Elliptic Curves, Group Law, and Efficient Computation

by

Hüseyin Hışıl

Bachelor of Computer Engineering *(Izmir Institute of Technology)* – 2003
Master of Computer Engineering *(Izmir Institute of Technology)* – 2005

Thesis submitted in accordance with the regulations for the Degree of Doctor of Philosophy

Information Security Institute
Faculty of Science and Technology
Queensland University of Technology

24 April 2010

Keywords

Elliptic curve, group law, point addition, point doubling, projective coordinates, rational maps, birational equivalence, Riemann-Roch theorem, rational simplification, ANSI C language, `x86` assembly language, scalar multiplication, cryptographic pairing computation, elliptic curve cryptography.

Abstract

This thesis is about the *derivation of the addition law* on an arbitrary elliptic curve and *efficiently adding points* on this elliptic curve using the derived addition law. The outcomes of this research guarantee *practical speedups* in higher level operations which depend on point additions. In particular, the contributions immediately find *applications in cryptology.*

Mastered by the 19^{th} century mathematicians, the study of the theory of elliptic curves has been active for decades. Elliptic curves over finite fields made their way into public key cryptography in late 1980's with independent proposals by Miller [Mil86] and Koblitz [Kob87]. Elliptic Curve Cryptography (ECC), following Miller's and Koblitz's proposals, employs

the *group of rational points on an elliptic curve*

in building discrete logarithm based public key cryptosystems. Starting from late 1990's, the emergence of the ECC market has boosted the research in computational aspects of elliptic curves. This thesis falls into this same area of research where the main aim is to speed up the additions of rational points on an arbitrary elliptic curve (over a field of large characteristic). The outcomes of this work can be used to speed up applications which are based on elliptic curves, including cryptographic applications in ECC.

The aforementioned goals of this thesis are achieved in five main steps. As the first step, this thesis brings together several algebraic tools in order to derive the unique group law of an elliptic curve. This step also includes an investigation of recent computer algebra packages relating to their capabilities. Although the group law is unique, its evaluation can be performed using abundant (in fact infinitely many) formulae. As the second step, this thesis progresses the finding of the best formulae for efficient addition of points. In the third step, the group law is stated explicitly by handling all possible summands. The fourth step presents the algorithms to be used for efficient point additions. In the fifth and final step, optimized software implementations of the proposed algorithms are presented in order to show that theoretical speedups of step four can be practically obtained. In each of the five steps, this thesis focuses on five forms of elliptic curves over finite fields of large characteristic. A list of these forms and their defining equations are given as follows:

(a) Short Weierstrass form, $y^2 = x^3 + ax + b$,

(b) Extended Jacobi quartic form, $y^2 = dx^4 + 2ax^2 + 1$,

(c) Twisted Hessian form, $ax^3 + y^3 + 1 = dxy$,

(d) Twisted Edwards form, $ax^2 + y^2 = 1 + dx^2y^2$,

(e) Twisted Jacobi intersection form, $bs^2 + c^2 = 1, as^2 + d^2 = 1$,

These forms are the most promising candidates for efficient computations and thus considered in this work. Nevertheless, the methods employed in this thesis are capable of handling arbitrary elliptic curves.

From a high level point of view, the following outcomes are achieved in this thesis.

- Related literature results are brought together and further revisited. For most of the cases several missed formulae, algorithms, and efficient point representations are discovered.
- Analogies are made among all studied forms. For instance, it is shown that two sets of affine addition formulae are sufficient to cover all possible affine inputs as long as the output is also an affine point in any of these forms. In the literature, many special cases, especially interactions with points at infinity were omitted from discussion. This thesis handles all of the possibilities.
- Several new point doubling/addition formulae and algorithms are introduced, which are more efficient than the existing alternatives in the literature. Most notably, the speed of extended Jacobi quartic, twisted Edwards, and Jacobi intersection forms are improved. New unified addition formulae are proposed for short Weierstrass form. New coordinate systems are studied for the first time.
- An optimized implementation is developed using a combination of generic `x86-64` assembly instructions and the plain C language. The practical advantages of the proposed algorithms are supported by computer experiments.
- All formulae, presented in the body of this thesis, are checked for correctness using computer algebra scripts together with details on register allocations.

Table of Contents

List of Figures

List of Tables

List of Algorithms

Declaration

The work contained in this thesis has not been previously submitted for a degree or diploma at any higher education institution. To the best of my knowledge and belief, the thesis contains no material previously published or written by another person except where due reference is made.

Signed: ..... **Date:** .....

Previously Published Material

The following papers have been published or presented, and contain material based on the content of this thesis.

[1] Huseyin Hisil, Gary Carter, and Ed Dawson. New formulae for efficient elliptic curve arithmetic. In *INDOCRYPT 2007*, volume 4859 of *LNCS*, pages 138–151. Springer, 2007.

[2] Huseyin Hisil, Kenneth Koon-Ho Wong, Gary Carter, and Ed Dawson. Faster group operations on elliptic curves. In *Australasian Information Security Conference (AISC 2009), Wellington, New Zealand, January 2009*, volume 98, pages 11–19. Conferences in Research and Practice in Information Technology (CRPIT), 2009, *The Best Student-paper Award*.

[3] Huseyin Hisil, Kenneth Koon-Ho Wong, Gary Carter, and Ed Dawson. Twisted Edwards curves revisited. In *ASIACRYPT 2008*, volume 5350 of *LNCS*, pages 326–343. Springer, 2008.

[4] Huseyin Hisil, Kenneth Koon-Ho Wong, Gary Carter, and Ed Dawson. Jacobi quartic curves revisited. In *ACISP 2009*, volume 5594 of *LNCS*, pages 452–468. Springer, 2009.

[5] Craig Costello, Huseyin Hisil, Colin Boyd, Juan Manuel Gonzales Nieto, and Kenneth Koon-Ho Wong. Faster pairings on special Weierstrass curves. In *Pairing 2009*, volume 5671 of *LNCS*, pages 89–101. Springer, 2009.

Acknowledgements

I would like to express my sincere gratitude to my principal supervisor Professor Emeritus Ed Dawson who generously devoted time to the research that I have carried out over the past three and a half years. Prof. Dawson has always looked at the larger picture when I was only able to focus on technical details, and continuously guided me with his expertise to a better understanding of my research topic.

I would like to thank my associate supervisors Dr. Gary Carter and Dr. Kenneth Koon-Ho Wong for their close interest in my research and uncountable number of helpful suggestions. Dr. Carter placed ultimate priority on every aspect of my research and provided improvements to several results of this work at their preliminary stage. Dr. Wong generously shared his knowledge, suggested many computer algebra tips, and even worked with me in writing several computer algebra scripts.

I would like to thank Prof. Colin Boyd, Mr. Craig Costello, and Dr. Juan Manuel Gonzalez Nieto for their joint work in Chapter 7.

I would like to thank Dr. Harry Bartlett, Dr. Siguna Mueller, Assoc. Prof. Mike Scott, and Dr. Douglas Stebila for taking part in the reading committee and for providing many useful ideas and suggestions.

Special thanks go to Prof. Saban Eren and Assoc. Prof. Ahmet Koltuksuz who provided invaluable support and motivation for doing post-graduate research abroad. Many thanks go to my friends and colleagues at Information Security Institute for forming such a great working environment. I would like to thank Mr. Hamdi Dibeklioglu who suggested visual improvements to the illustrations in this thesis.

I thank my parents and my sister for their continued love, for whom this thesis is dedicated to with the most sincere respect and gratitude. They are the ones who have truly understood my commitment and dedication into this thesis and have put enourmous efforts into my education over years with patience and support.

Last, but not least, thanks to all software developers: mostly, to the developers of Eclipse, MAGMA, Maple, Subversive, and Texlipse. Without these powerful tools it would have taken ages to make a long type setting and detailed algebraic investigation.

Chapter 1

Introduction

Elliptic curves have been of great interest to algebraists, algebraic geometers, and number theorists for numerous decades. Since the time of Jacobi (more than 150 years ago) and long before the emergence of modern cryptography, it was well-known that every elliptic curve is endowed with a unique *group law* which turns the points on an elliptic curve into an abelian group. The binary operation of this group, which is rationally expressed in terms of the coordinates of points of an elliptic curve, is called *the addition law*. The addition law turns out to be efficiently computable for elliptic curves defined over "suitable" fields. In the 20^{th} century, such elliptic curves found several applications in cryptography.

Cryptography *Elliptic Curve Cryptography* (ECC) is a main branch of public key cryptography which employs elliptic curves in the construction of cryptosystems. Fundamental notions of *public key cryptography* were developed by Diffie and Hellman [DH76] in 1976, based on a striking idea of using separate keys for encrypting a plaintext and decrypting the ciphertext; or for signing a document and verifying the digital signature of the signed document. Since then, several proposals have been made to realize public key cryptosystems. The discovery of ECC is due to independent works of Miller [Mil86] and Koblitz [Kob87], see Appendix B.1. ECC has gained commercial success starting in the 1990s with the release of internationally recognized standards (such as ANSI X9.62 [Ame05], ANSI X9.63 [Ame01], FIPS 186-2 [Nat00], IEEE-1363 [Ins00], ISO CD 14888-3 [Int06], and WAP WLTS [Ope99]). Today, several ECC hardware and software applications can be found in the market place. A typical ECC implementation has four logical layers: finite field layer, elliptic curve point operations layer, scalar multiplication layer, and protocol layer. This thesis focuses on improving the second layer. Cryptographic computations are mainly dominated by *point additions* and *point doublings*. Therefore, these operations are of vital importance for the efficiency of higher layers: the scalar multiplication layer and the protocol layer. The efficiency of ECC has been improved over decades and is still an active area of research. ECC related efficient computations largely cover the direction of this research.

Cryptanalysis The *discrete logarithm* of kP with respect to base point P is the integer k. Computing the discrete logarithm when only kP and P are known, is assumed to be *much harder* than computing a scalar multiplication, see Appendix B.2. The security of ECC is based on the hardness of computing discrete logarithms. In particular, all known methods of discrete-logarithm computation run in *exponential time* for carefully selected parameters. As a consequence, elliptic curve cryptosystems promote a maximum conjectured-security per key-bit among other types public key cryptosystems. Although this work does not contribute to faster computation of discrete logarithms, it should be emphasized that all methods of discrete-logarithm-computation are again dominated by point additions where *efficient point additions* are crucial for *successful attacks*. Elliptic curves are the main ingredients of Lenstra's integer factorization (ECM) algorithm [Len87] and Atkin and Morain's elliptic curve primality proving (ECPP) algorithm. At the heart of both algorithms, once more *efficient point additions* are of paramount importance.

1.1 Motivation

The emphasis on using elliptic curves in cryptography stands on the following observations:

i the *discrete logarithm* computation can be made intractable for the existing technology,

ii once **i** is satisfied, the cryptographic operations can still be carried out efficiently.

In the case of **i**, excluding the algebraic attacks which apply only to some very special elliptic curves (cf. [Sma99], [FR94], [MVO91]), computing an elliptic curve discrete logarithm in a large prime order subgroup of an elliptic curve still runs in *exponential time* with Pollard's rho method (cf. [Pol78], [Tes98], [vOW99]), the best algorithm known to date for computing generic discrete logarithms. As a side-effect the required key length and representation of the points require far fewer bits in comparison to other public key cryptosystems such as RSA. For instance, it is generally accepted that 160-bit ECC keys are as secure as 1024-bit RSA keys, cf. [HMV03]. For higher levels of security, 196-bit, 224-bit, 256-bit ECC keys can also be used in conjunction with ECC standards. The smaller key sizes is a powerful property of ECC, which has provided initial motivations for this thesis. Note that the corresponding RSA key sizes for "the same level of security" increase more rapidly since subexponential time attacks apply to the case of RSA crpytosystem.

In the case of **ii**, the cryptographic operations which are typically dominated by scalar multiplications, can be efficiently computed using the "traditional" Weierstrass form of an elliptic curve (cf. [CF05, Ch.13]). Roughly speaking (for now), each scalar multiplication is composed of a few hundred point additions each of which can be performed with 16 multiplications in the underlying finite field. At this stage, it is natural to ask whether the point addition can be performed using fewer field operations. If this can be achieved then higher-level operations will automatically be faster as desired. *As a prelude (to heighten the reader's interest), it can be stated here that this thesis introduces algorithms which require as few as 8 multiplications rather than 16 and are applicable to many cryptographically-interesting elliptic curves.*

Elliptic curves can be represented with several different types of defining equations. Over fields of large characteristic, the most celebrated five forms of elliptic curves are the following:

1 Short Weierstrass form, $y^2 = x^3 + ax + b$ (in variables x and y),

2 Montgomery form, $by^2 = x^3 + ax^2 + x$ (in variables x and y),

3 Extended Jacobi quartic form, $y^2 = dx^4 + 2ax^2 + 1$ (in variables x and y),

4 Hessian form, $x^3 + y^3 + 1 = 3dxy$ (in variables x and y),

5 Jacobi intersection form, $s^2 + c^2 = 1, as^2 + d^2 = 1$ (in variables s, c, and d).

Short Weierstrass form has been the choice in most cryptographic standards and in hardware realizations over decades. This is due to two advantages of the short Weierstrass form:

i every elliptic curve over a field of characteristic $\neq 2, 3$ can be represented in short Weierstrass form,

ii the existing algorithms for performing group operations on some Weierstrass curves were more efficient in comparison to the others, cf. [CC86] and [CMO98]. The only exception is the Montgomery form [Mon87] for which Montgomery's algorithm is faster in some applications.

Indeed, the picture in **ii** remained unchanged for a long time. After 2000, however, a series of studies were conducted to speed up other forms of elliptic curves in certain contexts such as preventing side channel information leak or parallelization of operations over the computational units of some hardware (cf. [LS01], [Sma01], [JQ01], [SW03], [BJ03a], [Duq07]). Among these works, [LS01] presented point doubling algorithms which were more efficient than the point doubling algorithms known for short Weierstrass form.

In 2006, Montgomery form was integrated into Diffie-Hellman key-exchange and secret-sharing in [Ber06b], with a speed record at the time. In the same year, competitive formulae were proposed in [Gau06] for elliptic Kummer lines. Again in the same year, two special cases of (general) Weierstrass form were considered in [DI06] for efficient computations. The point doubling and point tripling algorithms in [DI06] were faster than the algorithms known for short Weierstrass form. So, the list was updated with:

6 Kummer line of $y^2 = x(x-1)(x - \frac{a^4}{a^4-b^4})$ (in variables x and y),

7 Doche/Icart/Kohel-2 form, $y^2 = x^3 + ax(x + 16)$ (in variables x and y),

8 Doche/Icart/Kohel-3 form, $y^2 = x^3 + 3a(x + 1)^2$ (in variables x and y).

In 2007, Edwards form was introduced in [BL07b] adding a ninth entry to the list of famous forms:

9 Edwards form, $x^2 + y^2 = c^2(1 + dx^2y^2)$.

Besides its several advantages, Edwards form has also broken the speed barrier of the short Weierstrass form. Even faster algorithms were introduced in [BL07c]. Shortly after, Edwards form was generalized to twisted Edwards form $ax^2 + y^2 = 1 + dx^2y^2$ which covers more curves and provides additional benefits.

In conclusion, spanning the period 2000 to late 2006—when this research commenced—several studies provided evidence of a more efficient "future" for elliptic curves. These studies have been a major motivation of this thesis.

In this thesis, the computer algrebra has been used as a powerful tool for studying the selected models. Developments in this area have been an important motivation. Especially, the map compositions and newly added implementations of Riemann-Roch computations are performed with computer algebra packages MAGMA [BCP97] and Maple [MAP08]. In addition, the latest rational simplification techniques from 2006 are also used in making the investigation faster, see [MP06].

1.2 Aims and outcomes

The main aim of this thesis is revisiting the elliptic curve group law with an emphasis on designing *more* efficient point additions. To achieve this aim the research is split into the following successive tasks:

- Collecting algebraic tools in order to find maps between curves using the Riemann-Roch theorem;
- Developing computer algebra tools to automate the group law derivation using the derived maps and the well-known group law of Weierstrass form elliptic curves;
- Finding a systematic way of simplifying rational expressions to make a "simple" statement of the group law;
- Developing an algorithm for each form in order to make a complete description of the group law by appropriately handling all possible summands;
- Developing inversion-free algorithms in various coordinate systems for each form and comparing each coordinate system in terms of efficiency in suitable contexts;
- Developing optimized high-speed software implementations in order to support theoretical results.

In each of these tasks, this thesis focuses on five forms of elliptic curves over finite fields of large characteristic. These five forms are the following:

1 Short Weierstrass form, $y^2 = x^3 + ax + b$ (in variables x and y),

2 Extended Jacobi quartic form, $y^2 = dx^4 + 2ax^2 + 1$ (in variables x and y),

3 Twisted Hessian form, $ax^3 + y^3 + 1 = dxy$ (in variables x and y),

4 Twisted Edwards form, $ax^2 + y^2 = 1 + dx^2y^2$ (in variables x and y),

5 Twisted Jacobi intersection form, $bs^2 + c^2 = 1, as^2 + d^2 = 1$ (in variables s, c and d).

There are several other forms of elliptic curves which are not specifically studied (or not even mentioned) in this work. This is due to the experience gained in this work and in many other studies that the above five forms are the most promising candidates for efficient computations. Nevertheless, the methods employed in this thesis are capable of handling an arbitrary form of an elliptic curve defined over an arbitrary field, embedded in a two-or-more-dimensional algebraic space.

Partial outcomes of this thesis have already appeared in published papers and have been presented at relevant conferences which have been listed on page xvii. In particular, partial contributions of [1] and [2] appears in Chapter 5. The contributions of [3] and [4] specifically appear in §5.1 and §5.2 of Chapter 5, respectively. In addition, the implementation introduced in [4] is a part of a more general software project explained in Chapter 6. Finally, Chapter 7 is based on [5]. *The results in these papers will be used hereafter without further pointers.*

From a high level point of view, the following outcomes are achieved in this thesis.

- Fragmented results in the literature about the group law on elliptic curves are brought together. In this context, each form is revisited by following a common notation and a similar treatment. This approach leads to the discovery of several missed formulae, algorithms, and efficient point representations in the previous works.

- Analogies are made among all studied forms. For instance, it is shown that two sets of addition formulae are sufficient to cover the affine part of the curve given in any of these forms, see Chapter 4. This is an analogous observation to [BL95], which applies not only to short Weierstrass form but also to the other four forms.

- It is well-known that the addition law on an elliptic curve is a morphism, i.e. there always exists a way of adding two arbitrary points. In the literature, many special cases, especially interactions with points at infinity, were omitted from discussion. This thesis describes the corresponding morphism for each form explicitly in affine coordinates (see Chapter 4).

- ♣ Several new point doubling/addition formulae and algorithms are introduced, which are more efficient than the existing alternatives in the literature, see Chapters 4 and 5.

- An optimized implementation is developed using a combination of generic `x86` assembly and plain C languages. Several experiments using this implementation have supported the practical advantages of the proposed algorithms (see Chapter 6).

- All formulae presented in the body of this thesis are checked for correctness using computer algebra scripts which are also provided as an appendix for the convenience of the reader. In fact, the presented computer scripts make several details —such as the register allocations— accessible to programmers, see Appendix C.

Since the main contribution of the thesis is efficient computations, the ♣-marked item requires more discussion of the contributions as follows:

- *Short Weierstrass form* $y^2 = x^3 + ax + b$. Affine unified addition formulae (i.e. point addition formulae which can be used for almost all doublings and additions) were previously studied in homogeneous projective coordinates taking $12\mathbf{M} + 5\mathbf{S} + 1\mathbf{D}$ in [BJ02] and $11\mathbf{M} + 6\mathbf{S} + 1\mathbf{D}$ in [BL07a]. Here, the notations for **M**, **S**, **D**, and **a** are borrowed from [BL07b]. **M**, **S**, **D** means multiplication, squaring, addition, respectively. The operation count is improved to $11\mathbf{M} + 5\mathbf{S} + 1\mathbf{D}$ in the same coordinate system. Furthermore, a unified addition algorithm which takes $7\mathbf{M}+9\mathbf{S}+1\mathbf{D}$ is presented for the first time in modified Jacobian coordinates. The latter algorithm shows that modified Jacobian coordinates are more suitable for unified additions than homogeneous projective coordinates for most applications.

- *Special short Weierstrass form* $y^2 = cx^3+1$. A new point doubling formulae is introduced for the non-standard short Weierstrass curve $y^2 = cx^3 + 1$. The proposed doubling formulae shares several common subexpressions with the line functions which arise in Tate pairing computations. A combined point doubling and line computation takes only $(k + 3)\mathbf{M} + 5\mathbf{S}$ where k is the embedding degree. This is an improvement over the best literature proposal of $(k + 3)\mathbf{M} + 8\mathbf{S}$ in [ALNR09]. In terms of operation counts, this is so far the best for pairing computations.

- *Extended Jacobi quartic form* $y^2 = dx^4 + 2ax + 1$. A $2\mathbf{M} + 5\mathbf{S} + 1\mathbf{D}$ point doubling algorithm is proposed in homogeneous projective coordinates. The proposed algorithm is capable of working for arbitrary curve constants and in this context, improves upon the $3\mathbf{M} + 6\mathbf{S} + 3\mathbf{D}$ (reported as $11\mathbf{M}$) algorithm in [CC86]. Several point addition algorithms are proposed in this thesis in homogeneous projective coordinates and extended homogeneous projective coordinates. The best results are obtained in the latter coordinate system. In fact, this coordinate system is considered for extended Jacobi quartic curves for the first time in this thesis. A dedicated addition algorithm takes $7\mathbf{M} + 3\mathbf{S} + 3\mathbf{D}$ in extended homogeneous projective coordinates. This is also an improvement over the corresponding $16\mathbf{M} + 2\mathbf{D}$, $10\mathbf{M} + 3\mathbf{S} + 3\mathbf{D}$, and $9\mathbf{M} + 2\mathbf{S} + 3\mathbf{D}$ algorithms in [CC86], [BJ03a], and [Duq07], respectively. Furthermore, an efficient technique to benefit from fast doublings in homogeneous projective coordinates and fast additions in extended homogeneous projective coordinates is described, following a similar construction in [CMO98]. It is important to emphasize here that these results are obtained not only by considering different coordinate systems but also by searching for lower degree doubling and addition formulae in affine coordinates and then adapting them to suitable coordinate systems. The proposed algorithms are applicable for all elliptic curves having a point of order 2. For special cases of a and d, in particular for $a = \pm 1/2$ or $d = \pm 1$, even more striking results are obtained, which again improves upon other works in the literature.

- *Twisted Hessian form* $ax^3+y^3+1 = dxy$. An $11\mathbf{M}$ point addition algorithm is proposed, improving upon the $12\mathbf{M}+1\mathbf{D}$ algorithm in [BL07a]. For Hessian form, i.e. when $a = 1$, $7\mathbf{M} + 1\mathbf{S}$ and $3\mathbf{M} + 6\mathbf{S}$ doubling algorithms are proposed, improving upon the standard $6\mathbf{M} + 3\mathbf{S}$ algorithm, cf. [Sma01].

- *Twisted Edwards form* $ax^2 + y^2 = 1 + dx^2y^2$. A $9\mathbf{M} + 2\mathbf{D}$ point addition algorithm is proposed, improving upon the $9\mathbf{M} + 1\mathbf{S} + 1\mathbf{D}$ algorithm in [BL07c]. In the case $a = -1$ the operation count is further improved to a remarkable $8\mathbf{M}$ which is so far the best addition algorithm (excluding differential-additions which fall into a somewhat different "category" of comparison, cf. [Mon87]). Finally, efficient ways of mixing homogeneous projective and extended homogeneous projective coordinates are shown as in the extended Jacobi quartic case.

- *Twisted Jacobi intersection form* $bs^2 + c^2 = 1, as^2 + d^2 = 1$. Usually these curves were studied only for the case $b = 1$. This thesis revisits Jacobi intersection form with its generalized version, i.e. with arbitrary b. This modification only marginally increases the coverage of this form. On the other hand, describing elliptic curves which can be represented in this form becomes easier. In particular, every elliptic curve having 3 points of order 2 can be represented in twisted Jacobi intersection form. For the case $b = 1$, a $2\mathbf{M}+5\mathbf{S}+1\mathbf{D}$ doubling algorithm is presented which improves —for very small $\mathbf{D}$— upon the $4\mathbf{M} + 3\mathbf{S}$ and $3\mathbf{M} + 4\mathbf{S}$ alternatives given in [LS01] and [BL07a], respectively. An $11\mathbf{M}$ addition algorithm which improves upon the $13\mathbf{M}+2\mathbf{S}+1\mathbf{D}$ in [LS01] is introduced in extended homogeneous projective coordinates, based on new affine addition formulae.

Note that these remarks are only the most outstanding outcomes of this thesis. Additional contributions and comparisons for each form are omitted here. For full details see Chapter 5.

1.3 Roadmap

The rest of this thesis is structured as follows.

Chapter 2 provides formal definitions for background concepts which will be frequently accessed in the subsequent chapters. In particular, the most important concept of study in this thesis, *the group law*, is defined. Weierstrass forms of selected curves are presented along with birational maps.

Chapter 3 brings together several computational tools which are beneficial in developing efficient ways of deriving group laws on elliptic curves. These tools will be central to Chapters 4, 5, and 7.

Chapter 4 presents low-degree point addition formulae for fixed forms of elliptic curves and states a complete addition algorithm in affine coordinates for each form by suitably handling all division by zero exceptions and interactions with the point(s) at infinity.

Chapter 5 presents several new inversion-free point addition formulae together with various operation counts. Chapter 5 also provides a collection of selected formulae from the literature.

Chapter 6 provides several details on the implementation of elliptic curve arithmetic in the light of new results from Chapters 4 and 5. The aim is to show that ECC applications can benefit practically from the proposed methods. A secondary aim is to compare and contrast different elliptic curve models in terms of their efficiency, space consumption, and *sensitivity* to varying environmental constraints (e.g. minimizing side channel information leak or memory requirements).

Chapter 7 is a case study about efficient pairing computation on the j-invariant zero curve $y^2 = cx^3 + 1$. In particular, Chapter 7 introduces new formulae that facilitate a faster Tate pairing computation on this curve.

Chapter 8 concludes this thesis with a summary of the contributions and future research directions.

The thesis contains three appendices. Appendix A provides formal definitions for background algebraic concepts which will be frequently accessed in the main body of the thesis. In particular, the most important objects of study in this thesis –*elliptic curves*– are defined. Appendix A also summarizes the main results from Gröbner basis theory which will be needed in Chapter 3 for tool development. Appendix B reviews basic concepts of ECC, elliptic curve discrete logarithm problem, and cryptographic pairings. Appendix C provides computer scripts to verify the formulae presented in this thesis.

Chapter 2

Elliptic Curves

This chapter provides formal definitions for background concepts which will be frequently accessed in the subsequent chapters. In particular, the most important concept of study in this thesis, the so-called *group law*, is defined. This law makes the points forming an elliptic curve into a group on which *efficient* addition of points will gradually come into play as the main theme of this thesis. To assist this theme five celebrated forms of elliptic curves are presented together with their important properties. These five forms are the basis of the study in the remainder of the thesis for efficient implementation of point addition on elliptic curves.

> The definitions and notations for more fundamental concepts
> and important theorems can be found in Appendix A.

The notation is adapted from [Sil94], [Sti93], and [CF05]. The rest of the chapter is organized as follows. §2.1 describes the Weierstrass form of an elliptic curve. §2.2 summarizes the well known group law on Weierstrass curves. §2.3 presents the relation of the selected forms of elliptic curves to the general Weierstrass curve. Many of the maps in §2.3 are computed independently and documented in this thesis. §2.4 defines the scalar multiplication on an elliptic curve. Conclusions are drawn in §2.5.

2.1 Weierstrass form

Throughout this subsection, $\mathbb{K}$ denotes a field of *arbitrary characteristic* and $\mathbb{L}$ an algebraic extension of $\mathbb{K}$.

Definition 2.1.1. Let $a_1, a_3, a_2, a_4, a_6 \in \mathbb{K}$. A Weierstrass curve defined over $\mathbb{K}$ is the curve

$$E_{\mathbf{W},a_1,a_3,a_2,a_4,a_6} : v^2 + a_1uv + a_3v = u^3 + a_2u^2 + a_4u + a_6.$$

A Weierstrass curve is non-singular if and only if for every $u_1, v_1 \in \overline{\mathbb{K}}$ (closure of $\mathbb{K}$) with $v_1^2+a_1u_1v_1+a_3v_1-(u_1^3+a_2u_1^2+a_4u_1+a_6) = 0$, the partial derivatives $2v_1+a_1u_1+a_3$ and a_1v_1-

$3u_1^2 - 2a_2u_1 - a_4$ do not vanish simultaneously (see the Jacobi criterion in [CF05, Lemma 4.49]). The singularity check can be done algebraically by computing $\Delta = -b_2^2b_8 - 8b_4^3 - 27b_6^2 + 9b_2b_4b_6$ where $b_2 = a_1^2 + 4a_2$, $b_4 = a_1a_3 + 2a_4$, $b_6 = a_3^2 + 4a_6$, and $b_8 = a_1^2a_6 - a_1a_3a_4 + 4a_2a_6 + a_2a_3^2 - a_4^2$. A Weierstrass curve is non-singular if and only if $\Delta \neq 0$. The notation $E_{\mathbf{W},a_1,a_3,a_2,a_4,a_6}$ will be abbreviated as $E_{\mathbf{W}}$ when a_1, a_3, a_2, a_4, a_6 are understood. The projective closure of $E_{\mathbf{W}}$ is given by the equation

$$\overline{E}_{\mathbf{W},a_1,a_3,a_2,a_4,a_6} : V^2W + a_1UVW + a_3VW^2 = U^3 + a_2U^2W + a_4UW^2 + a_6W^3.$$

A point $(U\colon V\colon W)$ with $U, V \in \overline{\mathbb{K}}$ and $W \in \overline{\mathbb{K}}\backslash\{0\}$ on $\overline{E}_{\mathbf{W}}$ corresponds to the affine point $(U/W, V/W)$ on $E_{\mathbf{W}}$. The point $(0\colon 1\colon 0)$ on $\overline{E}_{\mathbf{W}}$ is non-singular. This point is called the point at infinity and is denoted by ∞. The point ∞ is $\mathbb{K}$-rational. There are no other points on $\overline{E}_{\mathbf{W}}$ with $W = 0$.

With a slight abuse of notation, $\overline{E}_{\mathbf{W}}(\mathbb{L})$, the set of $\mathbb{L}$-rational points on $\overline{E}_{\mathbf{W}}$ is denoted by

$$E_{\mathbf{W}}(\mathbb{L}) = \{(u, v) \in \mathbb{L}^2 \mid v^2 + a_1uv + a_3v = u^3 + a_2u^2 + a_4u + a_6\} \cup \{\infty\}.$$

An elliptic curve is denoted by its affine part hereafter by assuming that its projective closure is understood.

For a fixed field $\mathbb{K}$ and a fixed curve $C/\mathbb{K}$, the genus is an invariant of C and is a useful tool for classification purposes. Curves of a particular genus are typically loosely classified with respect to birational equivalence which preserves several algebraic properties but not necessarily an isomorphism. At this point, it is natural to ask whether an arbitrary genus-1 curve is birationally equivalent to a non-singular curve (preferably to a plane curve). The following theorem provides a celebrated answer.

Theorem 2.1.2 (Weierstrass form of an elliptic curve)**.** *Let $C/\mathbb{K}$ be a genus 1 curve with a $\mathbb{K}$-rational point. There exist $a_1, a_3, a_2, a_4, a_6 \in \mathbb{K}$ such that*

$$\mathbb{K}(C) \cong \mathbb{K}(E_{\mathbf{W},a_1,a_3,a_2,a_4,a_6}).$$

Thus, C is birationally equivalent over $\mathbb{K}$ to $E_{\mathbf{W}}$.

Proof. The proof follows from an application of Theorem A.3.6, see [Sil94, §III.3.3] and [CF05, §4.4.2 and §13.1]. Also see Appendix A for the notation and relevant definitions. □

It is also natural to ask when are two Weierstrass curves isomorphic over $\mathbb{K}$.

Theorem 2.1.3. *Let $E_{\mathbf{W},a_1,a_3,a_2,a_4,a_6}$ and $E_{\mathbf{W}',A_1,A_3,A_2,A_4,A_6}$ be Weierstrass curves defined over $\mathbb{K}$, as in Definition 2.1.1. $E_{\mathbf{W}}$ and $E_{\mathbf{W}'}$ are isomorphic over $\mathbb{K}$ if and only if there exists*

$c \in \mathbb{K}\backslash\{0\}$ and $r, s, t \in \mathbb{K}$ such that

$$\begin{aligned}
A_1 &= (a_1 + 2s)/c,\\
A_2 &= (a_2 - sa_1 + 3r - s^2)/c^2,\\
A_3 &= (a_3 + ra_1 + 2t)/c^3,\\
A_4 &= (a_4 - sa_3 + 2ra_2 - (t + rs)a_1 + 3r^2 - 2st)/c^4,\\
A_6 &= (a_6 + ra_4 + r^2a_2 + r^3 - ta_3 - t^2 - rta_1)/c^6.
\end{aligned}$$

If such c, r, s, t exist then the maps

$$\psi\colon E_{\mathbf{W}} \to E_{\mathbf{W}'},\ (u, v) \mapsto \Big(\frac{u - r}{c^2}, \frac{v - s(u - r) - t}{c^3}\Big), \tag{2.1}$$

$$\phi\colon E_{\mathbf{W}'} \to E_{\mathbf{W}},\ (u', v') \mapsto \Big(c^2u' + r, c^3v' + c^2su' + t\Big) \tag{2.2}$$

are the desired isomorphisms defined over $\mathbb{K}$.

Proof. See [Sil94, Table 1.2, III.1]. □

Definition 2.1.4. The morphism ϕ in Theorem 2.1.3 is called the **admissible change of variables**.

Definition 2.1.5. Assume that c, r, s, t exist such that $c \in \overline{\mathbb{K}}\backslash\{0\}$ and $r, s, t \in \overline{\mathbb{K}}$ in the setting of Theorem 2.1.3. Then, the curves $E_{\mathbf{W}}$ and $E_{\mathbf{W}'}$ are called **twists**. Let $\mathbb{L}$ be a finite extension of $\mathbb{K}$ with $[\mathbb{L}\colon \mathbb{K}] = d$. Assume that $c \in \mathbb{L}\backslash\{0\}$ and $r, s, t \in \mathbb{L}$. If $E_{\mathbf{W}}$ and $E_{\mathbf{W}'}$ are isomorphic over $\mathbb{L}$ where d is *minimal* then $E_{\mathbf{W}}$ and $E_{\mathbf{W}'}$ are called **twists of degree** d. Assuming that an appropriate d exists, **quadratic twists** are twists of degree 2, **cubic twists** are twists of degree 3, and so on.

Twists are identified with the j-invariant which will be defined next. Let $E_{\mathbf{W}}$ and $E_{\mathbf{W}'}$ be curves defined as in Definition 2.1.1. The j-invariant of $E_{\mathbf{W}}$ is given by $j(E_{\mathbf{W}}) = (b_2^2 - 24b_4)^3/\Delta$. This is a tool to algebraically check whether the desired c, r, s, t exist such that $c \in \overline{\mathbb{K}}\backslash\{0\}$ and $r, s, t \in \overline{\mathbb{K}}$. Now, if $j(E_{\mathbf{W}}) \neq j(E_{\mathbf{W}'})$ then $E_{\mathbf{W}}$ is not isomorphic over $\mathbb{K}$ (not even over $\overline{\mathbb{K}}$) to $E_{\mathbf{W}'}$. If $j(E_{\mathbf{W}}) = j(E_{\mathbf{W}'})$ then $E_{\mathbf{W}}$ is isomorphic over $\overline{\mathbb{K}}$ to $E_{\mathbf{W}'}$ (see any of the reference books given in Appendix A). For more details on the type of twists that elliptic curves admit, see [CF05].

2.2 Group law

This section presents the group law on elliptic curves. Let $E_{\mathbf{W}}$ be a Weierstrass form elliptic curve with the point at infinity $\infty \in E_{\mathbf{W}}(\mathbb{L})$. The identity element is the point ∞. To specify this choice the identity is denoted by $\mathcal{O}$. Every point in $E_{\mathbf{W}}(\mathbb{L})$ has a unique inverse which can be computed by the unary operation "$-$". A computation of this operation requires case distinctions. In particular, $-\mathcal{O} = \mathcal{O}$. Let $P_1 = (u_1, v_1) \in E_{\mathbf{W}}$. Then $-P_1 = (u_1, -v_1 - a_1u_1 - a_3)$. A computation of the binary operation "$+$" requires somewhat more case distinctions. These cases are summarized in Algorithm 2.2.1. Using this algorithm, it can be verified that $P_1 + P_2 = P_2 + P_1$ and $(P_0 + P_1) + P_2 = P_0 + (P_1 + P_2)$ for all $P_i \in E_{\mathbf{W}}(\mathbb{K})$. Geometric and algebraic verifications of the group axioms are given in many textbooks, cf. [Ful69] and [SS98].

Algorithm 2.2.1: The addition law for Weierstrass form in affine coordinates

input : $P_1, P_2, \mathcal{O} \in E_{\mathbf{W},a_1,a_3,a_2,a_4,a_6}(\mathbb{K})$.
output : $P_3 = P_1 + P_2$.

```
 1 if P1 = O then return P2.
 2 else if P2 = O then return P1.
 3 else if u1 = u2 then
 4     if v1 ≠ v2 then return O.
 5     else if 2v1 + a1u1 + a3 = 0 then return O.
 6     else
 7         λ ← (3u1^2 + 2a2u1 − a1v1 + a4)/(2v1 + a1u1 + a3).
 8         u3 ← λ^2 + a1λ − a2 − 2u1.
 9         v3 ← λ(u1 − u3) − v1 − a1u3 − a3.
10         return (u3, v3).
11     end
12 else
13     λ ← (v1 − v2)/(u1 − u2).
14     u3 ← λ^2 + a1λ − a2 − u1 − u2.
15     v3 ← λ(u1 − u3) − v1 − a1u3 − a3.
16     return (u3, v3).
17 end
```

Definition 2.2.1. The unary operation $-$ is called the **negation law**. The binary operation $+$ is called the **addition law**. Together with a fixed identity element these two laws become the building blocks of **the group law** which turns $E_{\mathbf{W}}$ in to an additively written abelian group.

Both the negation and addition laws require case distinctions. The sets of formulae handling some of these cases will be assigned special names hereafter.

Definition 2.2.2. If a set of formulae can only be used without any case distinction to carry out the operation

- "$-$" for all but finitely many points in $E_{\mathbf{W}}$ then such formulae are called the **point-negation formulae**. The operation carried out is called the **point-negation**.
- "$+$" for all but finitely many pairs of equal points and not for any pair of *distinct* points in $E_{\mathbf{W}} \times E_{\mathbf{W}}$ then such formulae are called the **point-doubling formulae**. For instance, see lines 7, 8, 9 in Algorithm 2.2.1. The operation carried out is called the **point-doubling**.
- "$+$" for all but finitely many pairs of distinct points in $E_{\mathbf{W}} \times E_{\mathbf{W}}$ then such formulae are called the **dedicated point-addition formulae**. For instance, see lines 13, 14, 15 in Algorithm 2.2.1. The operation carried out is called the **dedicated point-addition**.
- "$+$" for all but finitely many pairs of not necessarily distinct points in $E_{\mathbf{W}} \times E_{\mathbf{W}}$ then such formulae are called the **unified point-addition formulae**. For instance, see [SS98, Remark III.3.1]. The operation carried out is called the **unified point-addition**.

For economical reasons the "point-" and even the "formulae" part of each term will sometimes be dropped assuming that the meaning is clear from the context.

Theorem 2.2.3. *Let $E_{\mathbf{W}}/\mathbb{K}$ be an elliptic curve. Then the addition law and the negation law define morphisms*

$$+ : E_{\mathbf{W}} \times E_{\mathbf{W}} \to E_{\mathbf{W}} \qquad \text{and} \qquad - : E_{\mathbf{W}} \to E_{\mathbf{W}}$$
$$(P_1, P_2) \mapsto P_1 + P_2 \qquad\qquad P_1 \mapsto -P_1.$$

Proof. See [Sil94, Theorem III.3.6] for a proof. □

When speaking of one of these terms, say, a unified addition, it may be the case that the denominators vanish and produce division by zero in affine coordinates. Similarly, it may be the case in projective coordinates where all coordinates of the sum turn out to be zero. Since the addition law is a morphism by Theorem 2.2.3 it is always possible to switch to another set of formulae to compute the correct output. See also Definition A.2.2 or [Sil94, Remark 3.1]. Therefore, when stating the addition law on an elliptic curve all cases should be considered carefully. Chapter 4 will provide more details on this.

2.3 Forms of elliptic curves

This section provides examples of different forms of genus 1 curves:

- Short Weierstrass form, $y^2 = x^3 + ax + b$, §2.3.1
- Extended Jacobi quartic form, $y^2 = dx^4 + 2ax^2 + 1$, §2.3.2,
- Twisted Hessian form, $ax^3 + y^3 + 1 = dxy$, §2.3.3,
- Twisted Edwards form, $ax^2 + y^2 = 1 + dx^2y^2$, §2.3.4,
- Twisted Jacobi intersection form, $bs^2 + c^2 = 1, as^2 + d^2 = 1$, §2.3.5.

With a distinguished $\mathbb{K}$-rational point and resolved singularities (if any), the projective closure of all of these shapes are elliptic curves provided $\mathbb{K}$ has a suitable characteristic. These aspects will be detailed in each subsection.

In cryptography, two birationally equivalent genus 1 curves are usually treated as the same curve since both curves necessarily have the same group structure. In algebraic geometry, however, these two curves are distinct objects even if they are isomorphic. This thesis follows the latter nomenclature. To prevent ambiguity, the term "form" will be used instead of the term "curves".

This section explicitly describes the birational equivalence of each curve between each of these forms and some Weierstrass curve. Some of the birational maps are borrowed from the literature resources while some others are derived by computer algebra tools which use Theorem 2.1.2 for this purpose. Applied examples on the explicit derivation of the maps will be presented in §3.2 of Chapter 3. Therefore, further discussion is omitted in this section. On the other hand, pointers to the literature are provided in §2.3.6. It is convenient to note here that for each one of the studied forms the identity element and the presented maps comply with the revisited/computed/proposed formulae in Chapters 3, 4, 5, and 7.

It is still possible to substantially extend the list of the given forms. Indeed, a very recent preprint [CV09] explains a derivation of group laws for many more forms. However, the listed forms at the beginning of this section are still the best when it comes to efficient computations.

An illustration of sample curves over the real numbers $\mathbb{R}$ in each form is depicted in Figures 2.1 to 2.5 in the respective order of the list at the start of this section. The figures are drawn using Maple v.12. (Note that depending on the selected curve constants the presented shapes may look considerably different. The curve in Figure 2.5 lies in the intersection of the surfaces drawn.)

2.3.1 Short Weierstrass form

Throughout this subsection, $\mathbb{K}$ denotes a *fixed* field with $\mathrm{char}(\mathbb{K}) \neq 2, 3$ and $\mathbb{L}$ an algebraic extension of $\mathbb{K}$. Let $a, b \in \mathbb{K}$.

Definition 2.3.1. A short Weierstrass curve defined over $\mathbb{K}$ is the curve

$$E_{\mathbf{S},a,b} : \; y^2 = x^3 + ax + b.$$

This curve is non-singular if and only if $4a^3 + 27b^2 \neq 0$. The j-invariant is given by $6912a^3/(4a^3 + 27b^2) \in \mathbb{K}$. The projective closure of $E_{\mathbf{S}}$ is given by the equation

$$\overline{E}_{\mathbf{S},a,b} : \; Y^2Z = X^3 + aXZ^2 + bZ^3.$$

A point $(X : Y : Z)$ with $Z \neq 0$ on $\overline{E}_{\mathbf{S}}$ corresponds to the affine point $(X/Z, Y/Z)$ on $E_{\mathbf{S}}$. The point $\Omega = (0 : 1 : 0)$ on $\overline{E}_{\mathbf{S}}$ is non-singular and is always $\mathbb{L}$-rational. There are no other points on $\overline{E}_{\mathbf{S}}$ with $Z = 0$.

With a slight abuse of notation, $\overline{E}_{\mathbf{S}}(\mathbb{L})$, the set of $\mathbb{L}$-rational points on $\overline{E}_{\mathbf{S}}$ is denoted by

$$E_{\mathbf{S}}(\mathbb{L}) = \{(x, y) \in \mathbb{L}^2 \mid y^2 = x^3 + ax + b\} \cup \{\Omega\}.$$

To this end, the short Weierstrass form is nothing but a special case of $E_{\mathbf{W}}$. However, it turns out that every Weierstrass curve $E_{\mathbf{W},a_1,a_3,a_2,a_4,a_6}/\mathbb{K}$ is isomorphic over $\mathbb{K}$ to a short Weierstrass curve $E_{\mathbf{S},a',b'}$ where $a' = (24(2a_4 + a_1a_3) - (a_1^2 + 4a_2)^2)/48$ and $b' = ((a_1^2 + 4a_2)^3 - 36(2a_4 + a_1a_3)(a_1^2 + 4a_2) + 216(a_3^2 + 4a_6))/864$ via the admissible change of coordinates given by the polynomial maps

$$\phi\colon E_{\mathbf{W}} \to E_{\mathbf{S},a',b'},\ (u, v) \mapsto \Big(u + \frac{a_1^2 + 4a_2}{12}, v + \frac{a_1u + a_3}{2}\Big), \tag{2.3}$$

$$\psi\colon E_{\mathbf{S},a',b'} \to E_{\mathbf{W}},\ (x, y) \mapsto \Big(x - \frac{a_1^2 + 4a_2}{12}, y - \frac{a_1(x - \frac{a_1^2+4a_2}{12}) + a_3}{2}\Big). \tag{2.4}$$

See [Sil94, §III.1]. It is trivial to check that $\phi \circ \psi = \mathrm{id}_{E_{\mathbf{S}}}$ and $\psi \circ \phi = \mathrm{id}_{E_{\mathbf{W}}}$. Here id is the identity map. Both maps ϕ and ψ are regular at all points satisfying $E_{\mathbf{S},a',b'}$. Therefore, both ϕ and ψ define morphisms by Definition A.2.2. Also note that the point at infinity on $E_{\mathbf{W}}$ corresponds to the point at infinity on $E_{\mathbf{S},a',b'}$ and the curves are isomorphic over $\mathbb{K}$.

It would be possible to find a curve $E_{\mathbf{S},a'',b''}/\mathbb{K}$ which is birationally equivalent over $\mathbb{K}$ to one of the other selected forms given at the beginning of this section. On the other hand,

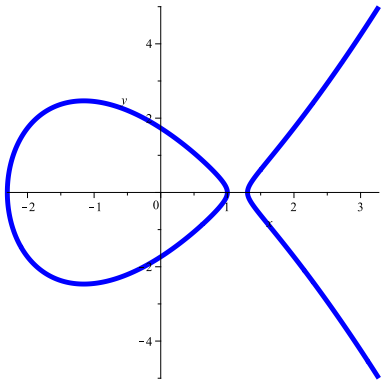

Figure 2.1: Short Weierstrass form elliptic curve $y^2 = x^3 - 4x + 3$ over $\mathbb{R}$.

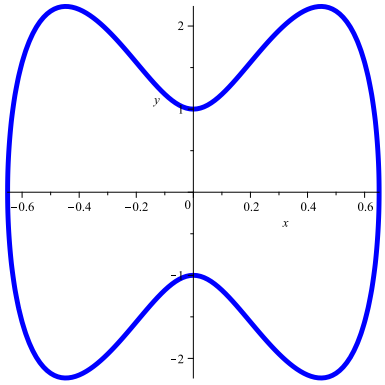

Figure 2.2: Extended Jacobi quartic form elliptic curve $y^2 = -100x^4 + 40x^2 + 1$ over $\mathbb{R}$.

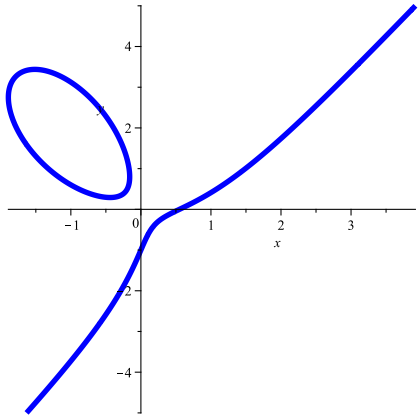

Figure 2.3: Twisted Hessian form elliptic curve $-6x^3 + y^3 + 1 = -4xy$ over $\mathbb{R}$.

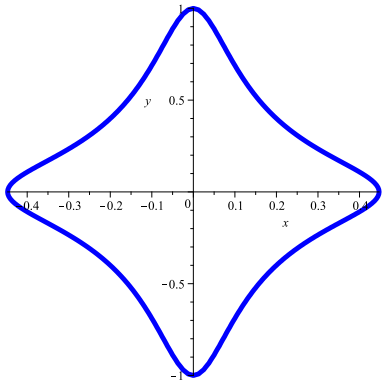

Figure 2.4: Twisted Edwards form elliptic curve $5x^2 + y^2 = 1 - 100x^2y^2$ over $\mathbb{R}$.

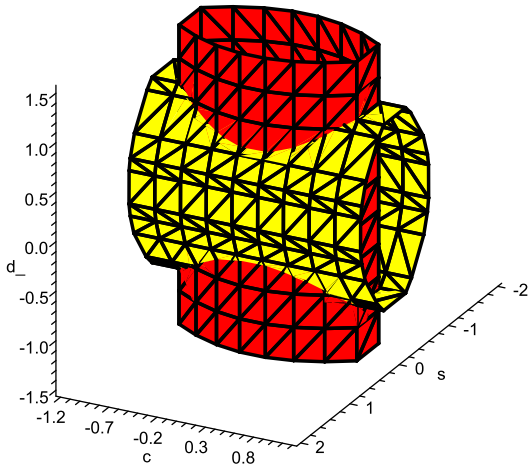

Figure 2.5: Twisted Jacobi intersection form elliptic curve $3.2s^2 + c^2 = 1, 2.2s^2 + d^2 = 1$ over $\mathbb{R}$.

using $E_{\mathbf{W}}$ for the very same purpose leads to "simpler" maps in some cases. Therefore, $E_{\mathbf{W}}$ is preferred hereafter. The cases where $E_{\mathbf{S}}$ is preferred are actually no different than the case $E_{\mathbf{W}}$ since $E_{\mathbf{S},a,b} = E_{\mathbf{W},0,0,0,a,b}$.

2.3.2 Extended Jacobi quartic form

Throughout this subsection, $\mathbb{K}$ denotes a *fixed* field of odd characteristic and $\mathbb{L}$ an algebraic extension of $\mathbb{K}$. Let $d, a \in \mathbb{K}$. Assume that d is a square in $\mathbb{L}$ unless stated otherwise.

Definition 2.3.2. An extended Jacobi quartic curve defined over $\mathbb{K}$ is the curve

$$E_{\mathbf{Q},d,a} : \; y^2 = dx^4 + 2ax^2 + 1.$$

This curve is non-singular if and only if $d(a^2 - d) \neq 0$. The j-invariant is given by $64(a^2 + 3d)^3/(d(a^2 - d)^2) \in \mathbb{K}$. The projective closure of $E_{\mathbf{Q}}$ is given by the equation

$$\overline{E}_{\mathbf{Q},d,a} : \; Y^2Z^2 = dX^4 + 2aX^2Z^2 + Z^4.$$

A point $(X : Y : Z)$ with $Z \neq 0$ on $\overline{E}_{\mathbf{Q}}$ corresponds to the affine point $(X/Z, Y/Z)$ on $E_{\mathbf{Q}}$. The point $(0 : 1 : 0)$ on $\overline{E}_{\mathbf{Q}}$ is singular. Using the standard "blow-up" techniques (see [Ful69, §7.3]) the singularity can be resolved. The resolution of singularities produces two points which are labeled as Ω_1 and Ω_2. It is convenient to note here that two "blow-up"s are necessary and sufficient to resolve the singularities. There are no other points on $\overline{E}_{\mathbf{Q}}$ with $Z = 0$.

A way of removing the singularity is by using the projective curve given by the equations

$$\widetilde{E}_{\mathbf{Q},d,a} : \; Y^2 = dT^2 + 2aX^2 + Z^2, X^2 = TZ.$$

A point $(X : Y : T : Z)$ with $Z \neq 0$ on $\widetilde{E}_{\mathbf{Q}}$ corresponds to the affine point $(X/Z, Y/Z)$ on $E_{\mathbf{Q}}$. Fix $\delta \in \mathbb{K}$ such that $\delta^2 = d$. The points $(0 : \delta : 1 : 0)$ and $(0 : -\delta : 1 : 0)$ correspond to Ω_1 and Ω_2 on the desingularization of $\overline{E}_{\mathbf{Q}}$. There is no other point on $\widetilde{E}_{\mathbf{Q}}$ with $Z = 0$.

Another way of removing the singularity is by using the weighted projective curve

$$\widehat{E}_{\mathbf{Q},d,a} : \; Y^2 = dX^4 + 2aX^2Z^2 + Z^4.$$

A point $(X : Y : Z)$ with $Z \neq 0$ on $\widehat{E}_{\mathbf{Q}}$ corresponds to the affine point $(X/Z, Y/Z^2)$ on $E_{\mathbf{Q}}$. The points $(1 : \delta : 0)$ and $(1 : -\delta : 0)$ on $\widehat{E}_{\mathbf{Q}}$ correspond to Ω_1 and Ω_2 on the desingularization of $\overline{E}_{\mathbf{Q}}$. There are no other points on $\widehat{E}_{\mathbf{Q}}$ with $Z = 0$.

With a slight abuse of notation, $\overline{E}_{\mathbf{Q}}(\mathbb{L})$, the set of $\mathbb{L}$-rational points on $\overline{E}_{\mathbf{Q}}$ is denoted by

$$E_{\mathbf{Q}}(\mathbb{L}) = \{(x, y) \in \mathbb{L}^2 \mid y^2 = dx^4 + 2ax^2 + 1\} \cup \{\Omega_1, \Omega_2\}$$

where Ω_1, Ω_2 are points at infinity.

Remark 2.3.3. The points Ω_1, Ω_2 on the desingularization of $\overline{E}_{\mathbf{Q}}$; the points $(1 : \delta : 0)$, $(1 : -\delta : 0)$ on $\widehat{E}_{\mathbf{Q}}$; and the points $(0 : \delta : 1 : 0)$, $(0 : -\delta : 1 : 0)$ on $\widetilde{E}_{\mathbf{Q}}$ are $\mathbb{L}$-rational if and only if d is a square in $\mathbb{L}$.

The curve $E_{\mathbf{Q}}$ is birationally equivalent over $\mathbb{K}$ to the Weierstrass curve

$$E_{\mathbf{W}}: v^2 = u(u^2 - 4au + 4a^2 - 4d)$$

via the maps

$$\psi: E_{\mathbf{Q}} \to E_{\mathbf{W}},\ (x,y) \mapsto \Big(\frac{2y+2}{x^2} + 2a, \frac{4y+4}{x^3} + \frac{4a}{x}\Big), \tag{2.5}$$

$$\phi: E_{\mathbf{W}} \to E_{\mathbf{Q}},\ (u,v) \mapsto \Big(2\frac{u}{v}, 2(u-2a)\frac{u^2}{v^2} - 1\Big). \tag{2.6}$$

It is trivial to check that $\phi \circ \psi = \mathrm{id}_{E_{\mathbf{Q}}}$ and $\psi \circ \phi = \mathrm{id}_{E_{\mathbf{W}}}$. The map ψ is regular at all points on $E_{\mathbf{Q}}$ except $(0,1)$ which corresponds to ∞ on $\overline{E}_{\mathbf{W}}$. At first glance, it may seem that ψ is not regular at $(0,-1)$. However, as explained in Definition A.2.2, it is possible to alter ψ to successfully map all points on $E_{\mathbf{Q}}$ except $(0,1)$. For instance, the point $(0,-1)$ can be sent to $E_{\mathbf{W}}$ with an alternative map given by

$$\psi': E_{\mathbf{Q}} \to E_{\mathbf{W}},\ (x,y) \mapsto \Big(\frac{2dx^2 + 2a(1+y)}{y-1}, \frac{4a(dx^2+2a) - 4d(1-y)}{(1-y)^2}x\Big). \tag{2.7}$$

The map ϕ is regular at all points on $E_{\mathbf{W}}$ except in one case. Before investigating this case observe that the point $(0,0)$ on $E_{\mathbf{W}}$ can be sent to $E_{\mathbf{Q}}$ with an alternative map given by

$$\phi': E_{\mathbf{W}} \to E_{\mathbf{Q}},\ (u,v) \mapsto \Big(\frac{2v}{(u-2a)^2 - 4d}, \frac{u^2 - 4(a^2-d)}{(u-2a)^2 - 4d}\Big). \tag{2.8}$$

The map ϕ is not regular at two points of the form (u,v) with $u \neq 0$ and $v = 0$. These exceptional points correspond to two points at infinity on the desingularization of $\overline{E}_{\mathbf{Q}}$. From Remark 2.3.3 and Definition A.2.2 it follows that ϕ is a morphism if d is a non-square in $\mathbb{K}$.

Every Weierstrass curve $v^2 = u^3 + a_2u^2 + a_4u$ is birationally equivalent over $\mathbb{K}$ to $E_{Q,(a_2^2-4a_4)/16,-a_2/4}$. The shape $v^2 = u^3 + a_2u^2 + a_4u$ covers all elliptic curves (over $\mathbb{K}$) having at least one point of order two. Therefore every elliptic curve of even order can be written in Jacobi quartic form. This extended model covers more isomorphism classes than the Jacobi model $E_{Q,k^2,-(k^2+1)/2}$.

Notes Jacobi and Abel worked on generalizing the results known for the circle $y^2 = (1-x^2)$ to the quartic curve $y^2 = (1-x^2)(1-k^2x^2)$. This form of elliptic curves is known as the Jacobi model. A Jacobi quartic curve given by $y^2 = x^4 + 2ax^2 + 1$ and its generalized version extended Jacobi quartic curve $y^2 = dx^4 + 2ax^2 + 1$ are studied in [BJ03a]. Billet and Joye remark that every elliptic curve of even order can be written in extended Jacobi quartic form.

2.3.3 Twisted Hessian form

Throughout this subsection, $\mathbb{K}$ denotes a *fixed* field with $\mathrm{char}(\mathbb{K}) \neq 2, 3$ and $\mathbb{L}$ an algebraic extension of $\mathbb{K}$. Let $a, d \in \mathbb{K}$. Assume that a has three distinct cube roots in $\mathbb{L}$ unless stated otherwise.

Definition 2.3.4. A twisted Hessian curve defined over $\mathbb{K}$ is the curve

$$E_{\mathbf{H},a,d}: ax^3 + y^3 + 1 = dxy.$$

This curve is non-singular if and only if $a(d^3 - 27a)^3 \neq 0$. The j-invariant is given by $d^3(d^3 + 216a)^3/(a(d^3 - 27a)^3) \in \mathbb{K}$. The projective closure of $E_{\mathbf{H}}$ is given by the equation

$$\overline{E}_{\mathbf{H},a,d} : \; aX^3 + Y^3 + Z^3 = dXYZ.$$

A point $(X : Y : Z)$ with $Z \neq 0$ on $\overline{E}_{\mathbf{H}}$ corresponds to the affine point $(X/Z, Y/Z)$ on $E_{\mathbf{H}}$. Fix $\alpha \in \mathbb{L}$ such that $\alpha^3 = a$. Fix $\omega \in \mathbb{L}$ such that $\omega^3 = 1$ and $\omega \neq 1$. The points $\Omega_1 = (-1/\alpha : 1 : 0)$, $\Omega_2 = (-\omega/\alpha : 1 : 0)$, and $\Omega_3 = (-1/(\omega\alpha) : 1 : 0)$ are non-singular. There are no other points on $\overline{E}_{\mathbf{H}}$ with $Z = 0$.

With a slight abuse of notation, $\overline{E}_{\mathbf{H}}(\mathbb{L})$, the set of $\mathbb{L}$-rational points on $\overline{E}_{\mathbf{H}}$ is denoted by

$$E_{\mathbf{H}}(\mathbb{L}) = \{(x, y) \in \mathbb{L}^2 \mid ax^3 + y^3 + 1 = dxy\} \cup \{\Omega_1, \Omega_2, \Omega_3\}$$

where $\Omega_1, \Omega_2, \Omega_3$ are points at infinity.

Remark 2.3.5. The points Ω_1, Ω_2, Ω_3 on $\overline{E}_{\mathbf{H}}$ are simultaneously $\mathbb{L}$-rational if and only if a has three distinct cube roots in $\mathbb{L}$. Only one of these three points is $\mathbb{L}$-rational if and only if a has a unique cube root in $\mathbb{L}$.

The curve $E_{\mathbf{H}}$ is birationally equivalent over $\mathbb{K}$ to the Weierstrass curve

$$E_{\mathbf{W}} : \; v^2 = u^3 - \tfrac{d^4+216da}{48}u + \tfrac{d^6-540d^3a-5832a^2}{864}$$

via the maps

$$\psi\colon E_{\mathbf{H}} \to E_{\mathbf{W}},\ (x, y) \mapsto \Big(\frac{(d^3 - 27a)x}{3(3 + 3y + dx)} - \frac{d^2}{4}, \frac{(d^3 - 27a)(1 - y)}{2(3 + 3y + dx)}\Big), \tag{2.9}$$

$$\phi\colon E_{\mathbf{W}} \to E_{\mathbf{H}},\ (u, v) \mapsto \Big(\frac{18d^2 + 72u}{d^3 - 12du - 108a + 24v}, 1 - \frac{48v}{d^3 - 12du - 108a + 24v}\Big). \tag{2.10}$$

It is trivial to check that $\phi \circ \psi = \mathrm{id}_{E_{\mathbf{H}}}$ and $\psi \circ \phi = \mathrm{id}_{E_{\mathbf{W}}}$. The map ψ is regular at all points on $E_{\mathbf{H}}$ except $(0, -1)$ which corresponds to ∞ on $\overline{E}_W$. The map ϕ is regular at all points on $E_{\mathbf{W}}$ except the three points of the form (u, v) with $d^3 - 12du - 108a + 24v = 0$ assuming that a is a cube in $\mathbb{K}$. These exceptional points correspond to the three points at infinity on $\overline{E}_{\mathbf{H}}$. From Remark 2.3.5 and Definition A.2.2 it follows that ϕ is a morphism if a is a non-cube in $\mathbb{K}$.

Notes Twisted Hessian form was introduced by Bernstein, Kohel, and Lange in [BKL09] as a generalization of Hessian curves.

2.3.4 Twisted Edwards form

Throughout this subsection, $\mathbb{K}$ denotes a *fixed* field of odd characteristic and $\mathbb{L}$ an algebraic extension of $\mathbb{K}$. Let $a, d \in \mathbb{K}$. Assume that both a and d are squares in $\mathbb{L}$ unless stated otherwise.

Definition 2.3.6. A twisted Edwards curve defined over $\mathbb{K}$ is the curve

$$E_{\mathbf{E},a,d} : \; ax^2 + y^2 = 1 + dx^2y^2.$$

This curve is non-singular if and only if $ad(a-d) \neq 0$. The j-invariant is given by $16(a^2 + 14ad + d^2)^3/(ad(a-d)^4) \in \mathbb{K}$. The projective closure of $E_{\mathbf{E}}$ is given by the equation

$$\overline{E}_{\mathbf{E},a,d} : \; aX^2Z^2 + Y^2Z^2 = Z^4 + dX^2Y^2.$$

A point $(X : Y : Z)$ with $Z \neq 0$ on $\overline{E}_{\mathbf{E}}$ corresponds to the affine point $(X/Z, Y/Z)$ on $E_{\mathbf{E}}$. The points $(0 : 1 : 0)$ and $(1 : 0 : 0)$ on $\overline{E}_{\mathbf{E}}$ are singular even if $ad(a-d) \neq 0$. Using the standard "blow-up" techniques (see [Ful69, §7.3]) the singularities can be resolved. The resolution of singularities produces four points (see [BBJ+08]) which are labeled as Ω_1, Ω_2, Ω_3, and Ω_4. It is convenient to note here that a single "blow-up" for each of the points $(0 : 1 : 0)$ and $(1 : 0 : 0)$ is necessary and sufficient to resolve the singularities. There are no other points on $\overline{E}_{\mathbf{E}}$ with $Z = 0$.

A way of removing the singularities is by using the projective curve given by the equations

$$\widetilde{E}_{\mathbf{E},a,d} : \; aX^2 + Y^2 = Z^2 + dT^2, XY = TZ.$$

A point $(X : Y : T : Z)$ with $Z \neq 0$ on $\widetilde{E}_{\mathbf{E}}$ corresponds to the affine point $(X/Z, Y/Z)$ on $E_{\mathbf{E}}$. Fix $\alpha, \delta \in \mathbb{K}$ such that $\alpha^2 = a$ and $\delta^2 = d$. The points $(\delta : 0 : \alpha : 0)$, $(-\delta : 0 : \alpha : 0)$, $(0 : \delta : 1 : 0)$, and $(0 : -\delta : 1 : 0)$ on $\widetilde{E}_{\mathbf{E}}$ correspond to Ω_1, Ω_2, Ω_3, and Ω_4 on the desingularization of $\overline{E}_{\mathbf{E}}$.

With a slight abuse of notation, $\overline{E}_{\mathbf{E}}(\mathbb{L})$, the set of $\mathbb{L}$-rational points on $\overline{E}_{\mathbf{E}}$ is denoted by

$$E_{\mathbf{E}}(\mathbb{L}) = \{(x, y) \in \mathbb{L}^2 \mid ax^2 + y^2 = 1 + dx^2y^2\} \cup \{\Omega_1, \Omega_2, \Omega_3, \Omega_4\}$$

where $\Omega_1, \Omega_2, \Omega_3, \Omega_4$ are points at infinity.

Remark 2.3.7. The points Ω_1, Ω_2, Ω_3, Ω_4 on the desingularization of $\overline{E}_{\mathbf{E}}$; and the points $(\delta : 0 : \alpha : 0)$, $(-\delta : 0 : \alpha : 0)$, $(0 : \delta : 1 : 0)$, $(0 : -\delta : 1 : 0)$ on $\widetilde{E}_{\mathbf{E}}$ are $\mathbb{L}$-rational if and only if both a and d are squares in $\mathbb{L}$.

