.

[CL15] Craig Costello and Patrick Longa. "FourQ: Four-Dimensional Decompositions on a Q-curve over the Mersenne Prime". In: *Advances in Cryptology – ASIACRYPT 2015*. Ed. by Tetsu Iwata and Jung Hee Cheon. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 214–235. DOI: 10.1007/978-3-662-48797-6_10.

[CL84] Henri Cohen and Hendrik W. Lenstra, Jr. "Heuristics on class groups of number fields". In: *Number Theory Noordwijkerhout 1983*. Ed. by Hendrik Jager. Springer, 1984, pp. 33–62. DOI: 10.1007/BFb0099440.

[CLG09] Denis X. Charles, Kristin E. Lauter, and Eyal Z. Goren. "Cryptographic Hash Functions from Expander Graphs". In: *Journal of Cryptology* 22.1 (2009), pp. 93–113. DOI: 10.1007/s00145-007-9002-x.

[CLN16a] Craig Costello, Patrick Longa, and Michael Naehrig. "Efficient Algorithms for Supersingular Isogeny Diffie-Hellman". In: *Advances in Cryptology – CRYPTO 2016*. Ed. by Matthew Robshaw and Jonathan Katz. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 572–601. DOI: 10.1007/978-3-662-53018-4_21.

[CLN16b] Craig Costello, Patrick Longa, and Michael Naehrig. *SIDH Library*. `http://research.microsoft.com/en-us/downloads/bd5fd4cd-61b6-458a-bd94-b1f406a3f33f/`. 2016.

[CMO98] Henri Cohen, Atsuko Miyaji, and Takatoshi Ono. "Efficient Elliptic Curve Exponentiation Using Mixed Coordinates". In: *Advances in Cryptology — ASIACRYPT'98*. Ed. by Kazuo Ohta and Dingyi Pei. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 51–65. DOI: 10.1007/3-540-49649-1_6.

[Com15] Committee on National Security Systems (CNSS). *Advisory Memorandum: Use of Public Standards for the Secure Sharing of Information Among National Security Systems*. `https://www.cnss.gov/CNSS/openDoc.cfm?Q5ww0Xu+7kg/OpTB/R2/MQ==`. 2015.

[Cor99] Jean-Sébastien Coron. "Resistance Against Differential Power Analysis For Elliptic Curve Cryptosystems". In: *Cryptographic Hardware and Embedded Systems*. Ed. by Çetin K. Koç and Christof Paar. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 292–302. DOI: 10.1007/3-540-48059-5_25.

[Cos11] Romain Cosset. "Applications of theta functions for hyperelliptic curve cryptography". https://tel.archives-ouvertes.fr/tel-00642951/file/main.pdf. PhD thesis. Université Henri Poincaré - Nancy I, 2011.

[COS86] Don Coppersmith, Andrew M. Odlzyko, and Richard Schroeppel. "Discrete Logarithms in GF(p)". In: *Algorithmica* 1.1 (1986), pp. 1–15. DOI: 10.1007/BF01840433.

[Cou06] Jean-Marc Couveignes. *Hard Homogeneous Spaces*. Cryptology ePrint Archive, Report 2006/291. https://eprint.iacr.org/2006/291. 2006.

[Cox13] David A. Cox. *Primes of the form $x^2 + ny^2$: Fermat, class field theory, and complex multiplication*. 2nd. Pure and applied mathematics. Wiley, 2013. DOI: 10.1002/9781118400722.

[CS17] Craig Costello and Benjamin Smith. "Montgomery curves and their arithmetic". In: *Journal of Cryptographic Engineering* 8 (2017), pp. 227–240. DOI: 10.1007/s13389-017-0157-6.

[DF17] Luca De Feo. "Mathematics of Isogeny Based Cryptography". In: *arXiv e-prints* (2017). https://arxiv.org/abs/1711.04062.

[DFG19] Luca De Feo and Steven D. Galbraith. "SeaSign: Compact Isogeny Signatures from Class Group Actions". In: *Advances in Cryptology – EUROCRYPT 2019*. Ed. by Yuval Ishai and Vincent Rijmen. Cham: Springer International Publishing, 2019, pp. 759–789. DOI: 10.1007/978-3-030-17659-4_26.

[DFJP14] Luca De Feo, David Jao, and Jérôme Plût. "Towards Quantum-Resistant Cryptosystems from Supersingular Elliptic Curve Isogenies". In: *Journal of Mathematical Cryptology* 8 (2014), pp. 209–247. DOI: 10.1515/jmc-2012-0015.

[DFKS18] Luca De Feo, Jean Kieffer, and Benjamin Smith. "Towards Practical Key Exchange from Ordinary Isogeny Graphs". In: *Advances in Cryptology – ASIACRYPT 2018*. Ed. by Thomas Peyrin and Steven Galbraith. Cham: Springer International Publishing, 2018, pp. 365–394. DOI: 10.1007/978-3-030-03332-3_14.

[DG16] Christina Delfs and Steven D. Galbraith. "Computing isogenies between supersingular elliptic curves over $\mathbb{F}_p$". In: *Designs, Codes and Cryptography* 78.2 (2016), pp. 425–440. DOI: 10.1007/s10623-014-0010-1.

[DH76] Whitfield Diffie and Martin E. Hellman. "New directions in cryptography". In: *IEEE Transactions on Information Theory* 22.6 (1976), pp. 644–654. DOI: 10.1109/TIT.1976.1055638.

[DPV18] Thomas Decru, Lorenz Panny, and Frederik Vercauteren. *Faster SeaSign signatures through improved rejection sampling*. Cryptology ePrint Archive, Report 2018/1109. https://eprint.iacr.org/2018/1109. 2018.

[Duq04] Sylvain Duquesne. "Montgomery Scalar Multiplication for Genus 2 Curves". In: *Algorithmic Number Theory*. Ed. by Duncan Buell. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 153–168. DOI: 10.1007/978-3-540-24847-7_11.

[Dwo15] Morris J. Dworkin. *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*. Tech. rep. http://www.nist.gov/manuscript-publication-search.cfm?pub_id=919061. National Institute of Standards and Technology (NIST), 2015.

[Dül+15] Michael Düll, Björn Haase, Gesine Hinterwälder, Michael Hutter, Christof Paar, Ana Helena Sánchez, and Peter Schwabe. "High-speed Curve25519 on 8-bit, 16-bit, and 32-bit microcontrollers". In: *Designs, Codes and Cryptography* 77.2 (2015), pp. 493–514. DOI: 10.1007/s10623-015-0087-1.

[ECC05] ECC Brainpool. *ECC Brainpool Standard Curves and Curve Generation*. http://www.ecc-brainpool.org/download/Domain-parameters.pdf. 2005.

[Edw07] Harold Edwards. "A normal form for elliptic curves". In: *Bulletin of the American Mathematical Society* 44.3 (2007), pp. 393–422. DOI: 10.1090/S0273-0979-07-01153-6.

[ElG85] Taher ElGamal. "A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms". In: *Advances in Cryptology*. Ed. by George Robert Blakley and David Chaum. Berlin, Heidelberg: Springer Berlin Heidelberg, 1985, pp. 10–18. DOI: 10.1109/TIT.1985.1057074.

[Eng14] Andreas Enge. "Bilinear pairings on elliptic curves". In: *arXiv e-prints* (2014). https://arxiv.org/abs/1301.5520.

[FA17] H. Fujii and D. F. Aranha. "Curve25519 for the Cortex-M4 and beyond". In: *Progress in Cryptology – LATINCRYPT 2017*. http://www.cs.haifa.ac.il/~orrd/LC17/paper39.pdf. 2017.

[FGV11] Junfeng Fan, Benedikt Gierlichs, and Frederik Vercauteren. "To Infinity and Beyond: Combined Attack on ECC Using Points of Low Order". In: *Cryptographic Hardware and Embedded Systems – CHES 2011*. Ed. by Bart Preneel and Tsuyoshi Takagi. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 143–159. DOI: 10.1007/978-3-642-23951-9_10.

[FH+17] Armando Faz-Hernández, Hayato Fujii, Diego F. Aranha, and Julio López. "A Secure and Efficient Implementation of the Quotient Digital Signature Algorithm (qDSA)". In: *Security, Privacy, and Applied Cryptography Engineering*. Ed. by Sk Subidh Ali, Jean-Luc Danger, and Thomas Eisenbarth. Cham: Springer International Publishing, 2017, pp. 170–189. DOI: 10.1007/978-3-319-71501-8_10.

[FH17] Reza Rezaeian Farashahi and Seyed Gholamhossein Hosseini. "Differential Addition on Twisted Edwards Curves". In: *Information Security and Privacy*. Ed. by Josef Pieprzyk and Suriadi Suriadi. Cham: Springer International Publishing, 2017, pp. 366–378. DOI: 10.1007/978-3-319-59870-3_21.

[FHL15] Armando Faz-Hernández and Julio López. "Fast Implementation of Curve25519 Using AVX2". In: *Progress in Cryptology – LATINCRYPT 2015*. Ed. by Kristin Lauter and Francisco Rodríguez-Henríquez. Cham: Springer International Publishing, 2015, pp. 329–345. DOI: 10.1007/978-3-319-22174-8_18.

[FMW12] Reza Rezaeian Farashahi, Dustin Moody, and Hongfeng Wu. "Isomorphism classes of Edwards curves over finite fields". In: *Finite Fields and Their Applications* 18.3 (2012), pp. 597 –612. DOI: 10.1016/j.ffa.2011.12.004.

[FO99] Eiichiro Fujisaki and Tatsuaki Okamoto. "Secure Integration of Asymmetric and Symmetric Encryption Schemes". In: *Advances in Cryptology — CRYPTO' 99*. Ed. by Michael Wiener. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 537–554. DOI: 10.1007/s00145-011-9114-1.

[Fre+13] Eduarda S. V. Freire, Dennis Hofheinz, Eike Kiltz, and Kenneth G. Paterson. "Non-Interactive Key Exchange". In: *Public-Key Cryptography – PKC 2013*. Ed. by Kaoru Kurosawa and Goichiro Hanaoka. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 254–271. DOI: 10.1007/978-3-642-36362-7_17.

[FS09] Reza Rezaeian Farashahi and Igor E. Shparlinski. "On the number of distinct elliptic curves in some families". In: *Designs, Codes and Cryptography* 54.1 (2009), p. 83. DOI: 10.1007/s10623-009-9310-2.

[FS87] Amos Fiat and Adi Shamir. "How To Prove Yourself: Practical Solutions to Identification and Signature Problems". In: *Advances in Cryptology — CRYPTO' 86*. Ed. by Andrew M. Odlyzko. Berlin, Heidelberg: Springer Berlin Heidelberg, 1987, pp. 186–194. DOI: 10.1007/3-540-47721-7_12.

[FV12] Junfeng Fan and Ingrid Verbauwhede. "An Updated Survey on Secure ECC Implementations: Attacks, Countermeasures and Cost". In: *Cryptography and Security: From Theory to Applications: Essays Dedicated to Jean-Jacques Quisquater on the Occasion of His 65th Birthday*. Ed. by David Naccache. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 265–282. DOI: 10.1007/978-3-642-28368-0_18.

[Gal+16] Steven D. Galbraith, Christophe Petit, Barak Shani, and Yan Bo Ti. "On the Security of Supersingular Isogeny Cryptosystems". In: *Advances in Cryptology – ASIACRYPT 2016*. Ed. by Jung Hee Cheon and Tsuyoshi Takagi. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 63–91. DOI: 10.1007/978-3-662-53887-6_3.

[Gal12] Steven D. Galbraith. *Mathematics of Public Key Cryptography*. Cambridge University Press, 2012. DOI: 10.1017/CBO9781139012843.

[Gal99] Steven D. Galbraith. "Constructing Isogenies between Elliptic Curves Over Finite Fields". In: *LMS Journal of Computation and Mathematics* 2 (1999), pp. 118–138. DOI: 10.1112/S1461157000000097.

[Gau+07] Pierrick Gaudry, Emmanuel Thomé, Nicolas Thériault, and Claus Diem. "A Double Large Prime Variation for Small Genus Hyperelliptic Index Calculus". In: *Mathematics of Computation* 76.257 (2007), pp. 475–492. DOI: 10.1090/S0025-5718-06-01900-4.

[Gau06] Pierrick Gaudry. *Variants of the montgomery form based on theta functions*. http://www.fields.utoronto.ca/audio/06-07/number_theory/gaudry/. 2006.

[Gau07] Pierrick Gaudry. "Fast genus 2 arithmetic based on Theta functions". In: *Journal of Mathematical Cryptology* 1.3 (2007), pp. 243–265. DOI: 10.1515/JMC.2007.012.

[GCT17] Gabriel Gallin, Turku Ozlum Celik, and Arnaud Tisserand. "Architecture Level Optimizations for Kummer Based HECC on FPGAs". In: *Progress in Cryptology – INDOCRYPT 2017*. Ed. by Arpita Patra and Nigel P. Smart. Cham: Springer International Publishing, 2017, pp. 44–64. DOI: 10.1007/978-3-319-71667-1_3.

[GHS02] Steven D. Galbraith, Keith Harrison, and David Soldera. "Implementing the Tate Pairing". In: *Proceedings of the 5th International Symposium on Algorithmic Number Theory*. ANTS-V. Berlin, Heidelberg: Springer-Verlag, 2002, pp. 324–337. DOI: 10.1007/3-540-45455-1_26.

[GL09] Pierrick Gaudry and David Lubicz. "The arithmetic of characteristic 2 Kummer surfaces and of elliptic Kummer lines". In: *Finite Fields and Their Applications* 15.2 (2009), pp. 246–260. DOI: 10.1016/j.ffa.2008.12.006.

[GLV01] Robert P. Gallant, Robert J. Lambert, and Scott A. Vanstone. "Faster Point Multiplication on Elliptic Curves with Efficient Endomorphisms". In: *Advances in Cryptology — CRYPTO 2001*. Ed. by Joe Kilian. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 190–200. DOI: 10.1007/3-540-44647-8_11.

[Gov01] Government Committee of Russia for Standards. *Information technology. Cryptographic data security. Signature and verification processes of [electronic] digital signature*. See https://tools.ietf.org/html/rfc5832. 2001.

[GP08] Tim Güneysu and Christof Paar. "Ultra High Performance ECC over NIST Primes on Commercial FPGAs". In: *Cryptographic Hardware and Embedded Systems – CHES 2008*. Ed. by Elisabeth Oswald and Pankaj

Rohatgi. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 62–78. DOI: 10.1007/978-3-540-85053-3_5.

[GPS17] Steven D. Galbraith, Christophe Petit, and Javier Silva. "Identification Protocols and Signature Schemes Based on Supersingular Isogeny Problems". In: *Advances in Cryptology – ASIACRYPT 2017*. Ed. by Tsuyoshi Takagi and Thomas Peyrin. Cham: Springer International Publishing, 2017, pp. 3–33. DOI: 10.1007/978-3-319-70694-8_1.

[Gro96] Lov K. Grover. "A Fast Quantum Mechanical Algorithm for Database Search". In: *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing*. New York, NY, USA: ACM, 1996, pp. 212–219. DOI: 10.1145/237814.237866.

[GS12] Pierrick Gaudry and Éric Schost. "Genus 2 Point Counting over Prime Fields". In: *Journal of Symbolic Computing* 47.4 (2012), pp. 368–400. DOI: 10.1016/j.jsc.2011.09.003.

[GT17] G. Gallin and A. Tisserand. "Hyper-threaded multiplier for HECC". In: *51st Asilomar Conference on Signals, Systems, and Computers*. 2017, pp. 447–451. DOI: 10.1109/ACSSC.2017.8335378.

[GV18] Steven D. Galbraith and Frederik Vercauteren. "Computational problems in supersingular elliptic curve isogenies". In: *Quantum Information Processing* 17.10 (2018), p. 265. DOI: 10.1007/s11128-018-2023-6.

[Hal05] Sean Hallgren. "Fast Quantum Algorithms for Computing the Unit Group and Class Group of a Number Field". In: *Proceedings of the Thirty-seventh Annual ACM Symposium on Theory of Computing*. New York, NY, USA: ACM, 2005, pp. 468–474. DOI: 10.1145/1060590.1060660.