Theorem 2.3.8 (Bernstein *et al.*, [BBJ+08])**.** *Every twisted Edwards curve over $\mathbb{K}$ is birationally equivalent over $\mathbb{K}$ to the Montgomery curve given by $By^2 = x^3 + Ax^2 + x$ for some $A, B \in \mathbb{K}$. Conversely, every Montgomery curve over $\mathbb{K}$ is birationally equivalent over $\mathbb{K}$ to a twisted Edwards curve.*

From Theorem 2.3.8, it can be deduced that every twisted Edwards curve is birationally equivalent over $\mathbb{K}$ to the Weierstrass curve

$$E_{\mathbf{W}} : \; = u^3 + 2(a+d)u^2 + (a-d)^2u$$

via the maps

$$\psi : E_{\mathbf{E}} \to E_{\mathbf{W}}, \; (x, y) \mapsto \Big((1+y)^2\frac{1-dx^2}{x^2}, 2(1+y)^2\frac{1-dx^2}{x^3}\Big), \tag{2.11}$$

$$\phi : E_{\mathbf{W}} \to E_{\mathbf{E}}, \; (u, v) \mapsto \Big(2\frac{u}{v}, \frac{u-a+d}{u+a-d}\Big). \tag{2.12}$$

It is trivial to check that $\phi \circ \psi = \mathrm{id}_{E_{\mathbf{E}}}$ and $\psi \circ \phi = \mathrm{id}_{E_{\mathbf{W}}}$. The map ψ is regular at all points on $E_{\mathbf{E}}$ except $(0, 1)$ which corresponds to ∞ on $\overline{E}_W$. At first glance, it may seem that ψ is

not regular at $(0,-1)$. However, as explained in Definition A.2.2, it is possible to alter ψ to successfully map all points on $E_{\mathbf{E}}$ except $(0,1)$. For instance, the point $(0,-1)$ can be sent to $E_{\mathbf{W}}$ with an alternative map given by

$$\psi' \colon E_{\mathbf{E}} \to E_{\mathbf{W}},\ (x,y) \mapsto \Big((a-d)\frac{1+y}{1-y}, 2(a-d)\frac{a-dy^2}{(1-y)^2}x\Big). \tag{2.13}$$

The map ϕ is regular at all points on $E_{\mathbf{W}}$ except in two cases. Before investigating these two cases observe that the point $(0,0)$ on $E_{\mathbf{W}}$ can be sent to $E_{\mathbf{E}}$ with an alternative map given by

$$\phi' \colon E_{\mathbf{W}} \to E_{\mathbf{E}},\ (u,v) \mapsto \Big(\frac{2v}{(u-2a)^2-4d}, \frac{u^2-4(a^2-d)}{(u-2a)^2-4d}\Big). \tag{2.14}$$

The map ϕ is not regular at two points of the form (u,v) with $u \neq 0$ and $v = 0$. These exceptional points correspond to two points at infinity on the desingularization of $\overline{E}_{\mathbf{E}}$. The map ϕ is not regular at two points of the form (u,v) with $u = d - a$. These exceptional points correspond to the other two points at infinity on the desingularization of $\overline{E}_{\mathbf{E}}$. From Remark 2.3.7 and Definition A.2.2 it follows that ϕ is a morphism if both d and ad are non-squares in $\mathbb{K}$.

Notes Building on the historical works of Euler and Gauss, Edwards introduced the normal form $x^2+y^2 = c^2(1+x^2y^2)$ of elliptic curves together with an explicit addition law on this curve in [Edw07]. Edwards also showed that every elliptic function field is equivalent to the function field of this curve for some c provided that $\mathbb{K}$ is algebraically closed. In [BL07b], Bernstein and Lange introduced Edwards form elliptic curves defined by $x^2 + y^2 = c^2(1 + dx^2y^2)$ where $c, d \in \mathbb{K}$ with $cd(1 - dc^4) \neq 0$, covering more curves than original Edwards curves when $\mathbb{K}$ is finite. Twisted Edwards form was introduced by Bernstein *et al.* in [BBJ+08] as a generalization of Edwards curves. The facts about the resolution of singularities or the points at infinity or the coverage of these curves or the group structure have already been studied in different generalities in [Edw07], [BL07b], [BBJ+08], and [BBLP08]. Also see [BL07a].

2.3.5 Twisted Jacobi intersection form

Throughout this subsection, $\mathbb{K}$ denotes a *fixed* field of odd characteristic and $\mathbb{L}$ an algebraic extension of $\mathbb{K}$. Let $b, a \in \mathbb{K}$. Assume that both $-a$ and $-b$ are squares in $\mathbb{L}$ unless stated otherwise.

Definition 2.3.9. A twisted Jacobi intersection curve defined over $\mathbb{K}$ is the curve

$$E_{\mathbf{I},b,a} \ :\ bs^2 + c^2 = 1, as^2 + d^2 = 1.$$

This curve is non-singular if and only if $ab(a-b) \neq 0$. The j-invariant is given by $256(a^2 - ab + b^2)^3/(ab(a-b))^2 \in \mathbb{K}$. The projective closure of $E_{\mathbf{I}}$ is given by the equations

$$\overline{E}_{\mathbf{I},b,a} \ :\ bS^2 + C^2 = Z^2, aS^2 + D^2 = Z^2.$$

A point $(S\colon C\colon D\colon Z)$ with $Z \neq 0$ on $\overline{E}_{\mathbf{I}}$ corresponds to the affine point $(S/Z, C/Z, D/Z)$

on $E_{\mathbf{I}}$. Fix $\alpha, \beta \in \mathbb{K}$ such that $\alpha^2 = -a$ and $\beta^2 = -b$. The points $\Omega_1 = (1\colon \beta\colon \alpha\colon 0)$, $\Omega_2 = (1\colon -\beta\colon \alpha\colon 0)$, $\Omega_3 = (1\colon \beta\colon -\alpha\colon 0)$, and $\Omega_4 = (1\colon -\beta\colon -\alpha\colon 0)$ are non-singular. There are no other points on $\overline{E}_{\mathbf{I}}$ with $Z = 0$.

With a slight abuse of notation, $\overline{E}_{\mathbf{I}}(\mathbb{L})$, the set of $\mathbb{L}$-rational points on $\overline{E}_{\mathbf{I}}$ is denoted by

$$E_{\mathbf{I}}(\mathbb{L}) = \{(s, c, d) \in \mathbb{L}^3 \mid bs^2 + c^2 = 1, as^2 + d^2 = 1\} \cup \{\Omega_1, \Omega_2, \Omega_3, \Omega_4\}$$

where $\Omega_1, \Omega_2, \Omega_3, \Omega_4$ are the points at infinity.

Remark 2.3.10. The points Ω_1, Ω_2, Ω_3, Ω_4 on $\overline{E}_{\mathbf{I}}$ are $\mathbb{L}$-rational if and only if both $-a$ and $-b$ are squares in $\mathbb{L}$.

The curve $E_{\mathbf{I}}$ is birationally equivalent over $\mathbb{K}$ to the Weierstrass curve

$$E_{\mathbf{W}}\colon\ v^2 = u(u-a)(u-b)$$

via the maps

$$\psi\colon E_{\mathbf{I}} \to E_{\mathbf{W}},\ (s, c, d) \mapsto \Big(\frac{(1+c)(1+d)}{s^2}, -\frac{(1+c)(1+d)(c+d)}{s^3}\Big), \tag{2.15}$$

$$\phi\colon E_{\mathbf{W}} \to E_{\mathbf{I}},\ (u, v) \mapsto \Big(\frac{2v}{ab-u^2}, 2u\frac{b-u}{ab-u^2} - 1, 2u\frac{a-u}{ab-u^2} - 1\Big). \tag{2.16}$$

It is trivial to check that $\phi \circ \psi = \mathrm{id}_{E_{\mathbf{I}}}$ and $\psi \circ \phi = \mathrm{id}_{E_{\mathbf{W}}}$. The map ψ is regular at all points on $E_{\mathbf{I}}$ except $(0, 1, 1)$ which corresponds to ∞ on $\overline{E}_W$. At first glance, it may seem that ψ is not regular at some other points with zero s-coordinate: $(0, -1, 1)$, $(0, 1, -1)$, and $(0, -1, -1)$. However, as explained in Definition A.2.2, it is possible to alter ψ to successfully map all points except $(0, 1, 1)$. For instance, the points $(0, -1, 1)$, $(0, 1, -1)$, $(0, -1, -1)$ can be sent to $E_{\mathbf{W}}$ with an alternative map given by

$$\psi'\colon E_{\mathbf{I}} \to E_{\mathbf{W}},\ (s, c, d) \mapsto \Big(b\frac{1+d}{1-c}, b\frac{a(1-c) - b(1+d)}{(1-c)^2}s\Big), \tag{2.17}$$

$$\psi''\colon E_{\mathbf{I}} \to E_{\mathbf{W}},\ (s, c, d) \mapsto \Big(a\frac{1+c}{1-d}, a\frac{b(1-d) - a(1+c)}{(1-d)^2}s\Big). \tag{2.18}$$

The map ϕ is regular at all points on $E_{\mathbf{W}}$ except the points of the form (u, v) with $u^2 = ab$. These exceptional points correspond to the four points at infinity on $\overline{E}_{\mathbf{I}}$ if ab is a square in $\mathbb{K}$. From Remark 2.3.10 and Definition A.2.2 it follows that ϕ is a morphism if ab is a non-square in $\mathbb{K}$.

Every elliptic curve having three points of order 2 is birational to a twisted Jacobi intersection curve.

Notes An elliptic curve can be represented generically as the intersection of two quadrics [Was03, §2.5.4]. See [LS01], [CC86], and [BJ03a] for cryptographic applications of Jacobi intersection form.

2.3.6 Coverage of different forms

Assume that $\mathbb{K}$ is a finite field with q elements. Further assume that $\mathrm{char}(\mathbb{K}) \neq 2, 3$. Table 2.1 and Table 2.2 summarize the coverage of selected forms over $\mathbb{K}$. Most of the results are collected

from approximations or statistics in the literature, cf. [BBJ+08], [GM00], [CV09]. For some of the entries such as $y^2 = x^3 + ax + b$ and $x^2 + y^2 = 1 + dx^2y^2$, the exact numbers of isomorphism classes are known. For instance, see [RS09]. The first columns of both tables describe the elliptic curve by its defining equation. The second and third columns estimate the number of distinct j-invariants covered by these forms. This information is typically used for a classification of elliptic curves over $\overline{\mathbb{K}}$. Note that equal j-invariants do not guarantee an isomorphism over $\mathbb{K}$ if $\mathbb{K}$ is not algebraically closed. The last column in each table estimates the number of distinct isomorphism classes for each form. There are approximately $2q$ isomorphism classes of all elliptic curves over $\mathbb{K}$, cf. [HMV03]. The rows are sorted with respect to the last column in descending order. The ordering of entries in the first column of Table 2.1 is also used for the ordering of the subsections of §2.3.

Table 2.1: Statistics on the coverage of some forms with two curve constants.

Curve equation	**Condition**	**# of isomorphism classes** ($\approx$)	
$y^2 = x^3 + ax + b$	-	$2q$	$2.00q$
$y^2 = dx^4 + 2ax^2 + 1$	-	$4q/3$	$1.33q$
$ax^3 + y^3 + 1 = dxy$	$q \equiv 1 \mod 3$ $q \equiv 2 \mod 3$	$0.76q$ q	$0.88q$
$ax^2 + y^2 = 1 + dx^2y^2$	$q \equiv 1 \mod 4$ $q \equiv 3 \mod 4$	$5q/6$ $3q/4$	$0.79q$
$bs^2 + c^2 = 1, as^2 + d^2 = 1$	-		$0.33q$

Table 2.2: Statistics on the coverage of some forms with a single curve constant.

Curve equation	**Condition**	**# of isomorphism classes** ($\approx$)	
$y^2 = dx^4 \pm x^2 + 1$	-		$0.80q$
$y^2 = x^3 - 3x + b$	-	$3q/4$	$0.75q$
$\pm x^2 + y^2 = 1 + dx^2y^2$	$q \equiv 1 \mod 4$ $q \equiv 3 \mod 4$	$2q/3$ $3q/4$	$0.71q$
$y^2 = -x^4 + 2ax^2 + 1$	-		$0.66q$
$\pm x^3 + y^3 + 1 = dxy$	$q \equiv 1 \mod 3$ $q \equiv 2 \mod 3$	$q/6$ q	$0.58q$
$y^2 = x^4 + 2ax^2 + 1$	-		$0.31q$
$\pm s^2 + c^2 = 1, as^2 + d^2 = 1$	-		$0.31q$

Note that the standardized NIST curves are only expressible in short Weiertsrass form $y^2 = x^3 - 3x + b$ over a standardized base field. The scope of this thesis is not limited to a

particular class of elliptic curves and therefore all of the listed forms are considered in detail.

2.4 Scalar multiplication

Let E be an elliptic curve over $\mathbb{K}$. A point P can be multiplied by a scalar as $[k]P = \sum_{i=1}^{k} P$. Efficient computation of $[k]P$ is an active research area in ECC implementations since a majority of the computational power is spent on this operation. A standard double-and-add technique to compute k-folds of P is presented in Algorithm 2.4.1.

Algorithm 2.4.1: Left-to-right binary method for scalar multiplication

input : $k = (k_{t-1}, \ldots, k_1, k_0)_2$, $P \in E(\mathbb{K})$.
output: $Q \leftarrow [k]P$.

```
1 Q ← O.
2 for i = t − 2 to 0 do
3     Q ← [2]Q.
4     if k_i = 1 then
5         Q ← Q + P.
6     end
7 end
8 return Q.
```

Although scalar multiplications are not the main topic of this thesis, they will come into play in Chapter 6 for performing efficiency oriented benchmarks of various other forms of elliptic curves over finite fields. In particular, a windowed version of Algorithm 2.4.1 will be used to decrease the number of point additions. For details on windowed scalar multiplications see [HMV03] and [BSS99]. In windowed scalar multiplication algorithms, the window width w typically controls the overall performance and the space needed for precomputation. The w-LtoR integer recoding algorithm in [Ava05] will also be incorporated to scan the digits of the scalar k from the most significant bit to the least. This algorithm allows the integer recoding to be performed on-the-fly as the steps of scalar multiplication progress.

2.5 Conclusion

This chapter reviewed elliptic curves with an emphasis on different ways of representing them. Although designed to serve as a background chapter, §2.3 of this chapter also provided contributions. §2.3 presented five forms of elliptic curves together with their transformation to the Weierstrass form via worked maps. The main contribution is that several fragmented results in the literature are brought together under a strictly followed notation using similar treatment for each case. The presented maps are further studied in detail to state all exceptional points where computational problems arise. §2.3 also contained a generalization for Jacobi intersection form to cover a slightly wider class of curves. Finally, §2.4 defined the scalar multiplication operation which will be needed in Chapter 6.

Chapter 3

A toolbox for group laws

Every elliptic curve is endowed with a group law[1] which turns the point set of an elliptic curve into an abelian group. The binary operation on this group is named *the addition law*. In the case of elliptic curves, the addition law can be rationally expressed.

The group operations are typically performed on a short Weiertrass curve. The reason is that every elliptic curve can be shaped into a Weierstrass form plane curve by preserving the initial group structure. As a consequence, a description of the group law for other forms of elliptic curves has not received as much attention as the Weierstrass form. On the other hand, other forms of elliptic curves might allow a more efficient group law than what can be achieved with the short Weierstrass form. Indeed, there have been proposals in the literature supporting this claim. In particular, see [Mon87], [LS01], [DI06], and [BL07b]. Therefore, it is worth studying the topic in detail to search for more results. At this stage, it is helpful to have a toolbox which makes the desired investigation easier.

This chapter brings together several computational tools which are beneficial in developing efficient ways of computation on elliptic curves. These tools will be central to Chapters 4, 5, and 7. The approach will be algebraic rather than geometric, and the emphasis lies on the development of computer algebra routines to derive the desired group laws. In this direction, §3.2 outlines an automated method to derive the group laws on elliptic curves and provides case studies. §3.3 revisits rational simplification techniques by Monagan and Pearce [MP06] in the context of efficient automated group law derivation to detect useful formulae. §3.4 shows how to validate worked formulae in MAGMA [BCP97] and Maple [MAP08] systems based on a similar strategy from [BL07a]. §3.5 provides a method to derive alternative formulae for point doubling and addition on elliptic curves. §3.6 brings together the most commonly used operation counting patterns to further optimize sequences of operations by trading multiplications with cheaper operations such as squarings or additions in field $\mathbb{K}$. Conclusions are drawn in §3.7.

[1] unique up to a choice of a rational point on the curve.

3.1 Computer Algebra Systems

Computer algebra systems have evolved very rapidly starting from 1970's with applications to different areas of science such as computer science, physics, mathematics. Relevant to the purpose of this thesis, the main breakthrough is the construction of Gröbner basis theory which have found many applications over the past fourty years and have been widely implemented.

Today's computer algebra systems contain high level routines that makes it possible to accomplish the aformentioned goal of this chapter in a succinct manner. In particular, this thesis uses MAGMA [BCP97] and Maple [MAP08] systems. Both system can compute the Weierstrass form of an elliptic curve. This capability will later be used in §3.2. MAGMA has useful functions to manipulate algebraic varieties in an arbitrary space. MAGMA is also supportive for numeric exercises such as point counting or finding the order of points on an elliptic curve. MAPLE includes simple map composition mechnanisms which shrinks the code size. Many ready-to-run examples are provied to show how the desired computations can be performed in a few lines of computer algebra commands.

3.2 Automated derivations

This section outlines how to derive the group law on an elliptic curve embedded in a suitable affine space. The method simply uses Riemann-Roch computations, see Appendix A. In a rough sense, this can be viewed as a "conversion" of the well known group law for Weierstrass curves to the corresponding group law on a birationally equivalent curve using rational mappings.

The following theorem shows how to find the affine part of the addition law on an arbitrary elliptic curve.

Theorem 3.2.1. *Let $W/\mathbb{K}$ and $M/\mathbb{K}$ be affine curves. Assume that $\overline{W}$ and $\overline{M}$, each with a fixed $\mathbb{K}$-rational point, are elliptic curves. Assume that W and M are birationally equivalent over $\mathbb{K}$. Let $\phi : W \to M$ and $\psi : M \to W$ be maps such that $\phi \circ \psi$ and $\psi \circ \phi$ are equal to the identity maps id_M and id_W, respectively. Let $+_W : W \times W \to W$ be the affine part of the unique addition law on $\overline{W}$. The affine part of the unique addition law on $\overline{M}$ is given by the compositions*

$$+_M \quad = \quad \phi \circ +_W \circ (\psi \times \psi). \tag{3.1}$$

Before giving the proof, the following lemma will be useful.

Lemma 3.2.2. *If two irreducible algebraic curves M and W are birationally equivalent then $\mathbb{K}(W) \cong \mathbb{K}(M)$ and $\mathbb{K}(W \times W) \cong \mathbb{K}(M \times M)$.*

Proof. For the isomorphism $\mathbb{K}(M) \cong \mathbb{K}(W)$ see the proof of Theorem 10 in [CLO07, §5.5]. The isomorphism $\psi^* : \mathbb{K}(W) \to \mathbb{K}(M)$ is constructed via the pullback map $\psi^*(f) = f \circ \psi$ where $f \in \mathbb{K}(W)$. In the same fashion, the map $\psi^* \times \psi^* \colon \mathbb{K}(W \times W) \to \mathbb{K}(M \times M)$ given by $(\psi^* \times \psi^*)(g) = g \circ (\psi \times \psi)$ where $g \in \mathbb{K}(W \times W)$, is an isomorphism by the universal property of products, cf. [Mus01, Theorem 28.5]. □

Proof of Theorem 3.2.1. Let P_1 and P_2 be points on M. By the definition of ϕ, ψ, and $+_W$, the following equalities hold

$$\begin{aligned} P_1 +_M P_2 &= (\mathrm{id}_M)(P_1 +_M P_2) = (\phi \circ \psi)(P_1 +_M P_2) = \phi(\psi(P_1 +_M P_2)) \\ &= \phi(\psi(P_1) +_W \psi(P_2)) \\ &= (\phi \circ +_W \circ (\psi \times \psi))(P_1, P_2) \text{ if defined.} \end{aligned} \tag{3.2}$$

The construction (3.2) works for all but finitely many pair of points. The rest of the claim follows from Lemma 3.2.2 and from the unicity of the addition law. □

The negation law can be computed accordingly.

For simplicity assume that W is in Weierstrass form

$$E_{\mathbf{W},a_1,a_3,a_2,a_4,a_6} : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

which is a nonsingular model for W. Assume also that the rational mapping $+_W$ defined by

$$\begin{aligned} +_W : W \times W &\to W \\ (P_1, P_2) &\mapsto P_1 + P_2, \end{aligned}$$

gives the group law. By Theorem 2.2.3 of Chapter 2, $+_W$ is a morphism, i.e. the group law is defined for all of $W \times W$. Noting that $+_W$ is already known explicitly for W, determining $+_M$ depends only on the definition of W, ϕ and ψ. Therefore, these definitions are crucial to have the automated derivation work. In the general case, Riemann-Roch computations always guarantee a transformation to a non-singular almost-Weierstrass form given by $c_0y^2 + a_1xy + a_3y = c_2x^3 + a_2x^2 + a_4x + a_6$ with $c_0, c_2, a_i \in \mathbb{K}$, cf. [Sil94, Theorem 3.1]. After this step, Nagell reduction can be applied (partially) to rescale c_0 and c_2 to 1, cf. Algorithm 7.4.10 in [Coh93]. An open source implementation by Bruin is available in MAGMA, see in particular the `CrvEll` package. An alternative method based on integral basis computations is given in [vH03]. An open source implementation by Hoeij is available in Maple, see in particular the `algcurves` package. The latter implementation requires M to be a plane curve. Also see [DL06] for more applications on SINGULAR [GPS09].

Next, several examples will be presented to show how to automate the group law derivation. This section is limited to examples on MAGMA and Maple. On the other hand, it should be possible to write similar scripts in other computer algebra systems.

Example 3.2.3. The following MAGMA script constructs a cubic curve defined by $M : bx^2y + xy^2 + ax + y = 0$; ensures that the curve is of genus 1; computes suitable $W : v^2 = u(u - 4/b)(u - 4a/b^2)$, $\phi = (2v/(4u - bu^2), b^2v/(2bu - 8))$, $\psi = (-4y/(b^2x), -8y(bx + y)/(b^3x))$; and verifies that M and W are birational:

```
> K<a,b>:=FieldOfFractions(PolynomialRing(Rationals(),2));
> SM<x,y>:=AffineSpace(K,2);
> M:=ProjectiveClosure(Curve(SM,[b*x^2*y+x*y^2+a*x+y]));
> assert Genus(M) eq 1;
> W,psi:=EllipticCurve(M,M![0,1,0]);
```

```
> phi:=Inverse(psi);
> assert Expand(phi*psi) eq IdentityMap(W);
> assert Expand(psi*phi) eq IdentityMap(M);
```

The definition of the maps ϕ and ψ depends on the point $(0:1:0)$ represented by `M![0,1,0]` in the script. However, the group structure of M is independent of this choice, cf. [Sti93, Ch.VI].

Example 3.2.4. The following MAGMA script constructs a curve M in twisted Jacobi intersection form; ensures that the curve is of genus 1; computes suitable $W\colon v^2 = u^3 + b^4(2a-b)/(a^2(a-b)^2)u^2 + b^8/(a^3(a-b)^3)u$, $\phi = (-2(a-b)^2a^2(ua^3 - uba^2 + b^4)b^2v/(v^2a^9 - 4v^2ba^8 + 6v^2b^2a^7 + a^6b^4u^2 - 4a^6v^2b^3 - 2a^5b^5u^2 + a^5v^2b^4 + b^6u^2a^4 + 2b^8ua^3 - 2b^9ua^2 + b^{12}), (v^2a^9 - 4v^2ba^8 + 6v^2b^2a^7 + a^6b^4u^2 - 4a^6v^2b^3 - 4a^5b^5u^2 + a^5v^2b^4 + 5b^6u^2a^4 - 2a^3b^7u^2 + 2b^8ua^3 - 4b^9ua^2 + 2b^{10}ua + b^{12})/(v^2a^9 - 4v^2ba^8 + 6v^2b^2a^7 + a^6b^4u^2 - 4a^6v^2b^3 - 2a^5b^5u^2 + a^5v^2b^4 + b^6u^2a^4 + 2b^8ua^3 - 2b^9ua^2 + b^{12}), -(v^2a^9 - 4v^2ba^8 + 6v^2b^2a^7 - 4a^6v^2b^3 - a^6b^4u^2 + a^5v^2b^4 + 2a^5b^5u^2 - b^6u^2a^4 - 2b^8ua^3 + 2b^9ua^2 - b^{12})/(v^2a^9 - 4v^2ba^8 + 6v^2b^2a^7 + a^6b^4u^2 - 4a^6v^2b^3 - 2a^5b^5u^2 + a^5v^2b^4 + b^6u^2a^4 + 2b^8ua^3 - 2b^9ua^2 + b^{12}))$, $\psi = (b^4(1-c)/(a(a-b)(ac+db-a+b)), -b^7s/(a^2(a-b)^2(ac+db-a+b)))$; and verifies that M and W are birational:

```
> K<a,b>:=FieldOfFractions(PolynomialRing(Rationals(),2));
> SM<s,c,d>:=AffineSpace(K,3);
> M:=ProjectiveClosure(Curve(SM,[b*s^2+c^2-1,a*s^2+d^2-1]));
> assert Genus(M) eq 1;
> W,psi:=EllipticCurve(M,M![0,1,1,1]);
> phi:=Inverse(psi);
> assert Expand(phi*psi) eq IdentityMap(W);
> assert Expand(psi*phi) eq IdentityMap(M);
```

This example uses a more general view of algebraic curves. In particular, an algebraic curve is the locus of points in $\mathbb{K}^n$ (affine n-space over $\mathbb{K}$), determined by independent polynomial functions $h_1, \ldots, h_{n-1}$ in n variables with coefficients in $\mathbb{K}$. The curve is defined by setting each $h_i = 0$. In this script, $n = 3$. The polynomial functions are defined as $h_1(s,c,d) = bs^2 + c^2 - 1$ and $h_2(s,c,d) = as^2 + d^2 - 1$ where a, b are in $\mathbb{K}$.

Example 3.2.5. Since every algebraic curve is of dimension 1, it is always possible to eliminate all but two of the variables (using hand calculations or resultants or Gröbner basis eliminations). This approach produces a birationally equivalent plane curve (up to isomorphism) defined by a single equation in two variables. Note that singularities may be introduced in this process. For example, the following Maple script constructs the quartic curve $M\colon y^2 = abx^4 - (a+b)x^2 + 1$ and computes suitable $W\colon v^2 = u^3 - (a^2 + 14ab + b^2)u^2/3 + 2(a^3 - 33ab^2 - 33a^2b + b^3)u/27$, $\phi = (18v/(a^2 - 34ab + b^2 - 6ua - 6ub + 9u^2), (-26ab - 12ua + 5a^2 - 12ub + 5b^2 - 9u^2)/(a^2 - 34ab + b^2 - 6ua - 6ub + 9u^2))$, $\psi = ((x^2b - 6 + x^2a + 6y)/(3x^2), (4 - 2x^2a - 2x^2b - 4y)/x^3)$:

```
> with(algcurves):
> t:=Weierstrassform(y^2-(a*b*x^4-(a+b)*x^2+1),x,y,u,v):
> W:=t[1]: psi:=(t[2],t[3]): phi:=(t[4],t[5]):
```

Examples 3.2.3, 3.2.4, and 3.2.5 showed how to prepare for the computation of the right-hand side of (3.1). It is now easy to derive group laws for elliptic curves defined by various

different type of equations with the help of computer algebra. The following examples show how to compute $+_M$ in (3.1).

Example 3.2.6. Consider the derivation of the group law for twisted Jacobi intersection curve $E_{I,b,a}$. The coordinate functions $s_1, c_1, d_1, s_2, c_2, d_2$ for $E_{I,b,a} \times E_{I,b,a}$ are labeled as `s1`, `c1`, `d1`, `s2`, `c2`, `d2`. The coordinate functions u_1, v_1, u_2, v_2 for $E_{W,0,0,-a-b,ab,0} \times E_{W,0,0,-a-b,ab,0}$ are labeled as `u1`, `v1`, `u2`, `v2`. The following Maple script defines `W`, `C`, `phi`, `psi`, `psipsi` to represent $W = E_{W,0,0,-a-b,ab,0}$, $M = E_{I,b,a}$, ϕ, ψ, and $\psi \times \psi$, respectively.

```
> a1:=0: a3:=0: a2:=-a-b: a4:=a*b: a6:=0:
> W:=(u,v)->(v^2+a1*u*v+a3*v-(u^3+a2*u^2+a4*u+a6)):
> C:=(s,c,d)->(b*s^2+c^2-1,a*s^2+d^2-1):
> phi:=(u,v)->(-2*v/(u^2-a*b),(u^2-2*b*u+a*b)/(u^2-a*b),(u^2-2*a*u+a*b)/(u^2-a*b)):
> psi:=(s,c,d)->(a*(1+c)/(1-d),-a^2*s*(1+c)*(c+d)/((1-d)^2*(1+d))):
> psipsi:=(s1,c1,d1,s2,c2,d2)->(psi(s1,c1,d1),psi(s2,c2,d2)):
```

In this example, W, ϕ, and ψ are copied from §2.3.5 of Chapter 2 to match a standard choice of the identity element. The example continues to find specialized negation, doubling and addition formulae.

Negation The following Maple script derives the corresponding negation formulae. The first line defines the negation formulae for W and the second line applies a simpler version of Theorem 3.2.1.

```
> negW:=(u1,v1)->(u1,-v1-a1*u1-a3):
> negM:=phi(negW(psi(s1,c1,d1))):
```

The negation formulae stored in `negM` are given by $-(s_1, c_1, d_1) = (s_3, c_3, d_3)$ where

$$s_3 = -2a^2 s_1(1+c_1)(c_1+d_1)/((1-d_1)^2(1+d_1)(a^2(1+c_1)^2/(1-d_1)^2 - ab)),$$

$$c_3 = (a^2(1+c_1)^2/(1-d_1)^2 - 2ab(1+c_1)/(1-d_1) + ab)/(a^2(1+c_1)^2/(1-d_1)^2 - ab),$$

$$d_3 = (a^2(1+c_1)^2/(1-d_1)^2 - 2a^2(1+c_1)/(1-d_1) + ab)/(a^2(1+c_1)^2/(1-d_1)^2 - ab).$$

Doubling Similarly, the following Maple script derives the corresponding doubling formulae. The first line defines the doubling formulae for W and the second line applies a simpler version of Theorem 3.2.1.

```
> dblW:=(u1,v1)->(((3*u1^2+2*a2*u1+a4-a1*v1)/(2*v1+a1*u1+a3))^2+a1*(3*u1^2+2*a2*u1+a4- \
                  a1*v1)/(2*v1+a1*u1+a3)-a2-2*u1,(3*u1^2+2*a2*u1+a4-a1*v1)/(2*v1+a1*u1+ \
                  a3)*(u1-(((3*u1^2+2*a2*u1+a4-a1*v1)/(2*v1+a1*u1+a3))^2+a1*(3*u1^2+ \
                  2*a2*u1+a4-a1*v1)/(2*v1+a1*u1+a3)-a2-2*u1))-v1-a1*u3-a3):
> dblM:=phi(dblW(psi(s1,c1,d1))):
```

The doubling formulae stored in `dblM` are given by $[2](s_1, c_1, d_1) = (s_3, c_3, d_3)$ where

$$\begin{aligned} s_3 = {} & -2(-(1/2)(3a^2(1+c_1)^2/(1-d_1)^2+2(-a-b)a(1+c_1)/(1-d_1)+ab)(1-d_1)^2(1+d_1)(3a(1+c_1)/(1-d_1)- \\ & (1/4)(3a^2(1+c_1)^2/(1-d_1)^2+2(-a-b)a(1+c_1)/(1-d_1)+ab)^2(1-d_1)^4(1+d_1)^2/(a^4s_1^2(1+c_1)^2(c_1+ \\ & d_1)^2) - a - b)/(a^2s_1(1+c_1)(c_1+d_1)) + a^2s_1(1+c_1)(c_1+d_1)/((1-d_1)^2(1+d_1)))/(((1/4)(3a^2(1+ \\ & c_1)^2/(1-d_1)^2 + 2(-a-b)a(1+c_1)/(1-d_1)+ab)^2(1-d_1)^4(1+d_1)^2/(a^4s_1^2(1+c_1)^2(c_1+d_1)^2) - \\ & 2a(1+c_1)/(1-d_1)+a+b)^2 - ab), \end{aligned}$$

$$c_3 = (((1/4)(3a^2(1+c_1)^2/(1-d_1)^2 + 2(-a-b)a(1+c_1)/(1-d_1) + ab)^2(1-d_1)^4(1+d_1)^2/(a^4s_1^2(1+c_1)^2(c_1+d_1)^2) - 2a(1+c_1)/(1-d_1) + a + b)^2 - 2b((1/4)(3a^2(1+c_1)^2/(1-d_1)^2 + 2(-a-b)a(1+c_1)/(1-d_1) + ab)^2(1-d_1)^4(1+d_1)^2/(a^4s_1^2(1+c_1)^2(c_1+d_1)^2) - 2a(1+c_1)/(1-d_1) + a + b) + ab)/(((1/4)(3a^2(1+c_1)^2/(1-d_1)^2 + 2(-a-b)a(1+c_1)/(1-d_1) + ab)^2(1-d_1)^4(1+d_1)^2/(a^4s_1^2(1+c_1)^2(c_1+d_1)^2) - 2a(1+c_1)/(1-d_1) + a + b)^2 - ab),$$

$$d_3 = (((1/4)(3a^2(1+c_1)^2/(1-d_1)^2 + 2(-a-b)a(1+c_1)/(1-d_1) + ab)^2(1-d_1)^4(1+d_1)^2/(a^4s_1^2(1+c_1)^2(c_1+d_1)^2) - 2a(1+c_1)/(1-d_1) + a + b)^2 - 2a((1/4)(3a^2(1+c_1)^2/(1-d_1)^2 + 2(-a-b)a(1+c_1)/(1-d_1) + ab)^2(1-d_1)^4(1+d_1)^2/(a^4s_1^2(1+c_1)^2(c_1+d_1)^2) - 2a(1+c_1)/(1-d_1) + a + b) + ab)/(((1/4)(3a^2(1+c_1)^2/(1-d_1)^2 + 2(-a-b)a(1+c_1)/(1-d_1) + ab)^2(1-d_1)^4(1+d_1)^2/(a^4s_1^2(1+c_1)^2(c_1+d_1)^2) - 2a(1+c_1)/(1-d_1) + a + b)^2 - ab).$$

Addition Similarly, the following Maple script derives the corresponding addition formulae. The first line defines the addition formulae for W and the second line applies Theorem 3.2.1.

```
> addW:=(u1,v1,u2,v2)->(((v2-v1)/(u2-u1))^2+a1*(v2-v1)/(u2-u1)-a2-u1-u2,(v2-v1)/(u2- \
                        u1)*(u1-(((v2-v1)/(u2-u1))^2+a1*(v2-v1)/(u2-u1)-a2-u1-u2))- \
                        v1-a1*u3-a3):
> addM:=phi(addW(psipsi(s1,c1,d1,s2,c2,d2))):
```

The addition formulae stored in `addM` are given by $(s_1, c_1, d_1) + (s_2, c_2, d_2) = (s_3, c_3, d_3)$ where

$$s_3 = -2((-a^2s_2(1+c_2)(c_2+d_2)/((1-d_2)^2(1+d_2)) + a^2s_1(1+c_1)(c_1+d_1)/((1-d_1)^2(1+d_1)))(2a(1+c_1)/(1-d_1) - (-a^2s_2(1+c_2)(c_2+d_2)/((1-d_2)^2(1+d_2)) + a^2s_1(1+c_1)(c_1+d_1)/((1-d_1)^2(1+d_1)))^2/(a(1+c_2)/(1-d_2) - a(1+c_1)/(1-d_1))^2 + a(1+c_2)/(1-d_2) - a - b)/(a(1+c_2)/(1-d_2) - a(1+c_1)/(1-d_1)) + a^2s_1(1+c_1)(c_1+d_1)/((1-d_1)^2(1+d_1)))/(((-a^2s_2(1+c_2)(c_2+d_2)/((1-d_2)^2(1+d_2)) + a^2s_1(1+c_1)(c_1+d_1)/((1-d_1)^2(1+d_1)))^2/(a(1+c_2)/(1-d_2) - a(1+c_1)/(1-d_1))^2 - a(1+c_1)/(1-d_1) - a(1+c_2)/(1-d_2) + a + b)^2 - ab),$$

$$c_3 = (((-a^2s_2(1+c_2)(c_2+d_2)/((1-d_2)^2(1+d_2)) + a^2s_1(1+c_1)(c_1+d_1)/((1-d_1)^2(1+d_1)))^2/(a(1+c_2)/(1-d_2) - a(1+c_1)/(1-d_1))^2 - a(1+c_1)/(1-d_1) - a(1+c_2)/(1-d_2) + a + b)^2 - 2b((-a^2s_2(1+c_2)(c_2+d_2)/((1-d_2)^2(1+d_2)) + a^2s_1(1+c_1)(c_1+d_1)/((1-d_1)^2(1+d_1)))^2/(a(1+c_2)/(1-d_2) - a(1+c_1)/(1-d_1))^2 - a(1+c_1)/(1-d_1) - a(1+c_2)/(1-d_2) + a + b) + ab)/(((-a^2s_2(1+c_2)(c_2+d_2)/((1-d_2)^2(1+d_2)) + a^2s_1(1+c_1)(c_1+d_1)/((1-d_1)^2(1+d_1)))^2/(a(1+c_2)/(1-d_2) - a(1+c_1)/(1-d_1))^2 - a(1+c_1)/(1-d_1) - a(1+c_2)/(1-d_2) + a + b)^2 - ab),$$

$$d_3 = (((-a^2s_2(1+c_2)(c_2+d_2)/((1-d_2)^2(1+d_2)) + a^2s_1(1+c_1)(c_1+d_1)/((1-d_1)^2(1+d_1)))^2/(a(1+c_2)/(1-d_2) - a(1+c_1)/(1-d_1))^2 - a(1+c_1)/(1-d_1) - a(1+c_2)/(1-d_2) + a + b)^2 - 2a((-a^2s_2(1+c_2)(c_2+d_2)/((1-d_2)^2(1+d_2)) + a^2s_1(1+c_1)(c_1+d_1)/((1-d_1)^2(1+d_1)))^2/(a(1+c_2)/(1-d_2) - a(1+c_1)/(1-d_1))^2 - a(1+c_1)/(1-d_1) - a(1+c_2)/(1-d_2) + a + b) + ab)/(((-a^2s_2(1+c_2)(c_2+d_2)/((1-d_2)^2(1+d_2)) + a^2s_1(1+c_1)(c_1+d_1)/((1-d_1)^2(1+d_1)))^2/(a(1+c_2)/(1-d_2) - a(1+c_1)/(1-d_1))^2 - a(1+c_1)/(1-d_1) - a(1+c_2)/(1-d_2) + a + b)^2 - ab).$$

The computer-derived formulae are overly involved with many terms which makes them inefficient in computations. For instance, both addition and doubling formulae have total degree of the fractions over 50.

Example 3.2.7. The analogous operations can also be performed using MAGMA. Since MAGMA is a declarative language all necessary objects should be constructed. The script starts with the construction of a field $\mathbb{K}$ = `K<a,b>` which is a transcendental extension of the rational field $\mathbb{Q}$. This way MAGMA treats `a` and `b` as fixed constants. So, the base field object is assigned to `K`. $\mathbb{K}^2$ = `SW` and $\mathbb{K}^3$ = `SM` are the spaces defined over `K`. These spaces are (defined as) the ambient of the curves W = `W` and M = `M`. The maps ϕ = `phi` and ψ = `phi` are defined using the `map` constructor. Note that the definitions of W, ϕ, and ψ are copied from Chapter 2

to match a standard choice of the identity element as was the case in Example 3.2.6. Two assertions after the definitions of `phi` and `psi` ensure that `W` and `M` are birationally equivalent over `K`.

```
> K<a,b>:=FieldOfFractions(PolynomialRing(Rationals(),2));
> a1:=0; a3:=0; a6:=0; a2:=-(a+b); a4:=a*b;
> SW<u,v>:=AffineSpace(K,2);
> W:=Curve(SW,[v^2+a1*u*v+a3*v-(u^3+a2*u^2+a4*u+a6)]);
> SM<s,c,d>:=AffineSpace(K,3);
> M:=Curve(SM,[b*s^2+c^2-1,a*s^2+d^2-1]);
> phi:=map<W->M|[2*v/(a*b-u^2),2*u*(b-u)/(a*b-u^2)-1,2*u*(a-u)/(a*b-u^2)-1]>;
> psi:=map<M->W|[a*(1+c)/(1-d),(-a*(1+c)/(1-d))*(c+d)/s]>;
> assert Expand(phi*psi) eq IdentityMap(W);
> assert Expand(psi*phi) eq IdentityMap(M);
```

The script continues to find specialized negation, doubling and addition formulae. The outputs are omitted here since they are equivalent to the formulae derived by the Maple example.

Negation The following MAGMA script derives the corresponding negation formulae. The first line defines the negation formulae for W and the second line applies a simpler version of Theorem 3.2.1.

```
> negW:=map<W->W|[u,-v-a1*u-a3]>;
> negM:=psi*negW*phi; //i.e. phi o negW o psi.
```

Doubling Similarly, the following MAGMA script derives the corresponding doubling formulae. The first two lines define the doubling formulae for W and the third line applies a simpler version of Theorem 3.2.1.

```
> L:=(3*u^2+2*a2*u+a4-a1*v)/(2*v+a1*u+a3); u3:=L^2+a1*L-a2-2*u; v3:=L*(u-u3)-v-a1*u3-a3;
> dblW:=map<W->W|[u3,v3]>;
> dblM:=psi*dblW*phi; //i.e. phi o dblW o psi.
```

Addition Similarly, the following Magma script derives the corresponding addition formulae. The script constructs $(\psi \times \psi)$ =`psipsi` and the products $W \times W$ =`WW`, $M \times M$ =`MM`.

```
> SWW<u1,v1,u2,v2>:=AffineSpace(K,4);
> WW:= Scheme(SWW,[v1^2+a1*u1*v1+a3*v1-(u1^3+a2*u1^2+a4*u1+a6), \
                              v2^2+a1*u2*v2+a3*v2-(u2^3+a2*u2^2+a4*u2+a6)]);
> SMM<s1,c1,d1,s2,c2,d2>:=AffineSpace(K,6);
> MM:= Scheme(SMM,[b*s1^2+c1^2-1,a*s1^2+d1^2-1,b*s2^2+c2^2-1,a*s2^2+d2^2-1]);
> psipsi:=map<MM->WW|[a*(1+c1)/(1-d1),(-a*(1+c1)/(1-d1))*(c1+d1)/s1, \
                            a*(1+c2)/(1-d2),(-a*(1+c2)/(1-d2))*(c2+d2)/s2]>;
```

The script continues to compute (3.1). The output is omitted here since it is equivalent to the formulae derived by the Magma example.

```
> L:=(v2-v1)/(u2-u1); u3:=L^2+a1*L-a2-u1-u2; v3:=L*(u1-u3)-v1-a1*u3-a3;
> addW:=map<WW->W|[u3,v3]>;
> addM:=psipsi*addW*phi; //i.e. phi o addW o (psi x psi).
```

The next section is a continuation of Examples 3.2.6 and 3.2.7 for finding more "suitable" representatives for s_3, c_3, and d_3 among each of the residue classes $[s_3]$, $[c_3]$, and $[d_3]$, respectively.

3.3 Minimal total degree

Let V be a variety over $\mathbb{K}$ and $\mathbb{K}(V)$ the function field on V. Recall from Appendix A that the elements of $\mathbb{K}(V)$ are represented by rational functions on V. Since the chief objects of study are efficient group laws on elliptic curves, V can be fixed to an elliptic curve E or to the product $E \times E$. Let $P \in V$ and $f \in \mathbb{K}(V)$ such that f is regular at P. Suppose that the aim is to evaluate f at P *efficiently*. It is then reasonable to find a suitable f' in the residue class $[f] \in \mathbb{K}(V)$ and then make the trivial substitution of P to compute $f(P) = f'(P)$ using more primitive field operations in $\mathbb{K}$. The computational effort for finding the suitable f' can be neglected here since f' can be fixed for many evaluations. Roughly speaking, the fewer the number of field operations are used for an evaluation of f' at P, the more *efficient* the evaluation is. The term efficiency here is usually understood as the *running time* or the space consumption of an algorithm. Note that other interpretations are possible such as the required transmission size of some data or consumed energy along the execution. The emphasis here is on the running time aspect. See also [CMO98].

A common experience for an efficient evaluation of a rational function on a variety at a randomly chosen non-singular point is that the evaluation takes less time if the numerator and denominator have lower total degrees, and preferably having no "common factor", cf. [CC86]. The numerator and denominator of a rational function in $\mathbb{K}(V)$ can be viewed as polynomial functions in $\mathbb{K}[V]$. These polynomial functions, for the purpose of this work, are the multivariate polynomial expressions arising in the group law of an elliptic curve. Therefore, the main emphasis of this chapter is on finding suitable fractions of polynomials with low/lowest total degree in making a description of the group law. Such expressions will eventually lead to efficient elliptic curve arithmetic. Note that the arithmetic of elliptic curves is known to be very efficient and it has attracted a lot of attention over the past few decades. Standard references are [HMV03] and [CF05]. In this sense, the present work is an attempt to improve previous upper bounds for efficient computations.

Concerning the numerator and denominator of s_3 (or c_3 or d_3) in Examples 3.2.6 and 3.2.7, it is intuitive to ask whether the denominator is a unit in the corresponding coordinate ring. If this is the case, the fraction reduces to a polynomial which allows working in affine coordinates without inverting elements of $\mathbb{K}$. In large characteristic fields, this turns out to be possible for negation formulae most of the time. However, for doubling and addition this is not possible in any of the five forms that are studied in this work. In characteristic 2, however, there are examples where this occurs for the doubling in some very special cases, cf. [BSS99].

It is also intuitive to ask whether there exists representatives having minimal total degrees for both the numerators and denominators. The two may not be possible simultaneously, i.e. among all possible fractions, a fraction with minimal total degree numerator may not have a minimal total degree denominator (and vice versa). However, there always exist

representatives with minimal total degree. This section collects necessary tools from literature to find such representatives. The computer algebra examples are also provided. Efficiency of the simplification process is not a major issue as long as the simplification can be done in a reasonable time.

Two algorithms for simplifying rational expressions, proposed by Monagan and Pearce in 2006 in [MP06], are adapted in this section since their algorithms perfectly fit the aforementioned goal of finding formulae with low/lowest total degree of fractions in the corresponding coordinate ring (i.e. $\mathbb{K}[M]$ for negation and doubling; and $\mathbb{K}[M \times M]$ for addition). Monagan and Pearce's first algorithm computes a reduced canonical form (RCF) of a fraction f/g modulo a proper prime ideal $I \subset \mathbb{K}[x_1, \ldots, x_n]$ where $f, g \in \mathbb{K}[x_1, \ldots, x_n]$. The coordinates $x_1, \ldots, x_n$ can be suitably renamed to $s_1, c_1, d_1, s_2, c_2, d_2$ in the case of Example 3.2.4. Using a prime ideal ensures that $\mathbb{K}[x_1, \ldots, x_n]/I$ is an integral domain and thus contains no zero divisors. Note that this is always the case for coordinate rings of elliptic curves (or their products). Now, let m/h be an RCF for a/b such that $ah - bm \equiv 0 \mod I$ where $a, b, m, h \in \mathbb{K}[x_1, \ldots, x_n]$ with $b, h \notin I$. The algorithm is built on three observations:

1. The colon ideal $J = (\langle g \rangle + I) : \langle f \rangle$ contains an h having no common components with g. Let $\{h_1, \ldots, h_t\}$ be a reduced Gröbner basis with respect to a graded monomial ordering. Each h_i is a candidate for h since $h \in \langle h_1, \ldots, h_t \rangle$.

2. By the definition of a colon ideal, b must divide $h_i a$. Thus, $m_i = h_i a/b$ can be computed using an exact division. Now selecting $m/h = m_i/h_i$ with $\min(\deg(h_i))$ gives a representation which guarantees a minimal total degree for the denominator of m/h and a removal of all common components. It is convenient to comment that for all curve models considered in this work, this computation yields a set of formulae having minimal total degree of fractions, which will be used in Chapters 4 and 5. However, to this end there is no guarantee that a minimal $\deg(m) + \deg(h)$ will be obtained.

3. Sometimes adding a common component to the numerator or denominator leads to a lower total degree sum. This idea is pursued by Monagan and Pearce by a computation of the reduced Gröbner basis of the module $\{[m, h] : fh - gm \equiv 0 \mod I\}$ with respect to a term-over-position order. Refer to the original paper [MP06] for details and to [AL96] for definitions of modules and term-over-position order.

   This last modification finds a "good" balance between numerator and denominator. However, there is still no guarantee that a minimal $\deg(m) + \deg(h)$ will be obtained.

An implementation of this algorithm comes with Maple v.11+. An open source Maple implementation is given in Pearce's thesis [Pea05].

Example 3.3.1. The following Maple script simplifies the automated formulae from Example 3.2.4 using Monagan/Pearce reduced canonical form algorithm:

```
> negM,dblM,addM:=simplify([negM,dblM,addM],[M(s1,c1,d1),M(s2,c2,d2)],tdeg(c1,c2,d1,d2));
```

Negation The simplified negation formulae are given by $-(s_1, c_1, d_1) = (-s_1, c_1, d_1)$. Clearly, the negation formulae are of minimal total degree since they are linear and they cannot be a constant map.

Doubling The simplified doubling formulae are given by

$$[2](s_1, c_1, d_1) = \Big(\frac{2s_1c_1d_1}{1-abs_1^4}, \frac{1-2bs_1^2+abs_1^4}{1-abs_1^4}, \frac{1-2as_1^2+abs_1^4}{1-abs_1^4}\Big).$$

Addition The simplified addition formulae are given by

$$(s_1, c_1, d_1) + (s_2, c_2, d_2) = \Big(\frac{s_1c_2d_2+c_1d_1s_2}{1-abs_1^2s_2^2}, \frac{c_1c_2-bs_1d_1s_2d_2}{1-abs_1^2s_2^2}, \frac{d_1d_2-as_1c_1s_2c_2}{1-abs_1^2s_2^2}\Big).$$

In all of these outputs, the total degrees of the denominators are minimized with respect to the fixed monomial ordering. Without a justification for now, it can be stated that the doubling formulae are of minimal total degree sum. However, the addition formulae are not.

It is experimentally observed that the rational simplifications for negation and doubling laws are quite efficient despite several cumbersome Gröbner basis computations. In particular, each simplification takes less than a second on a Core 2 processor running at $2.66\,GHz$. However, simplification for addition is rather slow. To speed up the derivation of the addition formulae the relevant script can be updated as follows to have simplifications for intermediate rational expressions.

```
> L:=(v1-v2)/(u1-u2):
> L:=simplify(L,[M(s1,c1,d1),M(s2,c2,d2)],tdeg(c1,c2,d1,d2));
> u3:=L^2+a1*L-a2-u1-u2:
> u3:=simplify(u3,[M(s1,c1,d1),M(s2,c2,d2)],tdeg(c1,c2,d1,d2));
> v3:=L*(u1-u3)-v1-a1*u3-a3:
> v3:=simplify(v3,[M(s1,c1,d1),M(s2,c2,d2)],tdeg(c1,c2,d1,d2));
> addM:=simplify([phi(u3,v3)],[M(s1,c1,d1),M(s2,c2,d2)],tdeg(c1,c2,d1,d2));
```

Monagan and Pearce's second algorithm always finds a fraction with minimal total degree sum of the numerator and denominator. Their algorithm makes a search among all possible m/n starting from lowest degree 0 assuming that the fraction can be simplified to a constant in $\mathbb{K}$. If the solution of the resulting system does not give the hypothesized numerator and denominator, the hypothesized degree is increased by one for both the numerator and denominator. The procedure is repeated until a solution is found. Then the remaining cases are explored in a recursive manner. For details see §4 in [Pea05]. An implementation of this algorithm comes with Maple v.11+. An open source Maple implementation is given in Pearce's thesis [Pea05].

Example 3.3.2. The following Maple script[2] simplifies the automated addition formulae using Monagan/Pearce minimal total degree algorithm:

```
> addM:=simplify(addM,[M(s1,c1,d1),M(s2,c2,d2)],mindeg);
```

Addition The simplified addition formulae are given by

$$(s_1, c_1, d_1) + (s_2, c_2, d_2) = \Big(\frac{s_1^2-s_2^2}{s_1c_2d_2-c_1d_1s_2}, \frac{s_1c_1d_2-d_1s_2c_2}{s_1c_2d_2-c_1d_1s_2}, \frac{s_1d_1c_2-c_1s_2d_2}{s_1c_2d_2-c_1d_1s_2}\Big).$$

[2] *Warning:* Maple v.11 and v.12 have internal bugs which are triggered by this example. The problem is that the minimal total degree implementation uses local variables which clash with the coordinate functions `c1`, `d1`, `c2`, and `d2` resulting in wrong outputs. To surpass these bugs, simply rename `c1` to `cc1`; `d1` to `dd1`; `c2` to `cc2`; and `d2` to `dd2` in all relevant scripts in this section.

It is experimentally observed that the rational simplification for finding minimal total degree addition formulae takes less than a second on a Core 2 processor running at $2.66\,GHz$ once the algorithm is fed with initial formulae in canonical form. Using the same algorithm, it can be checked that the doubling formulae computed above is really of minimal total degree. This also justifies the claims of Example 3.3.1.

Example 3.3.3. Finally, it is convenient to note that built-in mechanisms of MAGMA can also remove common components with respect to a monomial order (without a guarantee of minimal total degree sum of the numerator and denominator). In the following MAGMA script, the first two lines construct the ring of fractions of the corresponding coordinate rings $\mathbb{K}[M]$ and $\mathbb{K}[M \times M]$, respectively. The last three lines simplify the negation, doubling, and addition formulae which are triggered by the coercion operator, "`!`":

```
> KM:=RingOfFractions(quo<CoordinateRing(M)|Ideal(M)>);
> KMM:=RingOfFractions(quo<CoordinateRing(MM)|Ideal(MM)>);
> [KM!DefiningEquations(negM)[i] : i in [1..#DefiningEquations(negM)]];
> [KM!DefiningEquations(dblM)[i] : i in [1..#DefiningEquations(dblM)]];
> [KMM!DefiningEquations(addM)[i] : i in [1..#DefiningEquations(addM)]];
```

This script should be run after the relevant scripts of Example 3.2.7. The output is omitted here since the minimal degree formulae have already been computed and presented in Examples 3.3.1 and 3.3.2.

For other forms of elliptic curves (including the projective representations), it is easy to modify/parametrize the scripts of §3.2 and of this section, in order to detect "reduced" formulae for negation, doubling, and addition.

Let $\mathbb{L}$ be an algebraic extension of $\mathbb{K}$. The computer derived negation, doubling, and addition formulae will be used in Chapter 4 to make a complete description of the morphism $+_{M/\mathbb{L}}$ for desingularized curves of genus 1 in several different forms. Selected formulae will then be used in Chapter 5 for efficiency purposes.

3.4 Automated validations

It is useful to have a validation tool to decide whether two rational functions on a variety are equivalent. The key tool is described in the following lemma.

Lemma 3.4.1 (Ideal membership)**.** *Let G be a Gröbner basis for an ideal $I \subseteq \mathbb{K}[x_1, \ldots, x_n]$. Let f be a polynomial in $\mathbb{K}[x_1, \ldots, x_n]$. Then $f \in I$ if and only if the normal form of f by G is zero.*

Proof. The proof follows from basic properties of Gröbner basis. See [CLO07, §2.6]. □

Let V be a variety and $I(V)$ the ideal of V. Let $f, f', g, g' \in \mathbb{K}[V]$ such that $g, g' \notin I(V)$. Recall from Chapter 2 that the quotients f/g and f'/g' define the same function on V if and only if $f'g - fg' \in I(V)$. Now, applying Lemma 3.4.1 answers whether f/g and f'/g' are equivalent functions on V.

Example 3.4.2. It can be validated that the addition formulae in Examples 3.3.1 and 3.3.2 are coordinate-wise equivalent rational functions on $V = M \times M$. The following Maple script implements this validation. Note that the last line defines the quotient relations.

```
> simplify([
>     (s1*c2*d2+c1*d1*s2)*(s1*c2*d2-c1*d1*s2)-(s1^2-s2^2)*(1-a*b*s1^2*s2^2),
>     (c1*c2-b*s1*d1*s2*d2)*(s1*c2*d2-c1*d1*s2)-(s1*c1*d2-d1*s2*c2)*(1-a*b*s1^2*s2^2),
>     (d1*d2-a*s1*c1*s2*c2)*(s1*c2*d2-c1*d1*s2)-(s1*d1*c2-c1*s2*d2)*(1-a*b*s1^2*s2^2)
> ],[b*s1^2+c1^2-1,a*s1^2+d1^2-1,b*s2^2+c2^2-1,a*s2^2+d2^2-1]);
```

More implementations have already been developed in [BL07a] and several examples are given in that database.

3.5 Finding more formulae

In §3.3, it was noted how a computation of colon ideals was used for removing common components of the numerator and denominator of a rational expression modulo a polynomial ideal. For the purpose of this work, these rational expressions are rational functions on an elliptic curve M or rational functions on the product $M \times M$. By using a graded monomial order and by skipping the module construction phase in the Monagan/Pearce method, it is possible to minimize the total degree of either the numerator or denominator. More formulae can then be derived from the other low degree denominators that appear in the reduced Gröbner basis.

Example 3.5.1. It is convenient to continue with the investigation on twisted Jacobi intersection form. Consider the polynomials $f = s_1^2 - s_2^2$ and $g = s_1c_2d_2 - c_1d_1s_2$ in $\mathbb{K}[c_1, c_2, d_1, d_2, s_1, s_2]$ where $\mathbb{K} = \mathbb{Q}(a, b)$. Since $\mathrm{GCD}(f, g) = 1$, the fraction f/g does not simplify in $\mathbb{K}(c_1, c_2, d_1, d_2, s_1, s_2)$. Now assume that f/g is a function on $E_{I,b,a} : bs^2 + c^2 - 1, as^2 + d^2 - 1$ where $a, b \in \mathbb{K}$ with $ab(a - b) \neq 0$. Let K be the ideal generated by the relations $bs_1^2 + c_1^2 - 1, as_1^2 + d_1^2 - 1, bs_2^2 + c_2^2 - 1, as_2^2 + d_2^2 - 1$. The reduced Gröbner basis of the colon ideal $J = (\langle f \rangle + K) : \langle g \rangle$ with respect to any graded monomial order must contain a minimal total degree denominator, see §3.3. In addition, it *often* contains other low degree denominators because of the graded order which dominates in reducing the total degree of the generators. Indeed the generators of the reduced Gröbner basis of J with respect to graded reverse lexicographical order with $c > d > s$ are given by the sequence $G = [c_2d_1s_1^2s_2 - c_1d_2s_1s_2^2 + (1/b)c_1d_2s_1 - (1/b)c_2d_1s_2, c_1d_2s_1^2s_2 - c_2d_1s_1s_2^2 + (1/a)c_2d_1s_1 - (1/a)c_1d_2s_2, c_2s_1^3s_2 - (1/(ab))c_1d_1d_2 - (1/b)c_2s_1s_2, d_2s_1^3s_2 - (1/(ab))c_1c_2d_1 - (1/a)d_2s_1s_2, c_1s_1s_2^3 - (1/(ab))c_2d_1d_2 - (1/b)c_1s_1s_2, d_1s_1s_2^3 - (1/(ab))c_1c_2d_2 - (1/a)d_1s_1s_2, c_1c_2d_1d_2 - abs_1^3s_2 - abs_1s_2^3 + (a + b)s_1s_2, c_1c_2d_1s_2 + bd_2s_1s_2^2 - d_2s_1, c_1d_1d_2s_2 + ac_2s_1s_2^2 - c_2s_1, c_1c_2s_1s_2 + (1/a)d_1d_2, d_1d_2s_1s_2 + (1/b)c_1c_2, s_1^2s_2^2 - (1/(ab)), c_2d_2s_1 - c_1d_1s_2, c_1^2 + bs_1^2 - 1, c_2^2 + bs_2^2 - 1, d_1^2 + as_1^2 - 1, d_2^2 + as_2^2 - 1]$. By the definition of colon ideal, J trivially contains K. Therefore, the generators of G can be discarded if they are in K. This can be efficiently detected using Lemma 3.4.1. Observe that the initial denominator $s_1c_2d_2 - c_1d_1s_2$ is in G. On the other hand, there are several more low total degree entries which are other candidates for the denominator of equivalent fractions. For instance, select the entry $c_1c_2s_1s_2 + (1/a)d_1d_2$. Using a multivariate exact division algorithm

the new numerator is computed as $(c_1c_2s_1s_2 + (1/a)d_1d_2)f/g = (c_1s_2d_2 + s_1d_1c_2)/a$. So the alternative formula is given by $(c_1s_2d_2 + s_1d_1c_2)/(d_1d_2 + ac_1c_2s_1s_2)$. For an exact division algorithm see Pearce's thesis [Pea05]. Each one of the other entries gives rise to another fraction. Even more fractions can be obtained by changing the lexicographical ordering.

3.6 Brain teasers

When it comes to developing a computer program to carry out the group law on an elliptic curve E, the group law is typically described by an algorithm (possibly containing conditional statements). This algorithm can be viewed as a sequence of interdependent operations in $\mathbb{K}$. From a computational aspect, the basic operations associated with $\mathbb{K}$ are addition $\mathbf{a}$[3], squaring $\mathbf{S}$, multiplication $\mathbf{M}$, multiplication by small curve constants $\mathbf{D}$, and inversion $\mathbf{I}$. The most efficient algorithms are typically determined by how many of each field operations $\mathbf{I}$, $\mathbf{M}$, $\mathbf{S}$, $\mathbf{D}$, and $\mathbf{a}$ are sequentially accessed. Because of efficiency considerations, it is convenient to set some ordering relations between these operations. Practically, it is assumed that $\mathbf{I} > \mathbf{M}$, $\mathbf{M} > \mathbf{S}$, $\mathbf{M} > \mathbf{D}$, and $\mathbf{M} > \mathbf{a}$. To be more precise in comparisons, it is also convenient to set ratios for these operations such as $\mathbf{I} = 120\mathbf{M}$, $\mathbf{S} = 0.8\mathbf{M}$, $\mathbf{D} = 0.2\mathbf{M}$, and $\mathbf{a} = 0.1\mathbf{M}$. However, these ratios can vary from one implementation to another.

In this section, some short algorithms are isolated from different resources in the literature. Some of these algorithms turn out to be useful for computing group laws more efficiently. Each algorithm can be viewed as a brain teaser because it is not clear at the first glance what the underlying idea of the algorithm is. On the other hand, once the correctness of each approach is justified the challenge is simplified to recognizing these cases within group laws (hence the title of this section).

These algorithms will be employed as subalgorithms of several different addition laws in Chapter 5. There are more variations of each algorithm with altered signs which are omitted here. The last algorithm (named **Type** M_4) is a contribution of this thesis.

Type S Assume that the following algorithm is executed in three steps and the results are stored in r_1, r_2, and r_3;

$$r_1 \leftarrow A^2, \qquad r_2 \leftarrow B^2, \qquad r_3 \leftarrow 2A \cdot B. \tag{3.3}$$

A preliminary operation count yields $1\mathbf{M} + 2\mathbf{S} + 1\mathbf{a}$. On the other hand evaluating (3.3) can be done with $3\mathbf{S} + 3\mathbf{a}$;

$$r_1 \leftarrow A^2, \quad r_2 \leftarrow B^2, \quad r_3 \leftarrow (A+B)^2 - r_1 - r_2.$$

This algorithm is useful if $\mathbf{M} - \mathbf{S} > 2\mathbf{a}$. This algorithm has already been used in optimizing the arithmetic of several different elliptic curve forms, cf. [BL07a]. It will also be used in Chapter 5.

[3] $\mathbf{a}$ also stands for a subtraction or a multiplication or division by a small curve constant.

Type K This algorithm is attributed to Karatsuba in many resources, cf. [GL09]. This is due to its resemblance with Karatsuba multiplication;

$$r_1 \leftarrow A \cdot B + C \cdot D, \qquad r_2 \leftarrow A \cdot C - B \cdot D. \tag{3.4}$$

A preliminary operation count yields $4\mathbf{M} + 2\mathbf{a}$. On the other hand, evaluating (3.4) can be done with only $3\mathbf{M} + 5\mathbf{a}$;

$$t_1 \leftarrow A \cdot C, \quad t_2 \leftarrow B \cdot D, \quad r_1 \leftarrow (A + D) \cdot (B + C) - t_1 - t_2, \quad r_2 \leftarrow t_1 - t_2.$$

This algorithm is useful if $\mathbf{M} > 3\mathbf{a}$ and is a standard method used in many resources, cf. [BJ03a].

Type M_1 This algorithm is taken from [Mon87];

$$r_1 \leftarrow 2(A \cdot B - C \cdot D), \qquad r_2 \leftarrow 2(A \cdot C - B \cdot D). \tag{3.5}$$

A preliminary operation count yields $4\mathbf{M} + 4\mathbf{a}$. On the other hand, the sum and the difference of r_1 and r_2 both factor into two linear components each. Using this observation, evaluating (3.5) can be done with only $2\mathbf{M} + 6\mathbf{a}$ as in [Mon87];

$$\begin{array}{llll} t_1 \leftarrow A - C, & t_2 \leftarrow B + D, & t_1 \leftarrow t_1 \cdot t_2, & t_2 \leftarrow A + C, \\ t_3 \leftarrow B - D, & t_2 \leftarrow t_2 \cdot t_3, & r_1 \leftarrow t_1 + t_2, & r_2 \leftarrow t_1 - t_2. \end{array}$$

This algorithm is useful if $\mathbf{M} > \mathbf{a}$ and will be used for optimizing the arithmetic of twisted Edwards form with $a = -1$ and of twisted Jacobi intersection form in Chapter 5.

Type M_2 This algorithm is a special case of **Type** M_1 and has not been detected in any elliptic curve related optimizations yet;

$$r_1 \leftarrow 2(A^2 - C \cdot D), \qquad r_2 \leftarrow 2A \cdot (C - D). \tag{3.6}$$

A preliminary operation count yields $2\mathbf{M} + 1\mathbf{S} + 4\mathbf{a}$. On the other hand, evaluating (3.6) can be done with only $2\mathbf{M} + 6\mathbf{a}$;

$$\begin{array}{llll} t_1 \leftarrow A - D, & t_2 \leftarrow A + C, & t_1 \leftarrow t_1 \cdot t_2, & t_2 \leftarrow A + D, \\ t_3 \leftarrow A - C, & t_2 \leftarrow t_2 \cdot t_3, & r_1 \leftarrow t_1 + t_2, & r_2 \leftarrow t_1 - t_2. \end{array}$$

This algorithm is useful if $\mathbf{S} > 2\mathbf{a}$.

Type M_3 This algorithm is taken from [Mon87];

$$r_1 \leftarrow (A^2 - B^2)^2, \qquad r_2 \leftarrow 4A \cdot B. \tag{3.7}$$

A preliminary operation count yields $1\mathbf{M} + 3\mathbf{S} + 2\mathbf{a}$. On the other hand, evaluating (3.7) can be done with only $1\mathbf{M} + 2\mathbf{S} + 3\mathbf{a}$ as in [Mon87];

$$t_1 \leftarrow (A - B)^2, \quad t_2 \leftarrow (A + B)^2, \quad r_1 \leftarrow t_1 \cdot t_2, \quad r_2 \leftarrow t_2 - t_1.$$

This algorithm is useful if $\mathbf{S} > \mathbf{a}$.

Type M_4 This algorithm is closely related to **Type** M_1;

$$r_1 \leftarrow 2(A \cdot B - C \cdot D), \qquad r_2 \leftarrow 2(D \cdot E - A \cdot F), \qquad r_3 \leftarrow 2(C \cdot F - B \cdot E). \tag{3.8}$$

A preliminary operation count yields $6\mathbf{M} + 6\mathbf{a}$. On the other hand, observe that $2\mathbf{M} + 2\mathbf{a}$ is spent to compute $2(A \cdot B - C \cdot D)$. Using **Type**-M_1 pattern it is possible to generate $2(A \cdot B - C \cdot D)$ and $2(A \cdot C - B \cdot D)$ accessing $2\mathbf{M} + 6\mathbf{a}$. Similarly, $2(D \cdot E - A \cdot F)$ and $2(D \cdot A - E \cdot F)$ can be computed in $2\mathbf{M} + 6\mathbf{a}$. Now $2(C \cdot F - B \cdot E)$ can be written as a linear combination of $2(B - F) \cdot (C + E)$, $2(A \cdot C - B \cdot D)$, and $2(D \cdot A - E \cdot F)$. Evaluating (3.8) can be done with only $5\mathbf{M} + 17\mathbf{a}$;

$$\begin{gathered} t_1 \leftarrow (D + B) \cdot (A - C), \quad t_2 \leftarrow (D - B) \cdot (A + C), \quad t_3 \leftarrow (D + F) \cdot (A - E), \\ t_4 \leftarrow (D - F) \cdot (A + E), \quad r_1 \leftarrow t_1 - t_2, \quad r_2 \leftarrow t_4 - t_3, \\ r_3 \leftarrow t_3 + t_4 - t_1 - t_2 - 2(B - F) \cdot (C + E). \end{gathered}$$

This algorithm is useful if $\mathbf{M} > 11\mathbf{a}$ and will be used in the context of twisted Hessian curves in Chapter 5.

3.7 Conclusion

This chapter has prepared a toolbox for optimizing the arithmetic of elliptic curves given in some particular form. As the first tool, this chapter has provided a high-level method of finding group laws on elliptic curves using computer algebra. The method is composed of two stages. In the first stage, maps between birational curves are used in order to symbolically deduce the group law for some form of an elliptic curve. In the second stage, rational simplification methods are employed to find a lowest-degree group law. The notion of finding the lowest-degree rational expression modulo a prime ideal was developed in [MP06]. To the best of the authors knowledge, combining two stages and systematically finding the *lowest-degree* group laws is an outcome of this thesis. As the second tool, this chapter has isolated several algorithms from the literature which naturally arise in low-degree group laws. A new brain teaser has been contributed to the current body of knowledge which will be used in Chapter 5.

In conclusion, this work recommends going through five steps in order to make an efficiency related study of group laws on elliptic curves;

1. Fix a curve of genus 1 with a rational point lying in a suitable affine or projective space.
2. Derive the group law using Riemann-Roch computations, §3.2.
3. Simplify the negation, doubling, and addition formulae of the group law, §3.3.
4. Make a collection of several equivalent low degree formulae and ensure the equivalence, §3.4 and §3.5.
5. Detect algorithms to carry out operations efficiently for each of the collected formulae, §3.6.

As in other areas of mathematics and computer science, there may be several other ways to approach this goal. In this work, the suggested steps have been applied to all of the

studied curve models and in many cases the efficiency bounds of several literature resources are successfully improved. The subsequent chapters 4, 5, and 7 provide further details for each of the studied forms by making an effective use of the tools of this chapter.

Chapter 4

Group law in affine coordinates

The goal of this chapter is two-fold. The first part of the goal is to find low-degree point addition formulae for fixed representations of elliptic curves. Some of the formulae are obtained from literature resources where some others are derived with the tools from Chapter 3. In this context, each of the sections mainly concentrates on two denominators which naturally arise when searching for low degree group laws for each elliptic curve form. As the second part of the goal, the exceptional cases of the selected denominators are explicitly determined and practical ways of preventing division-by-zero exceptions are studied including pointers to the literature when possible. This work focuses on five aforementioned forms of elliptic curves in Chapter 2 which are the most commonly used ones in practical applications.

A complete addition algorithm is presented for each of the forms to handle all possible inputs including the point(s) at infinity. The complete description of addition law for all curves given in a particular form can be extracted from the relevant birational maps in §2.3 of Chapter 2 and the discussions on the exceptional points of the birational equivalence. In this context, exceptions can be handled by first sending the summands on a curve given in a particular form to the birationally equivalent Weierstrass curve, then carrying out the addition on the Weierstrass curve where a complete addition algorithm is present in the literature, and finally sending the sum on the Weierstrass curve to the desired sum on the original curve. Indeed, this approach *implicitly* describes a complete addition algorithm on all curves of a particular form. However, the arithmetic is now dependent on the arithmetic of Weierstrass curves. It is motivating to make a self-contained complete addition algorithm for each of these forms. The lemmas presented in §4.1-§4.5 investigate exceptional inputs and make the statement of a complete addition algorithm easier. These lemmas also provide useful information for an exception-free implementation. Since the same goals are set for each curve model, it is not surprising to have analogous results in each section. Therefore, some repetitions are unavoidable. However, it is still motivating to observe how similar ideas work for almost all studied forms.

All of the formulae in this chapter involve inversions in the underlying field. In cryptographic implementations, inversions tend to be significantly more costly than

multiplications and additions. Therefore, the operation counts are omitted in affine coordinates. In fact, this chapter serves as a preparation for efficient inversion-free algorithms which will be given later in Chapter 5.

The conclusions are drawn in §4.6. Further work on other forms are left as a future work.

4.1 Short Weierstrass form

This section presents the group law on $E_{\mathbf{S},a,b}$ in affine coordinates from literature resources. All formulae in this section plus the summarized results can be found in several books on elliptic curves. A standard reference is [Sil94]. This section investigates the exceptional summands for each set of formulae and properly handling the entire set of divide-by-zero exceptions.

Throughout this section, let $\mathbb{K}$ be a field with $\mathrm{char}(\mathbb{K}) \neq 2, 3$. Recall from Chapter 2 that a short Weierstrass curve is defined by

$$E_{\mathbf{S},a,b} : \; y^2 = x^3 + ax + b$$

where $a, b \in \mathbb{K}$ with $4a^3 + 27b^2 \neq 0$. Recall from Chapter 2 that the set of $\mathbb{K}$-rational points on $E_{\mathbf{S},a,b}$ is defined by

$$E_{\mathbf{S},a,b}(\mathbb{K}) = \{(x, y) \in \mathbb{K}^2 \mid y^2 = x^3 + ax + b\} \cup \{\Omega\}$$

where Ω is the point at infinity (which is also denoted by ∞ or by $\mathcal{O}$ in some other resources).

Identity element and negation The identity element can suitably be taken as Ω. In fact, this is the only $\mathbb{K}$-rational point on $E_{\mathbf{S},a,b}$ that does not depend on the field of definition. Let (x_1, y_1) be a point on $E_{\mathbf{S},a,b}$. The negative of (x_1, y_1) is $(x_1, -y_1)$.

Doubling The doubling formulae on $E_{\mathbf{S},a,b}$ is given by $[2](x_1, y_1) = (x_3, y_3)$ where

$$x_3 = ((3x_1^2 + a)/(2y_1))^2 - 2x_1, \tag{4.1}$$
$$y_3 = ((3x_1^2 + a)/(2y_1))(x_1 - x_3) - y_1 \tag{4.2}$$

assuming that $y_1 \neq 0$. Note, points with zero y-coordinate are of order 2. These formulae are *not* of minimal total degree.

Dedicated addition Further let (x_2, y_2) be a point on $E_{\mathbf{S},a,b}$. The addition formulae on $E_{\mathbf{S},a,b}$ are given by $(x_1, y_1) + (x_2, y_2) = (x_3, y_3)$ where

$$x_3 = ((y_1 - y_2)/(x_1 - x_2))^2 - x_1 - x_2, \tag{4.3}$$
$$y_3 = ((y_1 - y_2)/(x_1 - x_2))(x_1 - x_3) - y_1 \tag{4.4}$$

assuming that $x_1 - x_2 \neq 0$. These formulae do not work for identical summands hence the name *dedicated*. These formulae are of minimal total degree.

If $(x_1, y_1) + (x_2, y_2)$ is the point at infinity then $x_1 - x_2 = 0$. Otherwise, $(x_1, y_1) + (x_2, y_2)$ would be an affine point since it can be shown using the relations $y_1^2 = x_1^3 + ax_1 + b$ and $y_2^2 = x_2^3 + ax_2 + b$ that the algebraic expressions for (x_3, y_3) satisfy $y_3^2 = x_3^3 + ax_3 + b$. The converse, however, does not necessarily apply. This means that if $x_1 - x_2 = 0$ then $(x_1, y_1) + (x_2, y_2)$ may not be a point at infinity. Therefore it is worth investigating the exceptional cases. The denominators of (4.3) and (4.4) vanish for some summands which are described in the following lemma explicitly.

Lemma 4.1.1. *Fix $x_1, y_1 \in \mathbb{K}$ such that $y_1^2 = x_1^3 + ax_1 + b$. Let $x_2, y_2 \in \mathbb{K}$ such that $y_2^2 = x_2^3 + ax_2 + b$. It follows that $x_1 - x_2 = 0$ if and only if $(x_2, y_2) = (x_1, y_1)$ or $(x_2, y_2) = (x_1, -y_1) = -(x_1, y_1)$.*

Proof. Trivial. □

Unified addition Alternative addition formulae on $E_{\mathbf{S},a,b}$ are given by $(x_1, y_1) + (x_2, y_2) = (x_3, y_3)$ where

$$x_3 = ((x_1^2 + x_1x_2 + x_2^2 + a)/(y_1 + y_2))^2 - x_1 - x_2 \tag{4.5}$$

$$y_3 = ((x_1^2 + x_1x_2 + x_2^2 + a)/(y_1 + y_2))(x_1 - x_3) - y_1 \tag{4.6}$$

assuming $y_1 + y_2 \neq 0$, see [Sil94, Remark 3.6.1] and [BJ02]. These formulae work for identical summands in most of the cases hence the name *unified*.

If $(x_1, y_1) + (x_2, y_2)$ is the point at infinity then $y_1 + y_2 = 0$. Otherwise, $(x_1, y_1) + (x_2, y_2)$ would be an affine point since it can be shown using the relations $y_1^2 = x_1^3 + ax_1 + b$ and $y_2^2 = x_2^3 + ax_2 + b$ that the algebraic expressions for (x_3, y_3) satisfy $y_3^2 = x_3^3 + ax_3 + b$. The converse, however, does not necessarily apply. This means that if $y_1 + y_2 = 0$ then $(x_1, y_1) + (x_2, y_2)$ may not be a point at infinity. Therefore it is worth investigating the exceptional cases. The denominators of (4.5) and (4.6) vanish for some summands which are described in the following lemma explicitly.

Lemma 4.1.2. *Let $a, b \in \mathbb{K}$ with $4a^3 + 27b^2 \neq 0$. Fix $x_1, y_1 \in \mathbb{K}$ such that $y_1^2 = x_1^3 + ax_1 + b$. Fix $\theta \in \mathbb{K}$ such that $4\theta^2 x_1 = x_1^3 + 4b - 4y_1^2$. Let $x_2, y_2 \in \mathbb{K}$ such that $y_2^2 = x_2^3 + ax_2 + b$. It follows that $y_1 + y_2 = 0$ if and only if $(x_2, y_2) = (x_1, -y_1)$ or $(x_2, y_2) = (-x_1/2 + \theta, -y_1)$ or $(x_2, y_2) = (-x_1/2 - \theta, -y_1)$.*

Proof. $\Rightarrow$: Assume that $y_1 + y_2 = 0$. Then, y_2 is determined to be $-y_1$. Using the equation $y_2^2 = x_2^3 + ax_2 + b$ one gets $x_2^3 + ax_2 + b - (-y_1)^2 = (x_2 - x_1)(x_2 - x_1')(x_2 - x_1'')$. The exceptional cases are obtained by extracting x_1' and x_1'' in terms of x_1. $\Leftarrow$: The claims follow trivially by substitution. □

Exception handling in the general case Algorithm 4.1.1 provides a complete addition on all elliptic curves over $\mathbb{K}$ with $\mathrm{char}(\mathbb{K}) \neq 2, 3$. The point that appears as Ω is the point at infinity which serves as the identity element. This algorithm or its verbal explanation can be found in many literature resources. For instance, see [Sil94], [Ins00].

Algorithm 4.1.1: Addition law in affine coordinates for short Weierstrass form

input : $P_1, P_2, \Omega \in E_{\mathbf{S},a,b}(\mathbb{K})$.
output : $P_1 + P_2$.

if $P_1 = \Omega$ **then return** P_2.
else if $P_2 = \Omega$ **then return** P_1.
else if $x_1 = x_2$ **then**
    if $y_1 \neq y_2$ **then return** Ω.
    else if $y_1 = 0$ **then return** Ω.
    else
        $x_3 \leftarrow ((3x_1^2 + a)/(2y_1))^2 - 2x_1$.
        $y_3 \leftarrow ((3x_1^2 + a)/(2y_1))(x_1 - x_3) - y_1$.
        return (x_3, y_3).
    end
else
    $x_3 \leftarrow ((y_1 - y_2)/(x_1 - x_2))^2 - x_1 - x_2$.
    $y_3 \leftarrow ((y_1 - y_2)/(x_1 - x_2))(x_1 - x_3) - y_1$.
    return (x_3, y_3).
end

More formulae It has already been noted that the presented doubling formulae are not of minimal total degree. A set of minimal-degree point doubling formulae is given by $2(x_1, y_1) = (x_3, y_3)$ where

$$x_3 = \frac{x_1}{4} - \frac{9bx_1}{4y_1^2} - a\frac{3x_1^2 - a}{4y_1^2}, \tag{4.7}$$

$$y_3 = \frac{y_1}{2} - \frac{3b + 2ax_1}{2y_1} - x_3\frac{3x_1^2 + a}{2y_1} \tag{4.8}$$

assuming that $y_1 \neq 0$. These formulae are adapted from [CLN09]. The total degree of x_3 drops from 6 to 5 and y_3 from 9 to 7.

Furthermore, if $c \in \mathbb{K}$ such that $c^2 = b$ the doubling formulae can be written by

$$x_3 = x_1(\mu - \mu^2) + a\sigma, \tag{4.9}$$

$$y_3 = (y_1 - c)\mu^3 + a\delta - c \tag{4.10}$$

with $\mu = (y_1 + 3c)/(2y_1)$, $\sigma = (a - 3x_1^2)/(2y_1)^2$, $\delta = (3x_1(y_1 - 3c)(y_1 + 3c) - a(9x_1^2 + a))/(2y_1)^3$. A slightly modified version of these formulae will later be used in Chapter 7 for efficient Tate pairing computation.

4.2 Extended Jacobi quartic form

This section presents the group law on $E_{\mathbf{Q},d,a}$ in affine coordinates. It also investigates the exceptional summands for each formula and provides a complete addition algorithm for all extended Jacobi quartic curves by properly handling an entire set of divide-by-zero exceptions. In addition, practical ways of preventing these exceptions are explained.

Throughout this section, let $\mathbb{K}$ be a field of odd characteristic. Recall from Chapter 2 that

an extended Jacobi quartic curve is defined by

$$E_{\mathbf{Q},d,a} : \ y^2 = dx^4 + 2ax^2 + 1$$

where $a, d \in \mathbb{K}$ with $d(a^2 - d) \neq 0$. Assume that d is a square in $\mathbb{K}$. Recall from Chapter 2 that the set of $\mathbb{K}$-rational points on the desingularization of $E_{\mathbf{Q},d,a}$ is defined by

$$E_{\mathbf{Q},d,a}(\mathbb{K}) = \{(x, y) \in \mathbb{K}^2 \mid y^2 = dx^4 + 2ax^2 + 1\} \cup \{\Omega_1, \Omega_2\}$$

where Ω_1, Ω_2 are points at infinity.

Identity element and negation The identity element can suitably be taken as $(0, 1)$. Let (x_1, y_1) be a point on $E_{\mathbf{Q},d,a}$. The negative of (x_1, y_1) is $(-x_1, y_1)$.

Doubling The doubling formulae on $E_{\mathbf{Q},d,a}$ are given by $[2](x_1, y_1) = (x_3, y_3)$ where

$$x_3 = \frac{2x_1y_1}{2 - y_1^2 + 2ax_1^2}, \tag{4.11}$$

$$y_3 = \frac{2y_1^2(y_1^2 - 2ax_1^2)}{(2 - y_1^2 + 2ax_1^2)^2} - 1 \tag{4.12}$$

assuming that $2 - y_1^2 + 2ax_1^2 \neq 0$. These formulae do not depend on the curve constant d and are of minimal total degree. By the curve equation the denominator $2 - y_1^2 + 2ax_1^2$ is equivalent to $1 - dx_1^4$. This denominator can also be used if the total degree is not of concern.

The identity element is the point $(0, 1)$ which can be determined by solving $y_1^2 = dx_1^4 + 2ax_1^2 + 1$ and $(x_3, y_3) = (x_1, y_1)$ for x_1 and y_1 where x_3 and y_3 are given by (4.11) and (4.12).

Affine points of order 2 can be determined by solving $y_1^2 = dx_1^4 + 2ax_1^2 + 1$ and $(x_3, y_3) = (0, 1)$ for x_1 and y_1 where x_3 and y_3 are given by (4.11) and (4.12). The point $(0, -1)$ is of order 2. There are no other affine points of order 2. There are three points of order 2 in total (over a sufficiently large finite extension of $\mathbb{K}$). Therefore, both points at infinity Ω_1 and Ω_2 have to be of order 2.

The four points of the form $(x, 0)$ are of order 4 which can be determined by solving $y_1^2 = dx_1^4 + 2ax_1^2 + 1$ and $(x_3, y_3) = (0, -1)$ for x_1 and y_1 where x_3 and y_3 are given by (4.11) and (4.12). There are twelve points of order 4 in total (over a sufficiently large finite extension of $\mathbb{K}$). Therefore two-times-the-remaining-eight-points must be either Ω_1 or Ω_2. These eight affine points can be explicitly determined by solving $y_1^2 = dx_1^4 + 2ax_1^2 + 1$ and $2 - y_1^2 + 2ax_1^2 = 0$ for x_1 and y_1. These points are the only exceptions of (4.11) and (4.12). The following remark is immediate.

Remark 4.2.1. $[2](x_1, y_1)$ is a point at infinity if and only if $2 - y_1^2 + 2ax_1^2 = 0$.

Remark 4.2.1 does not extend to the case of generic additions. However, it is still useful in proving some lemmas regarding the generic addition formulae which will be presented next.

Dedicated addition Further let (x_2, y_2) be a point on $E_{\mathbf{Q},d,a}$. The addition formulae on $E_{\mathbf{Q},d,a}$ is given by $(x_1, y_1) + (x_2, y_2) = (x_3, y_3)$ where

$$x_3 = \frac{x_1^2 - x_2^2}{x_1y_2 - y_1x_2}, \tag{4.13}$$

$$y_3 = \frac{(x_1^2 + x_2^2)(y_1y_2 - 2ax_1x_2) - 2x_1x_2(1 + dx_1^2x_2^2)}{(x_1y_2 - y_1x_2)^2} \tag{4.14}$$

assuming that $x_1y_2 - y_1x_2 \neq 0$. These formulae are of minimal total degree. These formulae do not work for identical summands hence the name *dedicated*.

If $(x_1, y_1)+(x_2, y_2)$ is a point at infinity then $x_1y_2-y_1x_2 = 0$. Otherwise, $(x_1, y_1)+(x_2, y_2)$ would be an affine point since it can be shown using the relations $y_1^2 = dx_1^4 + 2ax_1^2 + 1$ and $y_2^2 = dx_2^4 + 2ax_2^2 + 1$ that the algebraic expressions for (x_3, y_3) satisfy $y_3^2 = dx_3^4 + 2ax_3^2 + 1$. The converse, however, does not necessarily apply. This means that if $x_1y_2 - y_1x_2 = 0$ then $(x_1, y_1) + (x_2, y_2)$ may not be a point at infinity. Therefore it is worth investigating the exceptional cases. The denominators of (4.13) and (4.14) vanish for some summands which are described in the following lemma explicitly.

Lemma 4.2.2. *Let $a, d \in \mathbb{K}$ with $d(a^2 - d) \neq 0$. Fix $\delta \in \mathbb{K}$ so that $\delta^2 = d$. Fix $x_1 \in \mathbb{K}\backslash\{0\}$ and $y_1 \in \mathbb{K}$ such that $y_1^2 = dx_1^4 + 2ax_1^2 + 1$. Let $x_2, y_2 \in \mathbb{K}$ such that $y_2^2 = dx_2^4 + 2ax_2^2 + 1$. Then $x_1y_2 - y_1x_2 = 0$ if and only if $(x_2, y_2) \in S$ where*

$$S = \left[(x_1, y_1), (-x_1, -y_1), \left(\frac{1}{\delta x_1}, \frac{y_1}{\delta x_1^2}\right), \left(\frac{-1}{\delta x_1}, \frac{-y_1}{\delta x_1^2}\right)\right].$$

Proof. $\Rightarrow$: Assume that $x_1y_2 - y_1x_2 = 0$. Solving the system of equations $x_1y_2 - y_1x_2 = 0$, $y_1^2 = dx_1^4 + 2ax_1^2 + 1$ for x_2 and y_2 gives S. All entries in S are defined since $x_1 \neq 0$.
$\Leftarrow$: The claim follows trivially by substitution. □

The following lemma shows that if one of the summands is of odd order then in the presence of an exception, the other summand is always of even order.

Lemma 4.2.3. *Let a, d, x_1, y_1, x_2, y_2 be defined as in Lemma 4.2.2. Assume that $P_1 = (x_1, y_1)$ is a fixed point of odd order. Assume that $P_2 \in S\backslash\{P_1\}$. Then P_2 is of even order.*

Proof. First note that points at infinity are of order 2. Assume that $P_1 = (x_1, y_1)$ is a fixed point of odd order hence not a point at infinity. Suppose that P_2 is of odd order hence not a point at infinity. It follows that $P_1 \pm P_2$, $M = 2P_1$, and $N = 2P_2$ are all of odd order hence not points at infinity.

Assume that $P_2 \in S\backslash\{P_1\}$. So, $P_2 \neq P_1$. In addition, $x_1y_2 - y_1x_2 = 0$ by Lemma 4.2.2. It follows that $P_1 \neq -P_2$, for otherwise, $x_1y_2 - y_1x_2 = 2x_1y_1 = 0$ which means that x or y is zero. But then P_1 would be of even order since $x_1 \neq 0$.

Note that $y_2 = y_1x_2/x_1$ is defined since $x_1 \neq 0$, by the definition. Using this relation

together with (4.11), (4.12), and the curve equation gives [1]

$$\begin{aligned} x(N)^2 &= \frac{(2x_2y_2)^2}{(2-y_2^2+2ax_2^2)^2} = \frac{(2x_2y_2)^2}{(2-y_2^2+2ax_2^2)^2+4(y_2^2-(dx_2^4+2ax_2^2+2))} = \\ &\frac{(2x_2y_2)^2}{(y_2^2-2ax_2^2)^2-4dx_2^4} = \frac{(2x_2\frac{y_1x_2}{x_1})^2}{((\frac{y_1x_2}{x_1})^2-2ax_2^2)^2-4dx_2^4} = \frac{(2x_1y_1)^2}{(2-y_1^2+2ax_1^2)^2} = x(M)^2, \\ y(N) &= \frac{2y_2^2(y_2^2-2ax_2^2)}{(2-y_2^2+2ax_2^2)^2}-1 = \frac{2y_2^2(y_2^2-2ax_2^2)}{(y_2^2-2ax_2^2)^2-4dx_2^4}-1 = \\ &\frac{2(\frac{y_1x_2}{x_1})^2((\frac{y_1x_2}{x_1})^2-2ax_2^2)}{((\frac{y_1x_2}{x_1})^2-2ax_2^2)^2-4dx_2^4}-1 = \frac{2y_1^2(y_1^2-2ax_1^2)}{(2-y_1^2+2ax_1^2)^2}-1 = y(M). \end{aligned}$$

Hence, $M = \pm N$. But then $M \mp N = 2P_1 \mp 2P_2 = 2(P_1 \mp P_2) = (0,1)$. Since $P_1 \neq \pm P_2$, it follows that $P_1 \mp P_2$ is a point of order 2, a contradiction. In conclusion, $P_2 \in S\backslash\{P_1\}$ is of even order provided that P_1 is of odd order. □

A practical solution is now provided to prevent the exceptional cases of (4.13) and (4.14).

Lemma 4.2.4. *Let $\mathbb{K}$ be a field of odd characteristic. Let $E_{\mathbf{Q},d,a}$ be an extended Jacobi quartic curve defined over $\mathbb{K}$. Let $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ be points on $E_{\mathbf{Q},d,a}$. Assume that P_1 and P_2 are of odd order with $P_1 \neq P_2$. It follows that $x_1y_2 - y_1x_2 \neq 0$.*

Proof. Assume that P_1 and P_2 are of odd order with $P_1 \neq P_2$. Suppose that $x_1 = 0$ and $x_2 = 0$. Then, $P_1 = P_2 = (0,1)$, contradiction. So, either $x_1 \neq 0$ or $x_2 \neq 0$. The claim then follows from Lemma 4.2.2 and Lemma 4.2.3 (by swapping P_1 and P_2 when necessary). □

Unified addition Alternative addition formulae on $E_{\mathbf{Q},d,a}$ are given by $(x_1, y_1) + (x_2, y_2) = (x_3, y_3)$ where

$$x_3 = \frac{x_1y_2 + y_1x_2}{1 - dx_1^2x_2^2}, \tag{4.15}$$

$$y_3 = \frac{(y_1y_2 + 2ax_1x_2)(1 + dx_1^2x_2^2) + 2dx_1x_2(x_1^2 + x_2^2)}{(1 - dx_1^2x_2^2)^2} \tag{4.16}$$

assuming that $1 - dx_1^2x_2^2 \neq 0$. These formulae work for identical summands in most of the cases hence the name *unified*.

If $(x_1, y_1) + (x_2, y_2)$ is a point at infinity then $1 - dx_1^2x_2^2 = 0$. Otherwise, $(x_1, y_1) + (x_2, y_2)$ would be an affine point since it can be shown using the relations $y_1^2 = dx_1^4 + 2ax_1^2 + 1$ and $y_2^2 = dx_2^4 + 2ax_2^2 + 1$ that the algebraic expressions for (x_3, y_3) satisfy $y_3^2 = dx_3^4 + 2ax_3^2 + 1$. The converse, however, does not necessarily apply. This means that if $1 - dx_1^2x_2^2 = 0$ then $(x_1, y_1) + (x_2, y_2)$ may not be a point at infinity. Therefore it is worth investigating the exceptional cases. The denominators of (4.15) and (4.16) vanish for some summands which are described in the following lemma explicitly.

Lemma 4.2.5. *Let $a, d \in \mathbb{K}$ with $d(a^2 - d) \neq 0$. Fix $\delta \in \mathbb{K}$ so that $\delta^2 = d$. Fix $x_1 \in \mathbb{K}\backslash\{0\}$ and $y_1 \in \mathbb{K}$ such that $y_1^2 = dx_1^4 + 2ax_1^2 + 1$. Let $x_2, y_2 \in \mathbb{K}$ such that $y_2^2 = dx_2^4 + 2ax_2^2 + 1$. Then*

[1] $x(N)$ means the x-coordinate of the point N.

$1 - dx_1^2x_2^2 = 0$ *if and only if* $(x_2, y_2) \in S'$ *where*

$$S' = [(\frac{1}{\delta x_1}, \frac{-y_1}{\delta x_1^2}), (\frac{-1}{\delta x_1}, \frac{y_1}{\delta x_1^2}), (\frac{1}{\delta x_1}, \frac{y_1}{\delta x_1^2}), (\frac{-1}{\delta x_1}, \frac{-y_1}{\delta x_1^2})].$$

Proof. $\Rightarrow$: Assume that $1 - dx_1^2x_2^2 = 0$. Solving the system of equations $1 - dx_1^2x_2^2 = 0$, $y_1^2 = dx_1^4 + 2ax_1^2 + 1$ for x_2 and y_2 gives S'. All entries in S' are defined since $x_1 \neq 0$.
$\Leftarrow$: The claim follows trivially by substitution. □

This lemma and Lemma 4.2.2 excludes $x_1 = 0$. If so, $1 - dx_1^2x_2^2 \neq 0$ as desired.

The following lemma shows that if one of the summands is of odd order then in the presence of a vanished denominator, the other summand is always of even order.

Lemma 4.2.6. *Let a, d, x_1, y_1, x_2, y_2 be defined as in Lemma 4.2.5. Assume that $P_1 = (x_1, y_1)$ is a fixed point of odd order. Assume that $P_2 = (x_2, y_2) \in S'$. Then P_2 is of even order.*

Proof. First note that points at infinity are of order 2. Assume that $P_1 = (x_1, y_1)$ is a fixed point of odd order hence not a point at infinity. Suppose that P_2 is of odd order hence not a point at infinity. It follows that $P_1 \pm P_2$, $M = 2P_1$, and $N = 2P_2$ are all of odd order hence not points at infinity.

Assume that $P_2 \in S'$. Then, $1 - dx_1^2x_2^2 = 0$ by Lemma 4.2.5 and it follows that $P_1 \neq \pm P_2$, for otherwise, $1 - dx_1^4 = 2 - y_1^2 + 2ax_1^2 = 0$ and P_1 would be of even order by Remark 4.2.1.

Note that $x_1 \neq 0$ since $1 - dx_1^2x_2^2 = 0$ (also true by definition). So, $x_2^2 = 1/(dx_1^2)$ is defined. Using this relation together with (4.15), (4.16), and the curve equation gives

$$\begin{aligned}
x(N)^2 &= \frac{(2x_2y_2)^2}{(1-dx_2^4)^2} = \frac{4x_2^2(dx_2^4 + 2ax_2^2 + 1)}{(1-dx_2^4)^2} = \frac{4\frac{1}{dx_1^2}(d(\frac{1}{dx_1^2})^2 + 2a\frac{1}{dx_1^2} + 1)}{(1-d(\frac{1}{dx_1^2})^2)^2} = \\
&\frac{4x_1^2(dx_1^4 + 2ax_1^2 + 1)}{(1-dx_1^4)^2} = \frac{(2x_1y_1)^2}{(1-dx_1^4)^2} = x(M)^2, \\
y(N) &= \frac{(y_2^2 + 2ax_2^2)(1+dx_2^4) + 4dx_2^4}{(1-dx_2^4)^2} = \frac{((dx_2^4 + 2ax_2^2 + 1) + 2ax_2^2)(1+dx_2^4) + 4dx_2^4}{(1-dx_2^4)^2} = \\
&\frac{((d(\frac{1}{dx_1^2})^2 + 2a\frac{1}{dx_1^2} + 1) + 2a\frac{1}{dx_1^2})(1 + d(\frac{1}{dx_1^2})^2) + 4d(\frac{1}{dx_1^2})^2}{(1-d(\frac{1}{dx_1^2})^2)^2} = \\
&\frac{((dx_1^4 + 2ax_1^2 + 1) + 2ax_1^2)(1+dx_1^4) + 4dx_1^4}{(1-dx_1^4)^2} = \frac{(y_1^2 + 2ax_1^2)(1+dx_1^4) + 4dx_1^4}{(1-dx_1^4)^2} = y(M).
\end{aligned}$$

Hence, $M = \pm N$. But then $M \mp N = 2P_1 \mp 2P_2 = 2(P_1 \mp P_2) = (0, 1)$. Since $P_1 \neq \pm P_2$, it follows that $P_1 \mp P_2$ is a point of order 2, contradiction. In conclusion, $P_2 \in S'$ is of even order provided that P_1 is of odd order. □

In the following lemma, with reasonable assumptions, it is shown that exceptions can be prevented regardless of any assumption on the curve constants.

Lemma 4.2.7. *Let $\mathbb{K}$ be a field of odd characteristic. Let $E_{\mathbf{Q},d,a}$ be an extended Jacobi quartic curve defined over $\mathbb{K}$. Let $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ be points on $E_{\mathbf{Q},d,a}$. Assume that P_1 and P_2 are of odd order. It follows that $1 - dx_1^2x_2^2 \neq 0$.*

Proof. Assume that P_1 and P_2 are of odd order. Assume that $x_1x_2 = 0$ then $1 - dx_1^2x_2^2 \neq 0$ as desired. From now on assume that $x_1x_2 \neq 0$. The claim follows from Lemma 4.2.5 and Lemma 4.2.6 (by swapping P_1 and P_2 when necessary). □

Exception handling in the general case Algorithm 4.2.1 provides a complete addition on all extended Jacobi quartic curves. The correctness of the algorithm follows from two

Algorithm 4.2.1: Addition law in affine coordinates for extended Jacobi quartic form

input : $P_1, P_2, \Omega_1, \Omega_2 \in E_{\mathbf{Q},d,a}(\mathbb{K})$ and fixed $\delta \in \mathbb{K}$ such that $\delta^2 = d$.
output : $P_1 + P_2$.

1 **if** $P_1 \in \{\Omega_1, \Omega_2\}$ **then** $P_t \leftarrow P_1$, $P_1 \leftarrow P_2$, $P_2 \leftarrow P_t$.
2 **if** $P_2 = \Omega_1$ **then**
3 **if** $P_1 = \Omega_1$ **then return** $(0, 1)$.
4 **else if** $P_1 = \Omega_2$ **then return** $(0, -1)$.
5 **else if** $P_1 = (0, 1)$ **then return** Ω_1.
6 **else if** $P_1 = (0, -1)$ **then return** Ω_2.
7 **else return** $(-1/(\delta x_1), y_1/(\delta x_1^2))$.
8 **else if** $P_2 = \Omega_2$ **then**
9 **if** $P_1 = \Omega_1$ **then return** $(0, -1)$.
10 **else if** $P_1 = \Omega_2$ **then return** $(0, 1)$.
11 **else if** $P_1 = (0, -1)$ **then return** Ω_1.
12 **else if** $P_1 = (0, 1)$ **then return** Ω_2.
13 **else return** $(1/(\delta x_1), -y_1/(\delta x_1^2))$.
14 **else if** $x_1y_2 - y_1x_2 \neq 0$ **then**
15 $x_3 \leftarrow (x_1^2 - x_2^2)/(x_1y_2 - y_1x_2)$.
16 $y_3 \leftarrow ((x_1^2 + x_2^2)(y_1y_2 - 2ax_1x_2) - 2x_1x_2(1 + dx_1^2x_2^2))/(x_1y_2 - y_1x_2)^2$.
17 **return** (x_3, y_3).
18 **else if** $1 - dx_1^2x_2^2 \neq 0$ **then**
19 $x_3 \leftarrow (x_1y_2 + y_1x_2)/(1 - dx_1^2x_2^2)$.
20 $y_3 \leftarrow ((y_1y_2 + 2ax_1x_2)(1 + dx_1^2x_2^2) + 2dx_1x_2(x_1^2 + x_2^2))/(1 - dx_1^2x_2^2)^2$.
21 **return** (x_3, y_3).
22 **else**
23 **if** $P_2 = (1/(\delta x_1), y_1/(\delta x_1^2))$ **then return** Ω_1.
24 **else return** Ω_2.
25 **end**

observations. Firstly, when a point at infinity is involved as the sum or as one of the summands along the lines 2 to 13, it is tedious but straightforward to check that the output of the algorithm is correct using the *implicit* technique mentioned at the start of the chapter. Line 1 conditionally swaps the inputs to eliminate half of the input-wise symmetric branches. The second observation is that *glueing* together the unified addition and the dedicated addition formulae is enough to handle all exceptions when both of the summands and the sum are affine points. This fact follows from Lemma 4.2.5 and Lemma 4.2.2 by observing that $\#(S' \cap S) = 2$. This means that if $(x_2, y_2) \in S' \cap S$ then the output must be a point at infinity (lines 23 and 24) since there are exactly two points at infinity. The remaining exceptional cases which occur at $(x_2, y_2) \in S' \backslash (S' \cap S)$ are handled by the dedicated addition formulae (lines 15 and 16). Similarly the exceptions at $(x_2, y_2) \in S \backslash (S' \cap S)$ are handled by the unified addition formulae (lines 19 and 20).

The points at infinity on the desingularized projective closure of $E_{\mathbf{Q},d,a}$ are not defined over $\mathbb{K}$ if d is not a square in $\mathbb{K}$. Having noted this, the following lemma implies that these addition

formulae are complete[2] provided that d is not a square in $\mathbb{K}$.

Lemma 4.2.8. *Let $d, x_1, x_2 \in \mathbb{K}$. Assume that d is non-square. Then $1 - dx_1^2x_2^2 \neq 0$.*

Proof. Suppose that $1 - dx_1^2x_2^2 = 0$. So $d, x_1, x_2 \neq 0$. But then $d = (1/(x_1x_2))^2$, contradiction. □

Algorithm 4.2.1 complies with the completeness criterion since only the lines 19 to 21 are necessary in this case. Note that the assumption on the curve constant d limits the number of curves in extended Jacobi quartic form for which the unified addition formulae are complete.

Algorithm 4.2.1 also complies with Lemma 4.2.7. If P_1 and P_2 are points of odd order then only the lines 19 to 21 are necessary. This technique applies to all extended Jacobi quartic curves.

Algorithm 4.2.1 also complies with Lemma 4.2.4. If P_1 and P_2 are distinct points of odd order then only the lines 15 to 17 are necessary. This technique applies to all extended Jacobi quartic curves. The doubling formulae (4.11) and (4.12) are enough to handle the special case $P_1 = P_2$.

More formulae Using the tools of Chapter 3, it is possible to derive other low-degree addition formulae;

$$x_3 = \frac{e(x_1^2 - x_2^2) + f(x_1y_2 + y_1x_2)}{e(x_1y_2 - y_1x_2) + f(1 - dx_1^2x_2^2)} \tag{4.17}$$

assuming that $e(x_1y_2 - y_1x_2) + f(1 - dx_1^2x_2^2) \neq 0$ where $e, f \in \mathbb{K}$ such that $ef \neq 0$. The analogous formulae for y_3 can be derived by using the square of the same denominator. In fact, it is still possible to derive many more low-degree addition formulae for y_3;

$$y_3 = \frac{y_1y_2 + 2ax_1x_2 \pm \sqrt{d}x_1^2 \pm \sqrt{d}x_2^2}{(1 \mp \sqrt{d}x_1x_2)^2} \mp \sqrt{d}x_3^2, \tag{4.18}$$

$$y_3 = \frac{(x_1^2 - x_2^2)^2 - (x_1y_2 - y_1x_2)(x_1^3y_2 - y_1x_2^3)}{x_1x_2(x_1y_2 - y_1x_2)^2}, \tag{4.19}$$

$$y_3 = \frac{(x_1 - x_2)^2}{(x_1y_2 - y_1x_2)^2}(y_1y_2 - 2ax_1x_2 + 1 + dx_1^2x_2^2) - 1, \tag{4.20}$$

$$y_3 = \frac{2(x_1y_1 - x_2y_2) - (x_1y_2 - y_1x_2)(y_1y_2 + 2ax_1x_2)}{(x_1y_2 - y_1x_2)(1 - dx_1^2x_2^2)}, \tag{4.21}$$

$$y_3 = \frac{(x_1 - x_2)(y_1 + y_2 + dx_1x_2(x_1^2y_2 + y_1x_2^2))}{(x_1y_2 - y_1x_2)(1 - dx_1^2x_2^2)} - 1, \tag{4.22}$$

$$y_3 = \frac{(1 \pm \sqrt{d}x_1x_2)(x_1y_1 - x_2y_2 \pm \sqrt{d}x_1^3y_2 \mp \sqrt{d}y_1x_2^3)}{(x_1y_2 - y_1x_2)(1 - dx_1^2x_2^2)} \mp \sqrt{d}x_3^2, \tag{4.23}$$

$$y_3 = \frac{(x_1 - x_2)(1 \pm \sqrt{d}x_1x_2)}{(x_1y_2 - y_1x_2)(1 - dx_1^2x_2^2)}(y_1 + y_2 \pm \sqrt{d}x_1^2y_2 \pm \sqrt{d}y_1x_2^2) \mp \sqrt{d}x_3^2 - 1. \tag{4.24}$$

The formulae (4.18) and (4.20) will be recalled in §5.2 of Chapter 5. The other formulae tend to include more subexpressions which are not commonly shared. Therefore, their evaluation turns out to be more costly and thus not considered hereafter.

[2] i.e. these formulae define the addition law

Literature notes Other results related to the affine formulae for extended Jacobi quartic form can be found in the literature. Some pointers are [Jac29], [WW27], [Yui88], and [MM99]. The dedicated addition formulae presented in this section are essentially the same formulae used by Chudnovsky and Chudnovsky in [CC86, 4.10i, p.418] with the minor detail that the formulae in this section are given in affine coordinates, the curve equation is $y^2 = dx^4+2ax^2+1$ rather than $y^2 = x^4+a'x^2+b'$, and the identity is the point $(0, 1)$ rather than a point at infinity.

4.3 Twisted Hessian form

This section presents the group law on $E_{\mathbf{H},a,d}$ in affine coordinates. It also investigates the exceptional summands for each set of formulae and provides a complete addition algorithm for all twisted Hessian curves by properly handling an entire set of divide-by-zero exceptions. In addition, practical ways of preventing these exceptions are explained.

Throughout this section, let $\mathbb{K}$ be a field with $\mathrm{char}(\mathbb{K}) \neq 2, 3$. Recall from Chapter 2 that a twisted Hessian curve is defined by

$$E_{\mathbf{H},a,d} : ax^3 + y^3 + 1 = dxy$$

where $a, d \in \mathbb{K}$ with $a(27a - d^3) \neq 0$. Assume that a is a cube in $\mathbb{K}$. Recall from Chapter 2 that the set of $\mathbb{K}$-rational points on $E_{\mathbf{H},a,d}$ is defined by

$$E_{\mathbf{H},a,d}(\mathbb{K}) = \{(x, y) \in \mathbb{K}^2 \mid ax^3 + y^3 + 1 = dxy\} \cup \{\Omega_1, \Omega_2, \Omega_3\}$$

where $\Omega_1, \Omega_2, \Omega_3$ are points at infinity.

Identity element and negation The identity element can suitably be taken as $(0, -1)$. In fact, this is the only $\mathbb{K}$-rational point on $E_{\mathbf{H},a,d}$ that does not depend on the field of definition. Let (x_1, y_1) be a point on $E_{\mathbf{H},a,d}$. The negative of (x_1, y_1) is $(x_1/y_1, 1/y_1)$.

Doubling The doubling formulae on $E_{\mathbf{H},a,d}$ are given by $[2](x_1, y_1) = (x_3, y_3)$ where

$$x_3 = (x_1 - y_1^3 x_1)/(ay_1x_1^3 - y_1), \tag{4.25}$$

$$y_3 = (y_1^3 - ax_1^3)/(ay_1x_1^3 - y_1). \tag{4.26}$$

assuming that $ay_1x_1^3 - y_1 \neq 0$, see [BKL09]. These formulae do not depend on the curve constant d and are of minimal total degree. By the curve equation the denominator $ay_1x_1^3 - y_1$ is equivalent to $y_1(dx_1y_1 - y_1^3 - 2)$ and is of the same degree. This denominator can also be used if desired.

To study the order of points at infinity tripling formulae are needed which are presented next.

Tripling Let (x_1, y_1) be a point on $E_{\mathbf{H},a,d}$. The tripling formulae on $E_{\mathbf{H},a,d}$ are given by $[3](x_1, y_1) = (x_3, y_3)$ where

$$x_3 = \frac{x_1y_1((1-ax_1^3)^2 + (y_1^3 - ax_1^3)(y_1^3-1))}{(1-ax_1^3)^2 + (y_1^3-ax_1^3)(y_1^3-1) - (y_1^3-1)^2(1-ax_1^3)}, \tag{4.27}$$

$$y_3 = \frac{y_1^3(1-ax_1^3)^2 - (y_1^3-ax_1^3)(1-y_1^3)}{(1-ax_1^3)^2 + (y_1^3-ax_1^3)(y_1^3-1) - (y_1^3-1)^2(1-ax_1^3)} \tag{4.28}$$

assuming that $(1-ax_1^3)^2 + (y_1^3-ax_1^3)(y_1^3-1) - (y_1^3-1)^2(1-ax_1^3) \neq 0$. These tripling formulae are adapted from [BKL09]. These formulae do not depend on the curve constant d.

The two points of the form $(0, y)$ (excluding the identity) and three points of the form $(x, 0)$ are of order 3 which can be determined by solving $ax_1^3 + y_1^3 + 1 = dx_1y_1$ and $(x_3, y_3) = (0, -1)$ for x_1 and y_1 where x_3 and y_3 are given by (4.27) and (4.28). There are eight points of order 3 in total (over a sufficiently large finite extension of $\mathbb{K}$). Therefore the remaining 3 points of order 3 have to be the points at infinity Ω_1, Ω_2, and Ω_3.

Dedicated addition Further let (x_2, y_2) be a point on $E_{\mathbf{H},a,d}$. The addition formulae on $E_{\mathbf{H},a,d}$ are given by $(x_1, y_1) + (x_2, y_2) = (x_3, y_3)$ where

$$x_3 = (y_1x_2^2 - x_1^2y_2)/(x_1y_2^2 - y_1^2x_2), \tag{4.29}$$

$$y_3 = (x_1y_1 - x_2y_2)/(x_1y_2^2 - y_1^2x_2). \tag{4.30}$$

assuming that $x_1y_2^2 - y_1^2x_2 \neq 0$. These formulae are of minimal total degree and do not depend on d. These formulae do not work for identical summands hence the name *dedicated*.

If $(x_1, y_1) + (x_2, y_2)$ is a point at infinity then $x_1y_2^2 - y_1^2x_2 = 0$. Otherwise, $(x_1, y_1) + (x_2, y_2)$ would be an affine point since it can be shown using the relations $ax_1^3 + y_1^3 + 1 = dx_1y_1$ and $ax_2^3 + y_2^3 + 1 = dx_2y_2$ that the algebraic expressions for (x_3, y_3) satisfy $ax_2^3 + y_2^3 + 1 = dx_2y_2$. The converse, however, does not necessarily apply. This means that if $x_1y_2^2 - y_1^2x_2 = 0$ then $(x_1, y_1) + (x_2, y_2)$ may not be a point at infinity. Therefore it is worth investigating the exceptional cases. The denominators of (4.29) and (4.30) vanish for some summands which are described in the following lemma explicitly.

Lemma 4.3.1. *Let $a, d \in \mathbb{K}$ with $a(27a - d^3) \neq 0$. Fix $\omega \in \mathbb{K}\backslash\{1\}$ so that $\omega^3 = 1$. Fix $\alpha \in \mathbb{K}$ so that $\alpha^3 = a$. Fix $x_1, y_1 \in \mathbb{K}\backslash\{0\}$ such that $ax_1^3 + y_1^3 + 1 = dx_1y_1$. Let $x_2, y_2 \in \mathbb{K}$ such that $ax_2^3 + y_2^3 + 1 = dx_2y_2$. Now, $x_1y_2^2 - y_1^2x_2 = 0$ if and only if $(x_2, y_2) \in S$ where*

$$S = \big[(x_1, y_1), (\omega x_1, \frac{y_1}{\omega}), (\frac{x_1}{\omega}, \omega y_1), (\frac{\alpha}{ax_1}, \frac{y_1}{\alpha x_1}), (\frac{\omega\alpha}{ax_1}, \frac{y_1}{\omega\alpha x_1}), (\frac{\alpha}{a\omega x_1}, \frac{\omega y_1}{\alpha x_1})\big].$$

Proof. $\Rightarrow$: Assume that $x_1y_2^2 - y_1^2x_2 = 0$. Solving the equations $x_1y_2^2 - y_1^2x_2 = 0$ and $ax_2^3 + y_2^3 + 1 = dx_2y_2$ simultaneously for x_2 and y_2 gives S. Note that the last three entries are equal to $-(\frac{1}{\alpha y_1}, \frac{\alpha x_1}{y_1})$, $-(\frac{1}{\omega\alpha y_1}, \frac{\omega\alpha x_1}{y_1})$, $-(\frac{\omega}{\alpha y_1}, \frac{\alpha x_1}{\omega y_1})$, respectively. All entries in S are defined since $x_1y_1 \neq 0$. $\Leftarrow$: The claim follows trivially by substitution. □

The following lemma shows that if one of the summands is of odd order then in the presence of an exception, the other summand is always of even order.

Lemma 4.3.2. *Let a, d, x_1, y_1, x_2, y_2 be defined as in Lemma 4.3.1. Assume that $P_1 = (x_1, y_1)$ is a fixed point of odd order such that $3 \nmid \#\langle P_1\rangle$. Assume that $P_2 = (x_2, y_2) \in S\backslash\{P_1\}$. Then, $P_2 \notin \langle P_1\rangle$.*

Proof. Note that the points at infinity (over the extension of $\mathbb{K}$ where they exist) are of order 3. Assume that $P_1 = (x_1, y_1)$ is a fixed point of odd order such that $3 \nmid \#\langle P_1\rangle$ hence not a point at infinity. Suppose that $P_2 \in \langle P_1\rangle$ hence not a point at infinity. It follows that $P_1 \pm P_2$, $M = 3P_1$, and $N = 3P_2$ are all in $\langle P_1\rangle$ hence not points at infinity.

Assume that $P_2 \in S\backslash\{P_1\}$. $P_2 \neq \pm P_1$. Using the dedicated tripling formulae, it is easy to check by substitutions (of five possible algebraic expressions for P_2 in S) that either $3P_1 = 3P_2$ or $3P_1 = -3P_2$. Then, either $3(P_1 - P_2) = (0, -1)$ or $3(P_1 + P_2) = (0, -1)$. Then, $P_1 \pm P_2 \notin \langle P_1\rangle$, contradiction. In conclusion, $P_2 \notin \langle P_1\rangle$. □

A practical solution is now provided to prevent the exceptional cases of (4.29) and (4.30).

Lemma 4.3.3. *Let $E_{\mathbf{H},a,d}$ be a twisted Hessian curve defined over $\mathbb{K}$. Let $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ be points on $E_{\mathbf{H},a,d}$. Assume that P_1 and P_2 are of odd order such that $P_1 \neq P_2$, $3 \nmid \#\langle P_1\rangle$, and $3 \nmid \#\langle P_2\rangle$. It follows that $x_1y_2^2 - y_1^2x_2 \neq 0$.*

Proof. Assume that $P_1 \neq P_2$, $3 \nmid \#\langle P_1\rangle$, and $3 \nmid \#\langle P_2\rangle$. Suppose that $x_1 = 0$ and $x_2 = 0$. Then, either $P_1 = P_2 = (0, -1)$ or $P_2 = (0, -\omega)$ or $P_2 = (0, -1/\omega)$, all are contradictions. So, either $x_1 \neq 0$ or $x_2 \neq 0$. Suppose that $y_1y_2 = 0$. Then, either P_1 or P_2 is of order 3, contradiction. So, $y_1y_2 \neq 0$. Therefore, either $x_1y_1 \neq 0$ or $x_2y_2 \neq 0$. The claim then follows from Lemma 4.3.1 and Lemma 4.3.2 (by swapping P_1 and P_2 when necessary). □

Unified addition Alternative addition formulae on $E_{\mathbf{H},a,d}$ are given by $(x_1, y_1) + (x_2, y_2) = (x_3, y_3)$ where

$$x_3 = (x_1 - y_1^2x_2y_2)/(ax_1y_1x_2^2 - y_2), \tag{4.31}$$

$$y_3 = (y_1y_2^2 - ax_1^2x_2)/(ax_1y_1x_2^2 - y_2). \tag{4.32}$$

assuming that $ax_1y_1x_2^2 - y_2 \neq 0$. These formulae work for identical summands in most of the cases hence the name *unified*.

If $(x_1, y_1)+(x_2, y_2)$ is a point at infinity then $ax_1y_1x_2^2-y_2 = 0$. Otherwise, $(x_1, y_1)+(x_2, y_2)$ would be an affine point since it can be shown using the relations $ax_1^3 + y_1^3 + 1 = dx_1y_1$ and $ax_2^3 + y_2^3 + 1 = dx_2y_2$ that the algebraic expressions for (x_3, y_3) satisfy $ax_3^3 + y_3^3 + 1 = dx_3y_3$. The converse, however, does not necessarily apply. This means that if $ax_1y_1x_2^2 - y_2 = 0$ then $(x_1, y_1) + (x_2, y_2)$ may not be a point at infinity. Therefore it is worth investigating the exceptional cases. The denominators of (4.31) and (4.32) vanish for some summands which are described in the following lemma explicitly.

Lemma 4.3.4. *Let $a, d \in \mathbb{K}$ with $a(27a - d^3) \neq 0$. Fix $\omega \in \mathbb{K}\backslash\{1\}$ so that $\omega^3 = 1$. Fix $\alpha \in \mathbb{K}$ so that $\alpha^3 = a$. Fix $x_1, y_1 \in \mathbb{K}\backslash\{0\}$ such that $ax_1^3 + y_1^3 + 1 = dx_1y_1$. Let $x_2, y_2 \in \mathbb{K}$ such that $ax_2^3 + y_2^3 + 1 = dx_2y_2$. Now, $ax_1y_1x_2^2 - y_2 = 0$ if and only if $(x_2, y_2) \in S'$ where*

$$S' = \big[(\frac{1}{\alpha y_1}, \frac{\alpha x_1}{y_1}), (\frac{1}{\omega\alpha y_1}, \frac{\omega\alpha x_1}{y_1}), (\frac{\omega}{\alpha y_1}, \frac{\alpha x_1}{\omega y_1}), (\frac{\alpha}{ax_1}, \frac{y_1}{\alpha x_1}), (\frac{\omega\alpha}{ax_1}, \frac{y_1}{\omega\alpha x_1}), (\frac{\alpha}{a\omega x_1}, \frac{\omega y_1}{\alpha x_1})\big].$$

Proof. $\Rightarrow$: Assume that $ax_1y_1x_2^2 - y_2 = 0$. Solving the system of equations $ax_1y_1x_2^2 - y_2 = 0$, $ax_2^3 + y_2^3 + 1 = dx_2y_2$ for x_2 and y_2 gives S'. Note that the last three entries are equal to $-(\frac{1}{\alpha y_1}, \frac{\alpha x_1}{y_1})$, $-(\frac{1}{\omega\alpha y_1}, \frac{\omega\alpha x_1}{y_1})$, $-(\frac{\omega}{\alpha y_1}, \frac{\alpha x_1}{\omega y_1})$, respectively. Then, all entries in S' are defined since $x_1y_1 \neq 0$. $\Leftarrow$: The claim follows trivially by substitution. □

This lemma and Lemma 4.3.1 excludes $x_1y_1 = 0$. The following lemma states the exceptional cases for the excluded situation.

Lemma 4.3.5. *In Lemma 4.3.4 assume that $x_1y_1 = 0$. Then $ax_1y_1x_2^2 - y_2 = 0$ if and only if $(x_2, y_2) \in [(\frac{-1}{\alpha}, 0), (\frac{-1}{\omega\alpha}, 0), (\frac{-\omega}{\alpha}, 0)]$. Each of these points is of order 3.*

Proof. Trivial. □

The following lemma shows that if one of the summands is selected suitably then in the presence of a vanished denominator, the other summand is not in the subgroup generated by the original summand.

Lemma 4.3.6. *Let a, d, x_1, y_1, x_2, y_2 be defined as in Lemma 4.3.4. Assume that $P_1 = (x_1, y_1)$ is a fixed point of odd order such that $3 \nmid \#\langle P_1\rangle$. Assume that $P_2 = (x_2, y_2) \in S'$. Then, $P_2 \notin \langle P_1\rangle$.*

Proof. See the proof of Lemma 4.3.2. The only difference is P_2 is selected from S'. □

In the following lemma, it is shown that exceptions can be prevented regardless of any assumption on the curve constants.

Lemma 4.3.7. *Let $E_{\mathbf{H},a,d}$ be a twisted Hessian curve defined over $\mathbb{K}$. Let $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ be points on $E_{\mathbf{H},a,d}$. Assume that P_1 and P_2 are of odd order such that $3 \nmid \#\langle P_1\rangle$ and $3 \nmid \#\langle P_2\rangle$. It follows that $ax_1y_1x_2^2 - y_2 \neq 0$.*

Proof. Assume that P_1 and P_2 are of odd order such that $3 \nmid \#\langle P_1\rangle$ and $3 \nmid \#\langle P_2\rangle$. If $x_1y_1 = 0$ then from Lemma 4.3.5 it follows that $ax_1y_1x_2^2 - y_2 \neq 0$. If $x_1y_1 \neq 0$ then the claim follows from Lemma 4.3.4 and Lemma 4.3.6 (by swapping P_1 and P_2 when necessary). □

Exception handling in the general case Algorithm 4.3.1 provides a complete addition on all twisted Hessian curves. The correctness of the algorithm follows from two observations. Firstly, when a point at infinity is involved as the sum or as one of the summands along the lines 2 to 25, it is tedious but straightforward to check that the output of the algorithm is correct using the *implicit* technique mentioned at the start of the chapter. Line 1 conditionally swaps the inputs to eliminate half of the input-wise symmetric branches. The second observation is that *glueing* together the unified addition and the dedicated addition formulae is enough to handle all exceptions when both of the summands and the sum are affine points. Also notice that the exceptions of Lemma 4.3.5 are handled properly. This fact follows from Lemma 4.3.1 and Lemma 4.3.4 by observing that $\#(S' \cap S) = 3$. This means that if $(x_2, y_2) \in S' \cap S$ then the output must be a point at infinity (lines 35 and 37) since there are exactly three points at infinity. The remaining exceptional cases which occur at $(x_2, y_2) \in S' \backslash (S' \cap S)$ are

Algorithm 4.3.1: Addition law in affine coordinates for twisted Hessian form

input : $P_1, P_2, \Omega_1, \Omega_2, \Omega_3 \in E_{\mathbf{H},a,d}(\mathbb{K})$ and fixed $\alpha, \omega \in \mathbb{K}$ such that $\alpha^3 = a$ and $\omega^3 = 1$ with $\omega \neq 1$.
output : $P_1 + P_2$.

```
1  if P1 ∈ {Ω1, Ω2, Ω3} then Pt ← P1, P1 ← P2, P2 ← Pt.
2  if P1 = Ω1 then
3      if P2 = Ω1 then return (−1/α, 0).
4      else if P2 = Ω2 then return (−1/(ωα), 0).
5      else if P2 = Ω3 then return (−ω/α, 0).
6      else if P2 = (0, −1) then return Ω1.
7      else if P2 = (0, −ω) then return Ω2.
8      else if P2 = (0, −1/ω) then return Ω3.
9      else return (αy2/(ax2), 1/(αx2)).
10 else if P1 = Ω2 then
11     if P2 = Ω1 then return (−1/(ωα), 0).
12     else if P2 = Ω2 then return (−ω/α, 0).
13     else if P2 = Ω3 then return (−1/α, 0).
14     else if P2 = (0, −1/ω) then return Ω1.
15     else if P2 = (0, −1) then return Ω2.
16     else if P2 = (0, −ω) then return Ω3.
17     else return (αy2/(ωax2), ω/(αx2)).
18 else if P1 = Ω3 then
19     if P2 = Ω1 then return (−ω/α, 0).
20     else if P2 = Ω2 then return (−1/α, 0).
21     else if P2 = Ω3 then return (−1/(ωα), 0).
22     else if P2 = (0, −ω) then return Ω1.
23     else if P2 = (0, −1/ω) then return Ω2.
24     else if P2 = (0, −1) then return Ω3.
25     else return (ωαy2/(ax2), 1/(ωαx2)).
26 else if x1y2^2 − y1^2x2 ≠ 0 then
27     x3 ← (y1x2^2 − x1^2y2)/(x1y2^2 − y1^2x2).
28     y3 ← (x1y1 − x2y2)/(x1y2^2 − y1^2x2).
29     return (x3, y3).
30 else if ax1y1x2^2 − y2 ≠ 0 then
31     x3 ← (x1 − y1^2x2y2)/(ax1y1x2^2 − y2).
32     y3 ← (y1y2^2 − ax1^2x2)/(ax1y1x2^2 − y2).
33     return (x3, y3).
34 else
35     if P2 = (α/(ax1), y1/(αx1)) then return Ω1.
36     else if P2 = (α/(ωax1), ωy1/(αx1)) then return Ω2.
37     else return Ω3.
38 end
```

handled by the dedicated addition formulae (lines 27 and 28). Similarly the exceptions at $(x_2, y_2) \in S \backslash (S' \cap S)$ are handled by the unified addition formulae (lines 31 and 32).

The points at infinity on the projective closure of $E_{\mathbf{H},a,d}$ are not defined over $\mathbb{K}$ if a is a non-cube in $\mathbb{K}$. Having noted this, it was pointed out in [Ber06a] that (4.31) and (4.32) are complete. Algorithm 4.3.1 complies with the completeness criterion since only the lines 31 to 32 are necessary in this case because these lines are input-wise symmetric versions of (4.31) and (4.32). Note that the assumption on the curve constant a limits the number of curves in twisted Hessian form for which the unified addition formulae are complete.

Algorithm 4.3.1 also complies with Lemma 4.3.7. If P_1 and P_2 are points of odd order then only the lines 19 to 21 are necessary. This technique applies to all twisted Hessian curves.

Algorithm 4.3.1 also complies with Lemma 4.3.3. If P_1 and P_2 are distinct points of odd order then only the lines 15 to 17 are necessary. This technique applies to all twisted Hessian

curves. The doubling formulae (4.25) and (4.26) are enough to handle the special case $P_1 = P_2$.

More formulae Using the tools of Chapter 3, it is possible to derive other low-degree addition formulae;

$$x_3 = \frac{e(x_1 - y_1^2x_2y_2) + f(x_1y_1y_2^2 - x_2) + g(y_1x_2^2 - x_1^2y_2)}{e(ax_1y_1x_2^2 - y_2) + f(y_1 - ax_1^2x_2y_2) + g(x_1y_2^2 - y_1^2x_2)}, \tag{4.33}$$

$$y_3 = \frac{h(y_1y_2^2 - ax_1^2x_2) + j(ax_1x_2^2 - y_1^2y_2) + k(x_1y_1 - x_2y_2)}{h(ax_1y_1x_2^2 - y_2) + j(y_1 - ax_1^2x_2y_2) + k(x_1y_2^2 - y_1^2x_2)} \tag{4.34}$$

assuming that $e(ax_1y_1x_2^2 - y_2) + f(y_1 - ax_1^2x_2y_2) + g(x_1y_2^2 - y_1^2x_2) \neq 0$ and $h(ax_1y_1x_2^2 - y_2) + j(y_1 - ax_1^2x_2y_2) + k(x_1y_2^2 - y_1^2x_2) \neq 0$ where $e, f, g, h, j, k \in \mathbb{K}$ such that at most one of e, f, g is zero and at most one of h, j, k is zero. These formulae tend to include more subexpressions which are not commonly shared. Therefore, their evaluation turns out to be more costly and thus not considered hereafter.

Literature notes Other results related to the affine formulae for Hessian form $x^3 + y^3 + 1 = dxy$ can be found in the literature. Those formulae typically use a point at infinity as the identity element. Moving the identity to $(0, -1)$ yields similar formulae presented in this section. Some pointers are [CC86], [Sma01], [JQ01]. Generalization from Hessian curves to twisted Hessian curves is due to Bernstein, Kohel, and Lange [BL07a].

4.4 Twisted Edwards form

This section presents the group law on $E_{\mathbf{E},a,d}$ in affine coordinates. It also investigates the exceptional summands for each set of formulae and provides a complete addition algorithm for all twisted Edwards curves by properly handling an entire set of divide-by-zero exceptions. In addition, practical ways of preventing these exceptions are explained.

Throughout this section, let $\mathbb{K}$ be a field of odd characteristic. Recall from Chapter 2 that a twisted Edwards curve is defined by

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

where $a, d \in \mathbb{K}$ with $ad(a - d) \neq 0$. Assume that both a and d are squares in $\mathbb{K}$. Recall from Chapter 2 that the set of $\mathbb{K}$-rational points on the desingularization of $E_{\mathbf{E},a,d}$ is defined by

$$E_{\mathbf{E},a,d}(\mathbb{K}) = \{(x, y) \in \mathbb{K}^2 \mid ax^2 + y^2 = 1 + dx^2y^2\} \cup \{\Omega_1, \Omega_2, \Omega_3, \Omega_4\}$$

where $\Omega_1, \Omega_2, \Omega_3, \Omega_4$ are points at infinity.

Identity element and negation The identity element can suitably be taken as $(0, 1)$. Let (x_1, y_1) be a point on $E_{\mathbf{E},a,d}$. The negative of (x_1, y_1) is $(-x_1, y_1)$.

Doubling The doubling formulae on $E_{\mathbf{E},a,d}$ are given by $[2](x_1, y_1) = (x_3, y_3)$ where

$$x_3 = 2x_1y_1/(y_1^2 + ax_1^2), \tag{4.35}$$
$$y_3 = (y_1^2 - ax_1^2)/(2 - y_1^2 - ax_1^2) \tag{4.36}$$

assuming that $(2 - y_1^2 - ax_1^2)(y_1^2 + ax_1^2) \neq 0$, see [BBJ+08] (also see [BL07b], [BBLP07], [BBLP08]). These formulae do not depend on the curve constant d and are of minimal total degree. By the curve equation the denominator $y_1^2 + ax_1^2$ is equivalent to $1 + dx_1^2y_1^2$. Similarly, the denominator $2 - y_1^2 - ax_1^2$ is equivalent to $1 - dx_1^2y_1^2$. These denominators can also be used if the total degree is not of concern.

The identity element is the point $(0, 1)$ which can be determined by solving $ax_1^2 + y_1^2 = 1 + dx_1^2y_1^2$ and $(x_3, y_3) = (x_1, y_1)$ for x_1 and y_1 where x_3 and y_3 are given by (4.35) and (4.36).

The point $(0, -1)$ is of order 2 which can be determined by solving $ax_1^2 + y_1^2 = 1 + dx_1^2y_1^2$ and $(x_3, y_3) = (0, 1)$ for x_1 and y_1 where x_3 and y_3 are given by (4.35) and (4.36). There are three points of order 2 in total (over a sufficiently large finite extension of $\mathbb{K}$). Therefore, two of the points at infinity have to be of order 2. Ω_1 and Ω_2 are taken to be of order 2 hereafter.

The two points of the form $(x, 0)$ are of order 4 which can be determined by solving $ax_1^2 + y_1^2 = 1 + dx_1^2y_1^2$ and $(x_3, y_3) = (0, -1)$ for x_1 and y_1 where x_3 and y_3 are given by (4.35) and (4.36). There are twelve points of order 4 in total (over a sufficiently large finite extension of $\mathbb{K}$). Eight of these points can be explicitly determined to be affine points by solving $ax_1^2 + y_1^2 = 1 + dx_1^2y_1^2$ and $(2 - y_1^2 - ax_1^2)(y_1^2 + ax_1^2) = 0$ for x_1 and y_1. Therefore, the remaining two points of order 4 have to be the points at infinity Ω_3 and Ω_4. The doubles of the eight points are either Ω_1 or Ω_2. The doubles of Ω_3 and Ω_4 are $(0, -1)$. These points are the only exceptions of (4.35) and (4.36). The following remark is immediate.

Remark 4.4.1. $[2](x_1, y_1)$ is a point at infinity if and only if $(2 - y_1^2 - ax_1^2)(y_1^2 + ax_1^2) = 0$.

Remark 4.4.1 does not extend to the case of generic additions. However, it is still useful in proving some lemmas regarding the generic addition formulae which will be presented next.