[Ham12] Mike Hamburg. *Fast and compact elliptic-curve cryptography*. Cryptology ePrint Archive, Report 2012/309. http://eprint.iacr.org/2012/309. 2012.

[Ham14] Mike Hamburg. *Twisting Edwards curves with isogenies*. Cryptology ePrint Archive, Report 2014/027. http://eprint.iacr.org/. 2014.

[Ham15a] Mike Hamburg. "Decaf: Eliminating Cofactors Through Point Compression". In: *Advances in Cryptology – CRYPTO 2015*. Ed. by Rosario Gennaro and Matthew Robshaw. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 705–723. DOI: 10.1007/978-3-662-47989-6_34.

[Ham15b] Mike Hamburg. *Ed448-Goldilocks, a new elliptic curve*. Cryptology ePrint Archive, Report 2015/625. http://eprint.iacr.org/. 2015.

[Ham17] Mike Hamburg. *The STROBE protocol framework*. Cryptology ePrint Archive, Report 2017/003. http://eprint.iacr.org/2017/003. 2017.

[Har77] Robin Hartshorne. *Algebraic Geometry*. Vol. 52. Springer-Verlag New York, 1977. DOI: 10.1007/978-1-4757-3849-0.

[Has36] Helmut Hasse. "Zur Theorie der abstrakten elliptischen Funktionenkörper III. Die Struktur des Meromorphismenrings. Die Riemannsche Vermutung." In: *Journal für die reine und angewandte Mathematik* 175 (1936), pp. 193–208. DOI: 10.2969/jmsj/00310045.

[HC14] Huseyin Hisil and Craig Costello. "Jacobian Coordinates on Genus 2 Curves". In: *Advances in Cryptology – ASIACRYPT 2014*. Ed. by Palash Sarkar and Tetsu Iwata. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 338–357. DOI: 10.1007/s00145-016-9227-7.

[Hes08] Florian Hess. "Pairing Lattices". In: *Pairing-Based Cryptography – Pairing 2008*. Ed. by Steven D. Galbraith and Kenneth G. Paterson. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 18–38. DOI: 10.1007/978-3-540-85538-5_2.

[HHK17] Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. "A Modular Analysis of the Fujisaki-Okamoto Transformation". In: *Theory of Cryptography*. Ed. by Yael Kalai and Leonid Reyzin. Cham: Springer International Publishing, 2017, pp. 341–371. DOI: 10.1007/978-3-319-70500-2_12.

[Hil90] David Hilbert. "Ueber die Theorie der algebraischen Formen". In: *Mathematische Annalen* 36.4 (1890), pp. 473–534. DOI: 10.1007/BF01208503.

[His+08] Huseyin Hisil, Kenneth Koon-Ho Wong, Gary Carter, and Ed Dawson. "Twisted Edwards Curves Revisited". In: *Advances in Cryptology - ASIACRYPT 2008*. Ed. by Josef Pieprzyk. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 326–343. DOI: 10.1007/978-3-540-89255-7_20.

[His10] Huseyin Hisil. "Elliptic curves, group law, and efficient computation". http://eprints.qut.edu.au/33233/. PhD thesis. Queensland University of Technology, 2010.

[HL10] Carmit Hazay and Yehuda Lindell. *Efficient Secure Two-Party Protocols*. Springer-Verlag Berlin Heidelberg, 2010. DOI: 10.1007/978-3-642-14303-8.

[HLX12] Zhi Hu, Patrick Longa, and Maozhi Xu. "Implementing the 4-dimensional GLV method on GLS elliptic curves with j-invariant 0". In: *Designs, Codes and Cryptography* 63.3 (2012), pp. 331–343. DOI: 10.1007/s10623-011-9558-1.

[HM89] James L. Hafner and Kevin S. McCurley. "A rigorous subexponential algorithm for computation of class groups". In: *Journal of the American Mathematical Society* 2.4 (1989), pp. 837–850. DOI: 10.1090/S0894-0347-1989-1002631-0.

[HMV06] Darrel Hankerson, Alfred J. Menezes, and Scott Vanstone. *Guide to elliptic curve cryptography*. Springer Science & Business Media, 2006. DOI: 10.1007/b97644.

[Hod+07] Alireza Hodjat, Lejla Batina, David Hwang, and Ingrid Verbauwhede. "HW/SW Co-design of a Hyperelliptic Curve Cryptosystem Using a Microcode Instruction Set Coprocessor". In: *Integration, the VLSI Journal - Special issue: Embedded cryptographic hardware* 40.1 (2007), pp. 45–51. DOI: 10.1016/j.vlsi.2005.12.011.

[HS13] Michael Hutter and Peter Schwabe. "NaCl on 8-Bit AVR Microcontrollers". In: *Progress in Cryptology – AFRICACRYPT 2013*. Ed. by Amr Youssef, Abderrahmane Nitaj, and Aboul Ella Hassanien. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 156–172. DOI: 10.1007/978-3-642-38553-7_9.

[HS15] Michael Hutter and Peter Schwabe. "Multiprecision multiplication on AVR revisited". In: *Journal of Cryptographic Engineering* 5.3 (2015), pp. 201–214. DOI: 10.1007/s13389-015-0093-2.

[HSV06] Florian Hess, Nigel P. Smart, and Frederik Vercauteren. "The Eta Pairing Revisited". In: *IEEE Transactions on Information Theory* 52.10 (2006), pp. 4595–4602. DOI: 10.1109/TIT.2006.881709.

[Hud05] Ronald W. H. T. Hudson. *Kummer's quartic surface*. Cambridge University Press, 1905.

[Hus04] Dale Husemöller. *Elliptic Curves*. Graduate Texts in Mathematics. Springer, 2004. DOI: 10.1007/b97292.

[ICA15] ICAO. *Doc 9303: Machine Readable Travel Documents – Part 12*. Tech. rep. 2015.

[IT02] Tetsuya Izu and Tsuyoshi Takagi. "Exceptional Procedure Attack on Elliptic Curve Cryptosystems". In: *Public Key Cryptography — PKC 2003*. Ed. by Yvo G. Desmedt. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 224–239. DOI: 10.1007/3-540-36288-6_17.

[Jao+16] David Jao, Reza Azarderakhsh, Matthew Campagna, Craig Costello, Luca De Feo, Basil Hess, Amir Jalali, Brian Koziel, Brian LaMacchia, Patrick Longa, Michael Naehrig, Joost Renes, Vladimir Soukharev, and David Urbanik. *SIKE. Supersingular Isogeny Key Encapsulation*. Submission to [Nat16]. http://sike.org. 2016.

[Jao+18] David Jao, Jason LeGrow, Christopher Leonardi, and Luis Ruiz-Lopez. *A subexponential-time, polynomial quantum space algorithm for inverting the CM group action*. MathCrypt. 2018.

[JDF11] David Jao and Luca De Feo. "Towards Quantum-Resistant Cryptosystems from Supersingular Elliptic Curve Isogenies". In: *Post-Quantum Cryptography*. Ed. by Bo-Yin Yang. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 19–34. DOI: 10.1007/978-3-642-25405-5_2.

[JMV09] David Jao, Stephen D. Miller, and Ramarathnam Venkatesan. "Expander graphs based on GRH with an application to elliptic curve cryptography". In: *Journal of Number Theory* 129.6 (2009), pp. 1491–1504. DOI: 10.1016/j.jnt.2008.11.006.

[JT09] Marc Joye and Michael Tunstall. "Exponent Recoding and Regular Exponentiation Algorithms". In: *Progress in Cryptology – AFRICACRYPT 2009*. Ed. by Bart Preneel. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 334–349. DOI: 10.1007/978-3-642-02384-2_21.

[Kal95] Burton S. Kaliski. "The Montgomery Inverse and Its Applications". In: *IEEE Transactions on Computers* 44.8 (1995), pp. 1064–1065. DOI: 10.1109/12.403725.

[KAMK16] Brian Koziel, Reza Azarderakhsh, and Mehran Mozaffari-Kermani. "Fast Hardware Architectures for Supersingular Isogeny Diffie-Hellman Key Exchange on FPGA". In: *Progress in Cryptology – INDOCRYPT 2016*. Ed. by Orr Dunkelman and Somitra Kumar Sanadhya. Cham: Springer International Publishing, 2016, pp. 191–206. DOI: 10.1007/978-3-319-49890-4_11.

[KD14] Sabyasachi Karati and Abhijit Das. "Faster Batch Verification of Standard ECDSA Signatures Using Summation Polynomials". In: *Applied Cryptography and Network Security*. Ed. by Ioana Boureanu, Philippe Owesarski, and Serge Vaudenay. Cham: Springer International Publishing, 2014, pp. 438–456. DOI: 10.1007/978-3-319-07536-5_26.

[Kel18] Julian Kelly. *Engineering superconducting qubit arrays for Quantum Supremacy*. APS March Meeting. 2018.

[Kie17] Jean Kieffer. "Étude et accélération du protocole d'échange de clés de Couveignes–Rostovtsev–Stolbunov". https://arxiv.org/abs/1804.10128. Mémoire du Master 2. Université Paris VI, 2017.

[Kir+15] Daniel Kirkwood, Bradley C. Lackey, John McVey, Mark Motley, Jerome A. Solinas, and David Tuller. *Failure is not an Option: Standardization issues for post-quantum key agreement*. Talk at NIST workshop on Cybersecurity in a Post-Quantum World: http://www.nist.gov/itl/csd/ct/post-quantum-crypto-workshop-2015.cfm. 2015.

[KJJ99] Paul Kocher, Joshua Jaffe, and Benjamin Jun. "Differential Power Analysis". In: *Advances in Cryptology — CRYPTO' 99*. Ed. by Michael Wiener. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 388–397. DOI: 10.1007/3-540-48405-1_25.

[Kob87] Neal Koblitz. "Elliptic curve cryptosystems". In: *Mathematics of Computation* 48 (1987), pp. 203–209. DOI: 10.1090/S0025-5718-1987-0866109-5.

[Kob88] Neal Koblitz. "A Family of Jacobians Suitable for Discrete Log Cryptosystems". In: *Advances in Cryptology — CRYPTO' 88*. Ed. by Shafi Goldwasser. New York, NY: Springer New York, 1988, pp. 94–99. DOI: 10.1007/0-387-34799-2_8.

[Koc96] Paul C. Kocher. "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems". In: *Advances in Cryptology — CRYPTO '96*. Ed. by Neal Koblitz. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 104–113. DOI: 10.1007/3-540-68697-5_9.

[Koh11] David Kohel. "Addition law structure of elliptic curves". In: *Journal of Number Theory* 131.5 (2011), pp. 894–919. DOI: 10.1016/j.jnt.2010.12.001.

[Koh96] David Kohel. "Endomorphism rings of elliptic curves over finite fields". http://iml.univ-mrs.fr/~kohel/pub/thesis.pdf. PhD thesis. University of California at Berkeley, 1996.

[Kop+18] Philipp Koppermann, Fabrizio De Santis, Johann Heyszl, and Georg Sigl. "Fast FPGA Implementations of Diffie-Hellman on the Kummer Surface of a Genus-2 Curve". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2018.1 (2018), pp. 1–17. DOI: 10.13154/tches.v2018.i1.1-17.

[KS17] Sabyasachi Karati and Palash Sarkar. "Kummer for Genus One over Prime Order Fields". In: *Advances in Cryptology – ASIACRYPT 2017*. Ed. by Tsuyoshi Takagi and Thomas Peyrin. Cham: Springer International Publishing, 2017, pp. 3–32. DOI: 10.1007/978-3-319-70697-9_1.

[KS19] Sabyasachi Karati and Palash Sarkar. "Connecting Legendre with Kummer and Edwards". In: *Advances in Mathematics of Communications* 13.1 (2019), pp. 41–66. DOI: 10.3934/amc.2019003.

[Kup05] Greg Kuperberg. "A Subexponential-Time Quantum Algorithm for the Dihedral Hidden Subgroup Problem". In: *SIAM Journal on Computing* 35.1 (2005), pp. 170–188. DOI: 10.1137/S0097539703436345.

[Kup13] Greg Kuperberg. "Another Subexponential-time Quantum Algorithm for the Dihedral Hidden Subgroup Problem". In: *TQC*. Vol. 22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2013, pp. 20–34. DOI: 10.4230/LIPIcs.TQC.2013.20.

[KW03] Jonathan Katz and Nan Wang. "Efficiency Improvements for Signature Schemes with Tight Security Reductions". In: *Proceedings of the 10th ACM Conference on Computer and Communications Security*. New York, NY, USA: ACM, 2003, pp. 155–164. DOI: 10.1145/948109.948132.

[Käs12] Emilia Käsper. "Fast Elliptic Curve Cryptography in OpenSSL". In: *Financial Cryptography and Data Security*. Ed. by George Danezis, Sven Dietrich, and Kazue Sako. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 27–39. DOI: 10.1007/978-3-642-29889-9_4.

[LD99] Julio López and Ricardo Dahab. "Fast Multiplication on Elliptic Curves Over $GF(2^m)$ without precomputation". In: *Cryptographic Hardware and Embedded Systems*. Ed. by Çetin K. Koç and Christof Paar. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 316–327. DOI: 10.1007/3-540-48059-5_27.

[Len87] Hendrik W. Lenstra. "Factoring Integers with Elliptic Curves". In: *The Annals of Mathematics* 126 (1987), pp. 649–673. DOI: 10.2307/1971363.

[LG10] Patrick Longa and Catherine Gebotys. "Efficient Techniques for High-Speed Elliptic Curve Cryptography". In: *Cryptographic Hardware and Embedded Systems, CHES 2010*. Ed. by Stefan Mangard and François-Xavier Standaert. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 80–94. DOI: 10.1007/978-3-642-15031-9_6.

[LHT16] Adam Langley, Mike Hamburg, and Sean Turner. *Elliptic Curves for Security*. RFC 7748. RFC Editor, 2016, pp. 1–22.

[Lic69] Stephen Lichtenbaum. "Duality theorems for curves over *P*-adic fields". In: *Inventiones Mathematicae* 7 (1969), pp. 120–136. DOI: 10.1007/BF01389795.

[Liu+13] Zhe Liu, Hwajeong Seo, Johann Großschädl, and Howon Kim. "Efficient Implementation of NIST-Compliant Elliptic Curve Cryptography for Sensor Nodes". In: *Information and Communications Security*. Ed. by Sihan Qing, Jianying Zhou, and Dongmei Liu. Cham: Springer International Publishing, 2013, pp. 302–317. DOI: 10.1007/978-3-319-02726-5_22.

[Liu+17] Zhe Liu, Patrick Longa, Geovandro C. C. F. Pereira, Oscar Reparaz, and Hwajeong Seo. "FourQ on Embedded Devices with Strong Countermeasures Against Side-Channel Attacks". In: *Cryptographic Hardware and Embedded Systems – CHES 2017*. Ed. by Wieland Fischer and Naofumi Homma. Cham: Springer International Publishing, 2017, pp. 665–686. DOI: 10.1007/978-3-319-66787-4_32.

[LL94] Chae Hoon Lim and Pil Joong Lee. "More Flexible Exponentiation with Precomputation". In: *Advances in Cryptology — CRYPTO '94*. Ed. by Yvo G. Desmedt. Berlin, Heidelberg: Springer Berlin Heidelberg, 1994, pp. 95–107. DOI: 10.1007/3-540-48658-5_11.

[LLL82] Hendrik W. Lenstra, Jr., Arjen K. Lenstra, and László Lovász. "Factoring Polynomials with Rational Coefficients". In: *Mathematische Annalen* 261 (1982), pp. 515–534. DOI: 10.1007/BF01457454.

[LR85] Herbert Lange and Wolfgang Ruppert. "Complete systems of addition laws on abelian varieties". In: *Inventiones mathematicae* 79.3 (1985), pp. 603–610. DOI: 10.1007/BF01388526.

[LV00] Arjen K. Lenstra and Eric R. Verheul. "The XTR Public Key System". In: *Advances in Cryptology — CRYPTO 2000*. Ed. by Mihir Bellare. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 1–19. DOI: 10.1007/3-540-44598-6_1.