Dedicated addition Further let (x_2, y_2) be a point on $E_{\mathbf{E},a,d}$. The addition formulae on $E_{\mathbf{E},a,d}$ are given by $(x_1, y_1) + (x_2, y_2) = (x_3, y_3)$ where

$$x_3 = (x_1y_1 + x_2y_2)/(y_1y_2 + ax_1x_2), \tag{4.37}$$
$$y_3 = (x_1y_1 - x_2y_2)/(x_1y_2 - y_1x_2) \tag{4.38}$$

assuming that $(y_1y_2 + ax_1x_2)(x_1y_2 - y_1x_2) \neq 0$. These formulae are of minimal total degree. These formulae do not work for identical summands hence the name *dedicated*.

If $(x_1, y_1) + (x_2, y_2)$ is a point at infinity then $(y_1y_2 + ax_1x_2)(x_1y_2 - y_1x_2) = 0$. Otherwise, $(x_1, y_1) + (x_2, y_2)$ would be an affine point since it can be shown using the relations $ax_1^2 + y_1^2 = 1 + dx_1^2y_1^2$ and $ax_2^2 + y_2^2 = 1 + dx_2^2y_2^2$ that the algebraic expressions for (x_3, y_3) satisfy $ax_2^2 + y_2^2 = 1 + dx_2^2y_2^2$. The converse, however, does not necessarily apply. This means that if $(y_1y_2 + ax_1x_2)(x_1y_2 - y_1x_2) = 0$ then $(x_1, y_1) + (x_2, y_2)$ may not be a point at infinity. Therefore it is worth investigating the exceptional cases. The denominators of (4.37) and (4.38) vanish for some summands which are described in the following lemma explicitly.

Lemma 4.4.2. *Let $a, d \in \mathbb{K}$ with $ad(a-d) \neq 0$. Fix $\alpha, \delta \in \mathbb{K}$ so that $\alpha^2 = a$ and $\delta^2 = d$. Fix $x_1, y_1 \in \mathbb{K}\backslash\{0\}$ such that $ax_1^2 + y_1^2 = 1 + dx_1^2y_1^2$. Let $x_2, y_2 \in \mathbb{K}$ such that $ax_2^2 + y_2^2 = 1 + dx_2^2y_2^2$. Now, $(y_1y_2 + ax_1x_2)(x_1y_2 - y_1x_2) = 0$ if and only if $(x_2, y_2) \in S$ where $S =$*

$$[(x_1, y_1), (-x_1, -y_1), (\frac{y_1}{\alpha}, -x_1\alpha), (\frac{-y_1}{\alpha}, x_1\alpha), (\frac{1}{\delta y_1}, \frac{1}{\delta x_1}), (\frac{-1}{\delta y_1}, \frac{-1}{\delta x_1})(\frac{1}{\alpha\delta x_1}, \frac{-\alpha}{\delta y_1}), (\frac{-1}{\alpha\delta x_1}, \frac{\alpha}{\delta y_1})].$$

Proof. $\Rightarrow$: Assume that $(y_1y_2 + ax_1x_2)(x_1y_2 - y_1x_2) = 0$. Solving the equations $(y_1y_2 + ax_1x_2)(x_1y_2 - y_1x_2) = 0$ and $ax_2^2 + y_2^2 = 1 + dx_2^2y_2^2$ simultaneously for x_2 and y_2 gives S. All entries in S are defined since $x_1y_1 \neq 0$. $\Leftarrow$: The claims follow trivially by substitution. □

The following lemma shows that if one of the summands is of odd order then in the presence of an exception, the other summand is always of even order.

Lemma 4.4.3. *Let a, d, x_1, y_1, x_2, y_2 be defined as in Lemma 4.4.2. Assume that $P_1 = (x_1, y_1)$ is a fixed point of odd order. Assume that $P_2 \in S\backslash\{P_1\}$. Then P_2 is of even order.*

Proof. In [BL07b] (where $a = 1$) and later in [BBJ+08], it is proven that the points at infinity (over the extension of $\mathbb{K}$ where they exist) are of even order. Assume that $P_1 = (x_1, y_1)$ is a fixed point of odd order hence not a point at infinity. Suppose that P_2 is of odd order hence not a point at infinity. It follows that $P_1 \pm P_2$, $M = 2P_1$, and $N = 2P_2$ are all of odd order hence not points at infinity.

Assume that $P_2 \in S\backslash\{P_1\}$. So, $P_1 \neq P_2$. Plus, $(y_1y_2 + ax_1x_2)(x_1y_2 - y_1x_2) = 0$ by Lemma 4.4.2. It follows that $P_1 \neq -P_2$, for otherwise, $(y_1y_2+ax_1x_2)(x_1y_2-y_1x_2) = 2x_1y_1(y_1^2-ax_1^2) = 0$ which means that $y_1^2 - ax_1^2 = 0$ since $x_1, y_1 \neq 0$. Using this relation, the doubling formulae simplifies to $(x_3, y_3) = (x_1/y_1, 0/(2-2y_1^2))$. The output x_3 is defined since $x_1y_1 \neq 0$. Whenever $0/(2-2y_1^2)$ is defined, it produces a point of order 4. But then P_1 would be of even order (in particular of order 8). If $y_1 = \pm 1$ then $0/(2-2y_1^2)$ is not defined. However these cases can be omitted since $x_1 \neq 0$ and the only points with $y_1 = \pm 1$ requires x_1 to be zero.

Now,

- In the case $y_1y_2 + ax_1x_2 = 0$, $x_2 = -y_1y_2/(ax_1)$ is defined since $x_1 \neq 0$ by definition. Using this relation together with (4.35) and the curve equation gives

$$x(N) = \frac{2x_2y_2}{1+dx_2^2y_2^2} = \frac{2x_2y_2}{y_1^2+ax_1^2} = \frac{2\frac{-y_1y_2}{ax_1}y_2}{y_2^2+a(\frac{-y_1y_2}{ax_1})^2} = -\frac{2x_1y_1}{y_1^2+ax_1^2} = -\frac{2x_1y_1}{1+dx_1^2y_1^2} = -x(M).$$

- In the case $x_1y_2 - y_1x_2 = 0$, $y_2 = y_1x_2/x_1$ is defined since $x_1 \neq 0$ by definition. Using this relation together with (4.35) and the curve equation gives

$$x(N) = \frac{2x_2y_2}{1+dx_2^2y_2^2} = \frac{2x_2y_2}{y_2^2+ax_2^2} = \frac{2x_2\frac{y_1x_2}{x_1}}{(\frac{y_1x_2}{x_1})^2+ax_2^2} = \frac{2x_1y_1}{y_1^2+ax_1^2} = \frac{2x_1y_1}{1+dx_1^2y_1^2} = x(M).$$

By the curve definition, $y(M) = \pm y(N)$ since $|x(M)| = |x(N)|$. Now,

- $x(M) = x(N)$ and $y(M) = y(N)$: $M - N = (0, 1)$. So, $M - N = 2P_1 - 2P_2 = 2(P_1 - P_2) = (0, 1)$.
- $x(M) = x(N)$ and $y(M) = -y(N)$: $M+N = (0, -1)$. So, $2(M+N) = 2(2P_1+2P_2) = 4(P_1+P_2) = (0, 1)$.
- $x(M) = -x(N)$ and $y(M) = y(N)$: $M + N = (0, 1)$. So, $M + N = 2P_1 + 2P_2 = 2(P_1 + P_2) = (0, 1)$.
- $x(M) = -x(N)$ and $y(M) = -y(N)$: $M - N = (0, -1)$. So, $2(M - N) = 2(2P_1 - 2P_2) = 4(P_1 - P_2) = (0, 1)$.

Since $P_1 \neq \pm P_2$, in all cases $P_1 \pm P_2$ is of even order, contradiction. In conclusion, $P_2 \in S \backslash \{P_1\}$ is of even order provided that P_1 is of odd order. □

A practical solution is now provided to prevent the exceptional cases of (4.37) and (4.38).

Lemma 4.4.4. *Let $\mathbb{K}$ be a field of odd characteristic. Let $E_{\mathbf{E},a,d}$ be a twisted Edwards curve defined over $\mathbb{K}$. Let $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ be points on $E_{\mathbf{E},a,d}$. Assume that P_1 and P_2 are of odd order with $P_1 \neq P_2$. It follows that $y_1 y_2 + a x_1 x_2 \neq 0$ and $x_1 y_2 - y_1 x_2 \neq 0$.*

Proof. Assume that P_1 and P_2 are of odd order with $P_1 \neq P_2$. Suppose that $x_1 = 0$ and $x_2 = 0$. Then, either $P_1 = P_2 = (0, 1)$ or $P_2 = (0, -1)$, both are contradictions. So, either $x_1 \neq 0$ or $x_2 \neq 0$. Suppose that $y_1 y_2 = 0$. Then, either P_1 or P_2 is of even order, contradiction. So, $y_1 y_2 \neq 0$. Therefore, either $x_1 y_1 \neq 0$ or $x_2 y_2 \neq 0$. The claim then follows from Lemma 4.4.2 and Lemma 4.4.3 (by swapping P_1 and P_2 when necessary). □

Unified addition Alternative addition formulae on $E_{\mathbf{E},a,d}$ are given by $(x_1, y_1) + (x_2, y_2) = (x_3, y_3)$ where

$$x_3 = (x_1 y_2 + y_1 x_2)/(1 + d x_1 y_1 x_2 y_2), \tag{4.39}$$

$$y_3 = (y_1 y_2 - a x_1 x_2)/(1 - d x_1 y_1 x_2 y_2) \tag{4.40}$$

assuming $(1 - d x_1 y_1 x_2 y_2)(1 + d x_1 y_1 x_2 y_2) \neq 0$, see [BBJ+08]. These formulae work for identical summands in most of the cases hence the name *unified*.

If $(x_1, y_1) + (x_2, y_2)$ is a point at infinity then $(1 - d x_1 y_1 x_2 y_2)(1 + d x_1 y_1 x_2 y_2) = 0$. Otherwise, $(x_1, y_1) + (x_2, y_2)$ would be an affine point by Theorem 3.1 of [BL07b] and by the remark in [BBJ+08, §6] stating that $E_{\mathbf{E},a,d}$ is isomorphic to $E_{\mathbf{E},1,d/a}$. The converse, however, does not necessarily apply. This means that if $(1 - d x_1 y_1 x_2 y_2)(1 + d x_1 y_1 x_2 y_2) = 0$ then $(x_1, y_1) + (x_2, y_2)$ may not be a point at infinity. Therefore it is worth investigating the exceptional cases. The denominators of (4.39) and (4.40) vanish for some summands which are described in the following lemma explicitly.

Lemma 4.4.5. *Let $a, d \in \mathbb{K}$ with $ad(a - d) \neq 0$. Fix $\alpha, \delta \in \mathbb{K}$ so that $\alpha^2 = a$ and $\delta^2 = d$. Fix $x_1, y_1 \in \mathbb{K} \backslash \{0\}$ such that $a x_1^2 + y_1^2 = 1 + d x_1^2 y_1^2$. Let $x_2, y_2 \in \mathbb{K}$ such that $a x_2^2 + y_2^2 = 1 + d x_2^2 y_2^2$. It follows that $d x_1 y_1 x_2 y_2 \in \{1, -1\}$ if and only if $(x_2, y_2) \in S'$ where $S' =$*

$$\left[\left(\frac{1}{\delta y_1}, \frac{-1}{\delta x_1}\right), \left(\frac{-1}{\delta y_1}, \frac{1}{\delta x_1}\right), \left(\frac{1}{\alpha\delta x_1}, \frac{\alpha}{\delta y_1}\right), \left(\frac{-1}{\alpha\delta x_1}, \frac{-\alpha}{\delta y_1}\right), \left(\frac{1}{\delta y_1}, \frac{1}{\delta x_1}\right), \left(\frac{-1}{\delta y_1}, \frac{-1}{\delta x_1}\right), \left(\frac{1}{\alpha\delta x_1}, \frac{-\alpha}{\delta y_1}\right), \left(\frac{-1}{\alpha\delta x_1}, \frac{\alpha}{\delta y_1}\right)\right].$$

Proof. $\Rightarrow$: Assume that $(1 - d x_1 y_1 x_2 y_2)(1 + d x_1 y_1 x_2 y_2) = 0$. Solving the equations $(1 - d x_1 y_1 x_2 y_2)(1 + d x_1 y_1 x_2 y_2) = 0$ and $a x_2^2 + y_2^2 = 1 + d x_2^2 y_2^2$ simultaneously for x_2 and y_2 gives S'. All entries in S' are defined since $x_1 y_1 \neq 0$. $\Leftarrow$: The claims follow trivially by substitution. □

This lemma and Lemma 4.4.2 excludes $x_1 y_1 = 0$. If so, $1 \pm d x_1 y_1 x_2 y_2 \neq 0$.

The following lemma shows that if one of the summands is of odd order then in the presence of a vanished denominator, the other summand is always of even order.

Lemma 4.4.6. *Let a, d, x_1, y_1, x_2, y_2 be defined as in Lemma 4.4.5. Assume that $P_1 = (x_1, y_1)$ is a fixed point of odd order. Assume that $P_2 = (x_2, y_2) \in S'$. Then, P_2 is of even order.*

Proof. In [BL07b] (where $a = 1$) and later in [BBJ+08], it is proven that the points at infinity (over the extension of $\mathbb{K}$ where they exist) are of even order. Assume that $P_1 = (x_1, y_1)$ is a fixed point of odd order hence not a point at infinity. Suppose that P_2 is of odd order hence not a point at infinity. It follows that $P_1 \pm P_2$, $M = 2P_1$, and $N = 2P_2$ are all of odd order hence not points at infinity.

Assume that $P_2 \in S'$. Then, $dx_1y_1x_2y_2 \in \{1, -1\}$ by Lemma 4.4.5 and it follows that $P_1 \neq \pm P_2$, for otherwise, $dx_1^2y_1^2 \in \{1, -1\}$ and P_1 would be of even order, see Remark 4.4.1.

Note that $x_1, y_1 \neq 0$ since $dx_1y_1x_2y_2 \in \{1, -1\}$ (also true by definition). So, $x_2y_2 = \pm 1/(dx_1y_1)$ are defined taking the signs independently. Using this relation together with (4.39) gives

$$x(N) = \frac{2x_2y_2}{1 + dx_2^2y_2^2} = \frac{2\frac{\pm 1}{dx_1y_1}}{1 + d(\frac{\pm 1}{dx_1y_1})^2} = \pm\frac{2x_1y_1}{1 + dx_1^2y_1^2} = \pm x(M).$$

By the curve definition, $y(M) = \pm y(N)$ since $|x(M)| = |x(N)|$. Now,

- $x(M) = x(N)$ and $y(M) = y(N)$: $M - N = (0, 1)$. So, $M - N = 2P_1 - 2P_2 = 2(P_1 - P_2) = (0, 1)$.
- $x(M) = x(N)$ and $y(M) = -y(N)$: $M+N = (0, -1)$. So, $2(M+N) = 2(2P_1+2P_2) = 4(P_1+P_2) = (0, 1)$.
- $x(M) = -x(N)$ and $y(M) = y(N)$: $M + N = (0, 1)$. So, $M + N = 2P_1 + 2P_2 = 2(P_1 + P_2) = (0, 1)$.
- $x(M) = -x(N)$ and $y(M) = -y(N)$: $M - N = (0, -1)$. So, $2(M - N) = 2(2P_1 - 2P_2) = 4(P_1 - P_2) = (0, 1)$.

Since $P_1 \neq \pm P_2$, in all cases $P_1 \pm P_2$ is of even order, contradiction. In conclusion, $P_2 \in S'$ is of even order provided that P_1 is of odd order. □

In the following lemma, with reasonable assumptions, it is shown that exceptions can be prevented regardless of any assumption on the curve constants.

Lemma 4.4.7. *Let $\mathbb{K}$ be a field of odd characteristic. Let $E_{\mathbf{E},a,d}$ be a twisted Edwards curve defined over $\mathbb{K}$. Let $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ be points on $E_{\mathbf{E},a,d}$. Assume that P_1 and P_2 are of odd order. It follows that $1 \pm dx_1x_2y_1y_2 \neq 0$.*

Proof. Assume that P_1 and P_2 are of odd order. If $x_1y_1x_2y_2 = 0$ then $1 \pm dx_1y_1x_2y_2 \neq 0$ as desired. If $x_1y_1x_2y_2 \neq 0$ then the claim follows from Lemma 4.4.5 and Lemma 4.4.6 (by swapping P_1 and P_2 when necessary). □

Exception handling in the general case Algorithm 4.4.1 provides a complete addition on all twisted Edwards curves. The correctness of the algorithm follows from two observations. Firstly, when a point at infinity is involved as the sum or as one of the summands along the lines 2 to 41, it is tedious but straightforward to check that the output of the algorithm is correct using the *implicit* technique mentioned at the start of the chapter. Line 1 conditionally swaps the inputs to eliminate half of the input-wise symmetric branches. The second observation is that *glueing* together the unified addition and the dedicated addition formulae is enough to handle all exceptions when both of the summands and the sum are affine points. This fact follows from Lemma 4.4.5 and Lemma 4.4.2 by observing that $\#(S' \cap S) = 4$. This means that if $(x_2, y_2) \in S' \cap S$ then the output must be a point at infinity (lines 51 to 54) since there are exactly four points at infinity. The remaining exceptional cases which occur at $(x_2, y_2) \in S' \backslash (S' \cap S)$ are handled by the dedicated addition formulae (lines 43 and 44).

Algorithm 4.4.1: Addition law in affine coordinates for twisted Edwards form

input : $P_1, P_2, \Omega_1, \Omega_2, \Omega_3, \Omega_4 \in E_{\mathbf{E},a,d}(\mathbb{K})$ and fixed $\alpha, \delta \in \mathbb{K}$ such that $\alpha^2 = a$ and $\delta^2 = d$.
output : $P_1 + P_2$.

if $P_1 \in \{\Omega_1, \Omega_2, \Omega_3, \Omega_4\}$ **then** $P_t \leftarrow P_1$, $P_1 \leftarrow P_2$, $P_2 \leftarrow P_t$.
if $P_2 = \Omega_1$ **then**
  if $P_1 = \Omega_1$ **then return** $(0, 1)$.
  else if $P_1 = \Omega_2$ **then return** $(0, -1)$.
  else if $P_1 = \Omega_3$ **then return** $(-1/\alpha, 0)$.
  else if $P_1 = \Omega_4$ **then return** $(1/\alpha, 0)$.
  else if $P_1 = (0, 1)$ **then return** Ω_1.
  else if $P_1 = (0, -1)$ **then return** Ω_2.
  else if $P_1 = (-1/\alpha, 0)$ **then return** Ω_3.
  else if $P_1 = (1/\alpha, 0)$ **then return** Ω_4.
  else return $(-1/(\alpha\delta x_1), -\alpha/(\delta y_1))$.
else if $P_2 = \Omega_2$ **then**
  if $P_1 = \Omega_1$ **then return** $(0, -1)$.
  else if $P_1 = \Omega_2$ **then return** $(0, 1)$.
  else if $P_1 = \Omega_3$ **then return** $(1/\alpha, 0)$.
  else if $P_1 = \Omega_4$ **then return** $(-1/\alpha, 0)$.
  else if $P_1 = (0, -1)$ **then return** Ω_1.
  else if $P_1 = (0, 1)$ **then return** Ω_2.
  else if $P_1 = (1/\alpha, 0)$ **then return** Ω_3.
  else if $P_1 = (-1/\alpha, 0)$ **then return** Ω_4.
  else return $(1/(\alpha\delta x_1), \alpha/(\delta y_1))$.
else if $P_2 = \Omega_3$ **then**
  if $P_1 = \Omega_1$ **then return** $(-1/\alpha, 0)$.
  else if $P_1 = \Omega_2$ **then return** $(1/\alpha, 0)$.
  else if $P_1 = \Omega_3$ **then return** $(0, -1)$.
  else if $P_1 = \Omega_4$ **then return** $(0, 1)$.
  else if $P_1 = (1/\alpha, 0)$ **then return** Ω_1.
  else if $P_1 = (-1/\alpha, 0)$ **then return** Ω_2.
  else if $P_1 = (0, 1)$ **then return** Ω_3.
  else if $P_1 = (0, -1)$ **then return** Ω_4.
  else return $(1/(\delta y_1), -1/(\delta x_1))$.
else if $P_2 = \Omega_4$ **then**
  if $P_1 = \Omega_1$ **then return** $(1/\alpha, 0)$.
  else if $P_1 = \Omega_2$ **then return** $(-1/\alpha, 0)$.
  else if $P_1 = \Omega_3$ **then return** $(0, 1)$.
  else if $P_1 = \Omega_4$ **then return** $(0, -1)$.
  else if $P_1 = (-1/\alpha, 0)$ **then return** Ω_1.
  else if $P_1 = (1/\alpha, 0)$ **then return** Ω_2.
  else if $P_1 = (0, -1)$ **then return** Ω_3.
  else if $P_1 = (0, 1)$ **then return** Ω_4.
  else return $(-1/(\delta y_1), 1/(\delta x_1))$.
else if $(y_1y_2 + ax_1x_2)(x_1y_2 - y_1x_2) \neq 0$ **then**
  $x_3 \leftarrow (x_1y_1 + x_2y_2)/(y_1y_2 + ax_1x_2)$.
  $y_3 \leftarrow (x_1y_1 - x_2y_2)/(x_1y_2 - y_1x_2)$.
  return (x_3, y_3).
else if $(1 - dx_1x_2y_1y_2)(1 + dx_1x_2y_1y_2) \neq 0$ **then**
  $x_3 \leftarrow (x_1y_2 + y_1x_2)/(1 + dx_1x_2y_1y_2)$.
  $y_3 \leftarrow (y_1y_2 - ax_1x_2)/(1 - dx_1x_2y_1y_2)$.
  return (x_3, y_3).
else
  if $P_2 = (1/(\alpha\delta x_1), -\alpha/(\delta y_1))$ **then return** Ω_1.
  else if $P_2 = (-1/(\alpha\delta x_1), \alpha/(\delta y_1))$ **then return** Ω_2.
  else if $P_2 = (1/(\delta y_1), 1/(\delta x_1))$ **then return** Ω_3.
  else return Ω_4.
end

Similarly the exceptions at $(x_2, y_2) \in S - (S' \cap S)$ are handled by the unified addition formulae (lines 47 and 48).

The points at infinity on the desingularized projective closure of $E_{\mathbf{E},a,d}$ are not defined over $\mathbb{K}$ if d is not a square in $\mathbb{K}$ and a is a square in $\mathbb{K}$, see [BBJ+08]. Having noted this, it was proven in [BL07b] (where $a = 1$) and later in [BBJ+08] that the unified addition formulae (4.39) and (4.40) are complete provided that d is not a square in $\mathbb{K}$ and a is a square in $\mathbb{K}$. Algorithm 4.4.1 complies with the completeness criterion since only the lines 47 to 49 are necessary in this case. Note that the assumptions on the curve constants a and d limit the number of curves in twisted Edwards form for which the unified addition formulae are complete. In [BBJ+08] such curves are named *complete Edwards curves*.

Algorithm 4.4.1 also complies with Lemma 4.4.7. If P_1 and P_2 are points of odd order only the lines 47 to 49 are necessary. This technique applies to all twisted Edwards curves.

Algorithm 4.4.1 also complies with Lemma 4.4.4. If P_1 and P_2 are distinct points of odd order then only the lines 43 to 45 are necessary. This technique applies to all twisted Edwards curves. The doubling formulae (4.35) and (4.36) are enough to handle the special case $P_1 = P_2$.

More formulae Using the tools of Chapter 3, it is possible to derive other low-degree addition formulae;

$$x_3 = \frac{e(x_1y_1 + x_2y_2) + f(x_1y_2 + y_1x_2)}{e(y_1y_2 + ax_1x_2) + f(1 + dx_1y_1x_2y_2)}, \tag{4.41}$$

$$y_3 = \frac{g(x_1y_1 - x_2y_2) + h(y_1y_2 - ax_1x_2)}{g(x_1y_2 - y_1x_2) + h(1 - dx_1y_1x_2y_2)} \tag{4.42}$$

assuming that $(e(y_1y_2 + ax_1x_2) + f(1 + dx_1y_1x_2y_2))(g(x_1y_2 - y_1x_2) + h(1 - dx_1y_1x_2y_2)) \neq 0$ where $e, f, g, h \in \mathbb{K}$ such that $efgh \neq 0$. These formulae tend to include more subexpressions which are not commonly shared. Therefore, their evaluation turns out to be more costly and thus not considered hereafter.

Literature notes Other results related to the affine formulae for twisted Edwards curves can be found in the literature. Bernstein *et al.* used Edwards curves (i.e. $a = 1$) in the ECM method of integer factorization in [BBLP08]. Bernstein *et al.* introduced the shape $d_1(x+y)+d_2(x^2+y^2) = (x+x^2)(y+y^2)$ and presented results on the arithmetic of these curves when $\mathrm{char}(\mathbb{K}) = 2$ in [BLR08]. These curve are named binary Edwards curves. In chronological order, Das and Sarkar [DS08], Ionica and Joux [IJ08], and Arène *et al.* [ALNR09] introduced successively faster formulae for pairing computations. The results in [DS08] and [IJ08] are based on the unified addition formulae and the doubling formulae with $a = 1$. The results in [ALNR09] are based on the dedicated addition formulae and the doubling formulae. The same reference also provided a geometric interpretation of the group law on twisted Edwards curves.

4.5 Twisted Jacobi intersection form

This section presents the group law on $E_{\mathbf{I},b,a}$ in affine coordinates. It also investigates the exceptional summands for each set of formulae and provides a complete addition algorithm for all twisted Jacobi intersection curves by properly handling an entire set of divide-by-zero exceptions. In addition, practical ways of preventing these exceptions are explained.

Throughout this section, let $\mathbb{K}$ be a field of odd characteristic. Recall from Chapter 2 that a twisted Jacobi intersection curve is defined by

$$E_{\mathbf{I},b,a} : \; bs^2 + c^2 = 1, as^2 + d^2 = 1$$

where $a, b \in \mathbb{K}$ with $ab(a-b) \neq 0$. Assume that both $-a$ and $-b$ are squares in $\mathbb{K}$. Recall from Chapter 2 that the set of $\mathbb{K}$-rational points on $E_{\mathbf{I},b,a}$ is defined by

$$E_{\mathbf{I},b,a}(\mathbb{K}) = \{(s,c,d) \in \mathbb{K}^3 \mid bs^2 + c^2 = 1, as^2 + d^2 = 1\} \cup \{\Omega_1, \Omega_2, \Omega_3, \Omega_4\}$$

where $\Omega_1, \Omega_2, \Omega_3, \Omega_4$ are points at infinity.

Identity element and negation The identity element can suitably be taken as $(0,1,1)$. Let (s_1, c_1, d_1) be a point on $E_{\mathbf{I},b,a}$. The negative of (s_1, c_1, d_1) is $(-s_1, c_1, d_1)$.

Doubling The doubling formulae on $E_{\mathbf{I},b,a}$ is given by $[2](s_1, c_1, d_1) = (s_3, c_3, d_3)$ where

$$s_3 = 2s_1c_1d_1/(c_1^2 + bs_1^2d_1^2), \tag{4.43}$$
$$c_3 = (c_1^2 - bs_1^2d_1^2)/(c_1^2 + bs_1^2d_1^2), \tag{4.44}$$
$$d_3 = (2d_1^2 - c_1^2 - bs_1^2d_1^2)/(c_1^2 + bs_1^2d_1^2). \tag{4.45}$$

assuming that $c_1^2 + bs_1^2d_1^2 \neq 0$. These formulae are of minimal total degree and do not depend on the curve constants a and b. By the curve equation the denominator $c_1^2 + bs_1^2d_1^2$ is equivalent to $1 - abs_1^4$ or $c_1^2 + d_1^2 - c_1^2d_1^2$ or $d_1^2 + as_1^2c_1^2$. These denominators can also be used.

The points $(0,-1,1)$, $(0,1,-1)$, and $(0,-1,-1)$ are of order 2. This can be determined by solving $bs_1^2 + c_1^2 = 1, as_1^2 + d_1^2 = 1$ and $(s_3, c_3, d_3) = (0,1,1)$ for s_1, c_1, and d_1 where s_3, c_3, and d_3 are given by (4.43), (4.44), and (4.45).

The four points of the form having the c-coordinates equal to zero are of order 4. This can be determined by solving $bs_1^2 + c_1^2 = 1, as_1^2 + d_1^2 = 1$ and $(s_3, c_3, d_3) = (0,-1,1)$ for s_1, c_1, and d_1. Another set of four points having the d-coordinates equal to zero are also of order 4. This can be determined by solving $bs_1^2 + c_1^2 = 1, as_1^2 + d_1^2 = 1$ and $(s_3, c_3, d_3) = (0,1,-1)$ for s_1, c_1, and d_1. There are twelve points of order 4 in total (over a sufficiently large finite extension of $\mathbb{K}$). Therefore the remaining four points of order 4 have to be the points at infinity Ω_1, Ω_2, Ω_3, and Ω_4. These points are the only exceptions of (4.43), (4.44), and (4.45). The following remark is immediate.

Remark 4.5.1. $[2](s_1, c_1, d_1)$ is a point at infinity if and only if $c_1^2 + bs_1^2d_1^2 = 0$.

Remark 4.5.1 does not extend to the case of generic additions. However, it is still useful in

proving some lemmas regarding the generic addition formulae which will be presented next.

Dedicated addition Further let (s_2, c_2, d_2) be a point on $E_{\mathbf{I},b,a}$. The addition formulae on $E_{\mathbf{I},b,a}$ are given by $(s_1, c_1, d_1) + (s_2, c_2, d_2) = (s_3, c_3, d_3)$ where

$$s_3 = (s_1^2 - s_2^2)/(s_1c_2d_2 - c_1d_1s_2), \tag{4.46}$$

$$c_3 = (s_1c_1d_2 - d_1s_2c_2)/(s_1c_2d_2 - c_1d_1s_2), \tag{4.47}$$

$$d_3 = (s_1d_1c_2 - c_1s_2d_2)/(s_1c_2d_2 - c_1d_1s_2) \tag{4.48}$$

assuming that $s_1c_2d_2 - c_1d_1s_2 \neq 0$. These formulae are of minimal total degree and do not depend on the curve constants a and b. These formulae do not work for identical summands hence the name *dedicated*.

If $(s_1, c_1, d_1) + (s_2, c_2, d_2)$ is a point at infinity then $s_1c_2d_2 - c_1d_1s_2 = 0$. Otherwise, $(s_1, c_1, d_1) + (s_2, c_2, d_2)$ would be an affine point since it can be shown using the relations $bs_1^2 + c_1^2 = 1, as_1^2 + d_1^2 = 1$ and $bs_2^2 + c_2^2 = 1, as_2^2 + d_2^2 = 1$ that the algebraic expressions for (s_3, c_3, d_3) satisfy $bs_3^2 + c_3^2 = 1, as_3^2 + d_3^2 = 1$. The converse, however, does not necessarily apply. This means that if $s_1c_2d_2 - c_1d_1s_2 = 0$ then $(s_1, c_1, d_1) + (s_2, c_2, d_2)$ may not be a point at infinity. Therefore it is worth investigating the exceptional cases. The denominators of (4.46), (4.47), and (4.48) vanish for some summands which are described in the following lemma explicitly.

Lemma 4.5.2. *Let $a, b \in \mathbb{K}$ with $ab(a-b) \neq 0$. Fix $\alpha, \beta \in \mathbb{K}$ so that $\alpha^2 = -a$ and $\beta^2 = -b$. Fix $s_1 \in \mathbb{K}\backslash\{0\}$ and $c_1, d_1 \in \mathbb{K}$ such that $bs_1^2 + c_1^2 = 1$ and $as_1^2 + d_1^2 = 1$. Let $s_2, c_2, d_2 \in \mathbb{K}$ such that $bs_2^2 + c_2^2 = 1$, $as_2^2 + d_2^2 = 1$. Now, $s_1c_2d_2 - c_1d_1s_2 = 0$ if and only if $(s_2, c_2, d_2) \in S$ where*

$$S = \big[(s_1, c_1, d_1), (s_1, -c_1, -d_1), (-s_1, -c_1, d_1), (-s_1, c_1, -d_1),$$

$$(\frac{1}{\alpha\beta s_1}, \frac{d_1}{\alpha s_1}, \frac{c_1}{\beta s_1}), (\frac{1}{\alpha\beta s_1}, \frac{-d_1}{\alpha s_1}, \frac{-c_1}{\beta s_1}), (\frac{-1}{\alpha\beta s_1}, \frac{-d_1}{\alpha s_1}, \frac{c_1}{\beta s_1}), (\frac{-1}{\alpha\beta s_1}, \frac{d_1}{\alpha s_1}, \frac{-c_1}{\beta s_1})\big].$$

Proof. $\Rightarrow$: Assume that $s_1c_2d_2 - c_1d_1s_2 = 0$. Solving the equations $s_1c_2d_2 - c_1d_1s_2 = 0$, $bs_2^2 + c_2^2 = 1$ and $as_2^2 + d_2^2 = 1$ simultaneously for s_2, c_2, and d_2 gives S. All entries in S are defined since $s_1 \neq 0$. $\Leftarrow$: The claims follow trivially by substitution. □

The following lemma shows that if one of the summands is of odd order then in the presence of an exception, the other summand is always of even order.

Lemma 4.5.3. *Let $a, b, s_1, c_1, d_1, s_2, c_2, d_2$ be defined as in Lemma 4.5.2. Assume that $P_1 = (s_1, c_1, d_1)$ is a fixed point of odd order. Assume that $P_2 \in S\backslash\{P_1\}$. Then P_2 is of even order.*

Proof. Note that points at infinity (over the extension of $\mathbb{K}$ where they exist) are of even order. Assume that $P_1 = (s_1, c_1, d_1)$ is a fixed point of odd order hence not a point at infinity. Suppose that P_2 is of odd order hence not a point at infinity. It follows that $P_1 \pm P_2$, $M = 2P_1$, and $N = 2P_2$ are all of odd order hence not points at infinity.

Assume that $P_2 \in S\backslash\{P_1\}$. So, $P_1 \neq P_2$. Plus, $s_1c_2d_2 - c_1d_1s_2 = 0$ by Lemma 4.5.2. It follows that $P_1 \neq -P_2$, for otherwise, $s_1c_2d_2 - c_1d_1s_2 = 2s_1c_1d_1 = 0$ which means that $c_1d_1 = 0$ since $s_1 \neq 0$. But then P_1 would be of even order.

It is possible to continue in a similar way used in the previous sections however this time computer algebra will be used. The following Maple script verifies that $s(M)^2 = s(N)^2$, $c(M)^2 = c(N)^2$, and $d(M)^2 = d(N)^2$.

```
> Q:=(s,c,d)->(b*s^2+c^2-1,a*s^2+d^2-1):
> sM:=2*s1*c1*d1/(c1^2+b*s1^2*d1^2):
> cM:=(c1^2-b*s1^2*d1^2)/(c1^2+b*s1^2*d1^2):
> dM:=(2*d1^2-c1^2-b*s1^2*d1^2)/(c1^2+b*s1^2*d1^2):
> sN:=2*s2*c2*d2/(c2^2+b*s2^2*d2^2):
> cN:=(c2^2-b*s2^2*d2^2)/(c2^2+b*s2^2*d2^2):
> dN:=(2*d2^2-c2^2-b*s2^2*d2^2)/(c2^2+b*s2^2*d2^2):
> simplify([sM^2-sN^2,cM^2-cN^2,dM^2-dN^2],[s1*c2*d2-c1*d1*s2=0,Q(s1,c1,d1),Q(s2,c2,d2)]);
[0,0,0]
```

Therefore, $s(M) = \pm s(N)$, $c(M) = \pm c(N)$, and $d(M) = \pm d(N)$ taking the signs independently. Now,

- $s(M) = s(N)$, $c(M) = c(N)$, $d(M) = d(N)$: $M - N = (0, 1, 1)$. So, $M - N = 2P_1 - 2P_2 = 2(P_1 - P_2) = (0, 1, 1)$;
- $s(M) = -s(N)$, $c(M) = c(N)$, $d(M) = d(N)$: $M + N = (0, 1, 1)$. So, $M + N = 2P_1 + 2P_2 = 2(P_1 + P_2) = (0, 1, 1)$;
- $s(M) = s(N)$, $c(M) = -c(N)$, $d(M) = d(N)$: $M + N = (0, -1, 1)$. So, $M + N = 2(2P_1 + 2P_2) = 4(P_1 + P_2) = (0, 1, 1)$;
- $s(M) = -s(N)$, $c(M) = -c(N)$, $d(M) = d(N)$: $M - N = (0, -1, 1)$. So, $M - N = 2(2P_1 + 2P_2) = 4(P_1 + P_2) = (0, 1, 1)$;
- $s(M) = s(N)$, $c(M) = c(N)$, $d(M) = -d(N)$: $M + N = (0, 1, -1)$. So, $M + N = 2(2P_1 + 2P_2) = 4(P_1 + P_2) = (0, 1, 1)$;
- $s(M) = -s(N)$, $c(M) = c(N)$, $d(M) = -d(N)$: $M - N = (0, 1, -1)$. So, $M - N = 2(2P_1 + 2P_2) = 4(P_1 + P_2) = (0, 1, 1)$;
- $s(M) = s(N)$, $c(M) = -c(N)$, $d(M) = -d(N)$: $M - N = (0, -1, -1)$. So, $M - N = 2(2P_1 - 2P_2) = 4(P_1 - P_2) = (0, 1, 1)$; and
- $s(M) = -s(N)$, $c(M) = -c(N)$, $d(M) = -d(N)$: $M + N = (0, -1, -1)$. So, $M + N = 2(2P_1 - 2P_2) = 4(P_1 - P_2) = (0, 1, 1)$.

Since $P_1 \neq \pm P_2$, in all cases $P_1 \pm P_2$ is of even order, contradiction. In conclusion, $P_2 \in S \backslash \{P_1\}$ is of even order provided that P_1 is of odd order. □

A practical solution is now provided to prevent the exceptional cases of (4.46), (4.47), and (4.48).

Lemma 4.5.4. *Let $\mathbb{K}$ be a field of odd characteristic. Let $E_{\mathbf{I},b,a}$ be a twisted Jacobi intersection curve defined over $\mathbb{K}$. Let $P_1 = (s_1, c_1, d_1)$ and $P_2 = (s_2, c_2, d_2)$ be points on $E_{\mathbf{I},b,a}$. Assume that P_1 and P_2 are of odd order with $P_1 \neq P_2$. It follows that $s_1c_2d_2 - c_1d_1s_2 \neq 0$.*

Proof. Assume that P_1 and P_2 are of odd order with $P_1 \neq P_2$. Suppose that $s_1 = 0$ and $s_2 = 0$. Then, $P_1 = P_2 = (0, 1, 1)$, contradiction. So, either $s_1 \neq 0$ or $s_2 \neq 0$. The claim then follows from Lemma 4.5.2 and Lemma 4.5.3 (by swapping P_1 and P_2 when necessary). □

Unified addition Alternative addition formulae on $E_{\mathbf{I},b,a}$ are given by $(s_1, c_1, d_1) + (s_2, c_2, d_2) = (s_3, c_3, d_3)$ where

$$s_3 = (s_1c_2d_2 + c_1d_1s_2)/(1 - abs_1^2s_2^2), \tag{4.49}$$

$$c_3 = (c_1c_2 - bs_1d_1s_2d_2)/(1 - abs_1^2s_2^2), \tag{4.50}$$

$$d_3 = (d_1d_2 - as_1c_1s_2c_2)/(1 - abs_1^2s_2^2) \tag{4.51}$$

assuming that $1 - abs_1^2s_2^2 \neq 0$. These formulae work for identical summands in most of the cases hence the name *unified*.

If $(s_1, c_1, d_1) + (s_2, c_2, d_2)$ is a point at infinity then $1 - abs_1^2s_2^2 = 0$. Otherwise, $(s_1, c_1, d_1) + (s_2, c_2, d_2)$ would be an affine point since it can be shown using the relations $bs_1^2 + c_1^2 = 1, as_1^2 + d_1^2 = 1$ and $bs_2^2 + c_2^2 = 1, as_2^2 + d_2^2 = 1$ that the algebraic expressions for (s_3, c_3, d_3) satisfy $bs_3^2 + c_3^2 = 1, as_3^2 + d_3^2 = 1$. The converse, however, does not necessarily apply. This means that if $1 - abs_1^2s_2^2 = 0$ then $(s_1, c_1, d_1) + (s_2, c_2, d_2)$ may not be a point at infinity. Therefore it is worth investigating the exceptional cases. The denominators of (4.49), (4.50), and (4.51) vanish for some summands which are described in the following lemma explicitly.

Lemma 4.5.5. *Let $a, b, s_1, c_1, d_1, s_2, c_2, d_2$ be defined as in Lemma 4.5.2. It follows that $1 - abs_1^2s_2^2 \neq 0$ if and only if $(s_2, c_2, d_2) \in S'$ where*

$$S' = [(\frac{1}{\alpha\beta s_1}, \frac{-d_1}{\alpha s_1}, \frac{c_1}{\beta s_1}), (\frac{1}{\alpha\beta s_1}, \frac{d_1}{\alpha s_1}, \frac{-c_1}{\beta s_1}), (\frac{-1}{\alpha\beta s_1}, \frac{d_1}{\alpha s_1}, \frac{c_1}{\beta s_1}), (\frac{-1}{\alpha\beta s_1}, \frac{-d_1}{\alpha s_1}, \frac{-c_1}{\beta s_1})$$

$$(\frac{1}{\alpha\beta s_1}, \frac{d_1}{\alpha s_1}, \frac{c_1}{\beta s_1}), (\frac{1}{\alpha\beta s_1}, \frac{-d_1}{\alpha s_1}, \frac{-c_1}{\beta s_1}), (\frac{-1}{\alpha\beta s_1}, \frac{-d_1}{\alpha s_1}, \frac{c_1}{\beta s_1}), (\frac{-1}{\alpha\beta s_1}, \frac{d_1}{\alpha s_1}, \frac{-c_1}{\beta s_1})].$$

Proof. $\Rightarrow$: Assume that $1 - abs_1^2s_2^2 = 0$. Solving the equations $1 - abs_1^2s_2^2 = 0$, $bs_2^2 + c_2^2 = 1$, and $as_2^2 + d_2^2 = 1$ simultaneously for s_2, c_2, and d_2 gives S'. All entries in S' are defined since $s_1 \neq 0$. $\Leftarrow$: The claims follow trivially by substitution. □

This lemma and Lemma 4.5.2 excludes $s_1 = 0$. If so, $1 - abs_1^2s_2^2 \neq 0$.

The following lemma shows that if one of the summands is of odd order then in the presence of a vanished denominator, the other summand is always of even order.

Lemma 4.5.6. *Let $a, b, s_1, c_1, d_1, s_2, c_2, d_2$ be defined as in Lemma 4.5.5. Assume that $P_1 = (s_1, c_1, d_1)$ is a fixed point of odd order. Assume that $P_2 = (s_2, c_2, d_2) \in S'$. Then, P_2 is of even order.*

Proof. The proof is similar to the proof of Lemma 4.5.3. The only difference is that the phrase `s1*c2*d2-c1*d1*s2=0` should be changed to `1-a*b*s1*s1*s2*s2=0` in the Maple script of the proof of Lemma 4.5.3 and the claim follows. □

In the following lemma, with reasonable assumptions, it is shown that exceptions can be prevented regardless of any assumption on the curve constants.

Lemma 4.5.7. *Let $\mathbb{K}$ be a field of odd characteristic. Let $E_{\mathbf{I},b,a}$ be a twisted Jacobi intersection curve defined over $\mathbb{K}$. Let $P_1 = (s_1, c_1, d_1)$ and $P_2 = (s_2, c_2, d_2)$ be points on $E_{\mathbf{I},b,a}$. Assume that P_1 and P_2 are of odd order. It follows that $1 - abs_1^2s_2^2 \neq 0$.*

Proof. Assume that P_1 and P_2 are of odd order. If $s_1 s_2 = 0$ then $1 - abs_1^2 s_2^2 \neq 0$ as desired. If $s_1 s_2 \neq 0$ then the claim follows from Lemma 4.5.5 and Lemma 4.5.6 (by swapping P_1 and P_2 when necessary). □

Exception handling in the general case Algorithm 4.5.1 provides a complete addition on all twisted Jacobi intersection curves. The correctness of the algorithm follows from two observations. Firstly, when a point at infinity is involved as the sum or as one of the summands along lines 2 to 13, it is tedious but straightforward to check that the output of the algorithm is correct using the *implicit* technique mentioned at the start of the chapter. Line 1 conditionally swaps the inputs to eliminate half of the input-wise symmetric branches. The second observation is that *glueing* together the unified addition and the dedicated addition formulae is enough to handle all exceptions when both of the summands and the sum are affine points. This fact follows from Lemma 4.5.5 and Lemma 4.5.2 by observing that $\#(S' \cap S) = 4$. This means that if $(s_2, c_2, d_2) \in S' \cap S$ then the output must be a point at infinity (lines 53 to 56) since there are exactly four points at infinity. The remaining exceptional cases which occur at $(s_2, c_2, d_2) \in S' \backslash (S' \cap S)$ are handled by the dedicated addition formulae (lines 43 to 45). Similarly the exceptions at $(s_2, c_2, d_2) \in S \backslash (S' \cap S)$ are handled by the unified addition formulae (lines 48 to 50).

The points at infinity on the projective closure of $E_{\mathbf{I},b,a}$ are not defined over $\mathbb{K}$ if a is not a square in $\mathbb{K}$. Having noted this, the following lemma implies that these addition formulae are complete if a is not a square in $\mathbb{K}$.

Lemma 4.5.8. *Let $a, b, s_1, s_2 \in \mathbb{K}$. Assume that ab is non-square. Then $1 - abs_1^2 s_2^2 \neq 0$.*

Proof. See the proof of Lemma 4.2.8 in §4.2. □

Algorithm 4.5.1 complies with the completeness criterion since only the lines 48 to 50 are necessary in this case. Note that the assumption on the curve constants a and d limits the number of curves in twisted Jacobi intersection form for which the unified addition formulae are complete.

Algorithm 4.5.1 also complies with Lemma 4.5.7. If P_1 and P_2 are points of odd order then all branches are eliminated and the lines 48 to 50 suffice. This technique applies to all twisted Jacobi intersection curves.

Algorithm 4.5.1 also complies with Lemma 4.5.4. If P_1 and P_2 are distinct points of odd order then all branches are eliminated and the lines 43 to 45 suffice. This technique applies to all twisted Jacobi intersection curves. The doubling formulae (4.55), (4.56), and (4.57) are enough to handle the special case $P_1 = P_2$.

More formulae Using the tools of Chapter 3, it is possible to derive many more low-degree addition formulae. In the case of doubling, the numerators can also be written in several alternative ways. However, there exist no formulae of lower total degree to compute any of these affine coordinates. The following examples are specifically chosen since each of them share many common subexpressions among their coordinates. More common subexpressions often reduces the complexity of the evaluation. One set of formulae (independent of a and b)

Algorithm 4.5.1: Addition law in affine coordinates for twisted Jacobi intersection form

input : $P_1, P_2, \Omega_1, \Omega_2, \Omega_3, \Omega_4 \in E_{\mathbf{I},b,a}(\mathbb{K})$ and fixed $\alpha, \beta \in \mathbb{K}$ such that $\alpha^2 = -a$ and $\beta^2 = -b$.
output : $P_1 + P_2$.

if $P_1 \in \{\Omega_1, \Omega_2, \Omega_3, \Omega_4\}$ **then** $P_t \leftarrow P_1$, $P_1 \leftarrow P_2$, $P_2 \leftarrow P_t$.
if $P_2 = \Omega_1$ **then**
  if $P_1 = \Omega_1$ **then return** $(0, -1, -1)$.
  else if $P_1 = \Omega_2$ **then return** $(0, -1, 1)$.
  else if $P_1 = \Omega_3$ **then return** $(0, 1, -1)$.
  else if $P_1 = \Omega_4$ **then return** $(0, 1, 1)$.
  else if $P_1 = (0, 1, 1)$ **then return** Ω_1.
  else if $P_1 = (0, 1, -1)$ **then return** Ω_2.
  else if $P_1 = (0, -1, 1)$ **then return** Ω_3.
  else if $P_1 = (0, -1, -1)$ **then return** Ω_4.
  else return $(-1/(\alpha\beta s_1), d_1/(\alpha s_1), c_1/(\beta s_1))$.
else if $P_2 = \Omega_2$ **then**
  if $P_1 = \Omega_1$ **then return** $(0, -1, 1)$.
  else if $P_1 = \Omega_2$ **then return** $(0, -1, -1)$.
  else if $P_1 = \Omega_3$ **then return** $(0, 1, 1)$.
  else if $P_1 = \Omega_4$ **then return** $(0, 1, -1)$.
  else if $P_1 = (0, 1, -1)$ **then return** Ω_1.
  else if $P_1 = (0, 1, 1)$ **then return** Ω_2.
  else if $P_1 = (0, -1, -1)$ **then return** Ω_3.
  else if $P_1 = (0, -1, 1)$ **then return** Ω_4.
  else return $(1/(\alpha\beta s_1), d_1/(\alpha s_1), -c_1/(\beta s_1))$.
else if $P_2 = \Omega_3$ **then**
  if $P_1 = \Omega_1$ **then return** $(0, 1, -1)$.
  else if $P_1 = \Omega_2$ **then return** $(0, 1, 1)$.
  else if $P_1 = \Omega_3$ **then return** $(0, -1, -1)$.
  else if $P_1 = \Omega_4$ **then return** $(0, -1, 1)$.
  else if $P_1 = (0, -1, 1)$ **then return** Ω_1.
  else if $P_1 = (0, -1, -1)$ **then return** Ω_2.
  else if $P_1 = (0, 1, 1)$ **then return** Ω_3.
  else if $P_1 = (0, 1, -1)$ **then return** Ω_4.
  else return $(1/(\alpha\beta s_1), -d_1/(\alpha s_1), c_1/(\beta s_1))$.
else if $P_2 = \Omega_4$ **then**
  if $P_1 = \Omega_1$ **then return** $(0, 1, 1)$.
  else if $P_1 = \Omega_2$ **then return** $(0, 1, -1)$.
  else if $P_1 = \Omega_3$ **then return** $(0, -1, 1)$.
  else if $P_1 = \Omega_4$ **then return** $(0, -1, -1)$.
  else if $P_1 = (0, -1, -1)$ **then return** Ω_1.
  else if $P_1 = (0, -1, 1)$ **then return** Ω_2.
  else if $P_1 = (0, 1, -1)$ **then return** Ω_3.
  else if $P_1 = (0, 1, 1)$ **then return** Ω_4.
  else return $(-1/(\alpha\beta s_1), -d_1/(\alpha s_1), -c_1/(\beta s_1))$.
else if $s_1c_2d_2 - c_1d_1s_2 \neq 0$ **then**
  $s_3 \leftarrow (s_1^2 - s_2^2)/(s_1c_2d_2 - c_1d_1s_2)$.
  $c_3 \leftarrow (s_1c_1d_2 - d_1s_2c_2)/(s_1c_2d_2 - c_1d_1s_2)$.
  $d_3 \leftarrow (s_1d_1c_2 - c_1s_2d_2)/(s_1c_2d_2 - c_1d_1s_2)$.
  return (s_3, c_3, d_3).
else if $1 - abs_1^2s_2^2 \neq 0$ **then**
  $s_3 \leftarrow (s_1c_2d_2 + c_1d_1s_2)/(1 - abs_1^2s_2^2)$.
  $c_3 \leftarrow (c_1c_2 - bs_1d_1s_2d_2)/(1 - abs_1^2s_2^2)$.
  $d_3 \leftarrow (d_1d_2 - as_1c_1s_2c_2)/(1 - abs_1^2s_2^2)$.
  return (s_3, c_3, d_3).
else
  if $P_2 = (1/(\alpha\beta s_1), -d_1/(\alpha s_1), -c_1/(\beta s_1))$ **then return** Ω_1.
  else if $P_2 = (-1/(\alpha\beta s_1), -d_1/(\alpha s_1), c_1/(\beta s_1))$ **then return** Ω_2.
  else if $P_2 = (-1/(\alpha\beta s_1), d_1/(\alpha s_1), -c_1/(\beta s_1))$ **then return** Ω_3.
  else return Ω_4.
end

is given as follows.

$$\begin{aligned} s_3 &= 2s_1c_1d_1/(c_1^2+d_1^2-c_1^2d_1^2), && (4.52)\\ c_3 &= (c_1^2-d_1^2+c_1^2d_1^2)/(c_1^2+d_1^2-c_1^2d_1^2), && (4.53)\\ d_3 &= (d_1^2-c_1^2+c_1^2d_1^2)/(c_1^2+d_1^2-c_1^2d_1^2). && (4.54) \end{aligned}$$

Another set of formulae is given as follows.

$$\begin{aligned} s_3 &= 2s_1c_1d_1/(1-abs_1^4), && (4.55)\\ c_3 &= (c_1^2-bs_1^2d_1^2)/(1-abs_1^4), && (4.56)\\ d_3 &= (d_1^2-as_1^2c_1^2)/(1-abs_1^4) && (4.57) \end{aligned}$$

Yet another set of formulae (independent of b) is given as follows.

$$\begin{aligned} s_3 &= 2s_1c_1d_1/(d_1^2+as_1^2c_1^2), && (4.58)\\ c_3 &= (2c_1^2-d_1^2-as_1^2c_1^2)/(d_1^2+as_1^2c_1^2), && (4.59)\\ d_3 &= (d_1^2-as_1^2c_1^2)/(d_1^2+as_1^2c_1^2). && (4.60) \end{aligned}$$

Some of these formulae will be used later in Chapter 5. It is possible to provide many more examples. Indeed some alternative examples will be further mentioned in the following chapter in which the aim is to investigate inversion-free operation counts on projective coordinates.

In the case of unified addition, observe that the denominator contains s_i^2. Using the defining equations $1-abs_1^2s_2^2$ can also be written as $1-as_1^2(1-c_2^2)=1-as_1^2+as_1^2c_2^2=d_1^2+as_1^2c_2^2$ or $1-bs_1^2(1-d_2^2)=1-bs_1^2+bs_1^2d_2^2=c_1^2+bs_1^2d_2^2$ or $1-(a/b)(1-c_1^2)(1-c_2^2)=1-(a/b)+(a/b)c_1^2+(a/b)c_2^2-(a/b)c_1^2c_2^2$ or $1-(b/a)(1-d_1^2)(1-d_2^2)=1-(b/a)+(b/a)d_1^2+(b/a)d_2^2-(b/a)d_1^2d_2^2$ or $1-(1-c_1^2)(1-d_2^2)=c_1^2+d_2^2-c_1^2d_2^2$. There exists even more alternatives. However all of these denominators have a total degree of 4. Plus, the exceptional cases for all of these denominators are the same. Similar arguments also apply to the case of dedicated addition.

Literature notes Other results related to the affine formulae for twisted Jacobi intersection form can be found in the literature. The group law on Jacobi intersection curves are typically derived directly from Jacobi elliptic functions, cf. [Jac29], [WW27], [CC86], and [BJ03a].

4.6 Conclusion

This chapter has provided low-total-degree formulae for elliptic curves in five basic forms of elliptic curves, namely, short Weierstrass form (§4.1), extended Jacobi quartic form (§4.2), twisted Hessian form (§4.3), twisted Edwards form (§4.4), and twisted Jacobi intersection form (§4.5). Besides bringing various formulae together in one chapter, the arithmetic properties of these formulae have also been studied algebraically. This is achieved by the determination of exceptional summands explicitly, the showing of how exceptional cases (such as additions involving the points at infinity) can be handled, the clarification of how these exceptions

can be prevented under reasonable assumptions, and a complete addition algorithm in affine coordinates.

Complete addition by exception handling Although the complete addition algorithms on Weierstrass curves were well documented in the literature, other representations had only been studied in the case of some particular addition formulae and in many resources the exceptional situations were simply omitted. This chapter has studied the topic from a wider perspective. In particular, complete addition algorithms; Algorithm 4.2.1 in §4.2, Algorithm 4.3.1 in §4.3 Algorithm 4.4.1 in §4.4, Algorithm 4.5.1 in §4.5, have been developed for the first time.

In order to build these algorithms each section has developed two alternative formulae covering the affine part of the curve. By covering the affine part it is meant that (the sequence of points) $S' \cap S$ composed of element(s) forcing the condition $P_1 + P_2$ being a point at infinity at an occurrence of a vanished denominator. See the lemmas in each section for detail. In fact, the addition of an affine point (x_1, y_1) with one of the entries in the first half of S' (or S) yields an affine point although they produce division-by-zero exceptions with the relevant formulae. The second half of entries also produce division-by-zero exceptions but the sum this time is a point at infinity. These facts have been left implicit in the text to minimize repetition but explicitly used while designing the complete addition algorithms for elliptic curves covered by a particular form.

Complete addition by assumption(s) on the curve constant(s) Excluding the Weierstrass form, new complete addition formulae are provided for each form using only a single set of addition formulae. This property was first stated for some Edwards curves by Bernstein and Lange in [BL07b]. Later the same idea was extended to suitable classes of elliptic curves in twisted Edwards form in [BBJ+08] and to twisted Hessian form in [BL07a]. This chapter has extended the same idea for suitable classes of elliptic curves in extended Jacobi quartic and twisted Jacobi intersection forms, see §4.2 and §4.5. This technique forces the point(s) at infinity to be defined over a proper extension of $\mathbb{K}$ but not over $\mathbb{K}$ itself. Therefore, all points on the selected curve are affine points. The second part of the technique is composed of finding a single set of addition formulae with denominators which cannot vanish for any pair of summands.

Exception prevention At this point, having dedicated addition formulae which do not work for some special summands might seem unnecessary. However, it will be shown in Chapter 5 that dedicated addition formulae lead to inversion-free algorithms which are as fast or even faster than unified addition formulae. For simplicity in implementations, it is desirable to have a condition for which the number of exceptional cases is minimal when using dedicated addition formulae. Each section has introduced sufficient assumptions to have $P_1 = P_2$ as the only exceptional case to be handled. (Of course, in the case of short Weierstrass form, the point at infinity is also handled separately since it is always involved in the arithmetic as the identity element.) In speed oriented implementations, distinguishing $P_1 = P_2$ is no problem because it is a standard method to switch to faster doubling formulae in inversion-free coordinate

systems, see Chapter 5. In addition, the same technique removes all exceptional cases for the unified addition formulae under suitable assumptions. In cryptographic applications, attention is typically restricted to a large prime order subgroup of the curve. This criterion covers a variety of different assumptions by further simplifying the exception prevention techniques introduced in this chapter.

Chapter 5

Group law in projective coordinates

Elliptic curves became increasingly employed in cryptology in the 1980's, and since then there has been a series of incremental studies to speed up the arithmetic of elliptic curves. This chapter uses the tools from Chapter 3 and the group laws in affine coordinates from Chapter 4 in order to develop efficient group laws for elliptic curves over fields of large characteristic. This chapter also provides a collection of selected algorithms from the literature to provide a summary of the recent advances. The work is limited to studying five forms of elliptic curves which have already been discussed in Chapter 4. It should be possible to extend this work to many other curve models including those of small characteristic. This is left as future work.

In what follows some terms related to the group law on elliptic curves will be extensively used. In particular, the term *unified* is used to emphasize that point addition formulae remain valid when two input points are identical, see [CF05, §29.1.2]. Therefore, unified addition formulae can be used for point doubling. The term *dedicated* is used to emphasize that point addition formulae are not unified. The term *re-addition* is used to emphasize that a point addition has already taken place and some of the previously computed data is cached, see [BL07b]. The term *redundant* is used to emphasize that unnecessary computational data is added to a projective representation of a point. The notations are adapted from [CMO98], [BL07b], and [BBJ+08].

This chapter studies the arithmetic of selected elliptic curve forms in suitable coordinate systems. Some coordinate systems (such as $\mathcal{Q}^e$ in §5.2.2) are introduced in this chapter. Some other coordinate systems (such as $\mathcal{J}$ in §5.5.2) are already studied in the literature. Even for such systems this chapter introduces new algorithms which are practical in some applications. For the quartic models $ax^2 + y^2 = 1 + dx^2y^2$ and $y^2 = dx^4 + 2ax^2 + 1$, the optimum speeds are achieved in a proposed approach using mixed homogeneous projective coordinates. For each of these forms the extended homogeneous projective coordinates are introduced which significantly improve the speed of point additions. The slowing down in

point doubling is remedied by mixing extended homogeneous projective coordinates with homogeneous projective coordinates, see §5.1 and §5.2 for details. The twisted Hessian form also exhibits some of these properties. However, in most situations, the proposed optimizations in homogeneous projective coordinates are more interesting than the proposed optimizations in extended homogeneous projective coordinates or modified homogeneous projective coordinates, see §5.4. Jacobi intersection form can provide improved speeds in both homogeneous projective coordinates and modified homogeneous projective coordinates. The speed of this form is close to the speed of Jacobi quartic form, see §5.3. The arithmetic of Weierstrass form has been extensively studied over decades. Therefore it is extremely unlikely to find faster algorithms with arbitrary curve constants. On the other hand, it is still possible to improve the speed of unified additions. The improvements for Weierstrass form are covered in §5.5. Each of these section contains tables to compare literature results with the results from this thesis. The entries which are not cited are the contributions of this thesis.

Since all formulae in this chapter are inversion-free they never cause division by zero exceptions. However, this does not mean that the correct output will always be produced. Some special inputs force all output coordinates be zero producing an n-tuple which is not even a point on the underlying space. When this occurs, the results from Chapter 4 will be recalled to show how to guarantee the correct output under some reasonable assumptions. If those assumptions are not satisfied then the complete affine addition algorithms in Chapter 4 can be incorporated to explicitly handle all such special cases.

5.1 Twisted Edwards form

The arithmetic of twisted Edwards curves is studied under four different point representations in this section, each system having its own advantages. The homogeneous projective coordinate system from [BBJ+08] (also see [BL07b]) is revisited with additional results in §5.1.1. This system allows fast doubling and additions and thus is interesting for efficient implementation. Similarly, the inverted coordinate system from [BBJ+08] (also see [BL07c]) is revisited with additional results in §5.1.2. This system allows faster additions in comparison to homogeneous projective coordinates. Both systems require three coordinates to represent points. The extended homogeneous projective coordinate system is proposed in §5.1.3. This system uses more space by representing each point using four coordinates. On the other hand, the additions in extended coordinates are faster than the three-coordinate systems. Finally, the mixed coordinate system which is proposed in §5.1.4 benefit from the faster doublings in homogeneous projective coordinates and faster addition algorithms in extended homogeneous projective coordinates. Conclusions are drawn in §5.1.5 together with pointers to the literature.

5.1.1 Homogeneous projective coordinates, $\mathcal{E}$

In [BL07b] where $a = 1$ and later in [BBJ+08], homogeneous coordinates are considered for efficiency. In this system, each point (x, y) on $ax^2 + y^2 = 1 + dx^2y^2$ is represented with the triplet $(X : Y : Z)$ which corresponds to the affine point $(X/Z, Y/Z)$ with $Z \neq 0$. These triplets satisfy the homogeneous projective equation $aX^2Z^2 + Y^2Z^2 = Z^4 + dX^2Y^2$. For all nonzero

$\lambda \in \overline{\mathbb{K}}$, $(X : Y : Z) = (\lambda X : \lambda Y : \lambda Z)$. The identity element is represented by $(0 : 1 : 1)$. The negative of $(X : Y : Z)$ is $(-X : Y : Z)$. This coordinate system is denoted by $\mathcal{E}$ throughout the text.

Doubling in $\mathcal{E}$ Let $(X_1 : Y_1 : Z_1)$ with $Z_1 \neq 0$ satisfy $aX^2Z^2 + Y^2Z^2 = Z^4 + dX^2Y^2$. Then $[2](X_1 : Y_1 : Z_1) = (X_3 : Y_3 : Z_3)$ where

$$\begin{aligned} X_3 &= 2X_1Y_1(2Z_1^2 - Y_1^2 - aX_1^2), \\ Y_3 &= (Y_1^2 - aX_1^2)(Y_1^2 + aX_1^2), \\ Z_3 &= (Y_1^2 + aX_1^2)(2Z_1^2 - Y_1^2 - aX_1^2) \end{aligned} \tag{5.1}$$

assuming $Z_3 \neq 0$. These formulae are obtained from (4.35) and (4.36). It was proven in [BBJ+08] that Z_3 is always nonzero if a is a square in $\mathbb{K}$ and d is not a square in $\mathbb{K}$. Also by Lemma 4.4.7, Z_3 is always nonzero if $(X_1 : Y_1 : Z_1)$ is of odd order regardless of the assumptions on a and d (of course, $ad(a-d) \neq 0$).

These formulae do not depend on d. Therefore keeping d arbitrary has no effect on the cost of (5.1). Evaluating (5.1) takes $3\mathbf{M} + 4\mathbf{S} + 1\mathbf{D} + 7\mathbf{a}$ in [BBJ+08];

$$\begin{gathered} B \leftarrow (X_1 + Y_1)^2, \quad C \leftarrow X_1^2, \quad D \leftarrow Y_1^2, \quad E \leftarrow aC, \quad F \leftarrow E + D, \quad H \leftarrow Z_1^2, \\ J \leftarrow F - 2H, \quad X_3 \leftarrow (B - C - D) \cdot J, \quad Y_3 \leftarrow F \cdot (E - D), \quad Z_3 \leftarrow F \cdot J. \end{gathered}$$

If $a = 1$ then the doubling takes $3\mathbf{M} + 4\mathbf{S} + 6\mathbf{a}$ by computing X_3 as $(B - F) \cdot J$, see [BL07b].

If $a = -1$ then the doubling again takes $3\mathbf{M} + 4\mathbf{S} + 6\mathbf{a}$;

$$\begin{gathered} A \leftarrow 2Z_1^2, \quad B \leftarrow Y_1^2, \quad C \leftarrow X_1^2, \quad D \leftarrow B + C, \quad E \leftarrow B - C, \quad F \leftarrow A - E, \\ X_3 \leftarrow ((X_1 + Y_1)^2 - D) \cdot F, \quad Y_3 \leftarrow D \cdot E, \quad Z_3 \leftarrow E \cdot F. \end{gathered}$$

Dedicated addition in $\mathcal{E}$ Further let $(X_2 : Y_2 : Z_2)$ with $Z_2 \neq 0$ satisfy $aX^2Z^2 + Y^2Z^2 = Z^4 + dX^2Y^2$. Then, $(X_1 : Y_1 : Z_1) + (X_2 : Y_2 : Z_2) = (X_3 : Y_3 : Z_3)$ where

$$\begin{aligned} X_3 &= (X_1Y_2 - Y_1X_2)(X_1Y_1Z_2^2 + X_2Y_2Z_1^2), \\ Y_3 &= (Y_1Y_2 + aX_1X_2)(X_1Y_1Z_2^2 - X_2Y_2Z_1^2), \\ Z_3 &= Z_1Z_2(X_1Y_2 - Y_1X_2)(Y_1Y_2 + aX_1X_2) \end{aligned} \tag{5.2}$$

assuming $Z_3 \neq 0$. These formulae are obtained from (4.37) and (4.38). By Lemma 4.4.4, $Z_3 \neq 0$ if $(X_1 : Y_1 : Z_1) \neq (X_2 : Y_2 : Z_2)$ and both summands are of odd order.

Evaluating (5.2) takes $11\mathbf{M} + 2\mathbf{D} + 9\mathbf{a}$;

$$\begin{gathered} A \leftarrow X_1 \cdot Z_2, \quad B \leftarrow Y_1 \cdot Z_2, \quad C \leftarrow Z_1 \cdot X_2, \quad D \leftarrow Z_1 \cdot Y_2, \quad E \leftarrow A \cdot B, \\ F \leftarrow C \cdot D, \quad G \leftarrow E + F, \quad H \leftarrow E - F, \quad J \leftarrow (A - C) \cdot (B + D) - H, \\ K \leftarrow (A + D) \cdot (B + aC) - E - aF, \quad X_3 \leftarrow G \cdot J, \quad Y_3 \leftarrow H \cdot K, \quad Z_3 \leftarrow J \cdot K. \end{gathered}$$

Additionally, if $Z_2 = 1$ then it takes $9\mathbf{M} + 2\mathbf{D} + 9\mathbf{a}$.

If $a = 1$ then the dedicated addition takes $11\mathbf{M} + 8\mathbf{a}$ by computing K as $(A+D)\cdot(B+C) - G$. Additionally, if $Z_2 = 1$ then it takes $9\mathbf{M} + 8\mathbf{a}$.

If $a = -1$ then the dedicated addition again takes $11\mathbf{M} + 8\mathbf{a}$ by computing K as $(A + D) \cdot (B - C) - H$. Additionally, if $Z_2 = 1$ then it takes $9\mathbf{M} + 8\mathbf{a}$.

Unified addition in $\mathcal{E}$ Alternatively, $(X_1 : Y_1 : Z_1) + (X_2 : Y_2 : Z_2) = (X_3 : Y_3 : Z_3)$ where

$$\begin{aligned} X_3 &= Z_1Z_2(X_1Y_2 + Y_1X_2)(Z_1^2Z_2^2 - dX_1X_2Y_1Y_2), \\ Y_3 &= Z_1Z_2(Y_1Y_2 - aX_1X_2)(Z_1^2Z_2^2 + dX_1X_2Y_1Y_2), \\ Z_3 &= (Z_1^2Z_2^2 - dX_1X_2Y_1Y_2)(Z_1^2Z_2^2 + dX_1X_2Y_1Y_2) \end{aligned} \tag{5.3}$$

assuming $Z_3 \neq 0$. These formulae are obtained from (4.39) and (4.40). It was proven in [BBJ+08] that $Z_3 \neq 0$ if a is a square in $\mathbb{K}$ and d is not a square in $\mathbb{K}$. Also by Lemma 4.4.7, $Z_3 \neq 0$ if both $(X_1 : Y_1 : Z_1)$ and $(X_2 : Y_2 : Z_2)$ are of odd order.

Evaluating (5.3) takes $10\mathbf{M} + 1\mathbf{S} + 2\mathbf{D} + 7\mathbf{a}$ in [BBJ+08];

$$\begin{gathered} A \leftarrow Z_1 \cdot Z_2, \quad B \leftarrow A^2, \quad C \leftarrow X_1 \cdot X_2, \quad D \leftarrow Y_1 \cdot Y_2, \quad E \leftarrow dC \cdot D, \quad F \leftarrow B - E, \\ G \leftarrow B + E, \quad X_3 \leftarrow A \cdot F \cdot ((X_1 + Y_1) \cdot (X_2 + Y_2) - C - D), \quad Y_3 \leftarrow A \cdot G \cdot (D - aC), \\ Z_3 \leftarrow F \cdot G. \end{gathered}$$

Additionally, if $Z_2 = 1$ then it takes $9\mathbf{M} + 1\mathbf{S} + 2\mathbf{D} + 7\mathbf{a}$.

If $a = 1$ then the unified addition takes $10\mathbf{M} + 1\mathbf{S} + 1\mathbf{D} + 7\mathbf{a}$ and, additionally, if $Z_2 = 1$ then it takes $9\mathbf{M} + 1\mathbf{S} + 1\mathbf{D} + 7\mathbf{a}$, see `add-2007-bl-3` in [BL07a].

If $a = -1$ then the unified addition takes $10\mathbf{M} + 1\mathbf{S} + 1\mathbf{D} + 6\mathbf{a}$ —saving an extra $\mathbf{a}$— by first computing $U = C + D$ then reusing it as $X_3 \leftarrow A \cdot F \cdot ((X_1 + Y_1) \cdot (X_2 + Y_2) - U)$ and $Y_3 \leftarrow A \cdot G \cdot U$. Additionally, if $Z_2 = 1$ then it takes $9\mathbf{M} + 1\mathbf{S} + 1\mathbf{D} + 6\mathbf{a}$.

5.1.2 Inverted coordinates, $\mathcal{E}^i$

In [BL07c] where $a = 1$ and later in [BBJ+08], inverted coordinates are also considered for efficiency. In this system, each point (x, y) on $ax^2 + y^2 = 1 + dx^2y^2$ is represented with the triplet $(X : Y : Z)$ which corresponds to the affine point $(Z/X, Z/Y)$ with $Z \neq 0$. These triplets satisfy the homogeneous projective equation $X^2Z^2 + aY^2Z^2 = X^2Y^2 + dZ^4$. For all nonzero $\lambda \in \overline{\mathbb{K}}$, $(X : Y : Z) = (\lambda X : \lambda Y : \lambda Z)$. The identity element is represented by the vector $(1, 0, 0)$. The negative of $(X : Y : Z)$ is $(-X : Y : Z)$. This coordinate system is denoted by $\mathcal{E}^i$ throughout the text.

Doubling in $\mathcal{E}^i$ Let $(X_1 : Y_1 : Z_1)$ with $X_1Y_1Z_1 \neq 0$ satisfy $X^2Z^2 + aY^2Z^2 = X^2Y^2 + dZ^4$. Then, $[2](X_1 : Y_1 : Z_1) = (X_3 : Y_3 : Z_3)$ where

$$\begin{aligned} X_3 &= (X_1^2 - aY_1^2)(X_1^2 + aY_1^2), \\ Y_3 &= 2X_1Y_1(X_1^2 + aY_1^2 - 2dZ_1^2), \\ Z_3 &= 2X_1Y_1(X_1^2 - aY_1^2) \end{aligned} \tag{5.4}$$

assuming $X_3Y_3Z_3 \neq 0$. These formulae are obtained from (4.35) and (4.36). In the case $a = 1$ and d a non-square in $\mathbb{K}$, handling the exceptions is described in [BL07c].

Evaluating (5.4) takes $3\mathbf{M} + 4\mathbf{S} + 2\mathbf{D} + 6\mathbf{a}$ in [BBJ+08];

$$\begin{gathered} A \leftarrow X_1^2, \quad B \leftarrow Y_1^2, \quad U \leftarrow aB, \quad C \leftarrow A + U, \quad D \leftarrow A - U, \quad E \leftarrow (X_1 + Y_1)^2 - A - B, \\ X_3 \leftarrow C \cdot D, \quad Y_3 \leftarrow E \cdot (C - (2d)Z_1^2), \quad Z_3 \leftarrow D \cdot E. \end{gathered}$$

If $a = 1$ then the doubling takes $3\mathbf{M} + 4\mathbf{S} + 1\mathbf{D} + 5\mathbf{a}$ by computing E as $(X_1 + Y_1)^2 - C$, see [BL07c].

If $a = -1$ then the doubling again takes $3\mathbf{M}+4\mathbf{S}+1\mathbf{D}+5\mathbf{a}$ by computing E as $(X_1+Y_1)^2-D$ and replacing $U \leftarrow aB$, $C \leftarrow A+U$, $D \leftarrow A-U$ with $C \leftarrow A-B$, $D \leftarrow A+B$.

Dedicated addition in $\mathcal{E}^i$ Further let $(X_2 : Y_2 : Z_2)$ with $X_1Y_1Z_1 \neq 0$ satisfy $X^2Z^2 + aY^2Z^2 = X^2Y^2 + dZ^4$. Then, $(X_1 : Y_1 : Z_1) + (X_2 : Y_2 : Z_2) = (X_3 : Y_3 : Z_3)$ where

$$\begin{aligned} X_3 &= Z_1Z_2(X_1X_2 + aY_1Y_2)(X_1Y_1Z_2^2 - Z_1^2X_2Y_2) \\ Y_3 &= Z_1Z_2(X_1Y_2 - Y_1X_2)(X_1Y_1Z_2^2 + Z_1^2X_2Y_2) \\ Z_3 &= (X_1Y_1Z_2^2 - Z_1^2X_2Y_2)(X_1Y_1Z_2^2 + Z_1^2X_2Y_2) \end{aligned} \tag{5.5}$$

assuming $X_3Y_3Z_3 \neq 0$. These formulae are obtained from (4.37) and (4.38).

Evaluating (5.5) takes $11\mathbf{M} + 2\mathbf{D} + 9\mathbf{a}$;

$$A \leftarrow X_1 \cdot Z_2, \quad B \leftarrow Y_1 \cdot Z_2, \quad C \leftarrow Z_1 \cdot X_2, \quad D \leftarrow Z_1 \cdot Y_2, \quad E \leftarrow A \cdot B, \quad F \leftarrow C \cdot D,$$
$$G \leftarrow E + F, \quad H \leftarrow E - F, \quad X_3 \leftarrow ((A + aD) \cdot (B + C) - E - aF) \cdot H,$$
$$Y_3 \leftarrow ((A - C) \cdot (B + D) - H) \cdot G, \quad Z_3 \leftarrow G \cdot H.$$

Additionally, if $Z_2 = 1$ then it takes $9\mathbf{M} + 2\mathbf{D} + 9\mathbf{a}$.

If $a = 1$ then the dedicated addition takes $11\mathbf{M} + 8\mathbf{a}$ by computing X_3 as $((A + D) \cdot (B + C) - G) \cdot H$. Additionally, if $Z_2 = 1$ then it takes $9\mathbf{M} + 8\mathbf{a}$.

If $a = -1$ then the dedicated addition takes $11\mathbf{M} + 8\mathbf{a}$ by computing X_3 as $((A - D) \cdot (B + C) - H) \cdot H$. Additionally, if $Z_2 = 1$ then it takes $9\mathbf{M} + 8\mathbf{a}$.

Unified addition in $\mathcal{E}^i$ Alternatively, $(X_1 : Y_1 : Z_1) + (X_2 : Y_2 : Z_2) = (X_3 : Y_3 : Z_3)$ where

$$\begin{aligned} X_3 &= (X_1X_2 - aY_1Y_2)(X_1Y_1X_2Y_2 + dZ_1^2Z_2^2) \\ Y_3 &= (X_1Y_2 + Y_1X_2)(X_1Y_1X_2Y_2 - dZ_1^2Z_2^2) \\ Z_3 &= Z_1Z_2(X_1Y_2 + Y_1X_2)(X_1X_2 - aY_1Y_2) \end{aligned} \tag{5.6}$$

assuming $X_3Y_3Z_3 \neq 0$. These formulae are obtained from (4.39) and (4.40). In the case $a = 1$ and d a non-square in $\mathbb{K}$, handling the exceptions is described in [BL07c].

Evaluating (5.6) takes $9\mathbf{M} + 1\mathbf{S} + 2\mathbf{D} + 7\mathbf{a}$ in [BBJ+08];

$$A \leftarrow Z_1 \cdot Z_2, \quad B \leftarrow dA^2, \quad C \leftarrow X_1 \cdot X_2, \quad D \leftarrow Y_1 \cdot Y_2, \quad E \leftarrow C \cdot D, \quad H \leftarrow C - aD,$$
$$F \leftarrow (X_1 + Y_1) \cdot (X_2 + Y_2) - C - D, \quad X_3 \leftarrow (E + B) \cdot H, \quad Y_3 \leftarrow (E - B) \cdot F,$$
$$Z_3 \leftarrow A \cdot H \cdot F.$$

Additionally, if $Z_2 = 1$ then it takes $8\mathbf{M} + 1\mathbf{S} + 2\mathbf{D} + 7\mathbf{a}$.

If $a = 1$ then the unified addition takes $9\mathbf{M} + 1\mathbf{S} + 1\mathbf{D} + 7\mathbf{a}$. Additionally, if $Z_2 = 1$ then it takes $8\mathbf{M} + 1\mathbf{S} + 1\mathbf{D} + 7\mathbf{a}$.

If $a = -1$ then the unified addition takes $9\mathbf{M} + 1\mathbf{S} + 1\mathbf{D} + 6\mathbf{a}$ by replacing $H \leftarrow C - aD$ with $H \leftarrow C + D$ and then computing F as $(X_1 + Y_1) \cdot (X_2 + Y_2) - H$. Additionally, if $Z_2 = 1$ then it takes $8\mathbf{M} + 1\mathbf{S} + 1\mathbf{D} + 6\mathbf{a}$.

5.1.3 Extended homogeneous projective coordinates, $\mathcal{E}^e$

In this system, each point (x, y) on $ax^2 + y^2 = 1 + dx^2y^2$ is represented with the quadruplet $(X : Y : T : Z)$ which corresponds to the affine point $(X/Z, Y/Z)$ with $Z \neq 0$. These

quadruplets satisfy the homogeneous projective equation $aX^2Z^2 + Y^2Z^2 = Z^4 + dX^2Y^2$. The auxiliary coordinate is defined by $T = XY/Z$. For all nonzero $\lambda \in \overline{\mathbb{K}}$, $(X : Y : T : Z) = (\lambda X : \lambda Y : \lambda T : \lambda Z)$. The identity element is represented by $(0 : 1 : 0 : 1)$. The negative of $(X : Y : T : Z)$ is $(-X : Y : -T : Z)$. This coordinate system is denoted by $\mathcal{E}^e$ throughout the text. Given $(X : Y : Z)$ in $\mathcal{E}$ passing to $\mathcal{E}^e$ can be performed in $3\mathbf{M} + 1\mathbf{S}$ by computing (XZ, YZ, XY, Z^2). Given $(X : Y : T : Z)$ in $\mathcal{E}^e$ passing to $\mathcal{E}$ is cost-free by simply ignoring T.

Doubling in $\mathcal{E}^e$ Let $(X_1 : Y_1 : T_1 : Z_1)$ with $Z_1 \neq 0$ and $T_1 = X_1Y_1/Z_1$ satisfy $aX^2Z^2 + Y^2Z^2 = Z^4 + dX^2Y^2$. Then $[2](X_1 : Y_1 : T_1 : Z_1) = (X_3 : Y_3 : T_3 : Z_3)$ where

$$\begin{aligned} X_3 &= 2X_1Y_1(2Z_1^2 - Y_1^2 - aX_1^2), \\ Y_3 &= (Y_1^2 - aX_1^2)(Y_1^2 + aX_1^2), \\ T_3 &= 2X_1Y_1(Y_1^2 - aX_1^2), \\ Z_3 &= (Y_1^2 + aX_1^2)(2Z_1^2 - Y_1^2 - aX_1^2) \end{aligned} \tag{5.7}$$

assuming $Z_3 \neq 0$. These formulae are obtained from (4.35) and (4.36). It can be deduced from [BBJ+08] that $Z_3 \neq 0$ if a is a square in $\mathbb{K}$ and d is not a square in $\mathbb{K}$. Also by Lemma 4.4.7, $Z_3 \neq 0$ if $(X_1 : Y_1 : T_1 : Z_1)$ is of odd order.

Evaluating (5.7) takes $4\mathbf{M} + 4\mathbf{S} + 1\mathbf{D} + 7\mathbf{a}$;

$$\begin{gathered} A \leftarrow X_1^2, \quad B \leftarrow Y_1^2, \quad C \leftarrow 2Z_1^2, \quad D \leftarrow aA, \quad E \leftarrow B + D, \quad F \leftarrow B - D, \\ G \leftarrow C - E, \quad H \leftarrow (X_1 + Y_1)^2 - A - B, \quad X_3 \leftarrow G \cdot H, \quad Y_3 \leftarrow E \cdot F, \quad T_3 \leftarrow F \cdot H, \\ Z_3 \leftarrow E \cdot G. \end{gathered}$$

This algorithm is essentially the same as the $3\mathbf{M} + 4\mathbf{S} + 1\mathbf{D}$ point doubling in [BBJ+08]. The extra multiplication $E \cdot H$ is required to generate the auxiliary coordinate T_3.

If $a = 1$ then the doubling takes $4\mathbf{M} + 4\mathbf{S} + 6\mathbf{a}$ by first removing $D \leftarrow aA$ and then replacing $E \leftarrow B + D$, $F \leftarrow B - D$, $H \leftarrow (X_1 + Y_1)^2 - A - B$ with $E \leftarrow B + A$, $F \leftarrow B - A$, $H \leftarrow (X_1 + Y_1)^2 - E$, respectively.

If $a = -1$ then the doubling again takes $4\mathbf{M} + 4\mathbf{S} + 6\mathbf{a}$ by first removing $D \leftarrow aA$ and then replacing $E \leftarrow B + D$, $F \leftarrow B - D$, $H \leftarrow (X_1 + Y_1)^2 - A - B$ with $E \leftarrow B - A$, $F \leftarrow B + A$, $H \leftarrow (X_1 + Y_1)^2 - F$, respectively.

Dedicated addition in $\mathcal{E}^e$ Further let $(X_2 : Y_2 : T_2 : Z_2)$ with $Z_2 \neq 0$ and $T_2 = X_2Y_2/Z_2$ satisfy $aX^2Z^2 + Y^2Z^2 = Z^4 + dX^2Y^2$. Then, $(X_1 : Y_1 : T_1 : Z_1) + (X_2 : Y_2 : T_2 : Z_2) = (X_3 : Y_3 : T_3 : Z_3)$ where

$$\begin{aligned} X_3 &= (X_1Y_2 - Y_1X_2)(T_1Z_2 + Z_1T_2), \\ Y_3 &= (Y_1Y_2 + aX_1X_2)(T_1Z_2 - Z_1T_2), \\ T_3 &= (T_1Z_2 - Z_1T_2)(T_1Z_2 + Z_1T_2), \\ Z_3 &= (X_1Y_2 - Y_1X_2)(Y_1Y_2 + aX_1X_2) \end{aligned} \tag{5.8}$$

assuming $Z_3 \neq 0$. These formulae are obtained from (4.37) and (4.38). By Lemma 4.4.4, $Z_3 \neq 0$ if $(X_1 : Y_1 : T_1 : Z_1) \neq (X_2 : Y_2 : T_2 : Z_2)$ and both summands are of odd order.

Evaluating (5.8) takes $9\mathbf{M} + 1\mathbf{D} + 7\mathbf{a}$;

$$A \leftarrow X_1 \cdot X_2, \quad B \leftarrow Y_1 \cdot Y_2, \quad C \leftarrow Z_1 \cdot T_2, \quad D \leftarrow T_1 \cdot Z_2, \quad E \leftarrow D + C,$$
$$F \leftarrow (X_1 - Y_1) \cdot (X_2 + Y_2) + B - A, \quad G \leftarrow B + aA, \quad H \leftarrow D - C, \quad X_3 \leftarrow E \cdot F,$$
$$Y_3 \leftarrow G \cdot H, \quad Z_3 \leftarrow F \cdot G, \quad T_3 \leftarrow E \cdot H.$$

Additionally, if $Z_2 = 1$ then it takes $8\mathbf{M} + 1\mathbf{D} + 7\mathbf{a}$.

If $a = 1$ then the dedicated addition takes $9\mathbf{M} + 7\mathbf{a}$ in the usual sense. Additionally, if $Z_2 = 1$ then it takes $8\mathbf{M} + 7\mathbf{a}$.

If $a = -1$ then the dedicated addition takes $8\mathbf{M} + 10\mathbf{a}$ by trading $1\mathbf{M}$ with $3\mathbf{a}$;

$$A \leftarrow (Y_1 - X_1) \cdot (Y_2 + X_2), \quad B \leftarrow (Y_1 + X_1) \cdot (Y_2 - X_2), \quad C \leftarrow 2Z_1 \cdot T_2,$$
$$D \leftarrow 2T_1 \cdot Z_2, \quad E \leftarrow D + C, \quad F \leftarrow B - A, \quad G \leftarrow B + A, \quad H \leftarrow D - C,$$
$$X_3 \leftarrow E \cdot F, \quad Y_3 \leftarrow G \cdot H, \quad T_3 \leftarrow E \cdot H, \quad Z_3 \leftarrow F \cdot G.$$

The optimization that leads to the removal of the extra multiplication is similar to the optimizations in [Mon87] and [BL07a, add-2007-bl-4]. Additionally, if $Z_2 = 1$ then it takes $7\mathbf{M} + 10\mathbf{a}$.

Unified addition in $\mathcal{E}^e$ Alternatively, $(X_1 : Y_1 : T_1 : Z_1) + (X_2 : Y_2 : T_2 : Z_2) = (X_3 : Y_3 : T_3 : Z_3)$ where

$$\begin{aligned} X_3 &= (X_1Y_2 + Y_1X_2)(Z_1Z_2 - dT_1T_2), \\ Y_3 &= (Y_1Y_2 - aX_1X_2)(Z_1Z_2 + dT_1T_2), \\ T_3 &= (X_1Y_2 + Y_1X_2)(Y_1Y_2 - aX_1X_2), \\ Z_3 &= (Z_1Z_2 - dT_1T_2)(Z_1Z_2 + dT_1T_2) \end{aligned} \tag{5.9}$$

assuming $Z_3 \neq 0$. These formulae are obtained from (4.39) and (4.40). It can be deduced from [BBJ+08] that $Z_3 \neq 0$ if a is a square in $\mathbb{K}$ and d is not a square in $\mathbb{K}$. Also by Lemma 4.4.7, $Z_3 \neq 0$ if $(X_1 : Y_1 : T_1 : Z_1)$ is of odd order.

Evaluating (5.9) takes $9\mathbf{M} + 2\mathbf{D} + 7\mathbf{a}$;

$$A \leftarrow X_1 \cdot X_2, \quad B \leftarrow Y_1 \cdot Y_2, \quad C \leftarrow dT_1 \cdot T_2, \quad D \leftarrow Z_1 \cdot Z_2,$$
$$E \leftarrow (X_1 + Y_1) \cdot (X_2 + Y_2) - A - B, \quad F \leftarrow D - C, \quad G \leftarrow D + C, \quad H \leftarrow B - aA,$$
$$X_3 \leftarrow E \cdot F, \quad Y_3 \leftarrow G \cdot H, \quad Z_3 \leftarrow F \cdot G, \quad T_3 \leftarrow E \cdot H.$$

Additionally, if $Z_2 = 1$ then it takes $8\mathbf{M} + 2\mathbf{D} + 7\mathbf{a}$.

If $a = 1$ then the unified addition takes $9\mathbf{M} + 1\mathbf{D} + 7\mathbf{a}$ in the usual sense. Additionally, if $Z_2 = 1$ then it takes $8\mathbf{M} + 2\mathbf{D} + 7\mathbf{a}$.

If $a = -1$ then the dedicated addition takes $8\mathbf{M} + 1\mathbf{D} + 9\mathbf{a}$;

$$A \leftarrow (Y_1 - X_1) \cdot (Y_2 - X_2), \quad B \leftarrow (Y_1 + X_1) \cdot (Y_2 + X_2), \quad C \leftarrow (2d)T_1 \cdot T_2,$$
$$D \leftarrow 2Z_1 \cdot Z_2, \quad E \leftarrow B - A, \quad F \leftarrow D - C, \quad G \leftarrow D + C, \quad H \leftarrow B + A,$$
$$X_3 \leftarrow E \cdot F, \quad Y_3 \leftarrow G \cdot H, \quad Z_3 \leftarrow F \cdot G, \quad T_3 \leftarrow E \cdot H.$$

The optimization that leads to the removal of the extra multiplication is similar to the optimizations in [Mon87] and [BL07a, add-2007-bl-4]. Additionally, if $Z_2 = 1$ then it takes $7\mathbf{M} + 1\mathbf{D} + 9\mathbf{a}$.

More Formulae Using the presented two different addition formulae for computing x_3 and another two for y_3 in affine coordinates, it is possible to produce hybrid addition formulae from

(4.37), (4.38), (4.39), and (4.40). The hybrid formulae are given by

$$(x_1, y_1) + (x_2, y_2) = \left(\frac{x_1y_1 + x_2y_2}{y_1y_2 + ax_1x_2}, \frac{y_1y_2 - ax_1x_2}{1 - dx_1y_1x_2y_2}\right) = (x_3, y_3), \tag{5.10}$$

$$(x_1, y_1) + (x_2, y_2) = \left(\frac{x_1y_2 + y_1x_2}{1 + dx_1y_1x_2y_2}, \frac{x_1y_1 - x_2y_2}{x_1y_2 - y_1x_2}\right) = (x_3, y_3). \tag{5.11}$$

$\mathcal{E}^e$ analogs of (5.10) and (5.11) lead to similar speeds and are excluded from further discussion.

Parameter selection Choosing curve constants with extremely small sizes or extremely low (or high) hamming weight can be used to eliminate the computational overhead of a field multiplication to some extent. For instance see [BJ03b], [BJ03a], [DIK06]. See also [BBJ+08, §7] for an alternative strategy for the selection of constants. When using $\mathcal{E}^e$ the situation is even better if $a = -1$; $1\mathbf{M}+1\mathbf{D}$ can be saved rather than just $1\mathbf{D}$. Consider a twisted Edwards curve given by

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

The map $(x, y) \mapsto (x/\sqrt{-a}, y)$ defines the curve,

$$-x^2 + y^2 = 1 + (-d/a)x^2y^2.$$

This map can be constructed if $-a$ is a square in $\mathbb{K}$. It is worth pointing out here that the curve $-x^2+y^2 = 1+(-d/a)x^2y^2$ corresponds to the Edwards curve $x^2+y^2 = 1+(d/a)x^2y^2$ via the map $(x, y) \mapsto (ix, y)$ if $i \in \mathbb{K}$ with $i^2 = -1$. For such curves the new addition algorithms in extended coordinates improve upon the $10\mathbf{M} + 1\mathbf{S} + 1\mathbf{D}$ point addition algorithm given in [BL07a, add-2007-bl-4] for $a = 1$.

Further benefits of the extended coordinates are summarized as follows.

Extreme optimizations On some architectures additions in $\mathbb{K}$ may not be negligible. Furthermore, it is always better if multiplications by curve constants can be eliminated. Otherwise a programmer is often forced to select curve constants of tiny sizes still having $\mathbf{D}$ as an overhead. In extended coordinates this problem can be solved. For instance, consider the $8\mathbf{M} + 1\mathbf{D} + 9\mathbf{a}$ unified addition algorithm with $a = -1$. Assuming that $(X_2 : Y_2 : T_2 : Z_2)$ is the base point one can cache $(Y_2 - X_2)$, $(Y_2 + X_2)$, $(2Z_2)$, and $(2dT_2)$ after the first addition to save $1\mathbf{D}+3\mathbf{a}$ in each of the consecutive re-additions. The unified re-additions will then take only $8\mathbf{M}+5\mathbf{a}$. The extra space required for these cached values can also be eliminated. Observe that none of the coordinates in $(X_2 : Y_2 : T_2 : Z_2)$ are ever accessed once all of the cached values are employed in unified re-additions. Therefore the cached values can be *destructively* stored within the registers holding X_2, Y_2, T_2, and Z_2. (It is interesting to note that the original coordinates X_2, Y_2, and Z_2 can still be recovered from the cached $(Y_2 - X_2)$, $(Y_2 + X_2)$, $(2Z_2)$ by linear operations if needed at some other time.) This is also common for the entries in a precomputed table of small multiples of the base point, i.e. the coordinates of each point in the lookup table should be altered in a similar way. Since the additions in this lookup table are accessed frequently the extra effort in computing $(Y_2 - X_2)$, $(Y_2 + X_2)$, $(2dT_2)$, and $(2Z_2)$

will be recovered starting from the second addition. Each subsequent access to this algorithm will then save $1\mathbf{D} + 4\mathbf{a}$. This approach eliminates the sensitivity of the algorithm to changes in $\mathbf{D}$. In addition, this approach also works for the dedicated addition algorithm with $a = -1$ in §5.1.3. This approach cannot be applied in full generality to the other algorithms. For instance, recall the $10\mathbf{M} + 1\mathbf{S} + 2\mathbf{D} + 7\mathbf{a}$ unified addition algorithm in homogeneous projective coordinates given in §5.1.1;

$$A \leftarrow Z_1 \cdot Z_2, \quad B \leftarrow A^2, \quad C \leftarrow X_1 \cdot X_2, \quad D \leftarrow Y_1 \cdot Y_2, \quad E \leftarrow dC \cdot D, \quad F \leftarrow B - E,$$
$$G \leftarrow B + E, \quad X_3 \leftarrow A \cdot F \cdot ((X_1 + Y_1) \cdot (X_2 + Y_2) - C - D), \quad Y_3 \leftarrow A \cdot G \cdot (D - aC),$$
$$Z_3 \leftarrow F \cdot G.$$

Here C and D are used for the computation of X_3 and Y_3. In addition, C and D depend on both summands. Therefore $dC \cdot D$ cannot be cached in a similar fashion for the addition algorithms in extended coordinates.

Parallelism in $\mathcal{E}^e$ with $a = -1$ The proposed extended coordinate system possesses a high level of parallelism if $a = -1$. Despite the computational overhead of the additional coordinate, the increase in the number of coordinates comes with lower total degree formulae. Exploiting this property the unified addition (5.9) costs an effective $2\mathbf{M} + 1\mathbf{D} + 2\mathbf{a}$ using four processors;

Cost	Step	Processor 1	Processor 2	Processor 3	Processor 4
	1	$R_1 \leftarrow Y_1 - X_1$	$R_2 \leftarrow Y_2 - X_2$	$R_3 \leftarrow Y_1 + X_1$	$R_4 \leftarrow Y_2 + X_2$
1**M**	2	$R_5 \leftarrow R_1 \cdot R_2$	$R_6 \leftarrow R_3 \cdot R_4$	$R_7 \leftarrow T_1 \cdot T_2$	$R_8 \leftarrow Z_1 \cdot Z_2$
1**D**	3	*idle*	*idle*	$R_7 \leftarrow (2d)R_7$	$R_8 \leftarrow 2R_8$
	4	$R_1 \leftarrow R_6 - R_5$	$R_2 \leftarrow R_8 - R_7$	$R_3 \leftarrow R_8 + R_7$	$R_4 \leftarrow R_6 + R_5$
1**M**	5	$X_3 \leftarrow R_1 \cdot R_2$	$Y_3 \leftarrow R_3 \cdot R_4$	$T_3 \leftarrow R_1 \cdot R_4$	$Z_3 \leftarrow R_2 \cdot R_3$

Note that the third step can be eliminated if $2Z_2$ and kT_2 are precomputed. Then $R_7 \leftarrow T_1 \cdot T_2$ is computed as $R_7 \leftarrow T_1 \cdot (kT_2)$ and $R_8 \leftarrow Z_1 \cdot Z_2$ as $R_8 \leftarrow Z_1 \cdot (2Z_2)$. The updated cost is then an effective $2\mathbf{M} + 2\mathbf{a}$.

The dedicated addition (5.8) costs an effective $2\mathbf{M} + 3\mathbf{a}$ using four processors;

Cost	Step	Processor 1	Processor 2	Processor 3	Processor 4
	1	$R_1 \leftarrow Y_1 - X_1$	$R_2 \leftarrow Y_2 + X_2$	$R_3 \leftarrow Y_1 + X_1$	$R_4 \leftarrow Y_2 - X_2$
1**M**	2	$R_5 \leftarrow R_1 \cdot R_2$	$R_6 \leftarrow R_3 \cdot R_4$	$R_7 \leftarrow Z_1 \cdot T_2$	$R_8 \leftarrow T_1 \cdot Z_2$
	3	*idle*	*idle*	$R_7 \leftarrow 2R_7$	$R_8 \leftarrow 2R_8$
	4	$R_1 \leftarrow R_8 + R_7$	$R_2 \leftarrow R_6 - R_5$	$R_3 \leftarrow R_6 + R_5$	$R_4 \leftarrow R_8 - R_7$
1**M**	5	$X_3 \leftarrow R_1 \cdot R_2$	$Y_3 \leftarrow R_3 \cdot R_4$	$T_3 \leftarrow R_1 \cdot R_4$	$Z_3 \leftarrow R_2 \cdot R_3$

Similarly the third step can be eliminated if $2Z_2$ and $2T_2$ are precomputed. Then $R_7 \leftarrow Z_1 \cdot T_2$ is computed as $R_7 \leftarrow Z_1 \cdot (2T_2)$ and $R_8 \leftarrow T_1 \cdot Z_2$ as $R_8 \leftarrow T_1 \cdot (2Z_2)$. The updated cost is then an effective $2\mathbf{M} + 2\mathbf{a}$.

The doubling (5.7) costs an effective $1\mathbf{M} + 1\mathbf{S} + 3\mathbf{a}$ using four processors;

Cost	Step	Processor 1	Processor 2	Processor 3	Processor 4
	1	*idle*	*idle*	*idle*	$R_1 \leftarrow X_1 + Y_1$
1**S**	2	$R_2 \leftarrow X_1^2$	$R_3 \leftarrow Y_1^2$	$R_4 \leftarrow Z_1^2$	$R_5 \leftarrow R_1^2$
	3	$R_6 \leftarrow R_2 + R_3$	$R_7 \leftarrow R_2 - R_3$	$R_4 \leftarrow 2R_4$	*idle*
	4	*idle*	$R_1 \leftarrow R_4 + R_7$	*idle*	$R_2 \leftarrow R_6 - R_5$
1**M**	5	$X_3 \leftarrow R_1 \cdot R_2$	$Y_3 \leftarrow R_6 \cdot R_7$	$T_3 \leftarrow R_2 \cdot R_6$	$Z_3 \leftarrow R_1 \cdot R_7$

From these algorithms the 2 processor case can be deduced by sequentially carrying the workload of processors 3 and 4 on processors 1 and 2.

5.1.4 Mixed homogeneous projective coordinates, $\mathcal{E}^x$

The extended coordinates allow much faster additions. For addition intensive computations using extended coordinates provides the fastest arithmetic. However, the doubling in extended coordinates comes with an extra multiplication. For a full speed implementation this extra computation should be eliminated. This section aims to solve this problem.

In [CMO98], Cohen *et al.* introduced the modified Jacobian coordinates and studied other systems in the literature, namely affine, projective, Jacobian, and Chudnovsky Jacobian coordinates. To gain better timings they proposed a technique of carefully mixing these coordinates. A similar approach is followed in this subsection. Note, the notations $\mathcal{E}$ and $\mathcal{E}^e$ follow the notation introduced in [CMO98]. On twisted Edwards curves, the speed of scalar multiplications which involve point doublings can be increased by mixing $\mathcal{E}^e$ with $\mathcal{E}$. The following technique replaces (slower) doublings in $\mathcal{E}^e$ with (faster) doublings in $\mathcal{E}$. In the execution of a scalar multiplication:

(i) If a point doubling is followed by another point doubling, use $\mathcal{E} \leftarrow 2\mathcal{E}$, i.e. point doubling in homogeneous projective coordinates.

(ii) If a point doubling is followed by a point addition, use

1. $\mathcal{E}^e \leftarrow 2\mathcal{E}$ for the point doubling step, i.e. point doubling in homogeneous projective coordinates by producing output coordinates in extended homogeneous projective coordinates; followed by,
2. $\mathcal{E} \leftarrow \mathcal{E}^e + \mathcal{E}^e$ for the point addition step, i.e. point addition in extended homogeneous projective coordinates by producing output coordinates in homogeneous projective coordinates.

$\mathcal{E} \leftarrow 2\mathcal{E}$ is performed using a $3\mathbf{M} + 4\mathbf{S} + 1\mathbf{D}$ doubling algorithm in [BBJ+08], see §5.1.1. $\mathcal{E}^e \leftarrow 2\mathcal{E}$ is performed using (5.7). In §5.1.3 it was noted that passing from $(X : Y : Z)$ to $(X : Y : T : Z)$ (i.e. passing from $\mathcal{E}$ to $\mathcal{E}^e$) can be performed in $3\mathbf{M} + 1\mathbf{S}$. From this, it might seem at the first glance that computing $\mathcal{E}^e \leftarrow 2\mathcal{E}$ will be more costly than expected. However, the doubling algorithm for (5.7) does not use the input T_1. So, it can be used for $\mathcal{E}^e \leftarrow 2\mathcal{E}$ without modification requiring only $4\mathbf{M} + 4\mathbf{S} + 1\mathbf{D}$. This operation is followed by $\mathcal{E} \leftarrow \mathcal{E}^e + \mathcal{E}^e$ which can be performed with either (5.8) or (5.9). Observe that one field multiplication can be saved by not computing T_3. This can be regarded as a remedy to the extra field multiplication

which appears in $\mathcal{E}^e \leftarrow 2\mathcal{E}$ to generate the intermediate coordinate T_3. For instance, if (5.8) is used with $a = -1$, the addition step effectively costs $(4\mathbf{M}+4\mathbf{S})+(8\mathbf{M})-(3\mathbf{M}+4\mathbf{S})-(1\mathbf{M}) = 8\mathbf{M}$ omitting the operation count for $\mathbf{a}$. It is also convenient to consider $\mathcal{E}^e \leftarrow 2\mathcal{E}$ followed by $\mathcal{E} \leftarrow \mathcal{E}^e + \mathcal{E}^e$ as a single composite operation as $\mathcal{E} \leftarrow 2\mathcal{E} + \mathcal{E}^e$ where $\mathcal{E}^e$ is the base point. See [ELM03] for a similar approach in affine Weierstrass coordinates.

5.1.5 Comparison and remarks

In §5.1.1 and §5.1.2 inversion-free arithmetic for twisted Edwards form is studied in coordinate systems from the literature. These subsections have reviewed the previous results and have added new and potentially more efficient results to the current body of knowledge. In particular, the dedicated addition formulae are derived for the point representations in $\mathcal{E}$ and $\mathcal{E}^i$ using the affine addition law (4.37) and (4.38) from Chapter 4. A useful feature of the dedicated addition formulae is that the curve constant d does not appear in computations thus eliminating the necessity of selecting d from a very restricted set of elements in $\mathbb{K}$. Most notably, the dedicated addition in $\mathcal{E}$ with $a = \pm 1$ and $Z_2 = 1$ takes only $9\mathbf{M} + 8\mathbf{a}$ improving upon the corresponding $9\mathbf{M} + 1\mathbf{S} + 1\mathbf{D} + 7\mathbf{a}$ algorithm in [BL07b]. The new algorithm has already been integrated into a very recent *ECM* (Elliptic Curve Method) implementation on graphic cards introduced at *EUROCRYPT'09* by Bernstein *et. al.*, see [BRCMC$^+$09].

In §5.1.3 a new point representation, $\mathcal{E}^e$, for twisted Edwards form is introduced. In this system each point is represented with four coordinates. With the additional coordinate the degrees of addition formulae decrease and consequently the dedicated and the unified addition can be performed faster than in $\mathcal{E}$ and $\mathcal{E}^i$. In particular, the dedicated addition takes only $9\mathbf{M} + 1\mathbf{D} + 7\mathbf{a}$ and the unified addition takes only $9\mathbf{M} + 2\mathbf{D} + 7\mathbf{a}$. Surprisingly, §5.1.3 shows that setting/rescaling $a = -1$ is more advantageous than setting $a = 1$ because this selection removes an additional multiplication. In particular, the dedicated addition takes only $8\mathbf{M}+10\mathbf{a}$ and the unified addition takes only $8\mathbf{M}+1\mathbf{D}+9\mathbf{a}$ if $a = -1$. With some extreme optimizations these costs can be further decreased to $8\mathbf{M} + 6\mathbf{a}$ without needing additional space, see §5.1.3.

The drawback of $\mathcal{E}^e$ is the slower doublings which take $4\mathbf{M} + 4\mathbf{S} + 1\mathbf{D} + 7\mathbf{a}$ for arbitrary a and d. This is slower by $1\mathbf{M}$ in comparison to $3\mathbf{M}+4\mathbf{S}+1\mathbf{D}+7\mathbf{a}$ doubling algorithm in $\mathcal{E}$. This problem is remedied in §5.1.4 which introduces the mixed coordinates, $\mathcal{E}^x$. This system uses faster doublings in $\mathcal{E}$ and faster additions in $\mathcal{E}^e$ without causing any computational overhead while switching between $\mathcal{E}$ and $\mathcal{E}^e$.

Various operation counts in $\mathcal{E}$, $\mathcal{E}^i$, $\mathcal{E}^e$, and $\mathcal{E}^x$ are summarized in Table 5.1 and Table 5.2.

Table 5.2 immediately shows the advantage of using $\mathcal{E}^x$ with $a = -1$. However, some pitfalls should be highlighted here. There are several elliptic curve applications each having its own memory requirement and pattern of access to elliptic curve operations. Therefore, a particular point representation may not be dominantly the fastest for all applications. On the other hand, the fastest system is often easy to determine once the environmental assumptions are made clear.

For instance, some applications such as the batch signature verification perform several scalar multiplications simultaneously, see [dR94]. These applications tend to utilize all fast-access-memory available on the hardware. In such applications using $\mathcal{E}$ might be preferable

Table 5.1: Operation counts for twisted Edwards form in different coordinate systems.

System	DBL	ADD
$\mathcal{E}$	3**M**+4**S**+1**D**+7**a**, [BKL09]	10**M**+1**S**+2**D**+7**a**, unified, [BKL09]
		11**M** +1**D**+8**a**, dedicated
$\mathcal{E}^i$	3**M**+4**S**+2**D**+6**a**, [BKL09]	9**M**+1**S**+2**D**+7**a**, unified, [BKL09]
		11**M** +1**D**+8**a**, dedicated
$\mathcal{E}^e$	4**M**+4**S**+1**D**+7**a**	9**M** +2**D**+7**a**, unified
		9**M** +1**D**+7**a**, dedicated
$\mathcal{E}^x$	3**M**+4**S**+1**D**+7**a**	9**M** +2**D**+7**a**, unified
		9**M** +1**D**+7**a**, dedicated

Table 5.2: Operation counts for twisted Edwards form with $a = \pm 1$ in different coordinate systems.

System	DBL	ADD
$\mathcal{E}$ $(a = \pm 1)$	3**M**+4**S** +7**a**, [BKL09]	10**M**+1**S**+1**D**+ 7**a**, unified, [BKL09]
		11**M** **D**+ 8**a**, dedicated
$\mathcal{E}^i$ $(a = \pm 1)$	3**M**+4**S**+1**D**+6**a**, [BKL09]	9**M**+1**S**+1**D**+ 7**a**, unified, [BKL09]
		11**M** + 8**a**, dedicated
$\mathcal{E}^e$ $(a = 1)$	4**M**+4**S** +7**a**	9**M** +1**D**+ 7**a**, unified
		9**M** + 7**a**, dedicated
$\mathcal{E}^e$ $(a = -1)$	4**M**+4**S** +7**a**	8**M** + 9**a**, unified
		8**M** +10**a**, dedicated
$\mathcal{E}^x$ $(a = 1)$	3**M**+4**S** +7**a**	9**M** +1**D**+ 7**a**, unified
		9**M** + 7**a**, dedicated
$\mathcal{E}^x$ $(a = -1)$	3**M**+4**S** +7**a**	8**M** + 9**a**, unified
		8**M** +10**a**, dedicated

since each point is represented with three coordinates rather than four, cf. [BCC+09]. For batch applications, $\mathcal{E}^i$ can also be interesting if doublings are not accessed, or rarely accessed, or multiplication by d in the doubling can be carried out very efficiently. However, on this system the implementor is always forced to handle some special cases separately, see §5.1.2. These special routines require some some hidden costs such as zero checks causing more branches within the execution.

Some other applications such as scalar multiplication with fixed base point can be implemented using a lot of memory. In such applications several multiples of the base point are precomputed and stored in affine coordinates in a large lookup table, cf. [HMV03, §3.3.2]. Note that the doublings are not frequently accessed and additions with $Z_2 = 1$ dominate the computation. For this case, the proposed $\mathcal{E}^e$ is the most preferable system. First of all note that $T_2 = X_2Y_2/Z_2$ can be computed on the fly requiring only one extra multiplication since

$Z_2 = 1$. This extra computation saves memory since only two coordinates (i.e. X_2 and Y_2) per entry are then needed to be stored in the lookup table. Now, $\mathcal{E}^e$ consumes the same amount of memory as $\mathcal{E}$ and $\mathcal{E}^i$. Moreover, the new dedicated addition with $a = -1$ and $Z_2 = 1$ takes $8\mathbf{M} + 10\mathbf{a}$ in $\mathcal{E}^x$ including the on-the-fly computation of T_2. This is always faster than all other addition algorithms with $Z_2 = 1$ in $\mathcal{E}$ or $\mathcal{E}^i$. Theoretically, the $\mathcal{E}^x$ system can save time over $\mathcal{E}^e$ since doublings are performed $1\mathbf{M}$ faster in comparison to $\mathcal{E}^e$. However, it was already mentioned that there is a very limited access to doublings. In Chapter 6 it will be shown by computer experiments that using $\mathcal{E}^e$ is better than using $\mathcal{E}^x$ when the base point is fixed and precomputation is performed.

Other applications which do not batch scalar multiplications use an optimum amount of memory and *cannot* utilize the rest of the fast-access-memory single instance of Diffie-Hellman key pair computation on modern processors such as Pentium 4 or x86-64. If the extra space required by the T-coordinate is available and only unified addition is being accessed then using the proposed $\mathcal{E}^e$ system with $a = -1$ is by far the best choice because this system provides the fastest additions, see Table 5.2. If the extra space required by the T-coordinate is available and the doubling is used for identical summands then the best choice this time is the proposed $\mathcal{E}^x$ system which benefits from fast doublings in $\mathcal{E}$ and fast additions in $\mathcal{E}^e$. For this case, it is up to the implementor whether or not to use the new dedicated addition algorithm or the new unified addition algorithm. Note, if the extreme optimizations proposed in §5.1.3 are not employed then the new dedicated addition algorithm is faster than the new unified addition algorithm.

Finally, it was noted in §5.1.3 that the proposed $\mathcal{E}^e$ system with $a = -1$ allows a high level of parallelism. A dedicated addition takes $4\mathbf{M} + 5\mathbf{a}$ on two processors and $2\mathbf{M} + 3\mathbf{a}$ on four processors. After extreme optimizations from the same subsection a dedicated re-addition takes $4\mathbf{M} + 4\mathbf{a}$ on two processors and $2\mathbf{M} + 3\mathbf{a}$ on four processors. Similarly, a unified addition takes $4\mathbf{M} + 1\mathbf{D} + 4\mathbf{a}$ on two processors and $2\mathbf{M} + 1\mathbf{D} + 2\mathbf{a}$ on four processors. After extreme optimizations as described in the same subsection a unified re-addition takes $4\mathbf{M} + 4\mathbf{a}$ on two processors and $2\mathbf{M} + 2\mathbf{a}$ on four processors. In addition, a doubling takes $2\mathbf{M} + 2\mathbf{S} + 4\mathbf{a}$ on two processors and $1\mathbf{M} + 1\mathbf{S} + 3\mathbf{a}$ on four processors.

Literature notes Bernstein *et al.* introduced the shape $d_1(x + y) + d_2(x^2 + y^2) = (x + x^2)(y + y^2)$ and presented results on the arithmetic of these curves when $\mathrm{char}(\mathbb{K}) = 2$ in [BLR08]. These curve are named binary Edwards curves. Inversion-free Tate pairing computations using twisted Edwards form were studied by Das and Sarkar [DS08], Ionica and Joux [IJ08], and Arène *et al.* [ALNR09] in chronological order. Gailbraith *et al.* proposed an efficiently computable homomorphism on a large class of curves in [GLS09a]. They extended their method to the case of twisted Edwards curves in [GLS09b]. Very recently, Bernstein and Lange adapted differential addition formulae and doubling algorithms by Gaudry and Lubics [GL08], [GL09] to Edwards curves, see [BL07a].

5.2 Extended Jacobi quartic form

The recent advances in efficient arithmetic for twisted Edwards form also shed light on the arithmetic of extended Jacobi quartic form. The arithmetic of extended Jacobi quartic form is studied in three different coordinate systems in this section. Similar to the practice in twisted Edwards curves, homogeneous projective coordinates allows very fast doubling in just three coordinates however the additions are quite costly, see §5.2.1. This disadvantage is eliminated in extended homogeneous projective coordinates which allows very fast additions, see §5.2.2. However, this time the doublings become slower. Finally, the mixed coordinates solve the efficiency problem by using fast doublings in homogeneous projective coordinates and fast additions in extended homogeneous projective coordinates, see §5.2.3. Weighted projective coordinates are reviewed in §5.2.4. Conclusions are drawn in §5.2.5 together with pointers to the literature.

5.2.1 Homogeneous projective coordinates, $\mathcal{Q}$

Projective coordinates are used as basic tools in designing inversion-free algorithms. In the case of (extended) Jacobi quartic curves, homogeneous projective coordinates $(X : Y : Z)$ are considered for efficiency purposes for the first time in this thesis.

In this system, each point (x, y) on $y^2 = dx^4 + 2ax^2 + 1$ is represented with the triplet $(X : Y : Z)$ which corresponds to the affine point $(X/Z, Y/Z)$ with $Z \neq 0$. These triplets satisfy the homogeneous projective equation $Y^2Z^2 = dX^4 + 2aX^2Z^2 + Z^4$. For all nonzero $\lambda \in \overline{\mathbb{K}}$, $(X : Y : Z) = (\lambda X : \lambda Y : \lambda Z)$. The identity element is represented by $(0 : 1 : 1)$. The negative of $(X : Y : Z)$ is $(-X : Y : Z)$. This coordinate system is denoted by $\mathcal{Q}$ throughout the text.

Doubling in $\mathcal{Q}$ Let $(X_1 : Y_1 : Z_1)$ with $Z_1 \neq 0$ satisfy $Y^2Z^2 = dX^4 + 2aX^2Z^2 + Z^4$. Then $[2](X_1 : Y_1 : Z_1) = (X_3 : Y_3 : Z_3)$ where

$$\begin{aligned} X_3 &= 2X_1Y_1(2Z_1^2 - Y_1^2 + 2aX_1^2), \\ Y_3 &= 2Y_1^2(Y_1^2 - 2aX_1^2) - (2Z_1^2 - Y_1^2 + 2aX_1^2)^2, \\ Z_3 &= (2Z_1^2 - Y_1^2 + 2aX_1^2)^2 \end{aligned} \tag{5.12}$$

assuming $Z_3 \neq 0$. These formulae are obtained from (4.11) and (4.12). By Lemma 4.2.8, Z_3 is always nonzero if d is not a square in $\mathbb{K}$. In this case, it is also important to note that the points at infinity on the desingularization of an extended Jacobi quartic curve are not defined over $\mathbb{K}$. Also by Lemma 4.4.7, Z_3 is always nonzero if $(X_1 : Y_1 : Z_1)$ is of odd order regardless of the assumption on d (of course, $d(a^2 - d) \neq 0$).

These formulae do not depend on d. Therefore keeping d arbitrary has no effect on the cost of (5.12). Evaluating (5.12) takes $2\mathbf{M} + 5\mathbf{S} + 1\mathbf{D} + 8\mathbf{a}$;

$$\begin{gathered} A \leftarrow 2Z_1^2, \quad B \leftarrow X_1^2, \quad C \leftarrow Y_1^2, \quad D \leftarrow B + C, \quad E \leftarrow (X_1 + Y_1)^2 - D, \quad F \leftarrow (2a)B, \\ G \leftarrow C - F, \quad H \leftarrow A - G, \quad Z_3 \leftarrow H^2, \quad X_3 \leftarrow E \cdot H, \quad Y_3 \leftarrow 2C \cdot G - Z_3. \end{gathered}$$

If $a = 1/2$ then the doubling takes $2\mathbf{M} + 5\mathbf{S} + 8\mathbf{a}$ by first removing $F \leftarrow (2a)B$ then replacing $G \leftarrow C - F$ with $G \leftarrow C - B$.

If $a = -1/2$ then the doubling takes $2\mathbf{M} + 5\mathbf{S} + 7\mathbf{a}$ —saving an extra $\mathbf{a}$— by first removing $F \leftarrow (2a)B$ and $G \leftarrow C - F$ then replacing $H \leftarrow A - G$ and $Y_3 \leftarrow 2C \cdot G - Z_3$ with $H \leftarrow A - D$ and $Y_3 \leftarrow 2C \cdot D - Z_3$, respectively.

Alternatively, evaluating (5.12) takes takes $3\mathbf{M} + 4\mathbf{S} + 1\mathbf{D} + 4\mathbf{a}$ by sacrifising one $\mathbf{M}/\mathbf{S}$ trade-off to save $4\mathbf{a}$;

$$A \leftarrow Z_1^2, \quad B \leftarrow X_1^2, \quad C \leftarrow Y_1^2, \quad D \leftarrow (C - (2a)B)/2, \quad E \leftarrow A - D, \quad Z_3 \leftarrow E^2,$$
$$X_3 \leftarrow X_1 \cdot Y_1 \cdot E, \quad Y_3 \leftarrow C \cdot D - Z_3.$$

If $a = \pm 1/2$ then $1\mathbf{D}$ is saved in the usual sense.

Dedicated addition in $\mathcal{Q}$ Further let $(X_2 : Y_2 : Z_2)$ with $Z_2 \neq 0$ satisfy $Y^2Z^2 = dX^4 + 2aX^2Z^2 + Z^4$. Then, $(X_1 : Y_1 : Z_1) + (X_2 : Y_2 : Z_2) = (X_3 : Y_3 : Z_3)$ where

$$\begin{aligned} X_3 &= (X_1Y_2 - Y_1X_2)(X_1^2Z_2^2 - Z_1^2X_2^2), \\ Y_3 &= (Y_1Y_2 - 2aX_1X_2)(X_1^2Z_2^2 + Z_1^2X_2^2) - 2X_1X_2(Z_1^2Z_2^2 + dX_1^2X_2^2), \\ Z_3 &= Z_1Z_2(X_1Y_2 - Y_1X_2)^2 \end{aligned} \tag{5.13}$$

assuming $Z_3 \neq 0$. These formulae are obtained from (4.13) and (4.14). By Lemma 4.2.4, $Z_3 \neq 0$ if $(X_1 : Y_1 : Z_1) \neq (X_2 : Y_2 : Z_2)$ and both summands are of odd order.

Evaluating (5.13) takes $10\mathbf{M} + 5\mathbf{S} + 2\mathbf{D} + 10\mathbf{a}$;

$$A \leftarrow Z_1 \cdot Z_2, \quad B \leftarrow Y_1 \cdot Y_2, \quad C \leftarrow X_1 \cdot X_2, \quad D \leftarrow (X_1 - Y_1) \cdot (X_2 + Y_2) + B - C,$$
$$F \leftarrow (X_1 \cdot Z_2)^2, \quad G \leftarrow (Z_1 \cdot X_2)^2, \quad X_3 \leftarrow D \cdot (F - G),$$
$$Y_3 \leftarrow (B - (2a)C) \cdot (F + G) - 2C \cdot (A^2 + dC^2), \quad Z_3 \leftarrow A \cdot D^2.$$

Additionally, if $Z_2 = 1$ then it takes $8\mathbf{M} + 5\mathbf{S} + 2\mathbf{D} + 10\mathbf{a}$.

If $a = 1/2$ then the dedicated addition takes $10\mathbf{M} + 5\mathbf{S} + 1\mathbf{D} + 9\mathbf{a}$ by first caching $U \leftarrow B - C$ then replacing $D \leftarrow (X_1 - Y_1) \cdot (X_2 + Y_2) + B - C$ and $Y_3 \leftarrow (B - (2a)C) \cdot (F + G) - 2C \cdot (A^2 + dC^2)$ with $D \leftarrow (X_1 - Y_1) \cdot (X_2 + Y_2) + U$ and $Y_3 \leftarrow U \cdot (F + G) - 2C \cdot (A^2 + dC^2)$, respectively. Additionally, if $Z_2 = 1$ then it takes $8\mathbf{M} + 5\mathbf{S} + 1\mathbf{D} + 9\mathbf{a}$.

If $a = -1/2$ then the dedicated addition takes $10\mathbf{M} + 5\mathbf{S} + 1\mathbf{D} + 10\mathbf{a}$. Additionally, if $Z_2 = 1$ then it takes $8\mathbf{M} + 5\mathbf{S} + 1\mathbf{D} + 10\mathbf{a}$.

Unified addition in $\mathcal{Q}$ Alternatively, $(X_1 : Y_1 : Z_1) + (X_2 : Y_2 : Z_2) = (X_3 : Y_3 : Z_3)$ where

$$\begin{aligned} X_3 &= Z_1Z_2(X_1Y_2 + Y_1X_2)(Z_1^2Z_2^2 - dX_1^2X_2^2), \\ Y_3 &= Z_1Z_2(Y_1Y_2 + 2aX_1X_2)(Z_1^2Z_2^2 + dX_1^2X_2^2) + 2dX_1X_2Z_1Z_2(X_1^2Z_2^2 + Z_1^2X_2^2), \\ Z_3 &= (Z_1^2Z_2^2 - dX_1^2X_2^2)^2 \end{aligned} \tag{5.14}$$

assuming $Z_3 \neq 0$. These formulae are obtained from (4.15) and (4.16). By Lemma 4.2.8, $Z_3 \neq 0$ if d is not a square in $\mathbb{K}$. Also by Lemma 4.2.7, $Z_3 \neq 0$ if both $(X_1 : Y_1 : Z_1)$ and $(X_2 : Y_2 : Z_2)$ are of odd order.

Evaluating (5.14) takes $10\mathbf{M} + 7\mathbf{S} + 3\mathbf{D} + 17\mathbf{a}$;

$$A \leftarrow Z_1 \cdot Z_2, \quad B \leftarrow Y_1 \cdot Y_2, \quad C \leftarrow X_1 \cdot X_2, \quad D \leftarrow (X_1 + Y_1) \cdot (X_2 + Y_2) - B - C,$$
$$F \leftarrow (X_1 \cdot Z_2)^2, \quad G \leftarrow (Z_1 \cdot X_2)^2, \quad H \leftarrow A^2, \quad J \leftarrow C^2, \quad K \leftarrow dJ, \quad L \leftarrow H - K,$$
$$M \leftarrow L^2, \quad N \leftarrow (A + L)^2 - H - M, \quad P \leftarrow (A + C)^2 - H - J, \quad X_3 \leftarrow D \cdot N,$$
$$Y_3 \leftarrow 2A \cdot (B + (2a) \cdot C) \cdot (H + K) + (2d)P \cdot (F + G), \quad Z_3 \leftarrow 2M.$$

Additionally, if $Z_2 = 1$ then it takes $8\mathbf{M} + 7\mathbf{S} + 3\mathbf{D} + 17\mathbf{a}$.

If $a = 1/2$ then the dedicated addition takes $10\mathbf{M}+7\mathbf{S}+2\mathbf{D}+16\mathbf{a}$ by first caching $U \leftarrow B+C$ then replacing $D \leftarrow (X_1 + Y_1) \cdot (X_2 + Y_2) - B - C$ and $Y_3 \leftarrow 2A \cdot (B + (2a) \cdot C) \cdot (H + K) + (2d)P \cdot (F+G)$ with $D \leftarrow (X_1+Y_1)\cdot(X_2+Y_2)-U$ and $Y_3 \leftarrow 2A\cdot U\cdot(H+K)+(2d)P\cdot(F+G)$, respectively. Additionally, if $Z_2 = 1$ then it takes $8\mathbf{M} + 7\mathbf{S} + 2\mathbf{D} + 16\mathbf{a}$.

If $a = -1/2$ then the dedicated addition takes $10\mathbf{M}+7\mathbf{S}+2\mathbf{D}+17\mathbf{a}$. Additionally, if $Z_2 = 1$ then it takes $8\mathbf{M} + 7\mathbf{S} + 2\mathbf{D} + 17\mathbf{a}$.

An alternative evaluation of (5.14) takes $12\mathbf{M} + 5\mathbf{S} + 3\mathbf{D} + 9\mathbf{a}$;

$$\begin{gathered} A \leftarrow X_1 \cdot Z_2, \quad B \leftarrow Y_1 \cdot Z_2, \quad C \leftarrow Z_1 \cdot X_2, \quad D \leftarrow Z_1 \cdot Y_2, \quad E \leftarrow A \cdot C, \\ F \leftarrow B \cdot D, \quad G \leftarrow (A + B) \cdot (C + D) - E - F, \quad H \leftarrow Z_1 \cdot Z_2, \quad J \leftarrow X_1 \cdot X_2, \\ K \leftarrow H^2, \quad L \leftarrow dJ^2, \quad M \leftarrow K - L, \quad X_3 \leftarrow G \cdot M, \quad Z_3 \leftarrow M^2, \\ Y_3 \leftarrow (K + L) \cdot (F + (2a)E) + (2d)E \cdot (A^2 + C^2). \end{gathered}$$

Additionally, if $Z_2 = 1$ then it takes $9\mathbf{M} + 5\mathbf{S} + 3\mathbf{D} + 9\mathbf{a}$.

If $a = \pm 1/2$ then $1\mathbf{D}$ is saved in the usual sense.

5.2.2 Extended homogeneous projective coordinates, $\mathcal{Q}^e$

Although homogeneous projective coordinates allow a very fast doubling, the additions require a lot of effort for computing multiplications by Z_1 and Z_2. Keeping track of some redundant data decreases this overload. There are several ways of extending the homogeneous projective coordinates such as representing $(X\colon Y\colon Z)$ in $\mathcal{Q}$ as $(X\colon Y\colon Z\colon X^2\colon Z^2)$ or $(X\colon Y\colon Z\colon X^2\colon Z^2\colon XZ)$, etc. However, this work is restricted to representing $(X\colon Y\colon Z)$ as $(X\colon Y\colon T\colon Z)$ with only one additional coordinate. This system performs faster than other choices and uses memory space wisely. Further comparison is omitted here.

In this system, each triplet $(X\colon Y\colon Z)$ with $Z \neq 0$ is represented as the quadruplet $(X\colon Y\colon T\colon Z)$ where $T = X^2/Z$. For all nonzero $\lambda \in \overline{\mathbb{K}}$, $(X\colon Y\colon T\colon Z) = (\lambda X\colon \lambda Y\colon \lambda T\colon \lambda Z)$. Such a quadruplet satisfies $Y^2Z^2 = dX^4 + 2aX^2Z^2 + Z^4$ and corresponds to the affine point $(X/Z, Y/Z)$. The identity element is represented by $(0\colon 1\colon 0\colon 1)$. The negative of $(X\colon Y\colon T\colon Z)$ is $(-X\colon Y\colon T\colon Z)$. Given $(X\colon Y\colon Z)$ in $\mathcal{Q}$ passing to $\mathcal{Q}^e$ can be performed in $1\mathbf{M} + 3\mathbf{S}$ by computing $(XZ\colon YZ\colon X^2\colon Z^2)$ noting that $XZ = ((X + Z)^2 - X^2 - Z^2)/2$. Given $(X\colon Y\colon T\colon Z)$ in $\mathcal{Q}^e$ passing to $\mathcal{Q}$ is cost-free by simply ignoring T.

Doubling in $\mathcal{Q}^e$ Let $(X_1\colon Y_1\colon T_1\colon Z_1)$ with $Z_1 \neq 0$ and $T_1 = X_1Y_1/Z_1$ satisfy $Y^2Z^2 = dX^4 + 2aX^2Z^2 + Z^4$. Then $[2](X_1\colon Y_1\colon T_1\colon Z_1) = (X_3\colon Y_3\colon T_3\colon Z_3)$ where

$$\begin{aligned} X_3 &= 2X_1Y_1(2Z_1^2 - Y_1^2 + 2aX_1^2), \\ Y_3 &= 2Y_1^2(Y_1^2 - 2aX_1^2) - (2Z_1^2 - Y_1^2 + 2aX_1^2)^2, \\ T_3 &= (2X_1Y_1)^2, \\ Z_3 &= (2Z_1^2 - Y_1^2 + 2aX_1^2)^2 \end{aligned} \tag{5.15}$$

assuming $Z_3 \neq 0$. These formulae are obtained from (4.11) and (4.12). By Lemma 4.2.8, $Z_3 \neq 0$ if d is not a square in $\mathbb{K}$. Also by Lemma 4.4.7, $Z_3 \neq 0$ if $(X_1\colon Y_1\colon T_1\colon Z_1)$ is of odd order.

Evaluating (5.15) takes $8\mathbf{S} + 2\mathbf{D} + 14\mathbf{a}$;

$$A \leftarrow Z_1^2, \quad B \leftarrow Y_1^2, \quad C \leftarrow X_1^2, \quad D \leftarrow B + C, \quad E \leftarrow (X_1 + Y_1)^2 - D,$$
$$F \leftarrow 2A - B + (2a)C, \quad G \leftarrow E^2, \quad H \leftarrow F^2, \quad X_3 \leftarrow (E + F)^2 - G - H, \quad T_3 \leftarrow 2G,$$
$$Z_3 \leftarrow 2H, \quad Y_3 \leftarrow (2B)^2 - (2a)G - Z_3.$$

If $a = 1/2$ then the doubling takes $8\mathbf{S} + 14\mathbf{a}$ by replacing $F \leftarrow 2A - B + (2a)C$ and $Y_3 \leftarrow (2B)^2 - (2a)G - Z_3$ with $F \leftarrow 2A - B + C$ and $Y_3 \leftarrow (2B)^2 - G - Z_3$, respectively.

If $a = -1/2$ then the doubling takes $8\mathbf{S} + 13\mathbf{a}$ —saving an extra $\mathbf{a}$— by replacing $F \leftarrow 2A - B + (2a)C$ and $Y_3 \leftarrow (2B)^2 - (2a)G - Z_3$ with $F \leftarrow 2A - D$ and $Y_3 \leftarrow (2B)^2 + G - Z_3$, respectively.

An alternative evaluation of (5.15) saves $1\mathbf{D}$, several $\mathbf{a}$ and takes $3\mathbf{M} + 5\mathbf{S} + 1\mathbf{D} + 4\mathbf{a}$ at the cost of three $\mathbf{M}/\mathbf{S}$ trade-offs;

$$A \leftarrow Z_1^2, \quad B \leftarrow Y_1^2, \quad C \leftarrow X_1^2, \quad D \leftarrow X_1 \cdot Y_1, \quad E \leftarrow (B - (2a)C)/2, \quad F \leftarrow A - E,$$
$$T_3 \leftarrow D^2, \quad Z_3 \leftarrow F^2, \quad X_3 \leftarrow D \cdot F, \quad Y_3 \leftarrow B \cdot E - Z_3.$$

If $a = \pm 1/2$ then $1\mathbf{D}$ is saved in the usual sense.

Dedicated addition in $\mathcal{Q}^e$ Further let $(X_2 : Y_2 : T_2 : Z_2)$ with $Z_2 \neq 0$ and $T_2 = X_2Y_2/Z_2$ satisfy $Y^2Z^2 = dX^4 + 2aX^2Z^2 + Z^4$. Then, $(X_1 : Y_1 : T_1 : Z_1) + (X_2 : Y_2 : T_2 : Z_2) = (X_3 : Y_3 : T_3 : Z_3)$ where

$$\begin{aligned} X_3 &= (X_1Y_2 - Y_1X_2)(T_1Z_2 - Z_1T_2), \\ Y_3 &= (Y_1Y_2 - 2aX_1X_2)(T_1Z_2 + Z_1T_2) - 2X_1X_2(Z_1Z_2 + dT_1T_2), \\ T_3 &= (T_1Z_2 - Z_1T_2)^2, \\ Z_3 &= (X_1Y_2 - Y_1X_2)^2 \end{aligned} \tag{5.16}$$

assuming $Z_3 \neq 0$. These formulae are obtained from (4.13) and (4.14). By Lemma 4.2.4, $Z_3 \neq 0$ if $(X_1 : Y_1 : T_1 : Z_1) \neq (X_2 : Y_2 : T_2 : Z_2)$ and both summands are of odd order.