[LWG14] Zhe Liu, Erich Wenger, and Johann Großschädl. "MoTE-ECC: Energy-Scalable Elliptic Curve Cryptography for Wireless Sensor Networks". In: *Applied Cryptography and Network Security*. Ed. by Ioana Boureanu, Philippe Owesarski, and Serge Vaudenay. Cham: Springer International Publishing, 2014, pp. 361–379. DOI: 10.1007/978-3-319-07536-5_22.

[McE78] Robert J. McEliece. "A Public-Key Cryptosystem Based On Algebraic Coding Theory". In: *The Deep Space Network Progress Report* 42.44 (1978), pp. 114–116.

[MCR18] Michael Meyer, Fabio Campos, and Steffen Reith. *On Lions and Elligators: An efficient constant-time implementation of CSIDH*. Cryptology ePrint Archive, Report 2018/1198. https://eprint.iacr.org/2018/1198. 2018.

[Mil04] Victor S. Miller. "The Weil Pairing, and Its Efficient Calculation". In: *Journal of Cryptology* 17.4 (2004), pp. 235–261. DOI: 10.1007/s00145-004-0315-8.

[Mil86] Victor S. Miller. "Use of Elliptic Curves in Cryptography". In: *Advances in Cryptology — CRYPTO '85 Proceedings*. Ed. by Hugh C. Williams. Berlin, Heidelberg: Springer Berlin Heidelberg, 1986, pp. 417–426. DOI: 10.1007/3-540-39799-X_31.

[Mon85] Peter L. Montgomery. "Modular Multiplication without Trial Division". In: *Mathematics of Computation* 44.170 (1985), pp. 519–521. DOI: 10.1090/S0025-5718-1985-0777282-X.

[Mon87] Peter L. Montgomery. "Speeding the Pollard and elliptic curve methods of factorization". In: *Mathematics of Computation* 48.177 (1987), pp. 243–264. DOI: 10.1090/S0025-5718-1987-0866113-7.

[Mor61] Louis J. Mordell. "The congruence $(p-1/2)! \equiv \pm 1 \bmod p$". In: *American Mathematical Monthly* 68.2 (1961), pp. 145–146. DOI: 10.2307/2312481.

[MOV91] Alfred Menezes, Tatsuaki Okamoto, and Scott Vanstone. "Reducing Elliptic Curve Logarithms to Logarithms in a Finite Field". In: *Proceedings of the Twenty-third Annual ACM Symposium on Theory of Computing*. New York, NY, USA: ACM, 1991, pp. 80–89. DOI: 10.1145/103418.103434.

[MR18] Michael Meyer and Steffen Reith. "A Faster Way to the CSIDH". In: *Progress in Cryptology – INDOCRYPT 2018*. Ed. by Debrup Chakraborty and Tetsu Iwata. Cham: Springer International Publishing, 2018, pp. 137–152. DOI: 10.1007/978-3-030-05378-9_8.

[MRa99] David M'Raïhi, David Naccache, David Pointcheval, and Serge Vaudenay. "Computational Alternatives to Random Number Generators". In: *Selected Areas in Cryptography*. Ed. by Stafford Tavares and Henk Meijer. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 72–80. DOI: 10.1007/3-540-48892-8_6.

[MS16] Dustin Moody and Daniel Shumow. "Analogues of Velu's Formulas for Isogenies on Alternate Models of Elliptic Curves". In: *Mathematics of Computation* 85.300 (2016), pp. 1929–1951. DOI: 10.1090/mcom/3036.

[Mum93] David Mumford. *Tata lectures on theta II*. Birkhäuser Boston, 1993. DOI: 10.1007/978-0-8176-4578-6.

[MW99] Ueli M. Maurer and Stefan Wolf. "The Relationship Between Breaking the Diffie–Hellman Protocol and Computing Discrete Logarithms". In: *SIAM Journal on Computing* 28.5 (1999), pp. 1689–1721. DOI: 10.1137/S0097539796302749.

[Nac+95] David Naccache, David M'Raïhi, Serge Vaudenay, and Dan Raphaeli. "Can D.S.A. be improved? — Complexity trade-offs with the digital signature standard —". In: *Advances in Cryptology — EUROCRYPT'94*. Ed. by Alfredo De Santis. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, pp. 77–85. DOI: 10.1007/BFb0053426.

[Nat00] National Institute for Standards and Technology (NIST). *Digital Signature Standard.* Federal Information Processing Standards Publication 186-2. http://csrc.nist.gov/publications/fips/archive/fips186-2/fips186-2.pdf. 2000.

[Nat13] National Institute for Standards and Technology (NIST). *Digital Signature Standard.* Federal Information Processing Standards Publication 186-4. http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf. 2013.

[Nat15] The National Security Agency. *Suite B Cryptography (fact sheet)*. https://www.nsa.gov/ia/programs/suiteb_cryptography/. 2015.

[Nat16] National Institute of Standards and Technology. *Post-Quantum Cryptography Standardization*. https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Post-Quantum-Cryptography-Standardization. 2016.

[NLD15] Erick Nascimento, Julio López, and Ricardo Dahab. "Efficient and Secure Elliptic Curve Cryptography for 8-bit AVR Microcontrollers". In: *Security, Privacy, and Applied Cryptography Engineering*. Ed. by Rajat Subhra Chakraborty, Peter Schwabe, and Jon Solworth. Cham: Springer International Publishing, 2015, pp. 289–309. DOI: 10.1007/978-3-319-24126-5_17.

[NV10] Phong Q. Nguyen and Brigitte Vallée, eds. *The LLL Algorithm. Survey and Applications*. Springer, 2010. DOI: 10.1007/978-3-642-02295-1.

[OS01] Katsuyuki Okeya and Kouichi Sakurai. "Efficient Elliptic Curve Cryptosystems from a Scalar Multiplication Algorithm with Recovery of the y-Coordinate on a Montgomery-Form Elliptic Curve". In: *Cryptographic Hardware and Embedded Systems — CHES 2001*. Ed. by Çetin K. Koç, David Naccache, and Christof Paar. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 126–141. DOI: 10.1007/3-540-44709-1_12.

[OT03] Katsuyuki Okeya and Tsuyoshi Takagi. "The Width-w NAF Method Provides Small Memory and Fast Elliptic Scalar Multiplications Secure against Side Channel Attacks". In: *Topics in Cryptology — CT-RSA 2003*. Ed. by Marc Joye. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 328–343. DOI: 10.1007/3-540-36563-X_23.

[OW99] Paul C. van Oorschot and Michael J. Wiener. "Parallel Collision Search with Cryptanalytic Applications". In: *Journal of Cryptology* 12.1 (1999), pp. 1–28. DOI: 10.1007/PL00003816.

[Pei14] Chris Peikert. "Lattice Cryptography for the Internet". In: *Post-Quantum Cryptography*. Ed. by Michele Mosca. Cham: Springer International Publishing, 2014, pp. 197–219. DOI: 10.1007/978-3-319-11659-4_12.

[Per] Thomas Perrin. *The XEdDSA and VXEdDSA Signature Schemes*. https://whispersystems.org/docs/specifications/xeddsa/.

[Pet17] Christophe Petit. "Faster Algorithms for Isogeny Problems Using Torsion Point Images". In: *Advances in Cryptology – ASIACRYPT 2017*. Ed. by Tsuyoshi Takagi and Thomas Peyrin. Cham: Springer International Publishing, 2017, pp. 330–353. DOI: 10.1007/978-3-319-70697-9_12.

[PH78] Stephen C. Pohlig and Martin E. Hellman. "An improved algorithm for computing logarithms over $GF(p)$ and its cryptographic significance". In: *IEEE Transactions on Information Theory* 24.1 (1978), pp. 106–110. DOI: 10.1109/TIT.1978.1055817.

[Piz90] Arnold K. Pizer. "Ramanujan graphs and Hecke operators". In: *Bulletin of the American Mathematical Society* 23.1 (1990), pp. 127–137. DOI: 10.1090/S0273-0979-1990-15918-X.

[Pol75] John M. Pollard. "A monte carlo method for factorization". In: *BIT Numerical Mathematics* 15.3 (1975), pp. 331–334. DOI: 10.1007/BF01933667.