Without any assumptions on the curve constants, Y_3 can alternatively be written as

$$Y_3 = (T_1Z_2 + Z_1T_2 - 2X_1X_2)(Y_1Y_2 - 2aX_1X_2 + Z_1Z_2 + dT_1T_2) - Z_3. \tag{5.17}$$

This formula is obtained from (4.20).

Evaluating (5.16) by replacing Y_3 with (5.17) takes $7\mathbf{M} + 3\mathbf{S} + 2\mathbf{D} + 19\mathbf{a}$;

$$A \leftarrow T_1 \cdot Z_2, \quad B \leftarrow Z_1 \cdot T_2, \quad C \leftarrow X_1 \cdot X_2, \quad D \leftarrow Y_1 \cdot Y_2,$$
$$E \leftarrow (X_1 - Y_1) \cdot (X_2 + Y_2) - C + D, \quad F \leftarrow A - B, \quad Z_3 \leftarrow E^2, \quad T_3 \leftarrow F^2,$$
$$X_3 \leftarrow ((E + F)^2 - T_3 - Z_3)/2,$$
$$Y_3 \leftarrow (A + B - 2C) \cdot (D - (2a)C + (Z_1 + T_1) \cdot (Z_2 + dT_2) - A - dB) - Z_3.$$

Additionally, if $Z_2 = 1$ then it takes $6\mathbf{M} + 3\mathbf{S} + 2\mathbf{D} + 19\mathbf{a}$.

If $a = 1/2$ then the dedicated addition takes $7\mathbf{M} + 3\mathbf{S} + 1\mathbf{D} + 18\mathbf{a}$ by first caching $U \leftarrow D - C$ then replacing $E \leftarrow (X_1 - Y_1) \cdot (X_2 + Y_2) - C + D$ and $Y_3 \leftarrow (A + B - 2C) \cdot (D - (2a)C + (Z_1 + T_1) \cdot (Z_2 + dT_2) - A - dB) - Z_3$ with $E \leftarrow (X_1 - Y_1) \cdot (X_2 + Y_2) + U$ and $Y_3 \leftarrow (A + B - 2C) \cdot (U + (Z_1 + T_1) \cdot (Z_2 + dT_2) - A - dB) - Z_3$, respectively. Additionally, if $Z_2 = 1$ then it takes $6\mathbf{M} + 3\mathbf{S} + 1\mathbf{D} + 18\mathbf{a}$.

If $a = -1/2$ then the dedicated addition takes $7\mathbf{M} + 3\mathbf{S} + 1\mathbf{D} + 19\mathbf{a}$. Additionally, if $Z_2 = 1$ then it takes $6\mathbf{M} + 3\mathbf{S} + 1\mathbf{D} + 19\mathbf{a}$.

Unified addition in $\mathcal{Q}^e$ Alternatively, $(X_1 : Y_1 : T_1 : Z_1) + (X_2 : Y_2 : T_2 : Z_2) = (X_3 : Y_3 : T_3 : Z_3)$ where

$$\begin{aligned} X_3 &= (X_1Y_2 + Y_1X_2)(Z_1Z_2 - dT_1T_2), \\ Y_3 &= (Y_1Y_2 + 2aX_1X_2)(Z_1Z_2 + dT_1T_2) + 2dX_1X_2(T_1Z_2 + Z_1T_2), \\ T_3 &= (X_1Y_2 + Y_1X_2)^2, \\ Z_3 &= (Z_1Z_2 - dT_1T_2)^2 \end{aligned} \tag{5.18}$$

assuming $Z_3 \neq 0$. These formulae are obtained from (4.15) and (4.16). By Lemma 4.2.8, $Z_3 \neq 0$ if d is not a square in $\mathbb{K}$. Also by Lemma 4.2.7, $Z_3 \neq 0$ if both $(X_1 : Y_1 : T_1 : Z_1)$ and $(X_1 : Y_1 : T_1 : Z_1)$ are of odd order.

Evaluating (5.18) takes $8\mathbf{M} + 3\mathbf{S} + 3\mathbf{D} + 17\mathbf{a}$;

$$\begin{gathered} A \leftarrow Z_1 \cdot Z_2, \quad B \leftarrow T_1 \cdot T_2, \quad C \leftarrow X_1 \cdot X_2, \quad D \leftarrow Y_1 \cdot Y_2, \quad E \leftarrow dB, \\ F \leftarrow (X_1 + Y_1) \cdot (X_2 + Y_2) - C - D, \quad G \leftarrow A - E, \quad Z_3 \leftarrow G^2, \quad T_3 \leftarrow F^2, \\ X_3 \leftarrow ((F + G)^2 - T_3 - Z_3)/2, \\ Y_3 \leftarrow (D + (2a)C) \cdot (A + E) + (2d)C \cdot ((T_1 + Z_1) \cdot (T_2 + Z_2) - A - B). \end{gathered}$$

Additionally, if $Z_2 = 1$ then it takes $7\mathbf{M} + 3\mathbf{S} + 3\mathbf{D} + 17\mathbf{a}$.

If $a = 1/2$ then the unified addition takes $8\mathbf{M} + 3\mathbf{S} + 2\mathbf{D} + 16\mathbf{a}$ by first caching $U \leftarrow C + D$ then replacing $F \leftarrow (X_1 + Y_1) \cdot (X_2 + Y_2) - C - D$ and $Y_3 \leftarrow (D + (2a)C) \cdot (A + E) + (2d)C \cdot ((T_1 + Z_1) \cdot (T_2 + Z_2) - A - B)$ with $F \leftarrow (X_1 + Y_1) \cdot (X_2 + Y_2) - U$ and $Y_3 \leftarrow U \cdot (A + E) + (2d)C \cdot ((T_1 + Z_1) \cdot (T_2 + Z_2) - A - B)$, respectively. Additionally, if $Z_2 = 1$ then it takes $7\mathbf{M} + 3\mathbf{S} + 2\mathbf{D} + 16\mathbf{a}$.

If $a = -1/2$ then the unified addition takes $8\mathbf{M} + 3\mathbf{S} + 2\mathbf{D} + 17\mathbf{a}$. Additionally, if $Z_2 = 1$ then it takes $7\mathbf{M} + 3\mathbf{S} + 2\mathbf{D} + 17\mathbf{a}$.

If d is a square in $\mathbb{K}$ then Y_3 can alternatively be written as

$$Y_3 = (Z_1Z_2 + dT_1T_2 \pm 2\sqrt{d}X_1X_2)(Y_1Y_2 + 2aX_1X_2 \pm \sqrt{d}T_1Z_2 \pm \sqrt{d}Z_1T_2) \mp \sqrt{d}T_3. \tag{5.19}$$

This formula is obtained from (4.18).

Evaluating (5.18) by replacing Y_3 with (5.19) takes $7\mathbf{M} + 3\mathbf{S} + 5\mathbf{D} + 17\mathbf{a}$ with the following sequence of operations;

$$\begin{gathered} A \leftarrow Z_1 \cdot Z_2, \quad B \leftarrow T_1 \cdot T_2, \quad C \leftarrow X_1 \cdot X_2, \quad D \leftarrow Y_1 \cdot Y_2, \quad E \leftarrow dB, \\ F \leftarrow (X_1 + Y_1) \cdot (X_2 + Y_2) - C - D, \quad G \leftarrow A - E, \quad Z_3 \leftarrow G^2, \quad T_3 \leftarrow F^2, \\ X_3 \leftarrow ((F + G)^2 - T_3 - Z_3)/2, \\ Y_3 \leftarrow (A + E + (2s)C) \cdot (D + (2a)C + s((T_1 + Z_1) \cdot (T_2 + Z_2) - A - B)) - sT_3. \end{gathered}$$

where $s^2 = d$. Additionally, if $Z_2 = 1$ then it takes $6\mathbf{M} + 3\mathbf{S} + 5\mathbf{D} + 17\mathbf{a}$.

5.2.3 Mixed homogeneous projective coordinates, $\mathcal{Q}^x$

The construction in this section is the same as §5.1.4 and is closely linked with [CMO98]. Therefore, only a brief outline of the technique is given in this section.

Most of the efficient scalar multiplication implementations are based on a suitable combination of signed integer recoding, fast precomputation and left-to-right sliding fractional-windowing techniques. The resulting algorithm is doubling intensive. Roughly for each bit of

the scalar one doubling is performed. Additions are accessed less frequently. Excluding the additions used in the precomputation phase, approximately $l/(w+1)$ additions are needed where l is the number of bits in the scalar and w is the window length. Parameter w is used to control space consumption and optimize the total running time.

An abstract view of the scalar multiplication is composed of several repeated-doublings each followed by a single addition. In the present case, these operations are performed in the following way:

(i) If a point doubling is followed by another point doubling, use $\mathcal{Q} \leftarrow 2\mathcal{Q}$.

(ii) If a point doubling is followed by a point addition, use

1. $\mathcal{Q}^e \leftarrow 2\mathcal{Q}$ for the point doubling step; followed by,
2. $\mathcal{Q} \leftarrow \mathcal{Q}^e + \mathcal{Q}^e$ for the point addition step.

Suppose that a repeated-doubling phase is composed of m doublings. In (i), $m-1$ successive doublings in $\mathcal{Q}$ are performed with (5.12) given in §5.2.1. In (ii), the remaining doubling is merged with the single addition phase to yield a combined double-and-add step; a similar approach to [ELM03]. To perform the double-and-add operation the doubling step is performed using (5.15) given in §5.2.1. This algorithm is suitable to compute $\mathcal{Q}^e \leftarrow 2\mathcal{Q}$ since the inputs are only composed of the coordinates X, Y, Z and the output is still produced in $\mathcal{Q}^e$. The doubling step is followed by the addition in $\mathcal{Q}^e$ using one of (5.16) or (5.17) or (5.18) or (5.19). Note that the computation of T_3 can be omitted to save $1\mathbf{S}$ since the result is in $\mathcal{Q}$ (not $\mathcal{Q}^e$).

For instance, if (5.16) with $a = -1/2$ is used by replacing Y_3 with (5.17), the dedicated addition step effectively costs $(8\mathbf{S}) + (8\mathbf{M} + 2\mathbf{S} + 2\mathbf{D}) - (2\mathbf{M} + 5\mathbf{S}) - (1\mathbf{S}) = 6\mathbf{M} + 4\mathbf{S} + 2\mathbf{D}$ omitting the operation count for $\mathbf{a}$.

Selection of curve constants The constant a can be rescaled to $-1/2$ via the map $(x, y) \mapsto (x/\sqrt{-2a}, y)$ provided that $\sqrt{-2a} \in \mathbb{K}$. This map transforms the curve $y^2 = dx^4 + 2ax^2 + 1$ to $y^2 = (d/(4a^2))x^4 - x^2 + 1$. Alternatively, a curve having $a = -1/2$ can be selected without rescaling. If doublings are frequently accessed then having $a = -1/2$ and keeping d arbitrary gives the optimum performance. If the doublings are not needed frequently then having $d = \pm 1$ and keeping a arbitrary is more efficient as the operation counts and comments in §5.2.2 indicate.

5.2.4 Weighted projective coordinates, $\mathcal{Q}^w$

All literature results prior to this work are based on weighted projective coordinates, $\mathcal{Q}^w$, e.g. [CC86] and [BJ03a]. In weighted projective coordinates each point (x, y) is represented by the triplet $(X : Y : Z)$ which satisfies the projective equation $Y^2 = dX^4 + 2aX^2Z^2 + Z^4$ and corresponds to the affine point $(X/Z, Y/Z^2)$ with $Z \neq 0$. For all nonzero $\lambda \in \overline{\mathbb{K}}$, $(X : Y : Z) = (\lambda X : \lambda^2 Y : \lambda Z)$. The identity element is represented by $(0 : 1 : 1)$. The negative of $(X : Y : Z)$ is $(-X : Y : Z)$. Unlike the homogeneous projective case, this curve is non-singular provided that $d(a^2 - d) \neq 0$.

Chudnovsky and Chudnovsky [CC86] proposed two inversion-free point addition and two inversion-free point doubling formulae using a slightly different quartic equation given by

$$E_{\tilde{J},a',b'} : y^2 = x^4 + a'x^2 + b'$$

and using weighted projective coordinates. The formulae in [CC86, (4.10i) on p.418] are analogous to (4.13) and (4.14) with the minor difference that the identity element is moved to the point at infinity $(1\colon 1\colon 0)$ on $Y^2 = X^4 + a'X^2Z^2 + b'Z^4$. The arithmetic of this curve is similar to that of $E_{\mathbf{Q},d,a}$ due to the symmetry in the right hand side of the weighted projective equations $Y^2 = X^4 + a'X^2Z^2 + b'Z^4$ and $Y^2 = dX^4 + 2aX^2Z^2 + Z^4$.

Billet and Joye [BJ03a] proposed a faster inversion-free unified addition algorithm on the curve $Y^2 = dX^4 + 2aX^2Z^2 + Z^4$ based on (4.15) and (4.16). Keeping in mind that the points at infinity are defined over $\mathbb{K}$ if and only if d is a square in $\mathbb{K}$, Lemma 4.2.8 implies that the Billet/Joye unified point addition algorithm is complete if d is non-square. This algorithm needs $10\mathbf{M} + 3\mathbf{S} + 3\mathbf{D}$ for arbitrary a and d. Note that no faster way of inversion-free general point addition is known to date in $(X\colon Y\colon Z)_{[1,2,1]}$ coordinates[1]. See EFD [BL07a] for a $2\mathbf{M} + 6\mathbf{S} + 1\mathbf{D}$ doubling algorithm by Feng/Wu in $(X\colon Y\colon Z)_{[1,2,1]}$ for the case $d = 1$.

The algorithms are omitted here since they are slower then the proposed mixed coordinates, $\mathcal{Q}^x$, in §5.2.3.

Redundant variants of $\mathcal{Q}^w$ It remains an open question whether it is possible to speed up the addition in $\mathcal{Q}^w$. However, the speed of the Billet/Joye algorithm (in weighted coordinates) was improved by Duquesne in [Duq07] with the proposal of $(X^2\colon XZ\colon Z^2\colon Y)$ coordinates. Duquesne's variant addition algorithm needs $9\mathbf{M} + 2\mathbf{S} + 3\mathbf{D}$ saving $1\mathbf{M} + 1\mathbf{S}$ over the Billet/Joye algorithm by using slightly more space to represent the points. For the case $d = 1$, Bernstein and Lange [BL07a] extended this representation to $(X\colon Y\colon Z\colon X^2\colon 2XZ\colon Z^2)$ and $(X\colon Y\colon Z\colon X^2\colon 2XZ\colon Z^2\colon X^2 + Z^2)$ saving an extra $\mathbf{M} - \mathbf{S}$ (i.e. **M**-**S** trade-off) over Duquesne's algorithm. A more detailed overview of these algorithms and operation counts can be found in the original papers or in [BL07a]. Duquesne coordinates $(X^2\colon XZ\colon Z^2\colon Y)_{[2,2,2,2]}$ use less space than redundant coordinates but need special treatment in the scalar multiplication to obtain the original coordinates $(X\colon Y\colon Z)_{[1,2,1]}$ of the final result. The original representation as $(X\colon Y\colon Z)_{[1,2,1]}$ in [BJ03a] uses even less space however this representation has to date been slower than the redundant coordinates.

The redundant representations such as

$$\begin{gathered}(X\colon Y\colon Z\colon X^2\colon 2XZ\colon Z^2\colon X^2 + Z^2)_{[1,2,1,2,2,2,2]},\\ (X\colon Y\colon Z\colon X^2\colon 2XZ\colon Z^2)_{[1,2,1,2,2,2]},\\ (X\colon Y\colon Z\colon X^2\colon Z^2\colon X^2 + Z^2)_{[1,2,1,2,2,2]},\\ (X\colon Y\colon Z\colon X^2\colon Z^2)_{[1,2,1,2,2]},\\ (X\colon Y\colon Z\colon X^2\colon Z^2\colon XZ)_{[1,2,1,2,2,2]}\end{gathered}$$

help in the development of faster algorithms for performing point operations and their overall performance only slightly differs from each other. However, they all share one serious drawback.

[1]The subscript provides information about the weights of the coordinates

They need more space for storing the points in comparison to earlier proposals. Despite the speed advantage of these coordinate systems, the large space requirement makes the practical use of Jacobi quartic form questionable since windowing techniques in scalar multiplication algorithms precompute and store several points. Since the extended homogeneous projective coordinates solve these problems without resulting in a speed penalty, all redundant versions of weighted coordinates (including Duquesne's approach) are not further considered. On the other hand, the Billet/Joye addition algorithm is still interesting since the arithmetic is done in three coordinates and is faster than its homogeneous projective analogue.

The algorithms are omitted here since they are slower than the proposed mixed coordinates, $\mathcal{Q}^x$, in §5.2.3.

5.2.5 Comparison and remarks

In §5.2.1 and §5.2.2 inversion-free arithmetic for extended Jacobi quartic form is studied in coordinate systems $\mathcal{Q}$ and $\mathcal{Q}^e$ for the first time for efficient computation. The most notable achievement of §5.2.1 is the $2\mathbf{M}+5\mathbf{S}+7\mathbf{a}$ and $3\mathbf{M}+4\mathbf{S}+4\mathbf{a}$ doubling algorithms in $\mathcal{Q}$ on the curve $E_{Q,d,-1/2}$. Comparing these result with previous results in $\mathcal{Q}^w$ (weighted projective coordinates) and its variants, the new doubling algorithm is faster than all other three-coordinate doubling algorithms in $\mathcal{Q}^w$. For instance, the new algorithm improves upon the $2\mathbf{M}+6\mathbf{S}+1\mathbf{D}+10\mathbf{a}$ doubling algorithm by Feng/Wu in $\mathcal{Q}^w$ on the curve $E_{Q,1,a}$, see [BL07a, `dbl-2007-fw-2`]. Feng/Wu point doubling algorithm fails for some special inputs. On the other hand, the proposed doubling algorithms work for all inputs provided that d is non-square in $\mathbb{K}$. In addition, the proposed algorithms do not depend on d. Therefore, selecting d of a large size does not affect the performance of the new algorithms. Note that in the case d is a square in $\mathbb{K}$, it is still possible to prevent failures by working in a subgroup of odd order; a result deduced in §5.2.1 from §4.2 of Chapter 4. The new doubling algorithms are also at least as fast as the previous doubling algorithms in redundant versions of $\mathcal{Q}^w$. (See Appendix C.2.) On the other hand, addition is performed in just 4 coordinates rather than 5, 6 or 7.

Although $\mathcal{Q}$ is ideal for point doublings, it severely suffers from inefficient point additions. The performance of point additions is improved in §5.2.2 by the proposal of the extended homogeneous projective coordinates, $\mathcal{Q}^e$. This system saves several multiplications in $\mathbb{K}$ over the best reported addition algorithms in $\mathcal{Q}$ and $\mathcal{Q}^w$. For instance, a dedicated addition in $\mathcal{Q}^e$ takes $7\mathbf{M}+3\mathbf{S}+1\mathbf{D}+19\mathbf{a}$ where essentially the same formulae takes $10\mathbf{M}+5\mathbf{S}+2\mathbf{D}+10\mathbf{a}$ in $\mathcal{Q}$. It is possible to draw a similar conclusion for the unified additions. The new addition algorithms are at least as fast as the previous addition algorithms in redundant versions of $\mathcal{Q}^w$. (See Appendix C.2.) However doubling is performed in 4 coordinates rather than 5, 6 or 7.

Although $\mathcal{Q}^e$ provides the fastest point additions, this time point doubling is slightly slower. This problem is remedied in §5.2.3 by the introduction of the mixed coordinates, $\mathcal{Q}^x$. This system uses faster doublings in $\mathcal{Q}$ and faster additions in $\mathcal{Q}^e$ without causing any computational overhead while switching between $\mathcal{Q}^e$ and $\mathcal{Q}$. In fact, the swichting speeds up the additions effectively after a combination of suitable algorithms. See §5.2.3 for details.

In §5.2.4, literature results are summarized for weighted projective coordinate system, $\mathcal{Q}^w$. In addition, newer and faster results are proposed for this specific system and some of its

variants in Appendix C.2. These results are separated from the main body of the discussion since the proposed $\mathcal{Q}^x$ systen has already achieved better performance.

Various operation counts in $\mathcal{Q}$, $\mathcal{Q}^w$, $\mathcal{Q}^e$, and $\mathcal{Q}^x$ are summarized in Table 5.3. Table 5.4 provides operation counts for the case $a = -1/2$.

Table 5.3: Operation counts for extended Jacobi quartic form in different coordinate systems.

System	DBL	ADD
$\mathcal{Q}^w$	-	10**M**+3**S**+3**D**+14**a**, unified, [BJ03a]
$\mathcal{Q}$	3**M**+4**S**+1**D**+ 4**a**	10**M**+7**S**+3**D**+17**a**, unified
	2**M**+5**S**+1**D**+ 8**a**	10**M**+5**S**+2**D**+10**a**, dedicated
$\mathcal{Q}^e$	3**M**+5**S**+1**D**+ 4**a**	8**M**+3**S**+3**D**+17**a**, unified
	8**S**+2**D**+14**a**	7**M**+3**S**+2**D**+19**a**, dedicated
$\mathcal{Q}^x$	3**M**+4**S**+1**D**+ 4**a**	7**M**+4**S**+4**D**+19**a**, unified
	2**M**+5**S**+1**D**+ 8**a**	6**M**+4**S**+3**D**+21**a**, dedicated

Table 5.4: Operation counts for extended Jacobi quartic form with $a = -1/2$ in different coordinate systems.

System	DBL	ADD
$\mathcal{Q}^w$	-	10**M**+2**S**+2**D**+14**a**, unified, [BJ03a]
$\mathcal{Q}$	3**M**+4**S**+ 4**a**	10**M**+7**S**+2**D**+17**a**, unified
	2**M**+5**S**+ 7**a**	10**M**+5**S**+1**D**+10**a**, dedicated
$\mathcal{Q}^e$	3**M**+5**S**+ 4**a**	8**M**+3**S**+2**D**+17**a**, unified
	8**S**+13**a**	7**M**+3**S**+1**D**+19**a**, dedicated
$\mathcal{Q}^x$	3**M**+4**S**+ 4**a**	7**M**+4**S**+3**D**+19**a**, unified
	2**M**+5**S**+ 7**a**	6**M**+4**S**+2**D**+21**a**, dedicated

If the slight memory overhead can be neglected then the mixed coordinates $\mathcal{Q}^x$ is clearly the fastest among all other coordinate systems if attention is restricted to the extended Jacobi quartic form. This can be observed in Table 5.3: The doubling operation is at least as fast as the other coordinate systems and the additions are always *effectively* faster, see §5.2.3.

In each of the coordinate systems, the dedicated addition again turned out to be faster than the unified addition as is the case for twisted Edwards curve. However, the gap between the performance of the dedicated addition and unified addition is wider in the case of extended Jacobi quartic form. This makes dedicated addition more suitable in speed oriented implementations.

Literature notes The results in the literature mainly focused on weighted projective coordinates. The relevant pointers are already given in §5.2.4.

5.3 Twisted Jacobi intersection form

The arithmetic of this form is translated from Jacobi elliptic functions where there is a large body of formulae available. Some of those formulae have already been given in affine coordinates in §4.5 of Chapter 4. In this section, the arithmetic of twisted Jacobi intersection form is studied in homogeneous projective coordinates and its redundant variants. The coordinates are not extended as was the case in the previous quartic forms. This is due to the fact that each inversion-free formula (i.e. a formula assigned to a single homogeneous projective coordinate) presented in §5.3.1 is readily of minimal-degree. On the other hand, §5.3.2 shows that keeping some redundant data helps in speeding up the additions. Conclusions are drawn in §5.3.3 together with pointers to the literature.

5.3.1 Homogeneous projective coordinates, $\mathcal{I}$

In this system, each point (s, c, d) on $bs^2+c^2 = 1, as^2+d^2 = 1$ is represented with the quadruplet $(S\colon C\colon D\colon Z)$ which corresponds to the affine point $(S/Z, D/Z, C/Z)$ with $Z \neq 0$. These quadruplets satisfy the homogeneous projective equations $bS^2 + C^2 = Z^2, aS^2 + D^2 = Z^2$. For all nonzero $\lambda \in \overline{\mathbb{K}}$, $(S\colon C\colon D\colon Z) = (\lambda S\colon \lambda C\colon \lambda D\colon \lambda Z)$. The identity element is represented by $(0\colon 1\colon 1\colon 1)$. The negative of $(S\colon C\colon D\colon Z)$ is $(-S\colon C\colon D\colon Z)$. This coordinate system is denoted by $\mathcal{I}$ throughout the text.

Doubling in $\mathcal{I}$ Let $(S_1\colon C_1\colon D_1\colon Z_1)$ with $Z_1 \neq 0$ satisfy $bS^2 + C^2 = Z^2, aS^2 + D^2 = Z^2$. Then $[2](S_1\colon C_1\colon D_1\colon Z_1) = (S_3\colon C_3\colon D_3\colon Z_3)$ where

$$\begin{aligned} S_3 &= 2S_1C_1D_1Z_1, \\ C_3 &= C_1^2Z_1^2 - bS_1^2D_1^2, \\ D_3 &= -C_1^2Z_1^2 - bS_1^2D_1^2 + 2D_1^2Z_1^2, \\ Z_3 &= C_1^2Z_1^2 + bS_1^2D_1^2 \end{aligned} \tag{5.20}$$

assuming that $Z_3 \neq 0$. These formulae are obtained from (4.55), (4.56), and (4.57). By Lemma 4.5.8, Z_3 is always nonzero if a is not a square in $\mathbb{K}$. In this case, it is also important to note that the points at infinity are not defined over $\mathbb{K}$. Also by Lemma 4.5.7, Z_3 is always nonzero if $(X_1\colon Y_1\colon Z_1)$ is of odd order regardless of the assumption on a and b (of course, $ab(a-b) \neq 0$).

Evaluating (5.20) takes $3\mathbf{M} + 4\mathbf{S} + 1\mathbf{D} + 7\mathbf{a}$;

$$\begin{gathered} U_1 \leftarrow S_1 \cdot D_1, \quad V_1 \leftarrow C_1 \cdot Z_1, \quad E \leftarrow D_1 \cdot Z_1, \quad F \leftarrow U_1^2, \quad G \leftarrow V_1^2, \\ S_3 \leftarrow (U_1 + V_1)^2 - G - F, \quad H \leftarrow bF, \quad C_3 \leftarrow G - H, \quad Z_3 \leftarrow G + H, \\ D_3 \leftarrow 2E^2 - Z_3. \end{gathered}$$

Evaluating (5.20) with $b = 1$ takes $3\mathbf{M} + 4\mathbf{S} + 6\mathbf{a}$ in [BL07a, `dbl-2007-bl`];

$$\begin{gathered} U_1 \leftarrow S_1 \cdot D_1, \quad V_1 \leftarrow C_1 \cdot Z_1, \quad E \leftarrow D_1 \cdot Z_1, \quad F \leftarrow U_1^2, \quad G \leftarrow V_1^2, \quad Z_3 \leftarrow G + F, \\ S_3 \leftarrow (U_1 + V_1)^2 - Z_3, \quad C_3 \leftarrow G - F, \quad D_3 \leftarrow 2E^2 - Z_3 \end{gathered}$$

with minor notation changes.

An alternative way of writing D_3 is as follows.

$$D_3 = -C_1^2Z_1^2 - bS_1^2D_1^2 + 2aS_1^2D_1^2 + 2D_1^4 \tag{5.21}$$

Evaluating (5.20) when D_3 is replaced with (5.21) takes $2\mathbf{M} + 5\mathbf{S} + 2\mathbf{D} + 8\mathbf{a}$;

$$U_1 \leftarrow S_1 \cdot D_1, \quad V_1 \leftarrow C_1 \cdot Z_1, \quad F \leftarrow U_1^2, \quad G \leftarrow V_1^2, \quad H \leftarrow bF, \quad C_3 \leftarrow G - H,$$
$$Z_3 \leftarrow G + H, \quad S_3 \leftarrow (U_1 + V_1)^2 - F - G, \quad D_3 \leftarrow 2(aF + D_1^4) - Z_3.$$

Evaluating (5.20) with $b = 1$ when D_3 is replaced with (5.21) takes $2\mathbf{M} + 5\mathbf{S} + 1\mathbf{D} + 7\mathbf{a}$;

$$U_1 \leftarrow S_1 \cdot D_1, \quad V_1 \leftarrow C_1 \cdot Z_1, \quad F \leftarrow U_1^2, \quad G \leftarrow V_1^2, \quad C_3 \leftarrow G - F, \quad Z_3 \leftarrow G + F,$$
$$S_3 \leftarrow (U_1 + V_1)^2 - Z_3, \quad D_3 \leftarrow 2(aF + D_1^4) - Z_3.$$

Dedicated addition in $\mathcal{I}$ Further let $(S_2 : C_2 : D_2 : Z_2)$ with $Z_2 \neq 0$ satisfy $bS^2 + C^2 = Z^2, aS^2 + D^2 = Z^2$. Then, $(S_1 : C_1 : D_1 : Z_1) + (S_2 : C_2 : D_2 : Z_2) = (S_3 : C_3 : D_3 : Z_3)$ where

$$\begin{aligned} S_3 &= S_1^2 Z_2^2 - Z_1^2 S_2^2, \\ C_3 &= S_1 C_1 D_2 Z_2 - D_1 Z_1 S_2 C_2, \\ D_3 &= S_1 D_1 C_2 Z_2 - C_1 Z_1 S_2 D_2, \\ Z_3 &= S_1 Z_1 C_2 D_2 - C_1 D_1 S_2 Z_2 \end{aligned} \tag{5.22}$$

assuming that $Z_3 \neq 0$. These formulae are obtained from (4.46), (4.47), and (4.48). By Lemma 4.5.7, Z_3 is always nonzero if $(S_1 : C_1 : D_1 : Z_1) \neq (S_2 : C_2 : D_2 : Z_2)$ and both summands are of odd order.

Evaluating (5.22) takes $12\mathbf{M} + 11\mathbf{a}$;

$$E \leftarrow S_1 \cdot Z_2, \quad F \leftarrow Z_1 \cdot S_2, \quad G \leftarrow C_1 \cdot D_2, \quad H \leftarrow D_1 \cdot C_2, \quad J \leftarrow E - F,$$
$$K \leftarrow E + F, \quad L \leftarrow G - H, \quad M \leftarrow G + H, \quad N \leftarrow K \cdot L, \quad P \leftarrow J \cdot M, \quad S_3 \leftarrow J \cdot K,$$
$$C_3 \leftarrow (N + P)/2, \quad D_3 \leftarrow (P - N)/2,$$
$$Z_3 \leftarrow (D_1 \cdot Z_2 + Z_1 \cdot D_2) \cdot (S_1 \cdot C_2 - C_1 \cdot S_2) - D_3.$$

Additionally, if $Z_2 = 1$ then it takes $10\mathbf{M} + 11\mathbf{a}$. Also note that it is possible to compute D_3 as $U_1 \cdot V_2 - V_1 \cdot U_2$ where $U_i = S_i \cdot D_i$ and $V_i = C_i \cdot Z_i$. Assuming that U_2 and V_2 are cached, a re-addition takes $11\mathbf{M} + 9\mathbf{a}$.

Unified addition in $\mathcal{I}$ Alternatively, $(S_1 : C_1 : D_1 : Z_1) + (S_2 : C_2 : D_2 : Z_2) = (S_3 : C_3 : D_3 : Z_3)$ where

$$\begin{aligned} S_3 &= S_1 Z_1 C_2 D_2 + C_1 D_1 S_2 Z_2, \\ C_3 &= C_1 Z_1 C_2 Z_2 - bS_1 D_1 S_2 D_2, \\ D_3 &= D_1 Z_1 D_2 Z_2 - aS_1 C_1 S_2 C_2, \\ Z_3 &= C_1^2 Z_2^2 + bS_1^2 D_2^2 \end{aligned} \tag{5.23}$$

assuming that $Z_3 \neq 0$. These formulae are obtained from (4.49), (4.50), and (4.51). By Lemma 4.5.8, $Z_3 \neq 0$ if a is not a square in $\mathbb{K}$. Also by Lemma 4.5.7, Z_3 is always nonzero if both $(S_1 : C_1 : D_1 : Z_1)$ and $(S_2 : C_2 : D_2 : Z_2)$ are of odd order.

Evaluating (5.23) takes $13\mathbf{M} + 2\mathbf{S} + 5\mathbf{D} + 13\mathbf{a}$;

$$E \leftarrow C_1 \cdot Z_2, \quad F \leftarrow Z_1 \cdot C_2, \quad G \leftarrow S_1 \cdot D_2, \quad H \leftarrow D_1 \cdot S_2, \quad J \leftarrow F \cdot H,$$
$$K \leftarrow E \cdot G, \quad S_3 \leftarrow (E + F) \cdot (G + H) - J - K, \quad C_3 \leftarrow (E - bH) \cdot (G + F) + bJ - K,$$
$$D_3 \leftarrow (D_1 \cdot Z_1 - aS_1 \cdot C_1) \cdot (S_2 \cdot C_2 + D_2 \cdot Z_2) - J + aK, \quad Z_3 \leftarrow E^2 + bG^2.$$

Additionally, if $Z_2 = 1$ then it takes $11\mathbf{M} + 1\mathbf{S} + 2\mathbf{D} + 15\mathbf{a}$.

Evaluating (5.20) with $b = 1$ takes $13\mathbf{M} + 1\mathbf{S} + 2\mathbf{D} + 15\mathbf{a}$;

$$U_1 \leftarrow S_1 \cdot C_1, \quad V_1 \leftarrow D_1 \cdot Z_1, \quad U_2 \leftarrow S_2 \cdot C_2, \quad V_2 \leftarrow D_2 \cdot Z_2, \quad E \leftarrow S_1 \cdot D_2,$$
$$F \leftarrow C_1 \cdot Z_2, \quad G \leftarrow D_1 \cdot S_2, \quad H \leftarrow Z_1 \cdot C_2, \quad J \leftarrow U_1 \cdot V_2, \quad K \leftarrow V_1 \cdot U_2,$$
$$S_3 \leftarrow (H+F) \cdot (E+G) - J - K, \quad C_3 \leftarrow (H+E) \cdot (F-G) - J + K,$$
$$D_3 \leftarrow (V_1 - aU_1) \cdot (U_2 + V_2) + aJ - K, \quad Z_3 \leftarrow (H+G)^2 - 2K.$$

Additionally, if $Z_2 = 1$ then it takes $11\mathbf{M} + 1\mathbf{S} + 2\mathbf{D} + 15\mathbf{a}$.

5.3.2 Modified homogeneous projective coordinates, $\mathcal{I}^m$

Suitable auxiliary coordinates speed up point additions. Three possible ways of modifying $\mathcal{I}$ are representing a sextuplet $(S\colon C\colon D\colon Z\colon U\colon V)$ satisfying $bS^2 + C^2 = Z^2, aS^2 + D^2 = Z^2$ as $(S\colon C\colon D\colon Z\colon SD\colon CZ)$ or $(S\colon C\colon D\colon Z\colon SC\colon DZ)$ or $(S\colon C\colon D\colon Z\colon CD\colon SZ)$. When suitable formulae are selected the performance of these coordinate systems is superior to $\mathcal{I}$ omitting the overhead of the extra space requirement. Because of similarity, only the system $\mathcal{I}^{m1}\colon (S\colon C\colon D\colon Z\colon SD\colon CZ)$, is considered for the doubling and dedicated addition. In the case of unified addition, $\mathcal{I}^{m2}\colon (S\colon C\colon D\colon Z\colon SC\colon DZ)$ is preferred for optimum performance. Further comparison is omitted here. In both systems, the identity element is represented by $(0\colon 1\colon 1\colon 1\colon 0\colon 1)$. Point negation is done by flipping the sign of the S-coordinate. The exception prevention techniques are identical to that of $\mathcal{I}$ for the modified coordinates.

Doubling in $\mathcal{I}^{m1}$ Let $(S_1\colon C_1\colon D_1\colon Z_1\colon U_1\colon V_1)$ with $Z_1 \neq 0$ satisfy $U = SD, V = CZ, bS^2 + C^2 = Z^2, aS^2 + D^2 = Z^2$. Then $[2](S_1\colon C_1\colon D_1\colon Z_1\colon U_1\colon V_1) = (S_3\colon C_3\colon D_3\colon Z_3\colon U_3\colon V_3)$ where S_3, C_3, D_3, and Z_3 are the same as the doubling formulae in §5.3.1 and $U_3 = S_3D_3$, and $V_3 = C_3Z_3$. The doubling in $\mathcal{I}^{m1}$ takes $3\mathbf{M} + 4\mathbf{S} + 1\mathbf{D} + 7\mathbf{a}$;

$$E \leftarrow D_1 \cdot Z_1, \quad F \leftarrow U_1^2, \quad G \leftarrow V_1^2, \quad S_3 \leftarrow (U_1 + V_1)^2 - G - F, \quad H \leftarrow bF,$$
$$C_3 \leftarrow G - H, \quad Z_3 \leftarrow G + H, \quad D_3 \leftarrow 2E^2 - Z_3. \quad U_3 \leftarrow S_3 \cdot D_3, \quad V_3 \leftarrow C_3 \cdot Z_3.$$

The doubling with $b = 1$ in $\mathcal{I}^{m1}$ takes $3\mathbf{M} + 4\mathbf{S} + 6\mathbf{a}$;

$$E \leftarrow D_1 \cdot Z_1, \quad F \leftarrow U_1^2, \quad G \leftarrow V_1^2, \quad Z_3 \leftarrow G + F, \quad S_3 \leftarrow (U_1 + V_1)^2 - Z_3,$$
$$C_3 \leftarrow G - F, \quad D_3 \leftarrow 2E^2 - Z_3, \quad U_3 \leftarrow S_3 \cdot D_3, \quad V_3 \leftarrow C_3 \cdot Z_3.$$

The doubling in $\mathcal{I}^{m1}$ alternatively takes $2\mathbf{M} + 5\mathbf{S} + 2\mathbf{D} + 8\mathbf{a}$;

$$F \leftarrow U_1^2, \quad G \leftarrow V_1^2, \quad H \leftarrow bF, \quad C_3 \leftarrow G - H, \quad Z_3 \leftarrow G + H,$$
$$S_3 \leftarrow (U_1 + V_1)^2 - F - G, \quad D_3 \leftarrow 2(aF + D_1^4) - Z_3, \quad U_3 \leftarrow S_3 \cdot D_3, \quad V_3 \leftarrow C_3 \cdot Z_3.$$

The alternative doubling with $b = 1$ in $\mathcal{I}^{m1}$ takes $2\mathbf{M} + 5\mathbf{S} + 1\mathbf{D} + 7\mathbf{a}$;

$$F \leftarrow U_1^2, \quad G \leftarrow V_1^2, \quad C_3 \leftarrow G - F, \quad Z_3 \leftarrow G + F, \quad S_3 \leftarrow (U_1 + V_1)^2 - Z_3,$$
$$D_3 \leftarrow 2(aF + D_1^4) - Z_3, \quad U_3 \leftarrow S_3 \cdot D_3, \quad V_3 \leftarrow C_3 \cdot Z_3.$$

Dedicated addition in $\mathcal{I}^{m1}$ Further let $(S_2\colon C_2\colon D_2\colon Z_2\colon U_2\colon V_2)$ with $Z_2 \neq 0$ satisfy $U = SD, V = CZ, bS^2 + C^2 = Z^2, aS^2 + D^2 = Z^2$. Then, $(S_1\colon C_1\colon D_1\colon Z_1\colon U_1\colon V_1) + (S_2\colon C_2\colon D_2\colon Z_2\colon U_2\colon V_2) = (S_3\colon C_3\colon D_3\colon Z_3\colon U_3\colon V_3)$ where S_3, C_3, D_3, and Z_3 are the same as the dedicated addition formulae in §5.3.1 and $U_3 = S_3D_3$, and $V_3 = C_3Z_3$. The dedicated addition in $\mathcal{I}^{m1}$ takes $11\mathbf{M}+1\mathbf{D}+9\mathbf{a}$ saving $1\mathbf{M}-1\mathbf{D}+2\mathbf{a}$ in comparison to the corresponding dedicated addition in $\mathcal{I}$;

$$E \leftarrow C_1 \cdot Z_2, \quad F \leftarrow Z_1 \cdot C_2, \quad G \leftarrow S_1 \cdot D_2, \quad H \leftarrow D_1 \cdot S_2, \quad J \leftarrow F - E,$$
$$K \leftarrow F + E, \quad L \leftarrow G - H, \quad M \leftarrow G + H, \quad N \leftarrow K \cdot L, \quad P \leftarrow J \cdot M,$$
$$S_3 \leftarrow (1/b)J \cdot K, \quad C_3 \leftarrow (N - P)/2, \quad Z_3 \leftarrow (N + P)/2, \quad D_3 \leftarrow U_1 \cdot V_2 - V_1 \cdot U_2,$$
$$U_3 \leftarrow S_3 \cdot D_3, \quad V_3 \leftarrow C_3 \cdot Z_3.$$

Additionally, if $Z_2 = 1$ then it takes $10\mathbf{M} + 1\mathbf{D} + 9\mathbf{a}$. Note that the equality $(1/b)(Z_1^2C_2^2 - C_1^2Z_2^2) = S_1^2Z_2^2 - Z_1^2S_2^2$ is exploited in this algorithm.

$$E \leftarrow C_1 \cdot Z_2, \quad F \leftarrow Z_1 \cdot C_2, \quad G \leftarrow S_1 \cdot D_2, \quad H \leftarrow D_1 \cdot S_2, \quad J \leftarrow F - E,$$
$$K \leftarrow F + E, \quad L \leftarrow G - H, \quad M \leftarrow G + H, \quad N \leftarrow K \cdot L, \quad P \leftarrow J \cdot M, \quad S_3 \leftarrow J \cdot K,$$
$$C_3 \leftarrow (N - P)/2, \quad Z_3 \leftarrow (N + P)/2, \quad D_3 \leftarrow U_1 \cdot V_2 - V_1 \cdot U_2, \quad U_3 \leftarrow S_3 \cdot D_3,$$
$$V_3 \leftarrow C_3 \cdot Z_3.$$

Additionally, if $Z_2 = 1$ then it takes $10\mathbf{M} + 9\mathbf{a}$.

Unified addition in $\mathcal{I}^{m2}$ It is possible to use $\mathcal{I}^{m1}$ to perform unified addition however $\mathcal{I}^{m2}$ is superior in speed. In addition, as long as only the unified addition is used, there is no need to switch to the other representations. Let $(S\colon C\colon D\colon Z\colon U\colon V)$ with $Z_i \neq 0$ satisfy $U = SC, V = DZ, S^2 + C^2 = Z^2, aS^2 + D^2 = Z^2$. Then, $(S_1\colon C_1\colon D_1\colon Z_1\colon U_1\colon V_1) + (S_2\colon C_2\colon D_2\colon Z_2\colon U_2\colon V_2) = (S_3\colon C_3\colon D_3\colon Z_3\colon U_3\colon V_3)$ where S_3, C_3, D_3, and Z_3 are the same as the unified addition formulae in §5.3.1 and $U_3 = S_3C_3$, and $V_3 = D_3Z_3$. The unified addition in $\mathcal{I}^{m2}$ takes $11\mathbf{M} + 2\mathbf{S} + 5\mathbf{D} + 13\mathbf{a}$ saving $2\mathbf{M}$ in comparison to the unified addition in $\mathcal{I}$;

$$E \leftarrow C_1 \cdot Z_2, \quad F \leftarrow Z_1 \cdot C_2, \quad G \leftarrow S_1 \cdot D_2, \quad H \leftarrow D_1 \cdot S_2, \quad J \leftarrow F \cdot H,$$
$$K \leftarrow E \cdot G, \quad S_3 \leftarrow (E + F) \cdot (G + H) - J - K, \quad C_3 \leftarrow (E - bH) \cdot (G + F) + bJ - K,$$
$$D_3 \leftarrow (V_1 - aU_1) \cdot (U_2 + V_2) - J + aK, \quad Z_3 \leftarrow E^2 + bG^2, \quad U_3 \leftarrow S_3 \cdot C_3,$$
$$V_3 \leftarrow D_3 \cdot Z_3.$$

The unified addition with $b = 1$ in $\mathcal{I}^{m2}$ takes $11\mathbf{M} + 1\mathbf{S} + 2\mathbf{D} + 15\mathbf{a}$;

$$E \leftarrow S_1 \cdot D_2, \quad F \leftarrow C_1 \cdot Z_2, \quad G \leftarrow D_1 \cdot S_2, \quad H \leftarrow Z_1 \cdot C_2, \quad J \leftarrow U_1 \cdot V_2,$$
$$K \leftarrow V_1 \cdot U_2, \quad S_3 \leftarrow (H + F) \cdot (E + G) - J - K, \quad C_3 \leftarrow (H + E) \cdot (F - G) - J + K,$$
$$D_3 \leftarrow (V_1 - aU_1) \cdot (U_2 + V_2) + aJ - K, \quad Z_3 \leftarrow (H + G)^2 - 2K, \quad U_3 \leftarrow S_3 \cdot C_3,$$
$$V_3 \leftarrow D_3 \cdot Z_3.$$

5.3.3 Comparison and remarks

In §5.3.1 inversion-free arithmetic for Jacobi intersection form was studied in homogeneous projective coordinate system, $\mathcal{I}$. This subsection improved the previous results. Most notably, dedicated addition formulae were proposed for the first time using the affine addition laws (4.46), (4.47), and (4.48) from §4.5 of Chapter 4. As in §5.1 and §5.2, dedicated addition surpasses unified addition in performance despite the proposed improvements for unified additions. In particular, a dedicated addition in $\mathcal{I}$ takes $12\mathbf{M} + 11\mathbf{a}$ where a unified addition takes $13\mathbf{M} + 1\mathbf{S} + 2\mathbf{D} + 15\mathbf{a}$. Both types of additions are further sped up in §5.3.2. Since essentially the same affine point doubling formulae in §4.5 of Chapter 4 can be written in many different ways, it would be possible to substantially increase the number of doubling formulae in §5.3.1 and §5.3.2. This section has only presented the fastest ones (up to further improvements). This argument also applies to the case of additions.

Various operation counts in $\mathcal{I}$, $\mathcal{I}^{m1}$, and $\mathcal{I}^{m2}$ are summarized in Table 5.5.

If just unified addition is being accessed and the extra memory is available then $\mathcal{I}^{m2}$ is the fastest system. For speed oriented implementations $\mathcal{I}^{m1}$ gives the best timings. If memory is a concern then $\mathcal{I}$ is preferable since all operations are done in four coordinates and without any speed penalty for doublings.

Table 5.5: Operation counts for (twisted) Jacobi intersection form with $b = 1$ in different coordinate systems.

System	DBL	ADD
$\mathcal{I}$	3**M**+4**S** +6**a**, [BL07a]	13**M**+2**S**+1**D**+ 7**a**, unified, [LS01]
	2**M**+5**S**+1**D**+7**a**	13**M**+1**S**+2**D**+15**a**, unified
		12**M** +11**a**, dedicated
$\mathcal{I}^{m1}$	3**M**+4**S** +6**a**, *	11**M** + 9**a**, dedicated
	2**M**+5**S**+1**D**+7**a**	-
$\mathcal{I}^{m2}$	-	11**M**+1**S**+2**D**+15**a**, unified

*: Adapted from [BL07a, dbl-2007-bl].

Literature notes The first inversion-free doubling algorithm appears in [CC86] where Chudnovsky and Chudnovsky use (4.52), (4.53), and (4.54) in homogeneous projective coordinates and provide a $5\mathbf{M} + 3\mathbf{S} + 6\mathbf{a}$ doubling algorithm. Later in [LS01], Liardet and Smart improve the doubling to $4\mathbf{M} + 3\mathbf{S} + 7\mathbf{a}$ based on (4.43), (4.44), and (4.45). In [BL07a], Bernstein and Lange report an even faster $3\mathbf{M}+4\mathbf{S}+6\mathbf{a}$ algorithm based on the same formulae used by Liardet and Smart. The unified addition is reported to take $14\mathbf{M} + 2\mathbf{S} + 1\mathbf{D} + 4\mathbf{a}$ in [CC86]; $13\mathbf{M}+2\mathbf{S}+1\mathbf{D}+7\mathbf{a}$ in [LS01]. In the case $\#\mathbb{K} \equiv 3 \bmod 4$, Bernstein *et al.* [BBJ+08]. show how to exploit isogenies between the curves $v^2 = u^3 + 2(a + d)u^2 + (a - d)^2u$ and $v^2 = u^3 - (a + d)u^2 + adu$ in order to provide the speed of twisted Edwards curves to all elliptic curves with 3 points of order two and which are not birationally equivalent over $\mathbb{K}$ to any twisted Edwards curve. Upon the completion of the write-up of this section, Feng *et al.* [FNW09] have independently studied twisted Jacobi intersection curves. The inversion-free formulae presented in [FNW09] are essentially identical to the formulae in this thesis. On the other hand, the algorithms given in this section are superior in operation counts.

5.4 Twisted Hessian form

This section contains new optimizations for performing arithmetic in homogeneous projective coordinates. In addition, the extended homogeneous projective coordinate system is considered for twisted Hessian form for the first time. This section studies the arithmetic of twisted Hessian curves using homogeneous projective coordinates in §5.4.1 and extended homogeneous projective coordinates in §5.4.2. In §5.1 and §5.2, the mixed coordinates have been very useful for finding the best balance between point doublings and point additions. The same idea is not useful for twisted Hessian curves. On the other hand, each coordinate system on its own can be advantageous for different applications. Additional remarks, comparisons, pointers to literature and conclusions are given in §5.4.3.

5.4.1 Homogeneous projective coordinates, $\mathcal{H}$

In this system, each point (x, y) on $ax^3+y^3+1 = dxy$ is represented with the triplet $(X : Y : Z)$ which corresponds to the affine point $(X/Z, Y/Z)$ with $Z \neq 0$. These triplets satisfy the homogeneous projective equation $aX^3+Y^3+Z^3 = dXYZ$. For all nonzero $\lambda \in \overline{\mathbb{K}}$, $(X : Y : Z) = (\lambda X : \lambda Y : \lambda Z)$. The identity element is represented by $(0 : -1 : 1)$. The negative of $(X : Y : Z)$ is $(X : Z : Y)$. This coordinate system is denoted by $\mathcal{H}$ throughout the text.

Doubling in $\mathcal{H}$ Let $(X_1 : Y_1 : Z_1)$ with $Z_1 \neq 0$ satisfy $aX^3 + Y^3 + Z^3 = dXYZ$. Then $[2](X_1 : Y_1 : Z_1) = (X_3 : Y_3 : Z_3)$ where

$$\begin{aligned} X_3 &= X_1(Z_1^3 - Y_1^3), \\ Y_3 &= Z_1(Y_1^3 - aX_1^3), \\ Z_3 &= Y_1(aX_1^3 - Z_1^3) \end{aligned} \tag{5.24}$$

assuming $Z_3 \neq 0$. These formulae are obtained from (4.25) and (4.26). It was pointed out in [Ber06a] that Z_3 is always nonzero if a is a non-cube in $\mathbb{K}$. Also by Lemma 4.3.7, Z_3 is always nonzero if $(X_1 : Y_1 : Z_1)$ is of order r such that $3 \nmid r$, regardless of the assumption on a and d (of course, $a(27a - d^3) \neq 0$).

These formulae do not depend on d. Therefore keeping d arbitrary has no effect on the cost of (5.24). Evaluating (5.24) takes $6\mathbf{M} + 3\mathbf{S} + 1\mathbf{D} + 3\mathbf{a}$ in [BL07a];

$$\begin{gathered} A \leftarrow X_1^2, \quad B \leftarrow Y_1^2, \quad C \leftarrow Z_1^2, \quad D \leftarrow X_1 \cdot A, \quad E \leftarrow Y_1 \cdot B, \quad F \leftarrow Z_1 \cdot C, \\ G \leftarrow aD, \quad X_3 \leftarrow X_1 \cdot (E - F), \quad Y_3 \leftarrow Z_1 \cdot (G - E), \quad Z_3 \leftarrow Y_1 \cdot (F - G). \end{gathered}$$

If $a = 1$ then the algorithm takes $6\mathbf{M} + 3\mathbf{S} + 3\mathbf{a}$. For this case, it is also possible to exploit further optimizations. First of all, it is easy to see that X_3 can be written as $(X_1Z_1 - X_1Y_1)(Y_1^2 + Y_1Z_1 + Z_1^2)$. If $a = 1$ then the same strategy also applies to Y_3 and Z_3. Based on this observation it is possible to trade 3 multiplications with 3 squarings and several additions if desired. The following $3\mathbf{M} + 6\mathbf{S} + 18\mathbf{a}$ doubling algorithm exploits this idea;

$$\begin{gathered} A \leftarrow X_1^2, \quad B \leftarrow Y_1^2, \quad C \leftarrow Z_1^2, \quad D \leftarrow A + B, \quad E \leftarrow A + C, \quad F \leftarrow B + C, \\ G \leftarrow ((X_1 + Y_1)^2 - D)/2, \quad H \leftarrow ((X_1 + Z_1)^2 - E)/2, \quad J \leftarrow ((Y_1 + Z_1)^2 - F)/2, \\ X_3 \leftarrow (H - G) \cdot (F + J), \quad Y_3 \leftarrow (J - H) \cdot (D + G), \quad Z_3 \leftarrow (G - J) \cdot (E + H). \end{gathered}$$

Similar ideas apply to the case $a = -1$.

The previous doubling algorithms spend too much effort in squaring each of the input coordinates. As an alternative optimization idea the following algorithm squares only Y_1 and takes $7\mathbf{M} + 1\mathbf{S} + 8\mathbf{a}$ assuming that $a = 1$;

$$\begin{gathered} A \leftarrow Y_1^2, \quad B \leftarrow (Z_1 - Y_1) \cdot (A + (Y_1 + Z_1) \cdot Z_1), \\ C \leftarrow (Y_1 - X_1) \cdot (X_1 \cdot (X_1 + Y_1) + A), \quad X_3 \leftarrow X_1 \cdot B, \quad Y_3 \leftarrow Z_1 \cdot C, \\ Z_3 \leftarrow Y_1 \cdot (-B - C). \end{gathered}$$

Similar ideas apply to the case $a = -1$.

For simplicity assume $a = 1$. So, $\mathbf{D} = 0$. Note that the formulae do not depend on d. The ratios $\mathbf{S}/\mathbf{M}$ and $\mathbf{a}/\mathbf{M}$ then determine the fastest of these three doubling algorithms. The $3\mathbf{M} + 6\mathbf{S} + 15\mathbf{a}$ algorithm is the fastest if $\mathbf{a}$ is very negligible and if $\mathbf{S}$ is cheaper than $\mathbf{M}$. Specifically, this algorithm takes over the $6\mathbf{M} + 3\mathbf{S} + 3\mathbf{a}$ algorithm if $\mathbf{M} - \mathbf{S} > 4\mathbf{a}$. The

$7\mathbf{M}+1\mathbf{S}+8\mathbf{a}$ algorithm takes over the $6\mathbf{M}+3\mathbf{S}+3\mathbf{a}$ algorithm if $2\mathbf{S}-\mathbf{M}>5\mathbf{a}$ and takes over the $3\mathbf{M}+6\mathbf{S}+15\mathbf{a}$ algorithm if $4\mathbf{M}-5\mathbf{S}<7\mathbf{a}$. For instance, in an implementation where $\mathbf{S}=0.8\mathbf{M}$ and $\mathbf{a}=0.1\mathbf{M}$, these algorithms cost $8.7\mathbf{M}$, $9.3\mathbf{M}$, and $8.6\mathbf{M}$ (in the respective order the algorithms are introduced).

Dedicated addition in $\mathcal{H}$ Further let $(X_2\colon Y_2\colon Z_2)$ with $Z_2\neq 0$ satisfy $aX^3+Y^3+Z^3=dXYZ$. Then, $(X_1\colon Y_1\colon Z_1)+(X_2\colon Y_2\colon Z_2)=(X_3\colon Y_3\colon Z_3)$ where

$$\begin{aligned} X_3 &= X_1^2Y_2Z_2-Y_1Z_1X_2^2,\\ Y_3 &= Z_1^2X_2Y_2-X_1Y_1Z_2^2,\\ Z_3 &= Y_1^2X_2Z_2-X_1Z_1Y_2^2 \end{aligned} \tag{5.25}$$

assuming $Z_3\neq 0$. By Lemma 4.4.4, $Z_3\neq 0$ if $(X_1\colon Y_1\colon Z_1)\neq(X_2\colon Y_2\colon Z_2)$ and both summands are of order r such that $3\nmid r$.

The addition law (5.25) is in fact the same law which was attributed to Cauchy and Sylvester in [CC86]. The only difference is that the identity element is moved from a point at infinity to $(0\colon -1\colon 1)$ and remarkably (5.25) works for all twisted Hessian curves.

Evaluating (5.25) which is obtained from (4.29) and (4.30), takes $12\mathbf{M}+3\mathbf{a}$;

$$A\leftarrow X_1\cdot Z_2,\quad B\leftarrow Y_1\cdot Z_2,\quad C\leftarrow Z_1\cdot X_2,\quad D\leftarrow Z_1\cdot Y_2,\quad E\leftarrow X_1\cdot Y_2,$$
$$F\leftarrow Y_1\cdot X_2,\quad X_3\leftarrow A\cdot E-C\cdot F,\quad Y_3\leftarrow C\cdot D-A\cdot B,\quad Z_3\leftarrow B\cdot F-D\cdot E.$$

Additionally, if $Z_2=1$ then it takes $10\mathbf{M}+3\mathbf{a}$.

Alternatively, evaluating (5.25) takes $11\mathbf{M}+17\mathbf{a}$;

$$A\leftarrow X_1\cdot Z_2,\quad B\leftarrow Y_1\cdot Z_2,\quad C\leftarrow Z_1\cdot X_2,\quad D\leftarrow Z_1\cdot Y_2,\quad E\leftarrow X_1\cdot Y_2,$$
$$F\leftarrow Y_1\cdot X_2,\quad G\leftarrow (C+E)\cdot(A-F),\quad H\leftarrow (C-E)\cdot(A+F),$$
$$J\leftarrow (C+B)\cdot(A-D),\quad K\leftarrow (C-B)\cdot(A+D),\quad X_3\leftarrow G-H,\quad Y_3\leftarrow K-J,$$
$$Z_3\leftarrow J+K-G-H-2(E-B)\cdot(F+D).$$

Here note that all coordinates are multiplied by 2. This algorithm is better than the conventional $12\mathbf{M}+3\mathbf{a}$ approach when $14\mathbf{a}<1\mathbf{M}$.

Unified addition in $\mathcal{H}$ Alternatively, $(X_1\colon Y_1\colon Z_1)+(X_2\colon Y_2\colon Z_2)=(X_3\colon Y_3\colon Z_3)$ where

$$\begin{aligned} X_3 &= X_1Z_1Z_2^2-Y_1^2X_2Y_2,\\ Y_3 &= Y_1Z_1Y_2^2-aX_1^2X_2Z_2,\\ Z_3 &= aX_1Y_1X_2^2-Z_1^2Z_2Y_2 \end{aligned} \tag{5.26}$$

assuming $Z_3\neq 0$. It was pointed out in [Ber06a] that $Z_3\neq 0$ if a is a non-cube in $\mathbb{K}$. Also by Lemma 4.3.7, $Z_3\neq 0$ if $(X_1\colon Y_1\colon Z_1)$ and $(X_2\colon Y_2\colon Z_2)$ are of order r such that $3\nmid r$.

Evaluating (5.26) which is obtained from (4.31) and (4.32), takes $12\mathbf{M}+1\mathbf{D}+3\mathbf{a}$ in [BKL09];

$$A\leftarrow X_1\cdot Z_2,\quad B\leftarrow Z_1\cdot Z_2,\quad C\leftarrow Y_1\cdot X_2,\quad D\leftarrow Y_1\cdot Y_2,\quad E\leftarrow Z_1\cdot Y_2,$$
$$F\leftarrow aX_1\cdot X_2,\quad X_3\leftarrow A\cdot B-C\cdot D,\quad Y_3\leftarrow D\cdot E-F\cdot A,\quad Z_3\leftarrow F\cdot C-B\cdot E.$$

Additionally, if $Z_2=1$ then it takes $10\mathbf{M}+1\mathbf{D}+3\mathbf{a}$.

Using the same optimization that removes a multiplication from the dedicated addition algorithm, evaluating (5.26) takes $11\mathbf{M}+1\mathbf{D}+17\mathbf{a}$;

$$A \leftarrow X_1 \cdot Z_2, \quad B \leftarrow Z_1 \cdot Z_2, \quad C \leftarrow Y_1 \cdot X_2, \quad D \leftarrow Y_1 \cdot Y_2, \quad E \leftarrow Z_1 \cdot Y_2,$$
$$F \leftarrow aX_1 \cdot X_2, \quad G \leftarrow (D+B) \cdot (A-C), \quad H \leftarrow (D-B) \cdot (A+C),$$
$$J \leftarrow (D+F) \cdot (A-E), \quad K \leftarrow (D-F) \cdot (A+E), \quad X_3 \leftarrow G-H, \quad Y_3 \leftarrow K-J,$$
$$Z_3 \leftarrow J+K-G-H-2(B-F) \cdot (C+E).$$

5.4.2 Extended homogeneous projective coordinates, $\mathcal{H}^e$

In this system, each point (x, y) on $ax^3 + y^3 + 1 = dxy$ is represented with the sextuplet $(X : Y : R : S : T : Z)$ which corresponds to the affine point $(X/Z, Y/Z)$ with $Z \neq 0$. These sextuplets satisfy the homogeneous projective equation $aX^3+Y^3+Z^3 = dXYZ$. The auxiliary coordinates are defined by $R = XY/Z$, $S = X^2/Z$, and $T = Y^2/Z$. For all nonzero $\lambda \in \overline{\mathbb{K}}$, $(X : Y : R : S : T : Z) = (\lambda X : \lambda Y : \lambda R : \lambda S : \lambda T : \lambda Z)$. The identity element is represented by $(0 : -1 : 0 : 0 : 1 : 1)$. The negative of $(X : Y : R : S : T : Z)$ is $(R : Y : X : S : Z : T)$. This coordinate system is denoted by $\mathcal{H}^e$ throughout the text. Given $(X : Y : Z)$ in $\mathcal{H}$ passing to $\mathcal{H}^e$ can be performed in $6\mathbf{S}$ by computing $(XZ, YZ, XY, X^2, Y^2, Z^2)$. Given $(X : Y : R : S : T : Z)$ in $\mathcal{H}^e$ passing to $\mathcal{H}$ is cost-free by simply ignoring R, S, T.

Doubling in $\mathcal{H}^e$ Let $(X_1 : Y_1 : R_1 : S_1 : T_1 : Z_1)$ with $Z_1 \neq 0$, $R_1 = X_1Y_1/Z_1$, $S_1 = X_1^2/Z_1$, $T_1 = Y_1^2/Z_1$ satisfy $aX^3 + Y^3 + Z^3 = dXYZ$. Then, $[2](X_1 : Y_1 : R_1 : S_1 : T_1 : Z_1) = (X_3 : Y_3 : R_3 : S_3 : T_3 : Z_3)$ where

$$\begin{aligned}
X_3 &= (X_1Z_1 - R_1T_1)(aR_1S_1 - Y_1Z_1),\\
Y_3 &= (Y_1T_1 - aX_1S_1)(aR_1S_1 - Y_1Z_1),\\
R_3 &= (X_1Z_1 - R_1T_1)(Y_1T_1 - aX_1S_1),\\
S_3 &= (X_1Z_1 - R_1T_1)^2,\\
T_3 &= (Y_1T_1 - aX_1S_1)^2,\\
Z_3 &= (aR_1S_1 - Y_1Z_1)^2
\end{aligned} \tag{5.27}$$

assuming $Z_3 \neq 0$. By Lemma 4.3.7, $Z_3 \neq 0$ if $(X_1 : Y_1 : R_1 : S_1 : T_1 : Z_1)$ is of order r such that $3 \nmid r$.

Evaluating (5.27) which is obtained from (4.25) and (4.26), takes $9\mathbf{M} + 3\mathbf{S} + 1\mathbf{D} + 3\mathbf{a}$;

$$A \leftarrow aS_1, \quad B \leftarrow X_1 \cdot Z_1 - T_1 \cdot R_1, \quad C \leftarrow Y_1 \cdot T_1 - A \cdot X_1, \quad D \leftarrow R_1 \cdot A - Z_1 \cdot Y_1,$$
$$X_3 \leftarrow B \cdot D, \quad Y_3 \leftarrow C \cdot D, \quad R_3 \leftarrow B \cdot C, \quad S_3 \leftarrow B^2, \quad T_3 \leftarrow C^2, \quad Z_3 \leftarrow D^2.$$

If $\mathbf{a}$ is not very costly then doubling can be improved to $5\mathbf{M} + 6\mathbf{S} + 1\mathbf{D} + 29\mathbf{a}$;

$$A \leftarrow aS_1, \quad B \leftarrow (T_1 + Z_1) \cdot (X_1 - R_1), \quad C \leftarrow (T_1 - Z_1) \cdot (X_1 + R_1),$$
$$D \leftarrow (T_1 + A) \cdot (X_1 - Y_1), \quad E \leftarrow (T_1 - A) \cdot (X_1 + Y_1), \quad F \leftarrow B - C, \quad G \leftarrow E - D,$$
$$H \leftarrow D + E - B - C - 2(Z_1 - A) \cdot (R_1 + Y_1), \quad S_3 \leftarrow F^2, \quad T_3 \leftarrow G^2, \quad Z_3 \leftarrow H^2,$$
$$X_3 \leftarrow ((F+H)^2 - S_3 - Z_3)/2, \quad Y_3 \leftarrow ((G+H)^2 - T_3 - Z_3)/2,$$
$$R_3 \leftarrow ((F+G)^2 - S_3 - T_3)/2.$$

Note here that if the $S = X^2/Z$-coordinate of $\mathcal{H}^e$ is changed to $S = aX^2/Z$ then doubling can also be performed in $9\mathbf{M} + 3\mathbf{S} + 1\mathbf{D} + 3\mathbf{a}$ or $5\mathbf{M} + 6\mathbf{S} + 1\mathbf{D} + 29\mathbf{a}$.

Dedicated addition in $\mathcal{H}^e$ Further let $(X_2 : Y_2 : R_2 : S_2 : T_2 : Z_2)$ with $Z_2 \neq 0$, $R_2 = X_2Y_2/Z_2$, $S_2 = X_2^2/Z_2$, $T_2 = Y_2^2/Z_2$ satisfy $aX^3 + Y^3 + Z^3 = dXYZ$. Then,

$(X_1 : Y_1 : R_1 : S_1 : T_1 : Z_1) + (X_2 : Y_2 : R_2 : S_2 : T_2 : Z_2) = (X_3 : Y_3 : R_3 : S_3 : T_3 : Z_3)$ where

$$\begin{aligned} X_3 &= (S_1Y_2 - Y_1S_2)(T_1X_2 - X_1T_2), \\ Y_3 &= (Z_1R_2 - R_1Z_2)(T_1X_2 - X_1T_2), \\ R_3 &= (S_1Y_2 - Y_1S_2)(Z_1R_2 - R_1Z_2), \\ S_3 &= (S_1Y_2 - Y_1S_2)^2, \\ T_3 &= (Z_1R_2 - R_1Z_2)^2, \\ Z_3 &= (T_1X_2 - X_1T_2)^2 \end{aligned} \tag{5.28}$$

assuming $Z_3 \neq 0$. By Lemma 4.3.3, $Z_3 \neq 0$ if $(X_1 : Y_1 : R_1 : S_1 : T_1 : Z_1) \neq (X_2 : Y_2 : R_2 : S_2 : T_2 : Z_2)$ and both summands are of order r such that $3 \nmid r$.

Evaluating (5.28) which is obtained from (4.29) and (4.30), takes $9\mathbf{M} + 3\mathbf{S} + 3\mathbf{a}$;

$$A \leftarrow S_1 \cdot Y_2 - Y_1 \cdot S_2, \quad B \leftarrow Z_1 \cdot R_2 - R_1 \cdot Z_2, \quad C \leftarrow T_1 \cdot X_2 - X_1 \cdot T_2, \quad X_3 \leftarrow A \cdot C,$$
$$Y_3 \leftarrow B \cdot C, \quad R_3 \leftarrow A \cdot B, \quad S_3 \leftarrow A^2, \quad T_3 \leftarrow B^2, \quad Z_3 \leftarrow C^2.$$

Additionally, if $Z_2 = 1$ then it takes $8\mathbf{M} + 3\mathbf{S} + 3\mathbf{a}$.

In addition, three multiplications can be traded with three squarings and several additions if desired. The alternative evaluation takes $6\mathbf{M} + 6\mathbf{S} + 15\mathbf{a}$;

$$A \leftarrow S_1 \cdot Y_2 - Y_1 \cdot S_2, \quad B \leftarrow Z_1 \cdot R_2 - R_1 \cdot Z_2, \quad C \leftarrow T_1 \cdot X_2 - X_1 \cdot T_2, \quad S_3 \leftarrow A^2,$$
$$T_3 \leftarrow B^2, \quad Z_3 \leftarrow C^2, \quad X_3 \leftarrow ((A + C)^2 - S_3 - Z_3)/2,$$
$$Y_3 \leftarrow ((B + C)^2 - T_3 - Z_3)/2, \quad R_3 \leftarrow ((A + B)^2 - S_3 - T_3)/2.$$

Additionally, if $Z_2 = 1$ then it takes $5\mathbf{M} + 6\mathbf{S} + 15\mathbf{a}$.

Unified addition in $\mathcal{H}^e$ Alternatively, $(X_1 : Y_1 : R_1 : S_1 : T_1 : Z_1) + (X_2 : Y_2 : R_2 : S_2 : T_2 : Z_2) = (X_3 : Y_3 : R_3 : S_3 : T_3 : Z_3)$ where

$$\begin{aligned} X_3 &= (X_1Z_2 - T_1R_2)(aR_1S_2 - Z_1Y_2), \\ Y_3 &= (Y_1T_2 - aS_1X_2)(aR_1S_2 - Z_1Y_2), \\ R_3 &= (X_1Z_2 - T_1R_2)(Y_1T_2 - aS_1X_2), \\ S_3 &= (X_1Z_2 - T_1R_2)^2, \\ T_3 &= (Y_1T_2 - aS_1X_2)^2, \\ Z_3 &= (aR_1S_2 - Z_1Y_2)^2 \end{aligned} \tag{5.29}$$

assuming $Z_3 \neq 0$. It can be deduced from [BBJ+08] that $Z_3 \neq 0$ if a is a square in $\mathbb{K}$ and d is not a square in $\mathbb{K}$. Also by Lemma 4.3.7, $Z_3 \neq 0$ if $(X_1 : Y_1 : R_1 : S_1 : T_1 : Z_1)$ and $(X_2 : Y_2 : R_2 : S_2 : T_2 : Z_2)$ are of order r such that $3 \nmid r$.

Evaluating (5.29) which is obtained from (4.31) and (4.32), takes $9\mathbf{M} + 3\mathbf{S} + 2\mathbf{D} + 3\mathbf{a}$;

$$A \leftarrow X_1 \cdot Z_2 - T_1 \cdot R_2, \quad B \leftarrow Y_1 \cdot T_2 - aS_1 \cdot X_2, \quad C \leftarrow aR_1 \cdot S_2 - Z_1 \cdot Y_2,$$
$$X_3 \leftarrow A \cdot C, \quad Y_3 \leftarrow B \cdot C, \quad R_3 \leftarrow A \cdot B, \quad S_3 \leftarrow A^2, \quad T_3 \leftarrow B^2, \quad Z_3 \leftarrow C^2.$$

Additionally, if $Z_2 = 1$ then it takes $8\mathbf{M} + 3\mathbf{S} + 2\mathbf{D} + 3\mathbf{a}$.

In addition, three multiplications can be traded with 3 squarings and several additions if desired. The alternative evaluation takes $6\mathbf{M} + 6\mathbf{S} + 2\mathbf{D} + 15\mathbf{a}$;

$$A \leftarrow X_1 \cdot Z_2 - T_1 \cdot R_2, \quad B \leftarrow Y_1 \cdot T_2 - aS_1 \cdot X_2, \quad C \leftarrow aR_1 \cdot S_2 - Z_1 \cdot Y_2, \quad S_3 \leftarrow A^2,$$
$$T_3 \leftarrow B^2, \quad Z_3 \leftarrow C^2, \quad X_3 \leftarrow ((A+C)^2 - S_3 - Z_3)/2,$$
$$Y_3 \leftarrow ((B+C)^2 - T_3 - Z_3)/2, \quad R_3 \leftarrow ((A+B)^2 - S_3 - T_3)/2.$$

Additionally, if $Z_2 = 1$ then it takes $5\mathbf{M} + 6\mathbf{S} + 2\mathbf{D} + 15\mathbf{a}$.

Note here that if the $S = X^2/Z$-coordinate of $\mathcal{H}^e$ is changed to $S = aX^2/Z$ then the unified addition naturally takes $9\mathbf{M} + 3\mathbf{S} + 1\mathbf{D} + 3\mathbf{a}$ or $6\mathbf{M} + 6\mathbf{S} + 1\mathbf{D} + 15\mathbf{a}$ eliminating one multiplication by a.

5.4.3 Comparison and remarks

In 5.4.1, inversion-free arithmetic for twisted Hessian form is studied in homogeneous projective coordinates, $\mathcal{H}$. This subsection reviewed the previous results and has added new and potentially more efficient results to the current body of knowledge. In particular, the proposed unified addition algorithm takes $11\mathbf{M} + 1\mathbf{D} + 17\mathbf{a}$ where an earlier algorithm in [BKL09] takes $12\mathbf{M} + 1\mathbf{D} + 3\mathbf{a}$. The new algorithm is faster when $1\mathbf{M} > 14\mathbf{a}$. As well, dedicated addition formulae are derived which eliminate multiplication by a. The proposed dedicated addition algorithms take $12\mathbf{M} + 3\mathbf{a}$ and $11\mathbf{M} + 17\mathbf{a}$. Moreover, in the case $a = 1$, it is shown that the $6\mathbf{M} + 3\mathbf{S} + 1\mathbf{D} + 3\mathbf{a}$ doubling algorithm in [BKL09] can be performed in $3\mathbf{M} + 6\mathbf{S} + 15\mathbf{a}$ or even $7\mathbf{M} + 1\mathbf{S} + 8\mathbf{a}$.

In §5.4.2, three additional coordinates are integrated into $\mathcal{H}$. The new system is denoted by $\mathcal{H}^e$. In this system, it is shown that the additions can benefit from three $\mathbf{M}/\mathbf{S}$ trade-offs without any additional cost. In particular, the new unified addition takes $9\mathbf{M} + 3\mathbf{S} + 2\mathbf{D} + 3\mathbf{a}$ which can be further improved to $9\mathbf{M} + 3\mathbf{S} + 1\mathbf{D} + 3\mathbf{a}$, see §5.4.2. This is an improvement over the $12\mathbf{M} + 1\mathbf{D} + 3\mathbf{a}$ algorithm in $\mathcal{H}$. In the case $\mathbf{a}$ is practically negligible, a $6\mathbf{M} + 6\mathbf{S} + 1\mathbf{D} + 15\mathbf{a}$ unified addition algorithm is proposed. This algorithm is faster than the proposed $11\mathbf{M} + 1\mathbf{D} + 17\mathbf{a}$ unified addition algorithm in $\mathcal{H}$. Similar arguments apply to the proposed dedicated addition.

Various operation counts in $\mathcal{H}$ and $\mathcal{H}^e$ are summarized in Table 5.6. Table 5.7 provides operation counts for the case $a = 1$.

The determination of the optimum system and algorithms for Hessian form is very sensitive to the changes in $\mathbf{S}/\mathbf{M}$, $\mathbf{a}/\mathbf{M}$ and target applications.

Clearly, the doublings in $\mathcal{H}^e$ are quite inefficient in comparison to $\mathcal{H}$ and switching between $\mathcal{H}$ and $\mathcal{H}^e$ is costly. Therefore, $\mathcal{H}^e$ is not suitable for speed oriented applications. On the other hand, the unified addition in $\mathcal{H}^e$ for arbitrary non-cube a is still interesting if just unified additions are being accessed, the cost of $\mathbf{a}$ and $\mathbf{D}$ are negligible, and $\mathbf{S}$ is much cheaper than $\mathbf{M}$. Other than these situations, $\mathcal{H}$ is better in performance. In addition, $\mathcal{H}$ needs three coordinates rather than six. If $7\mathbf{M} + 1\mathbf{S} + 8\mathbf{a}$ or $3\mathbf{M} + 6\mathbf{S} + 15\mathbf{a}$ doubling algorithms in Table 5.7 are desired for efficient implementation then parameter a should be rescaled to or selected as 1. In this case, the completeness will always be lost. On the other hand, possible incorrect outputs can be prevented by selecting the inputs from a suitable subgroup, see §5.4.1.

Literature notes The inversion-free arithmetic on Hessian curves $x^3 + y^3 + 1 = dxy$ was first studied by Chudnovsky and Chudnosky [CC86] with the addition laws attributed to Cauchy

Table 5.6: Operation counts for twisted Hessian form in different coordinate systems.

System	DBL	ADD
$\mathcal{H}$	6**M**+3**S**+1**D**+ 3**a**, [BKL09]	12**M** +1**D**+ 3**a**, unified, [BKL09]
		11**M** +1**D**+17**a**, unified
		12**M** + 3**a**, dedicated
		11**M** +17**a**, dedicated
$\mathcal{H}^e$	9**M**+3**S**+1**D**+ 3**a**	9**M**+3**S**+2**D**+ 3**a**, unified
		9**M**+3**S** + 3**a**, dedicated
	5**M**+6**S**+1**D**+29**a**	6**M**+6**S**+2**D**+15**a**, unified
		6**M**+6**S** +15**a**, dedicated

Table 5.7: Operation counts for (twisted) Hessian form with $a = 1$ in different coordinate systems.

System	DBL	ADD
$\mathcal{H}$	6**M**+3**S**+ 3**a**, [BKL09]	12**M** + 3**a**, unified, [BKL09]
	7**M**+1**S**+ 8**a**	11**M** +17**a**, unified
	3**M**+6**S**+18**a**	12**M** + 3**a**, dedicated
		11**M** +17**a**, dedicated
$\mathcal{H}^e$	9**M**+3**S**+ 3**a**	9**M**+3**S**+ 3**a**, unified
		9**M**+3**S**+ 3**a**, dedicated
	5**M**+6**S**+29**a**	6**M**+6**S**+15**a**, unified
		6**M**+6**S**+15**a**, dedicated

and Sylvester. Smart [Sma01] presented a 3-way parallel implementation of the dedicated addition on the curve $X^3 + Y^3 + Z^3 = dXYZ$. Joye and Quisquater [JQ01] showed how to use the dedicated addition formulae for point doubling by a permutation of coordinates and suggested using these curves for side channel resistant applications. At that time addition on Hessian curves was the fastest among all studied curve models. In all of these papers the identity element is a point other than $(0\colon -1\colon 1)$. In this section, [BKL09] was followed in order to benefit from a simplified concept of complete additions on the twisted Hessian curve $aX^3 + Y^3 + Z^3 = dXYZ$. Therefore, in all algorithms the identity element is moved to $(0\colon -1\colon 1)$.

5.5 Short Weierstrass form

In this section, short Weierstrass curves are reviewed with three coordinate systems. Most of the results are directly taken from the literature for the completeness of the discussion. The proposed improvements appear in unified addition.

5.5.1 Homogeneous projective coordinates, $\mathcal{P}$

In this system, each point (x, y) on $y^2 = x^3 + ax + b$ is represented with the triplet $(X : Y : Z)$ which corresponds to the affine point $(X/Z, Y/Z)$ with $Z \neq 0$. These triplets satisfy the homogeneous projective equation $Y^2Z = X^3 + aXZ^2 + bZ^3$. The identity element is represented by $(0 : 1 : 0)$. The negative of $(X : Y : Z)$ is $(X : -Y : Z)$. For all nonzero $\lambda \in \overline{\mathbb{K}}$, $(X : Y : Z) = (\lambda X : \lambda Y : \lambda Z)$. This coordinate system is denoted by $\mathcal{P}$ throughout the text.

Doubling in $\mathcal{P}$ Let $(X_1 : Y_1 : Z_1)$ with $Z_1 \neq 0$ satisfy $Y^2Z = X^3 + aXZ^2 + bZ^3$. Then $[2](X_1 : Y_1 : Z_1) = (X_3 : Y_3 : Z_3)$ where

$$\begin{aligned} X_3 &= 2Y_1Z_1((3X_1^2 + aZ_1^2)^2 - 8Y_1^2X_1Z_1), \\ Y_3 &= (3X_1^2 + aZ_1^2)(12Y_1^2X_1Z_1 - (3X_1^2 + aZ_1^2)^2) - 8Y_1^4Z_1^2, \\ Z_3 &= 8Y_1^3Z_1^3 \end{aligned} \tag{5.30}$$

assuming $Z_3 \neq 0$.

Evaluating (5.30) which is obtained from (4.1) and (4.2), takes $5\mathbf{M} + 6\mathbf{S} + 1\mathbf{D} + 11\mathbf{a}$ in [BL07a, `dbl-2007-bl`];

$$\begin{gathered} A \leftarrow 2Y_1 \cdot Z_1, \quad B \leftarrow Z_1^2, \quad C \leftarrow X_1^2, \quad D \leftarrow 4C + aB, \quad E \leftarrow Y_1 \cdot A, \quad F \leftarrow E^2, \\ G \leftarrow (X_1 + E)^2 - C - F, \quad H \leftarrow D^2 - 2G, \quad X_3 \leftarrow H \cdot A, \quad Y_3 \leftarrow D \cdot (G - H) - 2F, \\ Z_3 \leftarrow A \cdot A^2. \end{gathered}$$

If $a = -3$ then it takes $7\mathbf{M} + 3\mathbf{S} + 10\mathbf{a}$ in [BL07a, `dbl-2007-bl-2`];

$$\begin{gathered} A \leftarrow 3(X_1 - Z_1) \cdot (X_1 + Z_1), \quad B \leftarrow 2Y_1 \cdot Z_1, \quad C \leftarrow B^2, \quad D \leftarrow Y_1 \cdot B, \quad E \leftarrow D^2, \\ F \leftarrow 2X_1 \cdot D, \quad G \leftarrow A^2 - 2F, \quad X_3 \leftarrow B \cdot G, \quad Y_3 \leftarrow A \cdot (F - G) - 2E, \quad Z_3 \leftarrow B \cdot C. \end{gathered}$$

Dedicated addition in $\mathcal{P}$ Further let $(X_2 : Y_2 : Z_2)$ with $Z_2 \neq 0$ satisfy $Y^2Z = X^3 + aXZ^2 + bZ^3$. Then, $(X_1 : Y_1 : Z_1) + (X_2 : Y_2 : Z_2) = (X_3 : Y_3 : Z_3)$ where

$$\begin{aligned} X_3 &= (X_1Z_2 - Z_1X_2)(Z_1Z_2(Y_1Z_2 - Z_1Y_2)^2 - (X_1Z_2 + Z_1X_2)(X_1Z_2 - Z_1X_2)^2), \\ Y_3 &= (Y_1Z_2 - Z_1Y_2)((2X_1Z_2 + Z_1X_2)(X_1Z_2 - Z_1X_2)^2 - Z_1Z_2(Y_1Z_2 - Z_1Y_2)^2) - \\ &\quad Y_1Z_2(X_1Z_2 - Z_1X_2)^3, \\ Z_3 &= Z_1Z_2(X_1Z_2 - Z_1X_2)^3 \end{aligned} \tag{5.31}$$

assuming $Z_3 \neq 0$.

Evaluating (5.31) which is obtained from (4.3) and (4.4), takes $12\mathbf{M} + 2\mathbf{S} + 7\mathbf{a}$ in [CMO98];

$$\begin{gathered} A \leftarrow Z_1 \cdot Z_2, \quad B \leftarrow X_1 \cdot Z_2, \quad C \leftarrow Y_1 \cdot Z_2, \quad D \leftarrow B - Z_1 \cdot X_2, \quad E \leftarrow C - Z_1 \cdot Y_2, \\ F \leftarrow D^2, \quad G \leftarrow D \cdot F, \quad H \leftarrow F \cdot B, \quad J \leftarrow E^2 \cdot A + G - 2H, \quad X_3 \leftarrow D \cdot J, \\ Y_3 \leftarrow E \cdot (H - J) - G \cdot C, \quad Z_3 \leftarrow A \cdot G. \end{gathered}$$

Additionally, if $Z_2 = 1$ then it takes $9\mathbf{M} + 2\mathbf{S} + 7\mathbf{a}$, see [CMO98].

Unified addition in $\mathcal{P}$ Alternatively, $(X_1 : Y_1 : Z_1) + (X_2 : Y_2 : Z_2) = (X_3 : Y_3 : Z_3)$ where

$$\begin{aligned} X_3 &= Z_1Z_2(Y_1Z_2 + Z_1Y_2)(2((X_1Z_2 + Z_1X_2)^2 - Z_1Z_2(X_1X_2 - aZ_1Z_2))^2 - \\ &\quad 2Z_1Z_2(X_1Z_2 + Z_1X_2)(Y_1Z_2 + Z_1Y_2)^2), \\ Y_3 &= ((X_1Z_2 + Z_1X_2)^2 - Z_1Z_2(X_1X_2 - aZ_1Z_2))(3Z_1Z_2(X_1Z_2 + Z_1X_2)(Y_1Z_2 + Z_1Y_2)^2 - \\ &\quad 2((X_1Z_2 + Z_1X_2)^2 - Z_1Z_2(X_1X_2 - aZ_1Z_2))^2) - Z_1^2Z_2^2(Y_1Z_2 + Z_1Y_2)^4, \\ Z_3 &= 2Z_1^3Z_2^3(Y_1Z_2 + Z_1Y_2)^3 \end{aligned} \tag{5.32}$$

assuming $Z_3 \neq 0$.

Evaluating (5.32) which is obtained from (4.5) and (4.6), takes $11\mathbf{M} + 5\mathbf{S} + 1\mathbf{D} + 16\mathbf{a}$;

$$\begin{gathered} A \leftarrow X_1 \cdot X_2, \quad B \leftarrow Z_1 \cdot Z_2, \quad C \leftarrow (X_1 + Z_1) \cdot (X_2 + Z_2) - A - B, \\ D \leftarrow Y_1 \cdot Z_2 + Z_1 \cdot Y_2, \quad E \leftarrow B \cdot D, \quad F \leftarrow E \cdot D, \quad G \leftarrow C^2, \quad H \leftarrow F^2, \\ J \leftarrow G - B \cdot (A - aB), \quad K \leftarrow ((C + F)^2 - G - H)/2, \quad L \leftarrow 2(J^2 - K), \\ X_3 \leftarrow E \cdot L, \quad Y_3 \leftarrow J \cdot (K - L) - H, \quad Z_3 \leftarrow 2E \cdot E^2. \end{gathered}$$

Additionally, if $Z_2 = 1$ then it takes $9\mathbf{M} + 5\mathbf{S} + 1\mathbf{D} + 16\mathbf{a}$.

5.5.2 Jacobian coordinates, $\mathcal{J}$

In this system, each point (x, y) on $y^2 = x^3 + ax + b$ is represented with the triplet $(X : Y : Z)$ which corresponds to the affine point $(X/Z^2, Y/Z^3)$ with $Z \neq 0$. These triplets satisfy the weighted projective equation $Y^2 = X^3 + aXZ^4 + bZ^6$. The identity element is represented by $(1 : 1 : 0)$. The negative of $(X : Y : Z)$ is $(X : -Y : Z)$. For all nonzero $\lambda \in \mathbb{K}$, $(X : Y : Z) = (\lambda^2 X : \lambda^3 Y : \lambda Z)$. This coordinate system is denoted by $\mathcal{J}$ throughout the text.

Doubling in $\mathcal{J}$ Let $(X_1 : Y_1 : Z_1)$ with $Z_1 \neq 0$ satisfy $Y^2 = X^3 + aXZ^4 + bZ^6$. Then $[2](X_1 : Y_1 : Z_1) = (X_3 : Y_3 : Z_3)$ where

$$\begin{aligned} X_3 &= (3X_1^2 + aZ_1^4)^2 - 8X_1Y_1^2, \\ Y_3 &= (3X_1^2 + aZ_1^4)(4X_1Y_1^2 - X_3) - 8Y_1^4, \\ Z_3 &= 2Y_1Z_1 \end{aligned} \tag{5.33}$$

assuming $Z_3 \neq 0$.

Evaluating (5.33) which is obtained from (4.1) and (4.2), takes $1\mathbf{M} + 8\mathbf{S} + 1\mathbf{D} + 14\mathbf{a}$ in [BL07a, `dbl-2007-bl`];

$$\begin{gathered} A \leftarrow Z_1^2, \quad B \leftarrow Y_1^2, \quad C \leftarrow X_1^2, \quad D \leftarrow B^2, \quad E \leftarrow 2((X_1 + B)^2 - C - D), \\ F \leftarrow 3C + aA^2, \quad X_3 \leftarrow F^2 - 2E, \quad Y_3 \leftarrow F \cdot (E - X_3) - 8D, \\ Z_3 \leftarrow (Y_1 + Z_1)^2 - B - A. \end{gathered}$$

If $a = -3$ then it takes $3\mathbf{M} + 5\mathbf{S} + 12\mathbf{a}$ in [BL07a, `dbl-2001-b`];

$$\begin{gathered} A \leftarrow Z_1^2, \quad B \leftarrow Y_1^2, \quad C \leftarrow X_1 \cdot B, \quad D \leftarrow 3(X_1 - A) \cdot (X_1 + A), \quad X_3 \leftarrow D^2 - 8C, \\ Y_3 \leftarrow D \cdot (4C - X_3) - 8B^2, \quad Z_3 \leftarrow (Y_1 + Z_1)^2 - B - A. \end{gathered}$$

Dedicated addition in $\mathcal{J}$ Further let $(X_2 : Y_2 : Z_2)$ with $Z_2 \neq 0$ satisfy $Y^2 = X^3 + aXZ^4 + bZ^6$. Then, $(X_1 : Y_1 : Z_1) + (X_2 : Y_2 : Z_2) = (X_3 : Y_3 : Z_3)$ where

$$\begin{aligned} X_3 &= (Y_1Z_2^3 - Z_1^3Y_2)^2 - (X_1Z_2^2 + Z_1^2X_2)(X_1Z_2^2 - Z_1^2X_2)^2, \\ Y_3 &= (Y_1Z_2^3 - Z_1^3Y_2)(Z_1^2X_2(X_1Z_2^2 - Z_1^2X_2)^2 - X_3) - Z_1^3Y_2(X_1Z_2^2 - Z_1^2X_2)^3, \\ Z_3 &= Z_1Z_2(X_1Z_2^2 - Z_1^2X_2) \end{aligned} \tag{5.34}$$

assuming $Z_3 \neq 0$.

Evaluating (5.34) which is obtained from (4.3) and (4.4), takes $11\mathbf{M} + 5\mathbf{S} + 11\mathbf{a}$ in [BL07a, add-2007-bl];

$$\begin{gathered} U_2 \leftarrow Z_2^2, \quad V_2 \leftarrow U_2 \cdot Z_2, \quad U_1 \leftarrow Z_1^2, \quad V_1 \leftarrow U_1 \cdot Z_1, \quad A \leftarrow U_1 \cdot X_2, \quad B \leftarrow V_1 \cdot Y_2, \\ C \leftarrow X_1 \cdot U_2 - A, \quad D \leftarrow Y_1 \cdot V_2 - B, \quad E \leftarrow C^2, \quad F \leftarrow C \cdot E, \quad G \leftarrow A \cdot E, \\ X_3 \leftarrow D^2 - F - 2G, \quad Y_3 \leftarrow D \cdot (G - X_3) - B \cdot F, \\ Z_3 \leftarrow C \cdot ((Z_1 + Z_2)^2 - U_1 - U_2)/2. \end{gathered}$$

A $7\mathbf{M} + 4\mathbf{S} + 11\mathbf{a}$ algorithm is provided in [BL07a, madd-2007-bl] for the case $Z_2 = 1$.

Unified addition in $\mathcal{J}$ Alternatively, $(X_1 : Y_1 : Z_1) + (X_2 : Y_2 : Z_2) = (X_3 : Y_3 : Z_3)$ where

$$\begin{aligned} X_3 &= ((X_1Z_2^2 + Z_1^2X_2)^2 - Z_1^2Z_2^2(X_1X_2 - aZ_1^2Z_2^2))^2 - (X_1Z_2^2 + Z_1^2X_2)(Y_1Z_2^3 + Z_1^3Y_2)^2, \\ Y_3 &= \frac{1}{2}(((X_1Z_2^2 + Z_1^2X_2)^2 - Z_1^2Z_2^2(X_1X_2 - aZ_1^2Z_2^2)) \cdot \\ &\quad ((X_1Z_2^2 + Z_1^2X_2)(Y_1Z_2^3 + Z_1^3Y_2)^2 - 2X_3) - (Y_1Z_2^3 + Z_1^3Y_2)^4), \\ Z_3 &= Z_1Z_2(Y_1Z_2^3 + Z_1^3Y_2) \end{aligned} \tag{5.35}$$

assuming $Z_3 \neq 0$.

Evaluating (5.35) which is obtained from (4.5) and (4.6), takes $8\mathbf{M} + 10\mathbf{S} + 1\mathbf{D} + 24\mathbf{a}$;

$$\begin{gathered} U_2 \leftarrow Z_2^2, \quad V_2 \leftarrow U_2 \cdot Z_2, \quad U_1 \leftarrow Z_1^2, \quad V_1 \leftarrow U_1 \cdot Z_1, \quad A \leftarrow X_1 \cdot X_2, \\ B \leftarrow ((Z_1 + Z_2)^2 - U_1 - U_2)/2, \quad C \leftarrow B^2, \quad D \leftarrow (X_1 + U_1) \cdot (X_2 + U_2) - A - C, \\ E \leftarrow Y_1 \cdot V_2 + V_1 \cdot Y_2, \quad F \leftarrow E^2, \quad G \leftarrow D^2, \quad H \leftarrow F^2, \quad J \leftarrow G - C \cdot (A - aC), \\ K \leftarrow ((F + D)^2 - G - H)/2, \quad X_3 \leftarrow J^2 - K, \quad Y_3 \leftarrow (J \cdot (K - 2X_3) - H)/2, \\ Z_3 \leftarrow ((B + E)^2 - C - F)/2. \end{gathered}$$

Additionally, if $Z_2 = 1$ then it takes $6\mathbf{M} + 8\mathbf{S} + 1\mathbf{D} + 24\mathbf{a}$ by deleteing $B \leftarrow ((Z_1 + Z_2)^2 - U_1 - U_2)/2$ and by replacing $C \leftarrow B^2$ with $C \leftarrow Z_1^2$.

Chudnovsky Jacobian coordinates, $\mathcal{J}^c$ This system was introduced by Chudnovsky and Chudnovsky in [CC86]. In this system, each triplet in $\mathcal{J}$ is represented as $(X : Y : Z : Z^2 : Z^3)$. which saves time in additions however doublings are much slower. See [BL07b, §5] for further explanations and simplification of the concept by using re-additions. Further details are omitted here.

It is interesting to note that the proposed unified addition algorithm for Jacobian coordinates takes only $7\mathbf{M} + 9\mathbf{S} + 1\mathbf{D} + 24\mathbf{a}$ in Chudnovsky Jacobian coordinates.

Modified Jacobian coordinates, $\mathcal{J}^m$ This system was introduced by Cohen *et al.* in [CMO98]. In this system, each triplet in $\mathcal{J}$ is represented as $(X : Y : Z : aZ^4)$. This

representation saves time for arbitrary a in repeated doublings. On the other hand, the performance is identical to $\mathcal{J}$ when $a = -3$. Therefore, further details are omitted here.

Mixed coordinates, $\mathcal{J}^x$ Cohen *et al.* also suggest mixing different variants of Jacobian coordinates to obtain the optimum performance. See [CMO98] for details.

5.5.3 Comparison and remarks

Short Weierstrass curves are the most studied curve model in cryptology. There are several reasons for this:

- The short Weierstrass form, $y^2 = x^3 + ax + b$, covers all elliptic curves of large characteristic.
- The proposal of ECC was made using the short Weierstrass form, see [Mil86] and [Kob87].
- Several standards recommend (some even enforce) the use of short Weierstrass form in ECC applications.
- In affine coordinates, this form has the fastest point addition and point doubling formulae to date.
- Mixed Jacobian coordinates had been the fastest system in carrying out inversion-free point addition and point doubling for decades (ignoring the Montgomery ladder algorithm). The situation, however, changed after a sequence of results between 2007 and 2009. See §5.1, §5.2, §5.3, §5.4 in this chapter for details.

In this section, the most efficient formulae are brought together. EFD [BL07a] also displays all of these formulae. The contributions from this work appear in the case of unified addition. Previously, unified additions were only considered in homogeneous project coordinates, $\mathcal{P}$. In particular, Brier/Joye unified addition algorithm proposed in [BJ02] takes $12\mathbf{M}+5\mathbf{S}+1\mathbf{D}+10\mathbf{a}$. Bernstein and Lange announced $11\mathbf{M} + 6\mathbf{S} + 1\mathbf{D} + 15\mathbf{a}$ variant of the same algorithm in [BL07a]. The proposed unified addition algorithm in §5.5.1 takes $11\mathbf{M} + 5\mathbf{S} + 1\mathbf{D} + 16\mathbf{a}$. The new algorithm overtakes Brier/Joye algorithm when $\mathbf{M} > 6\mathbf{a}$ and is always faster than Bernstein/Lange algorithm.

Unified addition was never previously studied in Jacobian coordinates. In §5.5.1, a new unified addition algorithm is proposed in $\mathcal{J}$ which takes $8\mathbf{M}+10\mathbf{S}+1\mathbf{D}+24\mathbf{a}$. This algorithm takes $7\mathbf{M}+9\mathbf{S}+1\mathbf{D}+24\mathbf{a}$ in Chudnovsky Jacobian coordinates. If the extra space for storing Z^2 and Z^3 coordinates of Chudnovsky Jacobian coordinates is not an issue then the new unified addition algorithm is faster than unified addition in homogeneous projective coordinates by several $\mathbf{M}/\mathbf{S}$ trade-offs.

Various operation counts including the present contributions are summarized in Table 5.8 and Table 5.9.

It is already concluded in many works that the optimum system is $\mathcal{J}^x$ for arbitrary a in speed implementations. In the case $a = -3$, it is better to stay in $\mathcal{J}$ and use re-additions which correspond to addition timings in $\mathcal{J}^c$.

Table 5.8: Operation counts for short Weierstrass form in different coordinate systems.

System	DBL	ADD
$\mathcal{P}$, [CC86]	5**M**+6**S**+1**D**+11**a**, [BL07a]	12**M**+ 5**S**+1**D**+10**a**, unified, [BJ02]
		11**M**+ 6**S**+1**D**+15**a**, unified, [BL07a]
		11**M**+ 5**S**+1**D**+16**a**, unified
		12**M**+ 2**S** + 7**a**, dedicated, [CMO98]
$\mathcal{J}$, [CC86]	3**M**+6**S**+1**D**+ 9**a**, [HNM98]	8**M**+10**S**+1**D**+24**a**, unified
	1**M**+8**S**+1**D**+14**a**, [BL07a]	12**M**+ 4**S** + 7**a**, dedicated, [CMO98]
		11**M**+ 5**S** +11**a**, dedicated, [BL07a]
$\mathcal{J}^c$, [CC86]	4**M**+6**S**+1**D**+ 4**a**, [CMO98]	7**M**+ 9**S**+1**D**+24**a**, unified
		11**M**+ 3**S** + 7**a**, dedicated, [CMO98]
		10**M**+ 4**S** +13**a**, dedicated, [BL07a]
$\mathcal{J}^m$, [CMO98]	4**M**+4**S** +10**a**, [CMO98]	12**M**+ 6**S**+1**D**+ 7**a**, dedicated, [CMO98]
	3**M**+5**S** +13**a**, [BL07a]	11**M**+ 7**S**+1**D**+13**a**, dedicated, [BL07a]
$\mathcal{J}^x$, [CMO98]	4**M**+4**S** +10**a**, [CMO98]	10**M**+ 5**S**+1**D**+ 7**a**, dedicated, *
	3**M**+5**S** +13**a**, [BL07a]	8**M**+ 7**S**+1**D**+14**a**, dedicated, *

* Effective

Table 5.9: Operation counts for short Weierstrass form with $a = -3$ in different coordinate systems.

System	DBL	ADD
$\mathcal{P}$, [CC86]	7**M**+3**S**+10**a**, [BL07a]	12**M**+ 5**S**+1**D**+10**a**, unified, [BJ02]
		11**M**+ 6**S**+1**D**+15**a**, unified, [BL07a]
		11**M**+ 5**S**+1**D**+16**a**, unified
		12**M**+ 2**S** + 7**a**, dedicated, [CMO98]
$\mathcal{J}$, [CC86]	4**M**+4**S**+ 9**a**, [HMV03]	8**M**+10**S**+1**D**+24**a**, unified
	3**M**+5**S**+12**a**, [BL07a]	12**M**+ 4**S** + 7**a**, dedicated, [CMO98]
		11**M**+ 5**S** +11**a**, dedicated, [BL07a]
$\mathcal{J}^c$, [CC86]	4**M**+6**S**+ 4**a**, [CMO98]	7**M**+ 9**S**+1**D**+24**a**, unified
		11**M**+ 3**S** + 7**a**, dedicated, [CMO98]
		10**M**+ 4**S** +13**a**, dedicated, [BL07a]
$\mathcal{J}^m$, [CMO98]	4**M**+4**S**+10**a**, [CMO98]	12**M**+ 6**S**+1**D**+ 7**a**, dedicated, [CMO98]
	3**M**+5**S**+13**a**, [BL07a]	11**M**+ 7**S**+1**D**+13**a**, dedicated, [BL07a]
$\mathcal{J}^x$, [CMO98]	4**M**+4**S**+10**a**, [CMO98]	10**M**+ 5**S**+1**D**+ 7**a**, dedicated, *
	3**M**+5**S**+13**a**, [BL07a]	8**M**+ 7**S**+1**D**+14**a**, dedicated, *

* Effective

If just unified additions are being accessed then the proposed algorithms are faster than previous results in the literature. Selecting between $\mathcal{P}$ and $\mathcal{J}^c$ depends on the $\mathbf{S}/\mathbf{M}$ value. If $\mathbf{S} < \mathbf{M}$ then $\mathcal{J}^c$ will be the optimum choice.

5.6 Conclusion

This chapter studied the inversion-free arithmetic in five forms of elliptic curves using various point representations, alternative addition laws, and algorithms to obtain faster group operations. In the literature, only dedicated addition laws were known for some forms while only unified addition laws were known for some other forms. In this chapter, the gaps are closed by equipping each form with low-degree dedicated addition formulae, low-degree unified addition formulae, and low-degree doubling formulae. This chapter also reviewed many addition laws from the literature. Each set of formulae was further investigated for the best operations counts with optimized algorithms in various point representations. Moreover, this chapter discussed practical ways of preventing incorrect summands when a particular law is used, also see §4.6 of Chapter 4.

Bernstein *et al.*'s introduction of Edwards curves [BL07b] at Asiacrypt 2007 has focused a lot of attention for efficient elliptic curve based implementations on Edwards form. In §5.1, the speed of this form and its extended version twisted Edwards form is further improved in many different ways. For instance, the proposed dedicated addition algorithm is shown to take less effort for adding two distinct points, one of them given in homogeneous projective coordinates and the other in affine coordinates. With the proposal of extended coordinates, dedicated addition and unified addition are significantly sped up. The proposed mixed coordinate system for twisted Edwards form allows the use of new faster additions together with the fast doubling formulae by Bernstein *et al.* from [BBJ$^+$08]. For technical details see §5.1.

In §5.2, the arithmetic of extended Jacobi quartic form is substantially improved. In particular, the homogeneous projective coordinate system was shown to allow a very efficient point doubling. The proposed optimizations in §5.1 also perfectly matched with this quartic model. Exploiting the analogies between two quartic models, the speed of extended Jacobi quartic form is brought quite close to the speed of twisted Edwards curves. Noting that extended Jacobi quartic form covers all twisted Edwards curve plus all other curves of even order, the arithmetic of extended Jacobi quartic form is still very interesting for many curves which are not 1 or 2 isogenous to twisted Edwards curves. For technical details, see §5.2.

In §5.3 the speed of Jacobi intersection form has been improved for many $\mathbf{D}/\mathbf{M}$, $\mathbf{S}/\mathbf{M}$, and $\mathbf{a}/\mathbf{M}$ values. After proposed optimizations the speed of Jacobi intersection form has been brought close to extended Jacobi quartic form. However, every Jacobi intersection form elliptic curve is either 1 or 2 isogenous to a twisted Edwards curve, see [BBJ$^+$08, §5]. Therefore, twisted Edwards curves become more suitable representations for all elliptic curve which can be written in Jacobi intersection form once the proposed improvements from §5.1 are applied.

In §5.4 the arithmetic of twisted Hessian curves has been improved for some $\mathbf{a}/\mathbf{M}$, $\mathbf{D}/\mathbf{M}$, and $\mathbf{S}/\mathbf{M}$ values which can be exploited in a wide range of applications. For instance, one multiplication has been removed from both dedicated and unified addition algorithms at the expense of some more additions in $\mathbb{K}$. Note that in some implementations additions in $\mathbb{K}$ are practically negligible. With the proposed improvements for some elliptic curves, (twisted) Hessian form becomes a faster representation than short Weierstrass form.

A preliminary speed ranking between studied coordinate systems of elliptic curves is discussed next where plausible suggestions are provided for implementations.

Operation count based comparison The easiest way to get a speed ranking is to find a suitable elliptic curve for each form and compare operation counts for doubling and additions. Table 5.10 provides selected operation counts in **M** for selected coordinate systems under the assumptions $\mathbf{D} \approx 0.1\mathbf{M}$, $\mathbf{a} \approx 0.05\mathbf{M}$, and $\mathbf{S} = 0.8\mathbf{M}$. The columns are sorted in descending order with respect to the column `DBL`.

Table 5.10: Operation counts in selected coordinate systems for each form.

System	`DBL`	`reADD`	`ADD`	`ADD`,$Z_2 = 1$
$\mathcal{H}$ with $a = 1$	8.20**M**	11.85**M**	11.85**M**	9.85**M**
$\mathcal{J}^x$ with $a = -3$	7.60**M**	13.75**M**	15.55**M**	10.75**M**
$\mathcal{E}^x$ with $a = -1$	6.50**M**	8.30**M**	8.50**M**	7.50**M**
$\mathcal{I}$ with $b = 1$	6.45**M**	12.55**M**	11.00**M**	10.55**M**
$\mathcal{Q}^x$ with $a = -1/2$	6.35**M**	10.35**M**	10.55**M**	9.55**M**

Table 5.11 provides cost estimations per bit of the scalar for a variable single scalar multiplication with variable base point. The columns are sorted in descending order with respect to the column `NEW`, i.e. scalar multiplication cost in terms of **M**. For simplicity, the total cost is calculated as $0.98\times$`DBL` $+ 0.18\times$`reADD`. Although this is a rough estimation, the ranking between the systems does not change for a majority of plausible assumptions on the cost of **S**, **D**, and **a**. For a more detailed cost analysis see [BL08].

Table 5.11: Cost estimate of `SMUL` per bit of scalar in **M**.

System	`OLD`	`NEW`
Twisted Hessian form, $\mathcal{H}$ with $a = 1$	10.58**M**	10.17**M**
Short Weierstrass form, $\mathcal{J}^x$ with $a = -3$	9.92**M**	-
Jacobi intersection form, $\mathcal{I}$ with $a = 1$	9.01**M**	8.43**M**
Extended Jacobi quartic form, $\mathcal{Q}^x$ with $a = -1/2$	10.00**M**	8.07**M**
Twisted Edwards form, $\mathcal{E}^x$ with $a = -1$	8.31**M**	7.87**M**

This table highly benefits from the algorithms proposed in this chapter. For instance, if the system $\mathcal{Q}^w$ were considered for extended Jacobi quartic form with $d = 1$ using the doubling and re-addition algorithms in [BL07a, `dbl-2007-bl`] then the cost estimation would have been 10.00**M** per bit after several cached values. This would put $\mathcal{Q}^w$ even behind $\mathcal{J}$ with $a = -3$. Similarly, if the system $\mathcal{E}$ were used for twisted Edwards form as in [BBJ$^+$08] then the cost estimation for twisted Edwards form with $a = 1$ would be 8.40**M** per bit. If the system $\mathcal{E}^i$ were used for twisted Edwards form as in [BBJ$^+$08] then the cost estimation for twisted Edwards form with $a = 1$ would be 8.31**M** per bit. Similarly, if the system $\mathcal{H}$ is used for twisted Hessian form as described in [BKL09] then the cost estimation for twisted Hessian form with some extremely small non-cube a would be 10.58**M** per bit. Similarly, if the system $\mathcal{I}$ is used for Jacobi intersection form as described in [LS01] then the cost estimation for Jacobi intersection form with some extremely small non-cube a would be 9.01**M** per bit.

An implementation based ranking will be provided later in Chapter 6 where the results of this chapter are supported by experiments.

Curve based comparison This discussion gives a fairly satisfying idea of the speed ranking between different forms when a suitable elliptic curve is selected for each form. On the other hand, such a discussion leads to comparing speeds of elliptic curves which are possibly not isomorphic (or not birational) or not isogenous. From an cryptographic point of view this is no problem since over a large field, finding a suitable curve which resists attacks of concern (and which has small curve constants if desired), is enough to satisfy the cryptographic interest. From a mathematical point of view, however, it is better to determine the fastest form for a fixed elliptic curve. The discussion is now focused on this aspect.

First of all, short Weierstrass form is always a fallback model for fairly efficient implementation because any elliptic curve (of large characteristic) can be represented on the curve E_{W,a_4,a_6}, see Chapter 2. For arbitrary a_4, the optimum system is the mixed coordinates $\mathcal{J}^x$ introduced by Cohen *et al.* in [CMO98] which is further discussed and improved in [BL08]. If $a = -3$ then $\mathcal{J}$ is the optimum system with cached Z^2 for dedicated additions.

Now, fix an elliptic curve E of order r such that $2 \mid r$. Then, the elliptic curve is isomorphic (over the same field) to a Weierstrass curve $y^2 = x(x^2 + a_2'x + a_4')$ which is birationally equivalent (over the same field) to the extended Jacobi quartic form elliptic curve given by $E_{Q,-a_2'/4,(a_2'^2-4a_4')/16}$, see §2.3 of Chapter 2. It is suitable here to note that every extended Jacobi quartic curve $E_{Q,d,a}$ is birationally equivalent (over the same field) to a Weierstrass curve given by $y^2 = x(x^2 - 4ax + 4a^2 - 4d)$. Performing the arithmetic on this curve in extended Jacobi quartic form is always faster than performing the analogous operations on the corresponding short Weierstrass form because even for arbitrary a and d both point doublings and point additions in $\mathcal{Q}^x$ are faster than in $\mathcal{J}$ with $a_4 = -3$, (the fastest case in short Weierstrass). In particular, doubling in $\mathcal{Q}^x$ takes $2\mathbf{M} + 5\mathbf{S} + 1\mathbf{D} + 7\mathbf{a}$ where doubling in $\mathcal{J}$ with $a_4 = -3$ takes $3\mathbf{M} + 5\mathbf{S} + 12\mathbf{a}$. But, even if $\mathbf{D} = \mathbf{M}$ point doubling in $\mathcal{Q}^x$ is still faster. Similarly, a dedicated point addition in $\mathcal{Q}^x$ takes an effective $6\mathbf{M} + 4\mathbf{S} + 3\mathbf{D} + 17\mathbf{a}$ where the fastest dedicated point addition in $\mathcal{J}^c$ with $a_4 = -3$ takes $10\mathbf{M}+4\mathbf{S}+13\mathbf{a}$. So, even if $\mathbf{D} = \mathbf{M}$ point addition in $\mathcal{Q}^x$ is still faster. In conclusion, for elliptic curves of even order extended Jacobi quartic form becomes the new fallback curve model.

Now, further assume that $4 \mid r$. Then, the curve is either 1 or 2 isogenous to a twisted Edwards curve. Performing the arithmetic on this curve in twisted Edwards form is always faster than performing the analogous operations in the corresponding short Weierstrass form. In particular, a dedicated point addition in $\mathcal{E}^x$ takes an effective $9\mathbf{M}+1\mathbf{D}+7\mathbf{a}$ where the fastest dedicated point re-addition in $\mathcal{J}^c$ with $a_4 = -3$ takes $10\mathbf{M} + 4\mathbf{S} + 13\mathbf{a}$. So, even if $\mathbf{D} = \mathbf{M}$ point addition in $\mathcal{E}^x$ is faster by more than $4\mathbf{S}$. Point doubling in $\mathcal{E}^x$ takes $3\mathbf{M}+4\mathbf{S}+1\mathbf{D}+6\mathbf{a}$ where doubling in $\mathcal{J}$ with $a = -3$ takes $3\mathbf{M} + 5\mathbf{S} + 12\mathbf{a}$. It is ideal to rescale the constant a of a twisted curve to -1 for maximum performance. This requires $-a$ to be a square in the underlying field. Even if this is not possible it is always possible to rescale a to a very small constant. So, it makes sense to assume $\mathbf{D} < \mathbf{S}$. So, even if $\mathbf{D} = \mathbf{S}$ point doubling in $\mathcal{E}^x$ is faster than point doubling in $\mathcal{J}$ with $a_4 = -3$. In conclusion, for elliptic curves whose order is divisible by four, twisted Edwards form becomes a fallback curve model (using an isogenous

curve if necessary).

Since every twisted Edwards curve is isomorphic (over the same field) to an extended Jacobi quartic curve, it is also reasonable to make a speed comparison between these two models. An investigation of the worst case scenario for the twisted Edwards form is done first. The curve parameter a of the twisted Edwards curve can always be rescaled to a very small value so that $\mathbf{D} \leq \mathbf{a}$ and consequently a dedicated point addition in $\mathcal{E}^x$ takes at most an effective $9\mathbf{M} + 8\mathbf{a}$. In the best case a dedicated addition in $\mathcal{Q}^x$ will take an effective $6\mathbf{M} + 4\mathbf{S} + 17\mathbf{a}$ which can be faster than point addition in $\mathcal{E}^x$ for which the worst case rules apply. Plus, point doubling in $\mathcal{Q}^x$ is always faster than point doubling in $\mathcal{E}^x$ for the worst case scenario for the twisted Edwards curve. Therefore, for some twisted Edwards curves, the corresponding extended Jacobi quartic curve can be faster. Now, the worst case scenario for the extended Jacobi quartic form is investigated. Since $4 \mid r$ the extended Jacobi quartic curve is 1 or 2 isogenous to a twisted Edwards curve. The curve parameter a of the extended Jacobi quartic can always be rescaled to a very small value so that $\mathbf{D} \leq \mathbf{a}$ and consequently a dedicated point addition in $\mathcal{Q}^x$ takes at most an effective $6\mathbf{M} + 4\mathbf{S} + 2\mathbf{D} + 17\mathbf{a}$. In the best case a dedicated addition in $\mathcal{E}^x$ will take an effective $8\mathbf{M} + 10\mathbf{a}$ which is always faster than point addition in $\mathcal{Q}^x$ for which the worst case rules apply. Point doubling in $\mathcal{Q}^x$ takes $2\mathbf{M} + 5\mathbf{S} + 7\mathbf{a}$ in the worst case which can only be as fast as point doubling in $\mathcal{E}^x$ if $\mathbf{M} = \mathbf{S}$. Therefore, keeping in mind the assumption $4 \mid r$, for some extended Jacobi quartic curves, the corresponding 1 or 2 isogenous twisted Edwards curve can be faster. In conclusion, extended Jacobi quartic curves and twisted Edwards curves are not fallback curve models for each other in every situation.

Every Jacobi intersection curve is 1 or 2 isogenous to a twisted Edwards curve. After a suitable rescaling of constants, it is easy to conclude after similar discussions that twisted Edwards form is a faster alternative to Jacobi intersection form.

Now, assume that $3 \mid r$ but $6 \nmid r$. Such curves can be written in Weierstrass form or twisted Hessian form. It is easy to conclude after similar discussions that twisted Hessian form and short Weierstrass form are not fallback curve models for each other in every situation. On the other hand, if $6 \mid r$ then both extended Jacobi quartic form and twisted Edwards form become faster for such elliptic curves.

Chapter 6

Experimental results

This chapter provides details on the implementation of elliptic curve scalar multiplication in the light of new results from this work. The aim is to show that ECC applications can practically benefit from the proposed algorithms. To achieve this aim, one has to face an enormous number of design choices. These choices are investigated by categorizing different components of the implementation as follows:

- hardware,
- finite field and its construction,
- elliptic curve, its representation, doubling and addition algorithms,
- scalar multiplication algorithm

where each item has intrinsic interactions with the others. These interactions are briefly explained in the following sections. In particular, §6.1 is about the preferred programming languages and the hardware where the scalar multiplications are computed. §6.2 contains details about how the finite field arithmetic is developed. §6.3 makes an overview of best operation counts for each of the five forms studied in this thesis. §6.4 presents experimental results of the implemented scalar multiplication algorithms. The conclusions of this chapter are drawn and final remarks are provided in §6.5.

6.1 Hardware and programming environment

Features and restrictions of hardware influence strongly the code to be designed. Some hardware is so restricted that the programmer has to keep the compiled-code size to a minimum. Smart Cards and wireless sensor network gadgets are examples of this kind. On such hardware, the built-in word size tends to be small. Therefore inlining or loop-unrolling techniques are most likely to generate a large compiled-code. In addition, only very small lookup tables are practical. On the other hand, contemporary desktop processors come with large and fast

internal cache memories, multi-level pipelining mechanisms, specialized SIMD instruction sets and the like. This time the programmer's challenge is to eliminate branches and minimize the memory access by wisely using the internal registers and the available cache. Yet in some other cases (such as military applications) it is important to hide the side channel information leakage. As a consequence of these, the ratios $\mathbf{I}/\mathbf{M}$, $\mathbf{S}/\mathbf{M}$, $\mathbf{D}/\mathbf{M}$, and $\mathbf{a}/\mathbf{M}$ are affected by the way the underlying field arithmetic is implemented.

Due to ease of access, the experiments in this chapter are optimized for a desktop processor: a single core of an 64-bit Intel Core 2 Duo (E6550) processor. The finite field layer of the code is time-critical. Therefore, an assembly optimized finite field layer is an indispensable component of the implementation. This layer was written in assembly language using the `x86-64` instruction set excluding the SIMD instructions. All higher level operations were written in plain C language. For efficiency purposes, dynamic memory allocations were prevented. Therefore, the stack was used to store all points, look-up tables, etc. The code was compiled with the `GCC` compiler (the `-O2` flag set). The executables were run on an Ubuntu 9 platform.

6.2 Finite field arithmetic

The implementation of finite field arithmetic is a crucial step in developing efficient ECC applications. Although this thesis does not concentrate on improving finite field arithmetic, an optimized finite field layer is needed in order to test the performance of the proposed algorithms.

Every finite field $\mathbb{L}$ contains (up to isomorphism) a unique subfield $\mathbb{K} \cong \mathbb{Z}/p\mathbb{Z}$. In the case where p is large, it is reasonable to implement the arithmetic of $\mathbb{K}$ using multiprecision integers since p typically does not fit into a single computer word. See [HMV03, §2.2] and [CF05, §10,§11] for details. With modular integer arithmetic, the field operations in $\mathbb{K}$ such as addition, multiplication, and inversion, can be built up. See [BZ09] for details on modular and multiprecision arithmetic. If $\mathbb{L} = \mathbb{K}$ then there is nothing more to do. If $\mathbb{L} \supset \mathbb{K}$ then an efficient field towering technique can be adapted to perform operations in $\mathbb{L}$. For mathematical details of field constructions see [LN96]. For efficiency related discussions see [BS09]. The simplest and the most efficient finite field implementations have $\mathbb{L} = \mathbb{K}$ and p of extremely low or extremely high Hamming weight. Selecting p in this fashion, however, is not always possible. For instance, the latest techniques for generating pairing friendly elliptic curves often yield an arbitrary p. For arbitrary p, one of the best methods to follow is Montgomery's n-residue arithmetic, see [Mon85].

The code is designed to serve for other prime fields with characteristic p of the form $p = 2^{256} - c$ where c is a suitable positive integer with at most 64-bits. On a 64-bit processor each field element fits into only 4 computer words. Consequently, the assembly codes for basic field operations such as addition and multiplication are not very long. However, accurately measuring the cycle counts of each operation is practically hard due to high levels of the pipelining in a Core 2 processor. Each of these operations is partially overlapped in the execution by the preceding and the following operations. Plus compiler optimizations can change the position and the number of instructions. As a rough but consistent way of estimating

the cycle counts, one can count the number of instructions required for each operation. The details are depicted in Table 6.1. The first column explains the type of field operation being

Table 6.1: Estimated cost comparison of various field operations.

Operation	Count	Ratio
`an` × `bn`	165	1.00
`an`^{2}	119	0.72
`an` × `$b`	50	0.29
`an` + `bn`	31	0.19
`an` − `bn`	31	0.19
`an` × `$2`	30	0.18
`an` / `$2`	23	0.14

performed. For instance, `an` × `bn` means that two multiprecision field elements each of 4 words, `an` and `bn`, are being multiplied (including the modular reduction). The symbol `$` stands for a small constant that fits a single word. The second column reports the number of instructions used to carry out the corresponding operation. The third column scales the instruction counts to estimate the relative cost of each operation with respect to multiplication which appears in the first row. Therefore, from Table 6.1, the values **S**/**M**, **D**/**M**, **a**/**M** can be estimated.

In all experiments, the finite field $\mathbb{F}_{2^{256}-587}$ is used. The arithmetic of this field is faster than arbitrary fields since the prime $2^{256} - 587$ is of extremely low hamming weight and thus more suitable for computer implementations. To give a taste of the finite field code, the division-by-2 operation is given in Figure 6.1 as an example. In this operation, four consecutive computer words building `an` are continuously shifted right by 1 bit and the result is placed in registers `(r11,r10,r9,r8)`. If the carry flag is set, then the content of `an` is odd. In this case, $\lfloor (2^{256} - 587)/2 \rfloor + 1$ should be added to `(r11,r10,r9,r8)`. This done by subtracting $\lfloor 587/2 \rfloor$ plus some overhead to handle the most significant bit of `(r11,r10,r9,r8)`. Otherwise 0 is added to `(r11,r10,r9,r8)`. The addition of 0 might seem awkward since it does not alter the result. However, this is crucial in eliminating conditional statement instructions such as `jne`. At this stage, the conditional data movement instruction `cmovnc` which is available in all AMD64 architectures including Core 2 processors, is preferred. In both cases `zn` contains the final result. The registers `zn` and `an` are allowed to point to the same memory location. The other operations are designed in a similar fashion. Excluding the inversion, none of the field operations contains any loop or conditional statement. Therefore our software highly benefits from Core 2's multilevel pipelining.

Since each function is composed of only a few dozen assembly instructions it also makes sense to extend the assembly optimizations to the elliptic curve point doubling and point addition level so that the programmer has more flexibility in utilizing the internal registers of the processor and decreased memory access resulting in less data movement. On the other hand such an approach would significantly increase the code development time. Many point addition and point doubling formulae (even for different curve models) often share very similar higher

```
static INLINE void dv2(unsigned long *zn, unsigned long *an){
  __asm__ VOLATILE (
    "movq %2,%%rax; movq 8*3(%%rax),%%r11; shrq $0x1,%%r11; movq 8*2(%%rax),%%r10;
    rcrq $0x1,%%r10; movq 8*1(%%rax),%%r9; rcrq $0x1,%%r9; movq 8*0(%%rax),%%r8;
    rcrq $0x1,%%r8; movq %1,%%rax; movq %0,%%r12; movq $0x0,%%rdx; cmovnc %%rdx,%%r12;
    movq $0x8000000000000000,%%rbx; cmovnc %%rdx,%%rbx; subq %%r12,%%r8;
    movq %%r8,8*0(%%rax); sbbq $0x0,%%r9; movq %%r9,8*1(%%rax); sbbq $0x0,%%r10;
    movq %%r10,8*2(%%rax); sbbq %%rbx,%%r11; movq %%r11,8*3(%%rax);"
    : : "n" (587>>1), "m" (zn), "m" (an)
    : "%rax", "%rbx", "%rdx", "%r8", "%r9", "%r10", "%r11", "%r12", "memory"
  );
}
```

Figure 6.1: Sample $\mathbb{F}_{2^{256}-587}$ operation: Divide-by-2 with no conditional statement.

level operations e.g. $(\mathtt{an} \pm \mathtt{bn})^2$, $(\mathtt{an} \pm \mathtt{bn})^2\text{-}\mathtt{an}^2\text{-}\mathtt{bn}^2$, $\mathtt{an} \times \mathtt{bn} \pm \mathtt{cn}$ and $\mathtt{an} \times \mathtt{bn} \pm \mathtt{cn} \times \mathtt{dn}$. These operations when implemented properly can eliminate several instructions resulting in a slight speed-up. Therefore, it would be of interest implementing these additional functions as a future study.

6.3 Elliptic curve operations

The tools of Chapter 3 have led to a discovery of many new affine formulae which are presented together with the existing results in Chapter 4. In Chapter 5, several coordinate systems are reviewed/studied to find the best operation counts. In this section, the aim is to determine the best choice of those coordinate systems and algorithms that facilitate the fastest scalar multiplication on suitable elliptic curves.

Selecting the curve In the experiments, five curves are used. Each curve is selected in a way to contain a large prime order subgroup. The curve constants are small enough to fit a computer word so that multiplication with curve constants can be performed faster than a general multiplication. Table 6.2 shows the selected curves. The first column specifies the elliptic curve. The second shows the co-factor of the curve.

Table 6.2: Sample elliptic curves over $\mathbb{F}_{2^{256}-587}$.

Curve	**Equation**	h
Short Weierstrass, $E_{\mathbf{S}}$	$y^2 = x^3 - 3x + 2582$	1
Extended Jacobi quartic, $E_{\mathbf{Q}}$	$y^2 = 25629x^4 - x^2 + 1$	2
(Twisted) Hessian, $E_{\mathbf{H}}$	$x^3 + y^3 + 1 = 53010xy$	3
Twisted Edwards, $E_{\mathbf{E}}$	$-x^2 + y^2 = 1 + 3763x^2y^2$	4
(Twisted) Jacobi intersection, $E_{\mathbf{I}}$	$s^2 + c^2 = 1,\ 3764s^2 + d^2 = 1$	4

The curves in the last two entries are birationally equivalent. For the convenience of the reader, the number of points on each of the sample curves are given in the respective order as they appear in Table 6.2 as follows expressed as a product of prime powers:

```
 1157920892373161954235709850086879078528607202920494852544751702707832379894 37,
2x5789604461865809771178549250434395392635515090019661408280980817432597279668 7,
3x3859736307910539847452366166956263595114119665633948250925036338213600974684 7,
4x2894802230932904885589274625217197696345597600956913640490764780382365192994 9,
4x2894802230932904885589274625217197696345597600956913640490764780382365192994 9.
```

All of these curves can resist all known attacks which are summarized in Appendix B. These curves are of negative trace and the number of points on each curve has fewer bits than 256. Therefore, any scalar (modulo the group order) also fits into a 256-bit string.

Selecting the coordinate system Finding the most efficient coordinate system depends on several factors. The values $\mathbf{I}/\mathbf{M}$, $\mathbf{S}/\mathbf{M}$, $\mathbf{D}/\mathbf{M}$, $\mathbf{a}/\mathbf{M}$ affect how the most efficient point doubling and addition algorithms are determined. The frequency of point doubling and addition operations affect how the most efficient coordinate system is determined. To simplify this process, Table 6.3 provides a summary of operation counts for the most frequently accessed operations in scalar multiplication. This table contains only the selected coordinate systems and selected operation counts. For more alternatives and comparisons see Chapter 5.

Table 6.3: Selected operation counts for the most frequently accessed operations.

Curve	**Cond.**	**Coordinate System**	`DBL`	`reADD`
Short Weierstrass	$a=-1$	$\mathcal{P}$, $(X:Y:Z)$	$7\mathbf{M}+3\mathbf{S}$	$12\mathbf{M}+2\mathbf{S}$
	$a=-3$	$\mathcal{J}$, $(X:Y:Z)$	$3\mathbf{M}+5\mathbf{S}$	$10\mathbf{M}+4\mathbf{S}$
(Twisted) Hessian	$a=1$	$\mathcal{H}$, $(X:Y:Z)$	$7\mathbf{M}+1\mathbf{S}$ or $3\mathbf{M}+6\mathbf{S}$	$11\mathbf{M}$
(Twisted) Jacobi intersection	$b=1$	$\mathcal{I}^{m1}$, $(S:C:D:Z:U:V)$	$3\mathbf{M}+4\mathbf{S}$ or $2\mathbf{M}+5\mathbf{S}+1\mathbf{D}$	$11\mathbf{M}$
Extended Jacobi quartic	$d=1$	$\mathcal{Q}^{w}$, $(X:Y:Z)$	$2\mathbf{M}+6\mathbf{S}+1\mathbf{D}$	$10\mathbf{M}+3\mathbf{S}+1\mathbf{D}$
	$a=-1/2$	$\mathcal{Q}$, $(X:Y:Z)$	$2\mathbf{M}+5\mathbf{S}$	$10\mathbf{M}+5\mathbf{S}+2\mathbf{D}$
	$a=-1/2$	$\mathcal{Q}^{e}$, $(X:Y:T:Z)$	$8\mathbf{S}$	$7\mathbf{M}+3\mathbf{S}+2\mathbf{D}$
	$a=-1/2$	$\mathcal{Q}^{x}$	$2\mathbf{M}+5\mathbf{S}$	$6\mathbf{M}+4\mathbf{S}+2\mathbf{D}$*
Twisted Edwards	$a=-1$	$\mathcal{E}$, $(X:Y:Z)$	$3\mathbf{M}+4\mathbf{S}$	$10\mathbf{M}+1\mathbf{S}+1\mathbf{D}$
	$a=-1$	$\mathcal{E}^{i}$, $(X:Y:Z)$	$3\mathbf{M}+4\mathbf{S}+1\mathbf{D}$	$9\mathbf{M}+1\mathbf{S}+1\mathbf{D}$
	$a=-1$	$\mathcal{E}^{e}$, $(X:Y:T:Z)$	$4\mathbf{M}+4\mathbf{S}$	$8\mathbf{M}$
	$a=-1$	$\mathcal{E}^{x}$	$3\mathbf{M}+4\mathbf{S}$	$8\mathbf{M}$*

*Effective, see §5.2.3 and §5.1.4 in Chapter 5.

In Table 6.3, two systems are more efficient than the other entries in terms of operation counts regardless of $\mathbf{S}/\mathbf{M}$, $\mathbf{D}/\mathbf{M}$, $\mathbf{a}/\mathbf{M}$ values or the frequencies of doublings and additions. These systems are $\mathcal{Q}^{x}$ and $\mathcal{E}^{x}$; both are outcomes of this thesis. In the experiments, $\mathcal{Q}^{x}$ is used for the extended Jacobi quartic curve $y^2 = 25629x^4 - x^2 + 1$ and $\mathcal{E}^{x}$ is used for the twisted Edwards curve $-x^2 + y^2 = 1 + 3763x^2y^2$. For the twisted Hessian curve $x^3 + y^3 + 1 = 53010xy$, the coordinate system $\mathcal{H}$ is used. Similarly, for the Jacobi intersection curve $s^2 + c^2 = 1,\ 3764s^2 + d^2 = 1$, the coordinate system $\mathcal{I}^{m1}$ is used. For the Weierstrass curve

$y^2 = x^3 - 3x + 2582$ Jacobian coordinate system $\mathcal{J}$ is known to be more efficient than projective coordinate system $\mathcal{P}$. Therefore, Jacobian coordinates are used for the Weierstrass curve.

6.4 Scalar multiplication

The highest level task is to put the pieces of a scalar multiplication together. As in lower level operations, there are so many design issues such as the selection of a scalar recoding algorithm, powering algorithm, determining the optimal size of lookup tables, the elimination of unnecessary conditional statements, etc. Studying all possible scenarios would only complicate the experiments without changing most conclusions that can be derived from the results. To keep design alternatives to a minimum, two experiments are conducted. These experiments involve single-variable-point-single-variable-scalar multiplication and single-fixed-point-single-variable-scalar multiplication which are presented in §6.4.1 and §6.4.2, respectively.

6.4.1 Experiment1: Scalar multiplication with variable base-point

This section provides implementation timings for elliptic curve single-variable-point-single-variable-scalar multiplication. Algorithm 3.38 in [HMV03] is implemented for this pupose. As for the integer recoding part of the scalar multiplication, w-LtoR algorithm in [Ava05] is used. This part of the implementation runs on-the-fly as the main loop of the scalar multiplication is performed. The scalar multiplication algorithm starts with building a look-up table. To accommodate this computation $3P, 5P, \ldots, 15P$ are precomputed by the sequence of operations $2P, 2P + P, 2P + 3P, \ldots, 2P + 13P$. In this implementation $\mathbf{I}/\mathbf{M} \approx 121$. Therefore, the precomputed values are not normalized (i.e. conversion to affine form) following the analysis in [BL08]. Also following the same reference, double-and-add algorithms with $Z = 1$ are also used to obtain speed-ups of the scalar multiplication. Table 6.4 summarizes measured average clock cycles for a single-variable-point-single-variable-scalar multiplication on different representations of elliptic curves. The reverse-order of the entries in Table 6.4 was used to sort the main sections of Chapter 5.

Table 6.4: Cycle-counts (rounded to the nearest one thousand) for 256-bit scalar multiplication with variable base-point

Curve & coordinate system	w	**Approximate operation counts**	**Cycles**
Short Weierstrass ($a = -3$), $\mathcal{J}^x$	5	**I**+1598**M**+1156**S**+ 0**D**+2896**a**	468,000
(Twisted) Hessian ($a = 1$), $\mathcal{H}$	5	**I**+2093**M**+ 757**S**+ 0**D**+1177**a**	447,000
(Twisted) Jacobi intersection ($b = 1$), $\mathcal{I}^{m1}$	5	**I**+1295**M**+1011**S**+ 0**D**+2009**a**	383,000
Extended Jacobi quartic ($a = -1/2$), $\mathcal{Q}^x$	5	**I**+1162**M**+1110**S**+102**D**+1796**a**	376,000
Twisted Edwards ($a = -1$), $\mathcal{E}^x$	6	**I**+1202**M**+ 969**S**+ 0**D**+2025**a**	362,000

The implementation is developed only for the most efficient formulae for each form. For instance, weighted projective Jacobi quartic coordinates or homogeneous projective twisted Edwards coordinates are not implemented in these experiments. Therefore, the comparisons are made relative to the Jacobian coordinates.