[PS00] David Pointcheval and Jacques Stern. "Security Arguments for Digital Signatures and Blind Signatures". In: *Journal of Cryptology* 13.3 (2000), pp. 361–396. DOI: 10.1007/s001450010003.

[PS96] David Pointcheval and Jacques Stern. "Security Proofs for Signature Schemes". In: *Advances in Cryptology — EUROCRYPT '96*. Ed. by Ueli Maurer. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 387–398. DOI: 10.1007/3-540-68339-9_33.

[Reg04] Oded Regev. "A Subexponential Time Algorithm for the Dihedral Hidden Subgroup Problem with Polynomial Space". In: *arXiv e-prints* (2004). https://arxiv.org/abs/quant-ph/0406151.

[Reg05] Oded Regev. "On Lattices, Learning with Errors, Random Linear Codes, and Cryptography". In: *Proceedings of the Thirty-seventh Annual ACM Symposium on Theory of Computing*. New York, NY, USA: ACM, 2005, pp. 84–93. DOI: 10.1145/1060590.1060603.

[Res18] Eric Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3*. RFC 8446. RFC Editor, 2018, pp. 1–160.

[Roe+17] Martin Roetteler, Michael Naehrig, Krysta M. Svore, and Kristin Lauter. "Quantum Resource Estimates for Computing Elliptic Curve Discrete Logarithms". In: *Advances in Cryptology – ASIACRYPT 2017*. Ed. by Tsuyoshi Takagi and Thomas Peyrin. Cham: Springer International Publishing, 2017, pp. 241–270. DOI: 10.1007/978-3-319-70697-9_9.

[RS03] Karl Rubin and Alice Silverberg. "Torus-Based Cryptography". In: *Advances in Cryptology - CRYPTO 2003*. Ed. by Dan Boneh. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 349–365. DOI: 10.1007/978-3-540-45146-4_21.

[RS06] Alexander Rostovtsev and Anton Stolbunov. *Public-key Cryptosystem Based on Isogenies*. Cryptology ePrint Archive, Report 2006/145. https://eprint.iacr.org/2006/145. 2006.

[RSA78] Ron L. Rivest, Adi Shamir, and Leonard Adleman. "A Method for Obtaining Digital Signatures and Public-key Cryptosystems". In: *Communications of the ACM* 21.2 (1978), pp. 120–126. DOI: 10.1145/359340.359342.

[Sag18] The Sage Developers. *SageMath, the Sage Mathematics Software System*. https://sagemath.org. 2018.

[Sch87] René Schoof. "Nonsingular Plane Cubic Curves over Finite Fields". In: *Journal of Combinatorial Theory, Series A* 46.2 (1987), pp. 183–211. DOI: 10.1016/0097-3165(87)90003-3.

[Sch90] Claus P. Schnorr. "Efficient Identification and Signatures for Smart Cards". In: *Advances in Cryptology — CRYPTO' 89 Proceedings*. Ed. by Gilles Brassard. New York, NY: Springer New York, 1990, pp. 239–252. DOI: 10.1007/0-387-34805-0_22.

[Sco07] Michael Scott. "Implementing Cryptographic Pairings". In: *Proceedings of the First International Conference on Pairing-Based Cryptography*. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 177–196.

[Sem04] Igor Semaev. *Summation polynomials and the discrete logarithm problem on elliptic curves*. Cryptology ePrint Archive, Report 2004/031. https://eprint.iacr.org/2004/031. 2004.

[Sha71] Daniel Shanks. "Class number, a theory of factorization, and genera". In: *Proceedings of Symposia in Pure Mathematics*. Vol. 20. 1971, pp. 415–440.

[Sho94] Peter W. Shor. "Algorithms for quantum computation: Discrete logarithms and factoring". In: *Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on*. IEEE. 1994, pp. 124–134. DOI: 10.1109/SFCS.1994.365700.

[Sho97] Peter W. Shor. "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer". In: *SIAM Journal on Computing* 26.5 (1997), pp. 1484–1509.

[Sie35] Carl Siegel. "Über die Classenzahl quadratischer Zahlkörper". In: *Acta Arithmetica* 1.1 (1935), pp. 83–86.

[Sig] *Signal Protocol Specification*. `https://signal.org/docs/`.

[Sil09] Joseph H. Silverman. *The Arithmetic of Elliptic Curves, 2nd Edition*. Graduate Texts in Mathematics. Springer, 2009. DOI: `10.1007/978-0-387-09494-6`.

[SL03] Martijn Stam and Arjen K. Lenstra. "Efficient Subgroup Exponentiation in Quadratic and Sixth Degree Extensions". In: *Cryptographic Hardware and Embedded Systems - CHES 2002*. Ed. by Burton S. Kaliski, çetin K. Koç, and Christof Paar. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 318–332. DOI: `10.1007/3-540-36400-5_24`.

[SS04] Edward Schaefer and Michael Stoll. "How to do a p-descent on an elliptic curve". In: *Transactions of the American Mathematical Society* 356.3 (2004), pp. 1209–1231.

[SS95] Peter Smith and Christopher Skinner. "A public-key cryptosystem and a digital signature system based on the Lucas function analogue to discrete logarithms". In: *Advances in Cryptology — ASIACRYPT'94*. Ed. by Josef Pieprzyk and Reihanah Safavi-Naini. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, pp. 355–364. DOI: `10.1007/BFb0000447`.

[Sta03] Martijn Stam. "Speeding up subgroup cryptosystems". `http://alexandria.tue.nl/extra2/200311829.pdf?q=subgroup`. PhD thesis. Technische Universiteit Eindhoven, 2003.

[Sta04] Colin Stahlke. *Point Compression on Jacobians of Hyperelliptic Curves over* $\mathbb{F}_q$. Cryptology ePrint Archive, Report 2004/030. `https://eprint.iacr.org/2004/030`. 2004.

[Sto04] Anton Stolbunov. "Public-key encryption based on cycles of isogenous elliptic curves". MA thesis. Saint-Petersburg State Polytechnical University, 2004. DOI: `10.13140/RG.2.2.29215.05282`.

[Sto10] Anton Stolbunov. "Constructing public-key cryptographic schemes based on class group action on a set of isogenous elliptic curves". In: *Advances in Mathematics of Communications* 4.2 (2010), pp. 215–235. DOI: `10.3934/amc.2010.4.215`.

[Sto11] Anton Stolbunov. "Cryptographic Schemes Based on Isogenies". PhD thesis. Norwegian University of Science and Technology, 2011. DOI: 10.13140/RG.2.2.20826.44488.

[Sut07] Andrew V. Sutherland. "Order computations in generic groups". https://groups.csail.mit.edu/cis/theses/sutherland-phd.pdf. PhD thesis. Massachusetts Institute of Technology, 2007.

[Sut11] Andrew V. Sutherland. "Structure computation and discrete logarithms in finite abelian p-groups". In: *Mathematics of Compututation* 80.273 (2011), pp. 477–500. DOI: 10.1090/S0025-5718-10-02356-2.

[Sut12a] Andrew V. Sutherland. "Identifying supersingular elliptic curves". In: *LMS Journal of Computation and Mathematics* 15 (2012), pp. 317–325. DOI: 10.1112/S1461157012001106.

[Sut12b] Andrew V. Sutherland. "Isogeny volcanoes". In: *ANTS X*. Vol. 1. The Open Book Series. Mathematical Sciences Publishers, 2012, pp. 507–530. DOI: 10.2140/obs.2013.1.507.

[Szc+08] Piotr Szczechowiak, Leonardo B. Oliveira, Michael Scott, Martin Collier, and Ricardo Dahab. "NanoECC: Testing the Limits of Elliptic Curve Cryptography in Sensor Networks". In: *Wireless Sensor Networks*. Ed. by Roberto Verdone. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 305–320. DOI: 10.1007/978-3-540-77690-1_19.

[Tat66] John Tate. "Endomorphisms of abelian varieties over finite fields". In: *Inventiones mathematicae* 2.2 (1966), pp. 134–144. DOI: 10.1007/BF01404549.

[Tib14] Mehdi Tibouchi. "Elligator Squared: Uniform Points on Elliptic Curves of Prime Order as Uniform Random Strings". In: *Financial Cryptography and Data Security*. Ed. by Nicolas Christin and Reihaneh Safavi-Naini. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 139–156. DOI: 10.1007/978-3-662-45472-5_10.

[TTA18] Akira Takahashi, Mehdi Tibouchi, and Masayuki Abe. "New Bleichenbacher Records: Fault Attacks on qDSA Signatures". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2018.3 (2018), pp. 331–371. DOI: 10.13154/tches.v2018.i3.331-371.

[Unr12] Dominique Unruh. "Quantum Proofs of Knowledge". In: *Advances in Cryptology – EUROCRYPT 2012*. Ed. by David Pointcheval and Thomas Johansson. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 135–152. DOI: 10.1007/978-3-642-29011-4_10.

[Ver04] Eric R. Verheul. "Evidence that XTR Is More Secure than Supersingular Elliptic Curve Cryptosystems". In: *Journal of Cryptology* 17.4 (2004), pp. 277–296. DOI: 10.1007/s00145-004-0313-x.

[Vél71] Jacques Vélu. "Isogénies entre courbes elliptiques". In: *Comptes Rendus de l'Académie des Sciences des Paris* 273 (1971), pp. 238–241.

[Wat69] William C. Waterhouse. "Abelian varieties over finite fields". In: *Annales scientifiques de l'École Normale Supérieure* 2.4 (1969), pp. 521–560.

[Wha17] *WhatsApp Encryption Overview*. Tech. rep. https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf. 2017.

[WUW13] Erich Wenger, Thomas Unterluggauer, and Mario Werner. "8/16/32 Shades of Elliptic Curve Cryptography on Embedded Processors". In: *Progress in Cryptology – INDOCRYPT 2013*. Ed. by Goutam Paul and Serge Vaudenay. Cham: Springer International Publishing, 2013, pp. 244–261. DOI: 10.1007/978-3-319-03515-4_16.

[Yoo+17] Youngho Yoo, Reza Azarderakhsh, Amir Jalali, David Jao, and Vladimir Soukharev. "A Post-quantum Digital Signature Scheme Based on Supersingular Isogenies". In: *Financial Cryptography and Data Security*. Ed. by Aggelos Kiayias. Cham: Springer International Publishing, 2017, pp. 163–181. DOI: 10.1007/978-3-319-70972-7_9.

[Zan+18] Gustavo H. M. Zanon, Marcos A. Simplicio, Geovandro C. C. F. Pereira, Javad Doliskani, and Paulo S. L. M. Barreto. "Faster Isogeny-Based Compressed Key Agreement". In: *Post-Quantum Cryptography*. Ed. by Tanja Lange and Rainer Steinwandt. Cham: Springer International Publishing, 2018, pp. 248–268. DOI: 10.1007/978-3-319-79063-3_12.

Summary

This thesis works towards simple, secure and efficient curve-based primitives. After a review of the necessary background material in Part 1, the focus of Part 2 is on protocols secure against *classical* adversaries, i. e. those based on the discrete logarithm problem. In Part 3 we look towards cryptographic schemes based on isogeny problems that aim to be secure against *quantum* adversaries. This work is organized as a sequence of papers, whose content we briefly describe here.

Chapter III. This chapter considers the arithmetic of elliptic curves present in many standards. That is, curves in (short) Weierstrass form defined over a prime field $\mathbb{F}_p$ whose group of rational points over $\mathbb{F}_p$ has prime order. Such curves were known to have complete addition formulas (i. e. formulas that work on all pairs of points as input), but they incurred a tremendous slowdown compared to the incomplete formulas. In this chapter we significantly improve these formulas. Although a minor loss of efficiency remains (of about 30–40% in software depending on parameters), the implementation naturally simplifies and should give users more confidence in their security.

Chapter IV & V. The next two chapters consider the practicality of schemes based on the efficient arithmetic on Kummer varieties. In Chapter IV we present implementations of the Diffie–Hellman key exchange and Schnorr signature scheme based on a Kummer surface of a genus-2 hyperelliptic curve, outperforming all other existing schemes and demonstrating its applicability for low-resource devices. An arguably more elegant approach slightly modifies the signature scheme itself to be naturally instantiated with a Kummer variety. This leads to the qDSA signature scheme. On top of a theoretical security analysis of the scheme and highly efficient implementations, we also show how to efficiently instantiate the necessary operations (i. e. signature verification and public-key compression) with genus-2 Kummer surfaces.

Chapter VI. The final chapter of Part 2 studies the relations of the various Kummer lines that have appeared in the literature. In the presence of full rational 2-torsion, we provide explicit maps between Montgomery curves, (twisted) Edwards models and (squared) Kummer lines. We present an easy framework for moving between the different models with isomorphisms. This improves interoperability of the models and simplifies the task of an implementer. In particular, this allows for a straightforward generalization of the qDSA signature scheme to the squared Kummer line.

Chapter VII. The main advantage of SIDH compared to alternative post-quantum schemes is the relatively small size of its public keys. This chapter shows how to achieve even smaller public keys. We provide techniques for efficiently sampling torsion bases on a curve, significantly improve the pairing computations and obtain extremely efficient discrete logarithms in smooth cyclic groups. Moreover, we show how to compress the keys even further at essentially no cost.

Chapter VIII. The speed of supersingular-isogeny Diffie–Hellman is for a large part determined by the efficiency of the arithmetic of the elliptic-curve model and its isogeny formulas. A particularly popular form is the Montgomery form, which is used in the currently most optimal implementations of SIDH. This chapter studies the isogeny formulas between elliptic curves in Montgomery form, expanding on and simplifying the work of [CH17]. That is, we show that the isogeny formulas generalize to any group not containing the point $(0,0)$ (and in particular 2-isogenies) and provide simplifications to the proofs. We also include potential new models that could lead to elegant isogeny formulas, though they do not lead to faster implementations of SIDH as of yet.

Chapter IX. The last chapter proposes a new cryptographic primitive for key exchange, which is strongly related to the work of Couveignes [Cou06] and Rostovtsev and Stolbunov [RS06]. We replace the class group action arising from the endomorphism ring of an ordinary elliptic curve by a class group action related to the $\mathbb{F}_p$-rational endomorphism ring of a supersingular elliptic curve. The main upside is the fact that the group of rational points over $\mathbb{F}_p$ has exactly size $p+1$, allowing to easily select primes such that the curves have extremely smooth group orders. This leads to significantly faster evaluation of isogenies, while retaining the extremely small public keys. Moreover, we show how to efficiently validate public keys. As a result, the key exchange supports static public keys and is considered non-interactive.

Samenvatting (Dutch summary)

Dit proefschrift draagt bij aan de ontwikkeling van eenvoudige, veilige en efficiënte primitieven gebaseerd op algebraïsche krommen. Na in Deel 1 een overzicht te hebben gegeven van de benodigde achtergrondkennis, richt Deel 2 zich op protocollen die bestand zijn tegen *klassieke* aanvallers, i. e. die zijn gebaseerd op het discrete logaritme probleem. In Deel 3 kijken we naar cryptografische ontwerpen gebaseerd op isogenieproblemen met veiligheid ten opzichte van *kwantum*aanvallers als doel. Dit proefschrift is een verzameling van reeds gepubliceerde artikelen, die we nu in het kort beschrijven.

Hoofdstuk III. Dit hoofdstuk houdt zich bezig met de aritmetiek van elliptische krommen die in veel standaarden voor komt. Met andere woorden, krommen beschreven door een Weierstrass vorm gedefinieerd over een priemlichaam $\mathbb{F}_p$ waarvan de orde van de groep van rationale punten over $\mathbb{F}_p$ priem is. Het was bekend dat voor zulke krommen complete formules voor de optelling bestaan, maar alleen ten koste van een enorme vertraging ten opzichte van de incomplete varianten. In dit hoofdstuk verbeteren we deze formules significant. Hoewel een klein verlies van efficiëntie niet te voorkomen is (van ongeveer 30–40% in software afhankelijk van parameters), leidt ons resultaat tot natuurlijke vereenvoudigingen die het vertrouwen in de veiligheid verhogen.

Hoofdstuk IV & V. De volgende twee hoofdstukken richten zich op de uitvoerbaarheid van digitale handtekeningen gebaseerd op de efficiënte aritmetiek van Kummervariëteiten. In Hoofdstuk IV presenteren we implementaties van de sleuteluitwisseling van Diffie en Hellman en het protocol voor digitale handtekeningen van Schnorr gebaseerd op een Kummervariëteit van een hyperelliptische kromme van genus 2, die alle bestaande werken overtreft en de toepasbaarheid op kleine apparaten demonstreert. Een wellicht elegantere aanpak past het protocol voor digitale handtekeningen licht aan om zich gemakkelijker te lenen voor een Kummervariëteit, wat leidt tot het qDSA protocol. Naast

een theoretische analyse van de veiligheid en zeer efficiënte implementaties, laten we ook zien hoe de benodigde operaties (i. e. verificatie van een handtekening en compressie van een publieke sleutel) kunnen worden geïnstantieerd met een genus-2 Kummervariëteit.

Hoofdstuk VI. Het laatste hoofdstuk van Deel 2 bestudeert de relaties tussen de verschillende Kummerlijnen die in de literatuur voor komen. Onder de aanname dat de 2-torsie punten volledig rationaal zijn, verstrekken we expliciete afbeeldingen tussen krommen in Montgomery vorm, (getwiste) Edwards vorm en (gekwadrateerde) Kummerlijnen. We presenteren gemakkelijk te beschrijven isomorfismen tussen de verschillende modellen. Dit bevordert de interoperabiliteit en versimpelt de taken van ontwikkelaars. In het bijzonder kunnen we eenvoudig de theorie van qDSA generaliseren naar het domein van gekwadrateerde Kummerlijnen.

Hoofdstuk VII. Het grote voordeel van SIDH ten opzichte van alternatieve postkwantum ontwerpen is dat de publieke sleutels relatief klein zijn. Dit hoofdstuk laat zien hoe deze sleutels nog kleiner gemaakt kunnen worden. Hiervoor presenteren we technieken om efficiënt torsie-basissen te genereren, verbeteren we de berekeningen van de paringen en verschaffen we extreem efficiënte discrete logaritmen in gladde cyclische groepen. Bovendien laten we zien hoe de publieke sleutels vrijwel kosteloos nog verder gecomprimeerd kunnen worden.

Hoofdstuk VIII. De snelheid van SIDH is voor een groot deel bepaald door de efficiëntie van de aritmetiek van het model van de elliptische kromme en de bijbehorende formules voor isogenieën. Een bijzonder populaire vorm is het Montgomery model, welk momenteel gebruikt wordt in de meest optimale implementaties van SIDH. Dit hoofdstuk bestudeert de formules voor isogenieën tussen elliptische krommen in Montgomery vorm, waarmee we het werk van [CH17] uitbreiden en vereenvoudigen. We laten zien dat de formules generaliseren naar willekeurige groepen die het punt $(0, 0)$ niet bevatten (en in het bijzonder 2-isogenieën) en maken de bewijzen gemakkelijker. We stellen ook alternatieve modellen voor die tot elegante formules voor isogenieën kunnen leiden, hoewel dat tot op heden nog niet tot snellere implementaties van SIDH heeft geleid.

Hoofdstuk IX. Het laatste hoofdstuk stelt een nieuwe primitieve voor sleuteluitwisseling voor, die sterk gerelateerd is aan het werk van Couveignes [Cou06]

en Rostovtsev en Stolbunov [RS06]. We vervangen de actie van de klassengroep die ontstaat vanuit de endomorfisme ring van een ordinaire elliptische kromme door de klassengroepactie gerelateerd aan de $\mathbb{F}_p$-rationale endomorfisme ring van een supersinguliere elliptische kromme. Het belangrijkste voordeel is dat de groep van rationale punten over $\mathbb{F}_p$ precies $p + 1$ punten bevat, wat het gemakkelijk maakt om priemen te vinden waarvoor de krommen extreem gladde groepordes hebben. Dit leidt tot drastisch snellere evaluaties van isogenieën met behoud van de kleine publieke sleutels. Bovendien laten we zien hoe publieke sleutels geverifieerd kunnen worden. Het gevolg is dat de sleuteluitwisseling statische publieke sleutels ondersteunt en zodoende als niet-interactief kan worden beschouwd.

List of Publications

Conference proceedings

– Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. "CSIDH: An Efficient Post-Quantum Commutative Group Action". In: *Advances in Cryptology – ASIACRYPT 2018*. Ed. by Thomas Peyrin and Steven Galbraith. Cham: Springer International Publishing, 2018, pp. 395–427

– Joost Renes. "Computing Isogenies Between Montgomery Curves Using the Action of $(0,0)$". In: *Post-Quantum Cryptography*. Ed. by Tanja Lange and Rainer Steinwandt. Cham: Springer International Publishing, 2018, pp. 229–247

– Joost Renes and Benjamin Smith. "qDSA: Small and Secure Digital Signatures with Curve-Based Diffie–Hellman Key Pairs". In: *Advances in Cryptology – ASIACRYPT 2017*. Ed. by Tsuyoshi Takagi and Thomas Peyrin. Cham: Springer International Publishing, 2017, pp. 273–302

– Craig Costello, David Jao, Patrick Longa, Michael Naehrig, Joost Renes, and David Urbanik. "Efficient Compression of SIDH Public Keys". In: *Advances in Cryptology – EUROCRYPT 2017*. Ed. by Jean-Sébastien Coron and Jesper Buus Nielsen. Cham: Springer International Publishing, 2017, pp. 679–706

– Pedro Maat C. Massolino, Joost Renes, and Lejla Batina. "Implementing Complete Formulas on Weierstrass Curves in Hardware". In: *Security, Privacy, and Applied Cryptography Engineering*. Ed. by Claude Carlet, M. Anwar Hasan, and Vishal Saraswat. Cham: Springer International Publishing, 2016, pp. 89–108

– Joost Renes, Peter Schwabe, Benjamin Smith, and Lejla Batina. "μKummer: Efficient Hyperelliptic Signatures and Key Exchange on Microcontrollers". In: *Cryptographic Hardware and Embedded Systems – CHES 2016*. Ed. by Benedikt

Gierlichs and Axel Y. Poschmann. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 301–320

– Joost Renes, Craig Costello, and Lejla Batina. "Complete Addition Formulas for Prime Order Elliptic Curves". In: *Advances in Cryptology – EUROCRYPT 2016*. Ed. by Marc Fischlin and Jean-Sébastien Coron. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 403–428

Preprints

– Michael Naehrig and Joost Renes. *Dual Isogenies and Their Application to Public-key Compression for Isogeny-based Cryptography*. Cryptology ePrint Archive, Report 2019/499. `https://eprint.iacr.org/2019/499`. 2019

– Craig Costello, Patrick Longa, Michael Naehrig, Joost Renes, and Fernando Virdia. *Improved Classical Cryptanalysis of the Computational Supersingular Isogeny Problem*. Cryptology ePrint Archive, Report 2019/298. `https://eprint.iacr.org/2019/298`. 2019

– Huseyin Hisil and Joost Renes. *On Kummer Lines With Full Rational 2-torsion and Their Usage in Cryptography*. Cryptology ePrint Archive, Report 2018/839. `https://eprint.iacr.org/2018/839`. 2018

Other

– David Jao, Reza Azarderakhsh, Matthew Campagna, Craig Costello, Luca De Feo, Basil Hess, Amir Jalali, Brian Koziel, Brian LaMacchia, Patrick Longa, Michael Naehrig, Joost Renes, Vladimir Soukharev, and David Urbanik. *SIKE. Supersingular Isogeny Key Encapsulation*. Submission to [Nat16]. `http://sike.org`. 2016

Curriculum Vitae

Joost Renes

June 2019 –
Postdoctoral Researcher
Digital Security Group
Radboud University, The Netherlands

Summers of 2016 – 2018
Intern
Security and Cryptography Group
Microsoft Research, USA

January 2015 – May 2019
PhD Candidate
Digital Security Group
Radboud University, The Netherlands

September 2012 – July 2013
Master of Advanced Studies (*With Merit*)
Mathematics
University of Cambridge, United Kingdom

September 2009 – June 2012
Bachelor of Science (*Cum Laude*)
Mathematics
Utrecht University, The Netherlands