For $\mathcal{J}^x$, a $4\mathbf{M}+4\mathbf{S}$ point doubling algorithm is used from [HMV03], also see [BL07a, dbl-2004-hmv]. As for the additions, a $12\mathbf{M}+4\mathbf{S}$ algorithm from [CMO98] is used. For readditions this algorithm needs $11\mathbf{M}+3\mathbf{S}$. The $\mathbf{M}/\mathbf{S}$ trade-offs in Table 6.3 are not helpful for this implementation. The situation may be different in other implementations. For $\mathcal{H}$, only the classic algorithms are implemented. The proposed algorithms in this thesis are slightly slower in this implementation because $\mathbf{a}$ is not negligible. On the other hand, other implementations may still benefit from the proposed algorithms. For $\mathcal{I}^{m1}$, the $3\mathbf{M}+4\mathbf{S}$ doubling algorithm is preferred, see [BL07a, `dbl-2007-bl`]. Point additions are implemented as proposed in this thesis, see Table 6.3 and §5.3.2 of Chapter 5. For $\mathcal{Q}^x$ and $\mathcal{E}^x$, the operation counts from Table 6.3 apply except for a few $\mathbf{M}/\mathbf{S}$ trade-offs which are not used in order to limit the number of $\mathbf{a}$'s.

Using several coordinates to represent points might raise the belief that there will be a significant overhead of handling extra coordinates. It was empirically observed that these overheads are negligible. If they were non-negligible then there would be a speed problem with twisted Jacobi intersection implementation. In fact, once the field operations are removed, the remaining code including the integer recoding part of the scalar multiplication takes less than 3% of the execution time. With the integer recoding excluded, it takes less than 1% of the execution time. Furthermore, the extra coordinates also serve as local variables when performing point doubling and point additions. Therefore, having more than 3 coordinates does not constitute an efficiency issue in the case of modern desktop processors. If the scalar multiplication needs fewer field multiplications with the extra coordinates then it is always worth using the extended coordinate systems.

6.4.2 Experiment2: Scalar multiplication with fixed base-point

In the case where the base-point is fixed the timings can be dramatically improved by using Algorithm 3.44 or Algorithm 3.45 in [HMV03]. These algorithms are reported to be special cases of the exponentiation techniques in [LL94]. Both algorithms are point addition intensive, i.e. the point doublings are less important in overall efficiency.

So far the most efficient point addition algorithm among existing point addition algorithms[1] is an outcome of this thesis. This algorithm requires only $8\mathbf{M}$ for each point addition on a twisted Edwards curve. It is reasonable to implement Algorithm 3.45 in [HMV03] by incorporating this new algorithm and compare it to the efficiency of $\mathcal{J}$.

Table 6.5 provides measured cycles on Core 2. The first column specifies the coordinate system. The second column is the window length w of the Algorithm 3.45 in [HMV03]. This algorithm uses two look-up tables by default. Within this implementation, this was generalized to an arbitrary number of tables. The third column s tells how many look-up tables are used. For instance, $w=8$ and $s=4$ means that $s\times 2^w = 4\times 2^8 = 1024$ precomputed points are stored in look-up tables. Each coordinate fits into 32 bytes. So, an affine point (x,y) needs 64 bytes. Therefore, the look-up tables take 64 `KB` (kilobyte). The space consumption is reflected in the fourth column. The fifth column contains the averaged cycle counts on a single core of Core 2.

[1] Excluding differential point additions.

Table 6.5: Cycle-counts (rounded to the nearest one thousand) for 256-bit scalar multiplication with fixed base-point

Curve & coordinate system	w	s	**Look-up**	**Cycles**
Short Weierstrass ($a = -3$), $\mathcal{J}$	4	4	2 KB × 2	138,000
	8	1	8 KB × 2	121,000
	8	2	16 KB × 2	102,000
	8	4	32 KB × 2	92,000
	8	8	64 KB × 2	86,000
Twisted Edwards ($a = -1$), $\mathcal{E}^x$, precomputed t	4	4	2 KB × 3	131,000
	8	1	8 KB × 3	115,000
	8	2	16 KB × 3	96,000
	8	4	32 KB × 3	87,000
	8	8	64 KB × 3	79,000
Twisted Edwards ($a = -1$), $\mathcal{E}^e$, precomputed t	4	4	2 KB × 3	122,000
	8	1	8 KB × 3	107,000
	8	2	16 KB × 3	90,000
	8	4	32 KB × 3	81,000
	8	8	64 KB × 3	79,000
Twisted Edwards ($a = -1$), $\mathcal{E}^e$, on-the-fly t	4	4	2 KB × 2	124,000
	8	1	8 KB × 2	109,000
	8	2	16 KB × 2	92,000
	8	4	32 KB × 2	82,000
	8	8	64 KB × 2	79,000

Since the operations are point addition intensive, there is a little performance difference between $\mathcal{E}^e$ and $\mathcal{E}^x$. This can be observed from Table 6.5. The extra coordinate $t = xy$ can be computed on-the-fly to reduce the size of the look-up table. The additional computation for t only marginally increases the overall cycle counts. As expected, all implementations with twisted Edwards form are faster than Weierstrass form. Note that as the size of the look-up table increases, the overhead of memory access also increases. It is emprically observed that in all entries of Table 6.5, the cost of memory access is far less than the cost of performing arithmetic on the underlying field. For simplicity, $w = 8$ is selected to be a factor of 256. Better theoretic speed-ups are possible for higher values of $w < 16$ however such values will require a more complicated implementation and speed-ups may not be practically achieved. This is left as a future investigation.

6.5 Conclusion

This chapter introduced software implementation of elliptic curve scalar multiplication using different forms of elliptic curves. For each form the best algorithms proposed in Chapter 5 and also the best existing algorithms in the literature are used. These experiments show that the

proposed algorithms are practically useful in efficient implementations in most cases.

To the best of the author's knowledge, the implementation introduced in §6.4.1 is the first software realization of a left-to-right *on-the-fly* scalar recoding based on Avanzi's LtoR algorithm, see [Ava05]. The implementation is also the first attempt to compare the speeds of different forms in a software implementation.

The code is developed without an intention of a new speed record. On the other hand, the speeds obtained in §6.4.2 are better than the corresponding record in [GLS09a].

Additional experiments such as *multi-scalar multiplication* and a repeat of all experiments incorporating the GLV [GLV01] and GLS [GLS09a] homomorphisms are left as future work. A multi-scalar multiplication is an important ingredient of efficient digital signature verifications, see Appendix B. The algorithms from this thesis can be used to practically speed-up such kinds of computations. The fractional windowing technique [Möl03] is not incorporated in this implementation and left as a future work. In addition, a new precomputation strategy in [LG09] is of interest for implementation. This approach is not implemented since an adaptation of this technique requires more formulae derivation which has not been done to date. Note that none of these choices affect the main outcomes of this section.

All of these experiments exclude comparison with Montgomery-ladder type scalar multiplications based on differential addition chains, see [Mon87], [JY03], [Gau06], [Ber06b], [GT07], and [GL09]. In some instances, Montgomery ladder can be advantageous especially when the base point is of the form $(X : 1)$ where X and the curve constant(s) are extremely small. In the general case, the algorithms from this work are theoretically faster. Further experimental investigation is left as future work. Note also that none of the standardized NIST curves benefit from the proposed speed-ups. On the other many standards recommend elliptic curves with cofactor smaller or equal to 4. In this case, the proposed techniques can be applied to get the desired speed-ups.

Chapter 7

A case study on pairing computation

In this chapter, the j-invariant zero curve $y^2 = cx^3+1$ is studied for a faster pairing computation in conjuction with the techniques from this thesis. An overview of pairing computation and the relevant notation can be found in Appendix B.3. For a comprehensive survey on cryptographic pairings, the reader can consult with [Gal05]. The most efficient method of computing pairings is Miller's algorithm [Mil04]. Each iteration of this process requires three significant computations: (i) point operations, i.e. point doubling and/or point addition; (ii) Miller line function computations and (iii) updating the Miller function value. This chapter targets stage (ii) in order to decrease the number of computationally expensive field operations encountered in (ii).

For pairing computations with even embedding degree k, it will be shown in this chapter that the curve $y^2 = cx^3 + 1$ allows the Miller doubling stage to be computed in $(k+3)\mathbf{M} + 5\mathbf{S} + 1\mathbf{M_e} + 1\mathbf{S_e}$, where $\mathbf{M}$ and $\mathbf{S}$ denote the costs of multiplication and squaring in the base field while $\mathbf{M_e}$ and $\mathbf{S_e}$ denote the costs of multiplication and squaring in the extension field of degree k. For the more general j-invariant zero curve $y^2 = x^3 + b$, the fastest Miller doubling operation count recorded to date is $(k+3)\mathbf{M} + 8\mathbf{S} + 1\mathbf{M_e} + 1\mathbf{S_e}$ [ALNR09], meaning that the special curve $y^2 = cx^3 + 1$ offers an advantage of $3\mathbf{S}$ at the doubling stage.

In addition, practically useful examples of the curve $y^2 = cx^3 + 1$ are provided for different embedding degrees using the curve generation technique "Construction 6.6" from [FST06]. Comparisons are drawn between the curve $y^2 = cx^3+1$ and other special curves and discussions on where this curve model would be optimal in practice are presented.

The remainder of this chapter is organised as follows. §7.1 explains the search for a faster Weierstrass model and efficient group operations. §7.2 presents the optimization of the new formulae for the computation of the Tate pairing. §7.3 discusses curve generation and provides some practical examples. §7.4 summarizes the contributions and compares them with the literature. In the appendices, scripts that verify the main claims of §7.1 and §7.2 are presented.

Appendix C.6 also provides more intrinsic details on the realization of the proposed formulae.

7.1 Choice of curve and the group law

In this section the choice of curve which facilitates an efficient iteration of the Miller loop is specified.

Let E be a Weierstrass form elliptic curve $y^2 = x^3 + ax + b$. Let (x_1, y_1) be a point in $E(\mathbb{F}_q)\backslash\{\mathcal{O}\}$. Then, $(x_1, y_1) + (x_1, -y_1) = \mathcal{O}$. Further let (x_2, y_2) be a point in $E(\mathbb{F}_q)\backslash\{\mathcal{O}\}$ such that $y_2 \neq 0$ and $(x_2, y_2) \neq (x_1, -y_1)$. Then, $(x_1, y_1) + (x_2, y_2) = (x_3, y_3)$ where

$$x_3 = \lambda^2 - x_1 - x_2, \tag{7.1}$$

$$y_3 = \lambda(x_1 - x_3) - y_1 \tag{7.2}$$

with

$$\lambda = \begin{cases} (y_1 - y_2)/(x_1 - x_2) & \text{if } (x_1, y_1) \neq (x_2, y_2) \\ (3x_1^2 + a)/(2y_1) & \text{if } (x_1, y_1) = (x_2, y_2) \end{cases},$$

see also Algorithm 2.2.1 in Chapter 2.

In §4.1 of Chapter 4, it has been observed that it is possible to rewrite the doubling formulae as $[2](x_1, y_1) = (x_3, y_3)$ where

$$x_3 = x_1(\mu - \mu^2) + a\sigma, \tag{7.3}$$

$$y_3 = (y_1 - c)\mu^3 + a\delta - c \tag{7.4}$$

with $\mu = (y_1 + 3c)/(2y_1)$, $\sigma = (a - 3x_1^2)/(2y_1)^2$, $\delta = (3x_1(y_1 - 3c)(y_1 + 3c) - a(9x_1^2 + a))/(2y_1)^3$ provided that $b \neq 0$ is a square in $\mathbb{F}_q$ such that $c^2 = b$. Computer-aided proofs of the correctness of formulae (7.3) and (7.4) are provided in Appendix C.6.

In the derivation of these formulae the Monagan/Pearce minimal total degree algorithm was used, see [MP06] and Chapter 3. The total degrees[1] of x_3 and y_3 are less than those of the original point doubling formulae. Furthermore the total degrees of the new formulae are minimal. In particular, the total degree of x_3 and y_3 drops from 6 to 5 and from 9 to 7, respectively.

The evaluation of lower degree functions often requires fewer field operations. However, it seems that the original point doubling formulae are faster in affine coordinates. On the other hand, homogeneous projective or Jacobian coordinates will be used in most applications to prevent costly inversions. Therefore it is worthwhile to check operation counts on these coordinates. These results are delayed until §7.2.

On the elliptic curve $y^2 = x^3 + c^2$, i.e. $a = 0$, the formulae (7.3) and (7.4) become much simpler. In addition, in order to prevent the computational disadvantage of field operations with c in doubling formulae it is better to work with another representation of the same curve given by $y^2 = cx^3 + 1$. This curve is isomorphic over $\mathbb{F}_q$ to the Weierstrass curve $v^2 = u^3 + c^2$. The isomorphism from $y^2 = cx^3 + 1$ to $v^2 = u^3 + c^2$ is given by $\sigma\colon (x, y) \mapsto (u, v) = (cx, cy)$ with the inverse $\sigma^{-1}\colon (u, v) \mapsto (x, y) = (u/c, v/c)$.

[1] The total degree is defined as the sum of the degrees of the numerator and denominator of a rational function.

Again, the identity on $y^2 = cx^3 + 1$ is the point at infinity denoted by $\mathcal{O}$ and point negation is performed by negating the y coordinate. Using the same notation as in the original formulae, the doubling formulae become $[2](x_1, y_1) = (x_3, y_3)$ where

$$x_3 = x_1(\mu - \mu^2), \tag{7.5}$$
$$y_3 = (y_1 - 1)\mu^3 - 1 \tag{7.6}$$

with $\mu = (y_1 + 3)/(2y_1)$ and thus $(x_1, y_1) + (x_2, y_2) = (x_3, y_3)$ where

$$x_3 = c^{-1}\lambda^2 - x_1 - x_2, \tag{7.7}$$
$$y_3 = \lambda(x_1 - x_3) - y_1 \tag{7.8}$$

with $\lambda = (y_1 - y_2)/(x_1 - x_2)$. The point $(0, 1)$ is of order 3. Computer-aided proofs of the correctness of formulae (7.5), (7.6), (7.7), and (7.8) are provided in Appendix C.6.

7.2 Line computations for Tate pairing

In this section, the arithmetic of $y^2 = cx^3 + 1$ is further investigated in order to assist efficient computation of the Tate pairing. First, suitable line equations are derived to compute the Miller value at both the doubling and addition stages. Then, unnecessary computations are eliminated before converting all computations to projective representation to avoid inversions. For the relevant notation and definitions of constants and variables, see Appendix B.3.

Barreto *et al.* [BLS04a] show that it is possible to eliminate costly operations in Miller's algorithm provided the point where the Miller function is evaluated is chosen suitably. In the Tate pairing, the vertical line functions v (v_{dbl} and v_{add}) in Algorithm B.3.1 are evaluated at the point $Q = (x_Q, y_Q)$. These vertical line functions take the form $v = x_R - x_Q$, where $R = (x_R, y_R)$ is the intermediate point in Algorithm B.3.1. The computations in Miller's algorithm can be simplified if v takes a value in a proper subfield $\mathbb{F}_{q^d} \subset \mathbb{F}_{q^k}$. When computing the Tate pairing on curves with even embedding degrees $k = 2d$, Q is chosen to enable this simplification by choosing a point Q' on the quadratic twist E' of E and mapping Q' to Q under the twisting isomorphism, meaning that $x_Q \in \mathbb{F}_{q^d}$ and $y_Q = \tilde{y_Q}\sqrt{\nu}$, where $\tilde{y_Q} \in \mathbb{F}_{q^d}$ and ν is some quadratic non-residue in $\mathbb{F}_{q^d}$.

7.2.1 The Miller values

The line equations arising from the addition of (x_1, y_1) and (x_2, y_2) are given by

$$g_{\mathrm{add}} = c\frac{\lambda(x_2 - x_Q) - y_2 + y_Q}{c(x_1 + x_2 + x_Q) - \lambda^2} \tag{7.9}$$

where $\lambda = (y_1 - y_2)/(x_1 - x_2)$ and $g_{\mathrm{add}} = l_{\mathrm{add}}(Q)/v_{\mathrm{add}}(Q)$ (refer to Line 9 of Algorithm B.3.1). This formula shares several common subexpressions with (7.7) and (7.8).

Next, a new formula for the line computation which uses several shared common subexpressions with the new point doubling formulae (7.5) and (7.6) is proposed. The new

formula is given by

$$g_{\text{dbl}} = \frac{2cy_1(x_1 - x_Q)^2}{x_1^2(3cx_Q) - y_1^2 + 3 + 2y_1y_Q}, \tag{7.10}$$

where $g_{\text{dbl}} = l_{\text{dbl}}(Q)/v_{\text{dbl}}(Q)$ (refer to Line 5 of Algorithm B.3.1). Furthermore, if $(x_1, y_1) = -(x_2, y_2)$ we have

$$g_{\text{vert}} = -c(x_1 - x_Q). \tag{7.11}$$

Computer aided proofs of the correctness of the formulae are provided in Appendix C.6.

Irrelevant factors Some of the terms in equations (7.9) and (7.10) can be eliminated by adopting the denominator elimination technique in [BLS04b]. Recall that y_Q is the only element that appears in (7.9) and (7.10)[2] that is in the full extension field $\mathbb{F}_{q^k}$. The denominator of g_{add} in equation (7.9) is completely contained in $\mathbb{F}_{q^d}$ and can therefore be eliminated, to give

$$g'_{\text{add}} = (y_1 - y_2)(x_2 - x_Q) - (x_1 - x_2)(y_2 - y_Q). \tag{7.12}$$

With identical reasoning the numerator of g_{dbl} in equation (7.10) can be omitted. These eliminations are standard. Now, observe that since y_Q is of the form $y_Q = \tilde{y_Q}\sqrt{\nu}$, the denominator can be written as $1/(t_1 + t_2\sqrt{\nu})$ where $t_1 = x_1^2(3cx_Q) - y_1^2 + 3$ and $t_2 = 2y_1\tilde{y_Q}$. If the Miller value is computed in this fashion there will be an inversion at the end of the Miller loop. Even worse, both the numerator and the denominator of f_{var} would have to be updated at each iteration of the Miller loop since the addition step produces a non-trivial numerator. To prevent this the numerator and the denominator of $1/(t_1 + t_2\sqrt{\nu})$ are multiplied by the conjugate expression $t_1 - t_2\sqrt{\nu}$ to give $(t_1 - t_2\sqrt{\nu})/(t_1^2 - t_2^2\nu)$. Since $t_1^2 - t_2^2\nu \in \mathbb{F}_{q^d}$, the denominator can be simply omitted to give

$$g'_{\text{dbl}} = x_1^2(3cx_Q) - y_1^2 + 3 - 2y_1y_Q. \tag{7.13}$$

It also follows that if $(x_1, y_1) = -(x_2, y_2)$ then $g'_{\text{vert}} = 1$. If r is odd, the Miller loop always finishes in this fashion so the point addition in the final iteration is ignored.

All of the formulae presented in this section can be obtained with geometric approaches. However, because of their relevance to the topic this thesis they were derived and verified by computer algebra tools, see Chapter 3.

Next, point doubling and point addition formulae are presented together with their associated line formulae in homogeneous projective coordinates. The experiments showed that the best results are obtained in homogeneous coordinates rather than Jacobian coordinates for doubling and additions. While additions generally favour projective coordinates it is interesting to note that doublings on this curve are also faster in projective coordinates. In particular the number of field operations for the doubling is $4\mathbf{M}+3\mathbf{S}$ while the best known doubling speeds so far are $2\mathbf{M}+5\mathbf{S}$ but in Jacobian coordinates. So this representation achieves the best addition speed and the best doubling speed (up to some $\mathbf{M}/\mathbf{S}$ trade-offs) in *the same* coordinate system.

[2]The point (x_2, y_2) represents $P \in E(\mathbb{F}_q)$ and the point (x_1, y_1) represents $R \in E(\mathbb{F}_q)$ in Algorithm B.3.1, a multiple of P, so that $x_1, x_2, y_1, y_2 \in \mathbb{F}_q$.

7.2.2 Encapsulated computations in homogeneous projective coordinates

In homogeneous projective coordinates each point (x, y) is represented by the triplet $(X : Y : Z)$ which satisfies the projective equation $Y^2Z = cX^3 + Z^3$ and corresponds to the affine point $(X/Z, Y/Z)$ with $Z \neq 0$. The identity element is represented by $(0 : 1 : 0)$. The negative of $(X : Y : Z)$ is $(X : -Y : Z)$.

Point doubling with line computation Given $(X_1 : Y_1 : Z_1)$ with $Z_1 \neq 0$ the point doubling can be performed as $[2](X_1 : Y_1 : Z_1) = (X_3 : Y_3 : Z_3)$ where

$$\begin{aligned} X_3 &= 2X_1Y_1(Y_1^2 - 9Z_1^2), \\ Y_3 &= (Y_1 - Z_1)(Y_1 + 3Z_1)^3 - 8Y_1^3Z_1, \\ Z_3 &= 8Y_1^3Z_1. \end{aligned} \tag{7.14}$$

These formulae are derived from (7.5) and (7.6) in §7.1. Point doubling without line computation needs $4\mathbf{M} + 3\mathbf{S}$ using the following sequence of operations.

$$\begin{gathered} A = Y_1^2, \quad B = Z_1^2, \quad C = (Y_1 + Z_1)^2 - A - B, \quad Z_3 = 4A \cdot C, \\ X_3 = 2X_1 \cdot Y_1 \cdot (A - 9B), \quad Y_3 = (A - 3B + C) \cdot (A + 9B + 3C) - Z_3. \end{gathered}$$

The line formula derived from (7.13) is given by

$$\begin{aligned} g''_{\mathrm{dbl}} &= X_1^2(3cx_Q) - Y_1^2 + 3Z_1^2 - 2Y_1Z_1y_Q \\ &= E \cdot (3cx_Q) - A + 3B - 2C \cdot y_Q \end{aligned} \tag{7.15}$$

where $E = X_1^2$.

Assume that $3cx_Q$ is precomputed. If Q is chosen according to the discussion at the start of this section, then multiplication with $3cx_Q$ or with y_Q counts as $(k/2)\mathbf{M}$. The point doubling with line computation needs $(k + 3)\mathbf{M} + 5\mathbf{S}$ if k is even. In this operation count an additional $\mathbf{M}/\mathbf{S}$ trade-off is exploited when calculating $2X_1Y_1$ in the point doubling formulae, which can now be computed as $(X_1 + Y_1)^2 - E - A$.

See Appendix C.6 for further justifications and details of the operation scheduling.

Point addition with line computation Given $(X_1 : Y_1 : Z_1)$ and $(X_2 : Y_2 : Z_2)$ with $Z_1 \neq 0$ and $Z_2 \neq 0$ and $(X_1 : Y_1 : Z_1) \neq (X_2 : Y_2 : Z_2)$, an addition can be performed as $(X_1 : Y_1 : Z_1) + (X_2 : Y_2 : Z_2) = (X_3 : Y_3 : Z_3)$ where

$$\begin{aligned} X_3 &= (X_1Z_2 - Z_1X_2)(Z_1Z_2(Y_1Z_2 - Z_1Y_2)^2 - c(X_1Z_2 + Z_1X_2)(X_1Z_2 - Z_1X_2)^2), \\ Y_3 &= (Y_1Z_2 - Z_1Y_2)(c(2X_1Z_2 + Z_1X_2)(X_1Z_2 - Z_1X_2)^2 - Z_1Z_2(Y_1Z_2 - Z_1Y_2)^2) - \\ &\quad cY_1Z_2(X_1Z_2 - Z_1X_2)^3, \\ Z_3 &= cZ_1Z_2(X_1Z_2 - Z_1X_2)^3. \end{aligned} \tag{7.16}$$

These formulae are derived from (7.1) and (7.2) in §7.1. Point addition without line computation needs $12\mathbf{M} + 2\mathbf{S} + 1\mathbf{D}$ if Z_2 is arbitrary and $9\mathbf{M} + 2\mathbf{S} + 1\mathbf{D}$ if $Z_2 = 1$. Note that $\mathbf{D}$ stands for a multiplication with c.

The line formula derived from (7.12) is given by

$$\begin{aligned} g''_{\text{add}} &= (Y_1Z_2 - Z_1Y_2)(X_2 - x_QZ_2) - \\ &\quad (X_1Z_2 - Z_1X_2)Y_2 + (X_1Z_2 - Z_1X_2)Z_2y_Q. \end{aligned} \tag{7.17}$$

Assuming that Q is chosen according to the discussion at the start of this section, multiplication with $(X_2 - x_QZ_2)$ or with Z_2y_Q counts as $(k/2)\mathbf{M}$ each. Assuming that $Z_2 = 1$, point addition with line computation needs $(k+10)\mathbf{M} + 2\mathbf{S} + 1\mathbf{D}$ if k is even. Assume that Z_2 is arbitrary. Assume that $(X_2 - x_QZ_2)$ and Z_2y_Q are precomputed. The point addition with line computation needs $(k+13)\mathbf{M} + 2\mathbf{S} + 1\mathbf{D}$ if k is even.

The algorithm that is used for the point addition part is a slightly modified version of the Cohen/Miyaji/Ono algorithm [CMO98]. The details are omitted here. Refer to Appendix C.6 for justifications and details of the operation scheduling.

7.3 Curve generation

This section discusses generating pairing-friendly curves of the form $y^2 = cx^3 + 1$. A minor adjustment to the pairing definition is also given when employing this curve in the supersingular setting.

Implementing the Tate pairing on the curve $y^2 = cx^3 + 1$ requires the construction of the j-invariant zero curve $y^2 = x^3 + b$ where $b = c^2$ for $c \in \mathbb{F}_q$. All j-invariant zero curves have a special endomorphism ring and such curves have CM discriminant[3] $D = 3$. In Construction 6.6 of [FST06], Freeman *et al.* extend the results of Barreto *et al.* [BLS03] and Brezing and Weng [BW05] to efficiently construct $D = 3$ curves for all values of k where $18 \nmid k$. This technique is suitable to generate curves of the form that are of interest to this chapter. The details of Construction 6.6 are omitted here. Freeman *et al.* state that this construction achieves the best ρ-value[4] curve families for the majority of embedding degrees $k \leq 50$.

For most embedding degrees, this method of construction efficiently produces a curve of the desired form with the best ρ-value, however the extra condition on the curve constant is restrictive. For instance, a $k = 8$ curve with b as a square using this construction was unable to be obtained. For $k = 12$, constructing the curve $y^2 = cx^3 + 1$ gives $\rho \approx 3/2$, which is significantly larger than what can be obtained for Barreto-Naehrig (BN) curves [BW05] where b is non-square, for which D is also 3 but which have the optimal ρ-value of $\rho = 1$.

Nevertheless, there is a wide range of useful embedding degrees that would welcome the speedups offered on the curve $y^2 = cx^3 + 1$. Two pairing-friendly examples of the curve using Construction 6.6 of [FST06] are given by,

[3] Complex multiplication discriminant

[4] The ratio of the base field size (in bits) and the size of the prime-order subgroup on the curve.

$k = 12,\ \rho \approx 3/2,\ c = 1,$

$q = \texttt{0x55555583E6AAB5415B22F364648CF7D4A1A9716C687F053}\backslash$
$\texttt{39126A5FC2A09}$ (239 bits),

$r = \texttt{0x10000005D24000CB530E5C544B4E84E5B34F41BD1}$ (161 bits),

$t = \texttt{0x1000000174A} = q + 1 - \#E(F_q)$ (41 bits).

$k = 24,\ \rho \approx 5/4,\ c = 3,$

$q = \texttt{0x577380D96AF284FCF9200C2CC966EC756D86B4CBF2A3AAD}\backslash$
$\texttt{3C1}$ (199 bits),

$r = \texttt{0x105121CA61CB6CAF9EF3A835A4442784FFF816AF1}$ (161 bits),

$t = \texttt{0x100A0F} = q + 1 - \#E(F_q)$ (21 bits).

Supersingular curves When the characteristic of the underlying field is $p \equiv 2 \mod 3$, the curve $y^2 = cx^3 + 1$ is supersingular with $k = 2$. One would usually define the symmetric pairing as $\hat{e} : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ where $\hat{e}(P, Q) = e(P, \phi(Q))$ and ϕ is the distortion map $\phi(x, y) = (\xi x, y)$ for some non-trivial cube root of unity $\xi \in \mathbb{F}_{p^2}$. However, using the distortion map in this manner would not allow the use of the formulae derived in §7.2, since these formulae were derived under the assumption that it was the y-coordinate of the second argument in the pairing that was in the extension field. To solve this problem, Scott's technique [Sco04] is followed and the supersingular pairing is defined as $\tilde{e} : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ where $\tilde{e}(P, Q) = e(P, \theta(Q))$ and θ is defined as $\theta(Q) = \phi(Q) - \pi_p(\phi(Q))$, where π_p is the p-power Frobenius endomorphism. Then, $\pi_p(\phi(Q)) = \pi_p(\xi x_Q, y_Q) = (\xi^2 x_Q, y_Q)$ for $Q = (x_Q, y_Q)$ and $\theta(Q)$ becomes $\theta(x_Q, y_Q) = (\xi x_Q, y_Q) - (\xi^2 x_Q, y_Q)$. The map θ is an isomorphism from the base field subgroup to the trace zero subgroup [Sco04], where the x-coordinates lie in the base field and the y-coordinates are in the extension field. Therefore, the formulae from §7.2 can be applied. The inverse map from the trace zero subgroup to the base field subgroup is defined as $\theta^{-1}(Q) = \mathrm{Tr}(\phi(Q))$, where Tr is the trace map.

7.4 Comparison and conclusion

In this chapter pairing computations on a non-standard Weierstrass curve of the form $y^2 = cx^3 + 1$ are studied. This is the most specific curve model studied so far since there are only 3 isomorphism classes of curves for this shape in the general case where $p \equiv 1 \mod 3$. The main contribution of this chapter is a faster computation of the Tate pairing on this special curve. Practical examples of such curves can be achieved using Construction 6.6 of [FST06]. There are many examples of embedding degrees for which this construction gives the best known ρ-value [FST06], however it remains an open question to find suitable curves of this form having ρ-values very close to 1 with practically interesting embedding degrees, e.g. $k = 8$.

The following table summarizes the advantage of employing this new curve in the Tate pairing by comparing results from this work with the fastest results achieved on other j-invariant zero curves documented prior to this work. The formulae given by Arène *et al.*

[ALNR09] for j-invariant zero curves give an operation count that improves the operation count originally presented in [HSV06], so comparisons are drawn against these improved formulae. The trend of presenting the operation count for even k [KM05] is followed, since this is generally preferred in practice [BKLS02], [BLS04b]. The multiplications and squarings that take place in the extension field $\mathbb{F}_{q^k}$ are excluded, since these are common to all operation counts (see lines 5 and 9 of Algorithm B.3.1).

Tate pairing	`DBL`	`mADD`	`ADD`
Arène *et al.* [ALNR09]	$(k+3)\mathbf{M}+8\mathbf{S}$	$(k+6)\mathbf{M}+6\mathbf{S}$	$(k+12)\mathbf{M}+5\mathbf{S}$
This work	$(k+3)\mathbf{M}+5\mathbf{S}$	$(k+10)\mathbf{M}+2\mathbf{S}+1\mathbf{D}$	$(k+13)\mathbf{M}+2\mathbf{S}+1\mathbf{D}$

As k increases in the Tate pairing, the overall speed up that is achieved through using the curve $y^2 = cx^3 + 1$ becomes less, since the more difficult operations in $\mathbb{F}_{q^k}$ consume more computation relative to those operations in the base field.

Lastly, note that the EFD [BL07a] reports $2\mathbf{M} + 5\mathbf{S}$ point doubling formulae in Jacobian coordinates for j-invariant zero curves. Therefore a protocol requiring scalar multiplications should use Jacobian coordinates and should only switch to this proposal when the pairing is being computed. This conversion comes at the cost of $2\mathbf{M} + 1\mathbf{S} + 1\mathbf{D}$ by taking $(X : Y : Z)$ in Jacobian coordinates to $(XZ : Y : cZ^3)$ in homogeneous projective coordinates on the curve $y^2 = cx^3 + 1$.

Incremental results with applications to Ate pairing can be found in a recent preprint by Costello *et al.* [CLN09].

Chapter 8

Conclusion

Elliptic curve cryptography (ECC) is a main branch in public-key cryptography, based on the algebraic structure of elliptic curves over finite fields. The discovery of ECC is due to independent works of Miller [Mil86] and Koblitz [Kob87], see Appendix B.1. This thesis has focused on the group law on elliptic curves with an emphasis on efficiency issues. ECC-related high speed applications require high speed implementation of elliptic curve group laws. In this context, a systematic and automated method of finding cryptographically interesting point negation/doubling/addition formulae are described. Five forms of elliptic curves are revisited and many missed low-degree formulae are detected. With these new formulae a complete description of the group law is made in affine coordinates. The efficiency of these formulae are then investigated in suitable coordinate systems and comparisons are made in between. Theoretic results are supported with pratical applications on assembly-optimized computer programs. These contributions are summarized in §8.1 and new research directions are explained in §8.2.

8.1 Summary of research and outcomes

Chapter 2 reviewed the elliptic curve group law on Weierstrass curves together with definitions of frequently used technical terms. B Birational equivalences between selected curves and suitable Weierstrass curves were demonstrated using literature results and computer algebra. Chapter 2 also compared the estimated coverage of each studied form of elliptic curves.

Chapter 3 brought together several computational tools using fundamental results in algebraic geometry: the Riemann-Roch theorem and products of curves, and in arithmetic of function fields: Gröbner basis computations and rational simplifications. The final product is a toolbox for optimizing the group law arising from elliptic curves given in some particular form. The first tool is capable of finding group laws on elliptic curves using computer algebra. This is a high-level tool which produces massive and inefficient formulae. This tool uses birational maps between curves and symbolically deduces the group law for some form of an elliptic

curve. The second tool is responsible for rational simplification for finding lowest-degree point addition/doubling formulae. The notion of finding the lowest-degree rational expression modulo a prime ideal was developed in [MP06]. To the best of the author's knowledge, combining these two stages and systematically finding the *lowest-degree* group laws is an outcome of this thesis. The third tool contains isolated subalgorithms from the literature which naturally arise in low-degree group laws. These subalgorithms are named brain teasers in this work. A new subalgorithm has also been contributed to the current body of knowledge. This subalgorithm helps trading a multiplication with additions in the underlying field for twisted Hessian curves. The same chapter recommended the following steps when optimizing the group law on an elliptic curve.

1. Fix a curve of genus 1 with a rational point lying in a suitable affine or projective space.
2. Derive the group law using Riemann-Roch computations.
3. Simplify the negation, doubling, and addition formulae of the group law.
4. Make a collection of several equivalent low degree formulae and ensure the equivalence.
5. Find algorithms to carry out operations efficiently for each of the collected formulae.

Chapter 4 presented low-degree point addition formulae, some of which are outcomes of this thesis. The most important new formulae include dedicated addition formulae for extended Jacobi quartic, twisted Edwards, and twisted Jacobi intersection forms and a set of minimal-degree doubling formulae for extended Jacobi quartic form. A complete statement of the group law in affine coordinates was presented for each of the studied forms. These complete descriptions in affine coordinates cannot be found in the literature except for the Weierstrass and short Weierstrass forms which have been studied extensively in the past. The algorithms contributed are Algorithm 4.2.1 in §4.2, Algorithm 4.3.1 in §4.3, Algorithm 4.4.1 in §4.4, and Algorithm 4.5.1 in §4.5. In order to justify these algorithms each section contains a series of lemmas to systematically investigate exceptional situations that might appear in the computation. All of the proposed algorithms contain several conditional branches. In an optimized implementation these branches are best eliminated. This is achieved with two methods. The first method was initiated by Bernstein and Lange in [BL07b] for Edwards curves and was extended to suitable classes of elliptic curves in twisted Edwards form in [BBJ+08] and to twisted Hessian form in [BL07a]. This technique forces the point(s) at infinity to be defined over a proper extension of $\mathbb{K}$ but not defined over $\mathbb{K}$. Therefore, all points on the selected curve are affine points. The rest of the method is composed of finding a single set of addition formulae with a denominator which cannot vanish for any pair of summands. This chapter has extended the same idea for suitable classes of elliptic curves in extended Jacobi quartic and twisted Jacobi intersection forms, see §4.2 and §4.5 respectively. The second method selects a suitable subgroup of points which does not contain any points at infinity. The rest of the method is again composed of finding a single set of addition formulae with denominators which cannot vanish for any pair of summands. Using this method, it was shown how to prevent all exceptions of dedicated addition formulae for distinct inputs. This latter contribution is

perfectly suited to the context of fast scalar multiplications, since dedicated additions are more efficient than unified additions in almost all cases.

Chapter 5 proposed inversion-free point doubling and point addition algorithms using affine formulae presented in Chapter 4. Many sets of addition/doubling formulae were investigated for the best operation counts with optimized algorithms in various point representations. In particular the formulae (5.2), (5.5), (5.7), (5.8), (5.9) in §5.1; (5.12), (5.13), (5.14), (5.15), (5.16), (5.17), (5.18), (5.19) in §5.2; (5.21), (5.22) in §5.3; (5.27), (5.28), (5.29) in §5.4 are contributions of this thesis. All algorithms following these formulae are also new contributions. In the majority of cases the proposed algorithms are more efficient than the alternatives in the literature. For other formulae which are revisited from the literature, some improved operation counts are also achieved. For full details and comparison to previous results see §5.6. The same chapter has also contributed to overturning a common belief that the group laws arising from quartic forms are less efficient than that of cubic forms. Most notably, the latest results for the quartic forms, extended Jacobi quartic and twisted Edwards forms, were further improved by removal of several multiplications in suitable coordinate systems. Similar improvements were also obtained for twisted Jacobi intersection form. For the cubic forms –twisted Hessian and short Weierstrass forms–, conditional speed improvements were achieved. See Table 5.11 in §5.6 for a theoretical comparison of the efficiency of each form.

Chapter 6 showed that the proposed algorithms can practically speed up the scalar multiplications and thus the ECC applications. The details of these algorithms can be extracted from Appendix C. The efficiency of using different elliptic curve forms are compared. In this context, the best speeds are achieved with twisted Edwards form and extended Jacobi quartic form for variable-single-point variable-single-scalar multiplication. Twisted Edwards is fastest for addition intensive operations such as fixed-single-point variable-single-scalar multiplication. Other generalizations of scalar multiplications are left as future work. However, similar outcomes can be expected.

Chapter 7 considered, as a case study, the relevance of the techniques in this thesis to efficient Tate pairing computations. In particular, a non-standard Weierstrass curve $y^2 = cx^3 + 1$ is studied together with new minimal-degree doubling formulae which resulted in best-so-far operation counts for frequently accessed encapsulated point-doubling and line-computations. The techniques used in Chapter 5 are applied to derive these new formulae with an extension to the derivation of line formula. Computer implemention of these formulae is left as future work. On the other hand, it is natural to expect pratical speed-ups building on the the results of Chapter 6.

8.2 Future research ideas

This thesis answered many efficiency-related questions about elliptic curve group laws and made contributions to the current body of knowledge. In doing so, several research issues have arisen. These are summarized as follows.

Higher degree, higher genus, abelian varieties, embeddings to higher dimensional spaces, small characteristic The scope of this thesis is limited to five forms of elliptic curves over fields of large characteristic. It is possible to extend this work to elliptic curves over fields of characteristic 2 and 3. Similarly, other forms of elliptic curves are also open to the same treatment. Note that these other forms can be plane curves or embedding of a plane curve into a higher dimensional space. Yet another direction of research is studying higher genus curves or arbitrary abelian varieties.

More formulae/algorithms This thesis has focused on point doubling and point addition formulae. The emphasis has been on finding lowest degree rational expressions and on developing algorithms with the best operation counts. However, obtaining minimal degree formulae is not a concrete proof for the most efficient way of computation. Higher degree formulae in some specific cases may turn out to be more efficient. Therefore, further investigation of more formulae will be valuable.

Hybrid group laws The group law is usually interpreted as a mapping from $E \times E$ to E for an elliptic curve E. However, no efficiency-related work has been considered for the mappings of the form $E_1 \times E_2 \to E_3$ where E_1, E_2, and E_3 are birationally-equivalent/isogenous curves but not necessarily equal. It is worth extending the current research towards this idea. Moreover, like the point doublings, point tripling, point quintupling, etc. can also be of interest in some higher level computations. This is a big gap in the literature to be filled.

More experiments The experiments of Chapter 6 are limited to single-scalar multiplications with variable or fixed base-points. Further implementations for multi-scalar multiplication with variable or fixed base-points is left as future work. Chapter 6 does not make any comparisons with Montgomery ladder technique and recent advances in that area. A further implementation with a fair efficiency comparison is missing in the literature.

Assembly support for other processors The implementation introduced in Chapter 6, has a `C`-only mode where the project can be compiled for any processors using a suitable `C` compiler. On the other hand, assembly support for the finite field layer can significantly improve the throughput for some specific processors. This support is only provided for processors which can execute generic `x86-64` instructions and this part of the implementation was specifically optimized for Core 2 processors. Optimized support for other types of processors will shed more light on the usefulness of the algorithms proposed in this thesis, e.g. lightweight smart card applications or wireless sensor network devices.

Appendix A

Mathematical definitions

This appendix provides definitions for background concepts which are mainly used in Chapters 2, see §A.1, §A.2, and §A.3. In particular, the most important objects of study in this thesis –*elliptic curves*– are formally defined. In addition, basic results of the Gröbner basis theory are summarized in the context of the arithmetic of polynomial ideals, see §A.5.

A.1 Preliminaries

This section makes a formal definition of a *curve*. The relevant information is extracted from standard books [Ful69], [Har77], [Sil94], [Sti93], [CF05], and [CLO07]. The notation is adapted from [Sil94], [Sti93], and [CF05]. It is assumed that the reader is familiar with sets, groups, (polynomial) rings, ideals, quotient algebras, vector spaces, affine (projective) spaces and their topologies, fields, field extensions for which one can always consult the aforementioned books.

Throughout this chapter assume that $\mathbb{K}$ is a perfect field, $\overline{\mathbb{K}}$ its algebraic closure, $\mathbb{L}$ an extension of $\mathbb{K}$ contained in $\overline{\mathbb{K}}$. An affine (resp. projective) n-space over $\mathbb{K}$ will be denoted by $\mathbb{A}^n$ (resp. by $\mathbb{P}^n$). The "resp. " part in the previous sentence will be omitted for brevity hereafter. It is useful to keep in mind that a point of an n-space over $\mathbb{K}$ has coordinates in $\overline{\mathbb{K}}$. The $\mathbb{L}$-rational points of $\mathbb{A}^n$ ($\mathbb{P}^n$) will be denoted by $\mathbb{A}^n(\mathbb{L})$ $\big(\mathbb{P}^n(\mathbb{L})\big)$. The projective closure of an algebraic set V will be denoted by $\overline{V}$. The ideal of an affine (projective) algebraic set V will be denoted by $I(V)$ or by I for shorthand. Note that $I(V)$ is an ideal of $\overline{\mathbb{K}}[x_1, \ldots, x_n]$ $\big(\overline{\mathbb{K}}[X_0, \ldots, X_n]\big)$ for which the variables $x_1, \ldots, x_n$ $(X_0, \ldots, X_n)$ will sometimes be abbreviated as "." when clear from the context. It is useful to keep in mind that V *defined over* $\mathbb{K}$ means $I(V)$ is an ideal of $\mathbb{K}[\,.\,]$. An algebraic set V defined over $\mathbb{K}$ will be denoted by $V/\mathbb{K}$. The set of $\mathbb{L}$-rational points of $V/\mathbb{K}$ will be denoted by $V(\mathbb{L})$.

Definition A.1.1. An irreducible affine (projective) algebraic set V (with respect to the Zariski topology), is called an **affine (projective) variety**.

For an explanation of open and closed sets defining the Zariski topology of affine and projective n-spaces see [CF05, §4.1.1]. An affine (projective) algebraic set V is an affine

(projective) variety if and only if $I(V)$ is a (homogeneous) prime ideal of $\mathbb{K}[\,.\,]$, see [CF05, Proposition 4.11].

Definition A.1.2. An affine (projective) variety V defined over $\overline{\mathbb{K}}$ is called **absolutely irreducible**.

Definition A.1.3. Let $V/\mathbb{K}$ be a variety and I its ideal. The quotient ring $\mathbb{K}[V] \stackrel{\text{def}}{=} \mathbb{K}[\,.\,]/I$ is called the **coordinate ring of** V. An element of $\mathbb{K}[V]$ is called a **polynomial function on** V.

Let $V/\mathbb{L}$ be a variety. The coordinate ring $\mathbb{K}[V]$ is an integral domain (i.e. a commutative ring with unity containing no zero divisors) since I is a prime ideal in $\mathbb{K}[\,.\,]$. Note that $\overline{\mathbb{K}}[V]$ is not an integral domain unless V is absolutely irreducible.

Definition A.1.4. The set of all fractions r/s such that $r, s \in \mathbb{K}[V]$ and $s \neq 0$ form the **function field of** $\mathbb{K}[V]$, denoted by $\mathbb{K}(V)$. An element of $\mathbb{K}(V)$ is called a **rational function** on V.

Definition A.1.5. The transcendence degree of $\mathbb{K}(V)$ over $\mathbb{K}$ is called the **dimension of** V. The dimension of V is denoted by $\dim V$.

For a definition of *transcendence degree* consult [Hun74, VI.1.10] or [BGM+93, §9.5].

Definition A.1.6. An affine (projective) variety V with $\dim V = 1$ is called an **affine (projective) curve**.

A.2 Birational equivalence and isomorphism

Definition A.2.1. Let m and n be positive integers. Let $C_1 \subset \mathbb{A}^m(\mathbb{K})$ and $C_2 \subset \mathbb{A}^n(\mathbb{K})$ be affine curves. A **rational map** ϕ from C_1 to C_2 is a fraction of two polynomial functions given by

$$\phi\colon C_1 \to C_2, (x_1, \ldots, x_m) \mapsto \Big(\frac{f_1(x_1, \ldots, x_m)}{g_1(x_1, \ldots, x_m)}, \ldots, \frac{f_n(x_1, \ldots, x_m)}{g_n(x_1, \ldots, x_m)}\Big)$$

where the $f_i/g_i \in \mathbb{K}(x_1, \ldots, x_m)$ satisfy:

(i) each ϕ is defined at some point of C_1.

(ii) For every $(\alpha_1, \ldots, \alpha_m) \in C_1$ where ϕ is defined, $\phi(\alpha_1, \ldots, \alpha_m) \in C_2$.

The notion $\phi(x_1, \ldots, x_m)$ will be abbreviated as ϕ when $x_1, \ldots, x_m$ is implicit in the meaning. The notation for rational maps is borrowed from [Sil94].

Definition A.2.2. A rational map

$$\phi\colon C_1 \to C_2,\ (.) \mapsto (f_0(.), f_1(.), \ldots, f_n(.))$$

is **regular** (or defined) at $P \in C_1$ if there is a function $g \in \overline{\mathbb{K}}(C_1)$ such that

(i) each gf_i is regular at P;

(ii) there is some i for which $(gf_i)(P) \neq 0$.

If such a g exists then ϕ is given by

$$\phi(P) = [(gf_0)(P), \ldots, (gf_n)(P)].$$

It may be necessary to take different g's for different points.

Definition A.2.3. A rational map that is regular at every point is called a **morphism**.

The rational maps are used for defining birational equivalence between affine curves in Chapter 4. Therefore, Definitions A.2.1 and A.2.2 are not extended for projective curves. For relevant definitions see [Sti93], [CF05].

Definition A.2.4. Let m and n be positive integers. Let $C_1 \subset \mathbb{A}^m(\mathbb{K})$ and $C_2 \subset \mathbb{A}^n(\mathbb{K})$ be curves. Let $\phi : C_1 \to C_2$ and $\psi : C_2 \to C_1$ be rational maps defined over $\mathbb{K}$ such that $\psi \circ \phi$ and $\phi \circ \psi$ are equal to the identity maps on C_1 and C_2, respectively. The identity maps are typically denoted by id. Then, C_1 and C_2 are said to be **birationally equivalent over** $\mathbb{K}$ or simply **birational**.

Note that alternative notations $g * f$ or $f(g)$ or fg are preferred in some other resources for the very same composition $f \circ g$.

Definition A.2.5. Let C_1, C_2, ϕ, ψ be defined as in Definition A.2.4. Then C_1 and C_2 are said to be **isomorphic over** $\mathbb{K}$ provided that both ϕ and ψ are morphisms.

A.3 Riemann-Roch theorem

This section makes a formal definition of an *elliptic curve* with the help of Riemann-Roch theorem (for curves) which has several applications in algebraic geometry. The role of Riemann-Roch theorem in this chapter is limited to finding the Weierstrass form of elliptic curves. Details of the topic can be found in standard references [Ful69], [Sil94], and [CF05]. General treatments can be found in [Har77] and [Sti93]. Before stating the theorem some definitions are required.

Let P be a point on a curve C. A subring of $\overline{\mathbb{K}}(C)$ defined by

$$\overline{\mathbb{K}}[C]_P \stackrel{\text{def}}{=} \{f \in \overline{\mathbb{K}}(C) \mid f = \frac{r}{s} \text{ for some } r, s \in \overline{\mathbb{K}}[C] \text{ with } s(P) \neq 0\}$$

is a (local) ring with a maximal ideal $\mathfrak{m}_P \stackrel{\text{def}}{=} \{f \in \overline{\mathbb{K}}[C]_P \mid f(P) = 0\}$, see [Sti93, Definition I.1.8] and [Sil94].

Definition A.3.1. A point P having $\overline{\mathbb{K}}[C]_P$ as a discrete valuation ring is called **non-singular**. A curve C having no singular point is non-singular.

For a definition of *discrete valuation* consult [Sti93, Definition I.1.9]. Let C be a curve. A point P on C is non-singular if and only if $\dim_{\overline{\mathbb{K}}} \mathfrak{m}_P/\mathfrak{m}_P^2 = 1$, see [Har77, §I.5.1].

Definition A.3.2. Let C be a curve, $P \in C$ a non-singular point, and f a rational function on C. The integer defined by

$$\mathrm{ord}_P : \overline{\mathbb{K}}[C]_P \to \{0, 1, 2, \ldots\} \cup \{\infty\},\ f \mapsto \max\{i \in \mathbb{Z} \mid f \in \mathfrak{m}_P^i\},$$

is the **valuation of f at P**.

Note that the elements of $\overline{\mathbb{K}}[C]_P$ are rational functions. Let $f' = r/s \in \overline{\mathbb{K}}(C)$ such that $r, s \in \overline{\mathbb{K}}[C]$. It is convenient to redefine ord_P by defining

$$\mathrm{ord}_P : \overline{\mathbb{K}}(C) \to \mathbb{Z} \cup \{\infty\},\ f' \mapsto \mathrm{ord}_P(r) - \mathrm{ord}_P(s).$$

Definition A.3.3. Let C be a curve. The free abelian group generated by the points of a smooth curve C, is called the **divisor group of** C. The divisor group of C is denoted by $\mathrm{Div}(C)$. An element of $\mathrm{Div}(C)$ defined by the formal sum

$$D \stackrel{\text{def}}{=} \sum_{P \in C} n_P P$$

where $n_p \in \mathbb{Z}$ and $n_P = 0$ for all but finitely many P, is called a **divisor on** C. The integer defined by $\deg(D) \stackrel{\text{def}}{=} \sum n_P$ is called the **degree of** D. A divisor $D = \sum_{P \in C} n_P P$ such that $n_P \geq 0$ for every $P \in C$ is called **effective**. An effective divisor is denoted by

$$D \geq 0.$$

Let $D_1, D_2 \in \mathrm{Div}(C)$. The meaning of $D_1 - D_2$ being effective is implied by $D_1 \geq D_2$. Let f be a rational function in $\overline{\mathbb{K}}(C) \backslash \{0\}$. The divisor defined by

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

is called the **divisor of** f.

Definition A.3.4. Let D be divisor in $\mathrm{Div}(C)$. The set of functions on C defined by

$$L(D) \stackrel{\text{def}}{=} \{f \in \overline{\mathbb{K}}(C) \backslash \{0\} \mid \mathrm{div}(f) \geq -D\} \cup \{\infty\},$$

is called the **Riemann-Roch space of** D.

$L(D)$ is a finite dimensional $\overline{\mathbb{K}}$-vector space, see [Ful69, §8.1].

Definition A.3.5. Let D be divisor in $\mathrm{Div}(C)$. The integer $\ell(D)$ defined by the dimension of $L(D)$ as

$$\ell(D) \stackrel{\text{def}}{=} \dim_{\overline{\mathbb{K}}} L(D),$$

is called the **dimension of** $L(D)$.

For a definition of the dimension of a vector space consult [Fra76, §8.2].

Theorem A.3.6 (Riemann-Roch)**.** *Let $C/\mathbb{K}$ be a curve with function field $\mathbb{K}(C)$. There exists a non-negative integer g such that*

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

for every divisor $D \in \mathrm{Div}(C)$. Furthermore, if $D \in \mathrm{Div}(C)$ and $\deg(D) \geq 2g - 2$ then $\ell(D) = \deg(D) - g + 1$.

Several books contain the full version of Theorem A.3.6, cf. [Har77] or [Sti93]. This is a shortened version from [CF05]. Practical applications of this theorem are used to state birational maps between curves in §2.3 of Chapter 2 and to state group laws in Chapter 3. For computational aspects of the theorem see [Hes02].

Definition A.3.7. The positive integer g in Theorem A.3.6 is called the **genus** of C.

The chief object of study can now be defined as follows.

Definition A.3.8. A curve $C/\mathbb{K}$ which

(i) is projective, (ii) is of genus 1,

(iii) is absolutely irreducible, (iv) is non-singular,

(v) contains at least one $\mathbb{K}$-rational point,

is called an **elliptic curve**.

Example A.3.9. The plane curve $C : y^2 = dx^4 + 2ax^2y^2 + y^4$ over $\mathbb{Q}$ is of genus 1 and is absolutely irreducible. The point $(0, 1)$ is a $\mathbb{Q}$-rational point. The point $(0, 0)$ is singular. The desingularization $\widetilde{C}$ of the projective closure $\overline{C}$ is an *elliptic curve*.

Note that $\widetilde{C}$ in Example A.3.9 is not a plane curve (emphasizing that the projection is homogeneous), cf. see [CF05, §4.4.1].

A.4 Divisor class group

Throughout this section $C/\mathbb{K}$ denotes a smooth curve, f a rational function in $\overline{\mathbb{K}}(C)\backslash\{0\}$.

Definition A.4.1. Let $C/\mathbb{K}$ be a smooth curve, f a rational function in $\overline{\mathbb{K}}(C)\backslash\{0\}$. A **principal divisor** $D \in \mathrm{Div}(C)$ is a divisor of the form $D = \mathrm{div}(f)$. Two divisors D_1, D_2 are called **linearly equivalent** provided that $D_1 - D_2$ is a principal ideal.

Two linearly equivalent divisors are denoted by $D_1 \sim D_2$. The divisors of degree 0 form a subgroup of $\mathrm{Div}(C)$, denoted by $\mathrm{Div}^0(C)$. The set of all principal divisors on C is denoted by $\mathrm{Prin}(C)$. If two divisors are principal so is their sum. $\mathrm{Prin}(C)$ is a subgroup of $\mathrm{Div}(C)$, where each principal divisor is of degree 0. Therefore $\mathrm{Prin}(C)$ is also a subgroup of $\mathrm{Div}^0(C)$.

Definition A.4.2. The **divisor class group (Picard group) of** C is the quotient group $\mathrm{Pic}(C) \stackrel{\text{def}}{=} \mathrm{Div}(C)/\mathrm{Prin}(C)$. The **degree zero part of** $\mathrm{Pic}(C)$ is the subgroup $\mathrm{Pic}^0(C) \stackrel{\text{def}}{=} \mathrm{Div}^0(C)/\mathrm{Prin}(C)$.

Theorem A.4.3. *Let $E_{\mathbf{W}}$ be an elliptic curve with a fixed $\mathbb{K}$-rational point $\mathcal{O}$.* $\mathrm{Pic}^0(E_{\mathbf{W}})$ *is isomorphic over $\mathbb{L}$ to $E_{\mathbf{W}}(\mathbb{L})$.*

See [Sil94, Proposition 3.4(a,b,c,d)] for a proof.

Since $E(\mathbb{L})$ is a group by Theorem A.4.3, it is natural to ask how to perform negation —i.e. inversion in multiplicative groups— and addition —i.e. multiplication in multiplicative groups— in $E(\mathbb{L})$. A succinct way of describing this group law in terms of the coordinates of points is presented in §2.2 of Chapter 2. See [Sil94, Proposition 3.4(e)] for backtracking the connection between the group law and the Picard group.

A.5 Arithmetic of ideals

This section initially defines polynomial division then provides well-known results for the arithmetic of polynomial ideals. The reader is assumed to be familiar with polynomial rings and monomial orders. For a comprehensive background, standard references are [BW93] and [CLO07]. Throughout this section $\mathbb{K}$ denotes a field and $\mathbb{K}[x_1, \ldots, x_n]$ denotes a polynomial ideal over $\mathbb{K}$ in n variables.

Definition A.5.1. Fix a monomial order $>$. Let $f = \sum_i a_i x^i$ be a nonzero polynomial in $\mathbb{K}[x_1, \ldots, x_n]$.

(i) The **multi-degree of** f is $\mathrm{MD}(f) \stackrel{\text{def}}{=} \max\{i \in \{0, 1, 2, \ldots\} \mid a_i \neq 0\}$ where the maximum is taken with respect to $>$.

(ii) The **leading coefficient of** f is $\mathrm{LC}(f) \stackrel{\text{def}}{=} a_{\mathrm{MD}(f)}$.

(iii) The **leading monomial of** f is $\mathrm{LM}(f) \stackrel{\text{def}}{=} x^{\mathrm{MD}(f)}$ with coefficient 1.

(iv) The **leading term of** f is $\mathrm{LT}(f) \stackrel{\text{def}}{=} \mathrm{LC}(f) \cdot \mathrm{LM}(f)$.

Theorem A.5.2 (Division in a polynomial ring)**.** *Fix a monomial order. Let $(f_1, \ldots, f_s)$ be an ordered s-tuple of polynomials in $\mathbb{K}[x_1, \ldots, x_n]$. Then every $f \in \mathbb{K}[x_1, \ldots, x_n]$ can be written as*

$$f = a_1 f_1 + \ldots + a_s f_s + r$$

where $a_i, r \in \mathbb{K}[x_1, \ldots, x_n]$ and either $r = 0$ or r is a linear combination of monomials (with coefficients in $\mathbb{K}$) none of which is divisible by any of $\langle \mathrm{LT}(f_1), \ldots, \mathrm{LT}(f_s) \rangle$. Furthermore, if $a_i f_i \neq 0$ then $\mathrm{MD}(f) \geq \mathrm{MD}(a_i f_i)$.

See [CLO07, Theorem 3, §2.3] for the proof.

Definition A.5.3. The polynomial r is called a **remainder** of f on division by $(f_1, \ldots, f_s)$. An **exact division** is the division which produces $r = 0$.

Theorem A.5.4 (Hilbert basis)**.** *Every ideal $I \subset \mathbb{K}[x_1, \ldots, x_n]$ is finitely generated.*

See [CLO07, Theorem 4, §2.5] for the proof.

Definition A.5.5. Fix a monomial order. A **Gröbner basis** is a finite subset $G = \{g_1, \ldots, g_t\}$ of an ideal I with $\langle \mathrm{LT}(g_1), \ldots, \mathrm{LT}(g_t) \rangle = \langle \mathrm{LT}(I) \rangle$.

Theorem A.5.6 (Existence of a Gröbner basis)**.** *Fix a monomial order. Every ideal $I \subset \mathbb{K}[x_1, \ldots, x_n]$ other than $\{0\}$ has at least one Gröbner basis. Any Gröbner basis for an ideal I is a basis of I.*

See [CLO07, Corollary 6, §2.5] for the proof.

Definition A.5.7. A **reduced Gröbner basis** for a polynomial I is a Gröbner basis G for I such that for all $g \in G$, $\mathrm{LC}(g) = 1$ and for all $g \in G$, no monomial of g lies in $\langle \mathrm{LT}(G \backslash \{g\}) \rangle$.

Theorem A.5.8 (Uniqueness of the reduced Gröbner basis)**.** *Let $I \neq \{0\}$ be a polynomial ideal. Then, for a given monomial ordering, I has a unique reduced Gröbner basis.*

See [CLO07, Proposition 6, §2.7] for the proof. A reduced Gröbner basis can be computed by Buchberger's algorithm [Buc65] or its variants.

Definition A.5.9. The **normal form** of a polynomial f is the remainder of f on division by a Gröbner basis.

The following definitions will be used in Chapter 3. Not all operations are given here. For details see [BW93] and [CLO07].

Definition A.5.10. If I and J are ideals in $\mathbb{K}[x_1, \ldots, x_n]$ then $I \cap J$ is the set

$$f \in \mathbb{K}[x_1, \ldots, x_n] \colon f \in I \text{ and } f \in J$$

and is called the **intersection** of I and J.

If I and J are ideals in $\mathbb{K}[x_1, \ldots, x_n]$ then $I \cap J$ is also an ideal, see [CLO07, Proposition 9, §4.3].

Theorem A.5.11 (Computing intersections of ideals)**.** *Let I, J be ideals in $\mathbb{K}[x_1, \ldots, x_n]$. Then $I \cap J = (tI + (1-t)J) \cap \mathbb{K}[x_1, \ldots, x_n]$.*

See [CLO07, Theorem 11, §4.3] for the proof.

Definition A.5.12. If I and J are ideals in $\mathbb{K}[x_1, \ldots, x_n]$ then $I \colon J$ is the set

$$f \in \mathbb{K}[x_1, \ldots, x_n] \colon fg \in I \text{ for all } g \in J$$

and is called the **colon ideal** of I by J.

Theorem A.5.13 (Computing colon ideals)**.** *Let I be an ideal and g an element of $\mathbb{K}[x_1, \ldots, x_n]$. If $\{h_1, \ldots, h_m\}$ is a basis of the ideal $I \cap \langle g \rangle$ then $\{h_1/g, \ldots, h_m/g\}$ is a basis of $I \colon \langle g \rangle$. Each operation h_i/g is an exact division.*

See [CLO07, Theorem 11, §4.4] for the proof.

Definition A.5.14. Let $\mathbb{K}$ be a perfect field, $V/\mathbb{K}$ a variety, I the ideal of V. The **dimension of** I is the dimension of V, see §A.1.

Appendix B

Elliptic Curve Cryptography

The need for information security has been emerging as an indispensable component of communication between parties through public channels. An ancient science, cryptography, plays a central role in this context since its main concerns are hiding the meaning of messages, signing messages, providing message authentication, non-repudiation and integrity. Thus, much attention has been focused on cryptography as an important asset in providing information security.

The boundaries of cryptography grew considerably after Diffie and Hellman [DH76] introduced the concept of public-key (asymmetric-key) cryptography in 1976 as an alternative to secret-key (symmetric-key) cryptography. Public-key cryptography drew attention quickly since it provided a solution to the key distribution problem[1] which is the most serious disadvantage of secret-key cryptography. The idea behind public-key cryptography is to use two different (but mathematically related) keys where one of the keys is used as an input in a cryptographic function (e.g. encrypt, sign) and the other key contains the necessary information to invert the steps carried out previously. These functions are also called one-way trapdoor functions which are *easy*[2] to compute but *hard*[3] to invert. However, they can be *easily* inverted with the trapdoor information.

The security of asymmetric cryptosystems is based on the intractability of one-way functions. The discrete logarithm problem (DLP) is one of the most promising instances of such functions that are used in cryptography. The use of DLP over elliptic curves was proposed independently by Miller [Mil86] and Koblitz [Kob87] in 1986. Starting from the early 1990's Elliptic Curve Cryptography (ECC) has received great interest from cryptologists and has been enhanced in theoretical and practical ways.

[1] The secret key which is to be used by two parties, must be kept secret. However, one of the two parties needs a new secret key to transfer the secret key to the other party as a cipher-text. This results in a chicken-egg problem.

[2] *Easy* means the function can be evaluated in polynomial time (in the input size) by some algorithm.

[3] *Hard* means no probabilistic polynomial-time algorithm exists.

B.1 Elliptic curve cryptosystems

A one-way function f is called a trapdoor one-way function if there exists trapdoor information under which f is easily inverted. Without the knowledge of the trapdoor information it should be computationally hard to perform the inversion. Elliptic Curve Discrete Logarithm Problem (ECDLP) satisfies these requirements. Thus, it is a good candidate to be embedded into the well known public key schemes (which are based on cyclic groups). This section contains the description of common schemes that make use of elliptic curves.

Diffie and Hellman [DH76] initiated so-called *Public Key Cryptography* by showing how two independent parties can define a shared secret key over a public channel. Section B.1.2 discusses the ECC analogue of the *Diffie-Hellman Key Exchange*. ElGamal [ElG85] used the discrete logarithm problem in a cyclic subgroup, which is generated by a primitive element, to build an alternative encryption-decryption-signature scheme to Rivest-Shamir-Adleman [RSA78] (RSA) cryptosystem. The elliptic curve analogue of the ElGamal cryptosystem is given in Section B.1.3. Section B.1.4 gives the basics of the *Elliptic Curve Digital Signature Algorithm* (ECDSA) which is the elliptic curve analogue of the *Digital Signature Algorithm* (DSA)[4]. The protocol attacks are omitted in this appendix since the main interest is efficient computation.

B.1.1 Key-pair generation

Public key systems require the selection of a public key and a private key as inputs to the encryption and decryption schemes respectively. The public and the private keys are algebraically related to each other by $Q = [m]P$ where Q is the public key, m is the private key and P is the primitive (base) point of $\langle P \rangle$. The order of $\langle P \rangle$ is denoted by $|\langle P \rangle|$. Algorithm B.1.1 shows the steps.

Algorithm B.1.1: Elliptic Curve Key-pair Generation.

input : All necessary parameters for $P \in E(\mathbb{F}_q)$.
output : Public key Q and private key m.

1 Select a random m, $0 < m < |\langle P \rangle|$.
2 Compute $Q = [m]P$.
3 **return** (Q, m).

The random selection of m is of crucial importance to the security of the whole system. However, generating selections scientifically at random is not possible with the contemporary architecture. The notion of randomness goes much deeper. Nevertheless, a pseudo-random selection is suitable for cryptographic use. Such a selection can be performed with the help of a secret seed and a random bit-generation function (see Wagstaff [Wag02, Chapter 15]).

[4]DSA is a United States Federal Government standard for digital signatures (FIPS 186). It was proposed by the National Institute of Standards and Technology (NIST) in August 1991.

B.1.2 Diffie-Hellman key exchange

Suppose that Alice and Bob (the security protocols' common figures) want to share a key which will be used later for some secret key scheme. This can be achieved over an authenticated public communication environment by following the steps of Algorithm B.1.2.

Algorithm B.1.2: Elliptic Curve Diffie-Hellman Key Exchange Scheme.

input : All necessary parameters for $P \in E(\mathbb{F}_q)$.

1. Alice selects a random m_A, $0 < m_A < |\langle P \rangle|$, the order of $\langle P \rangle$.
2. Alice sends $Q_A = [m_A]P$ to Bob.
3. Bob selects a random m_B, $0 < m_B < |\langle P \rangle|$.
4. Bob sends $Q_B = [m_B]P$ to Alice.
5. Alice computes $[m_A]Q_B$.
6. Bob computes $[m_B]Q_A$.
7. Now both have $[m_A m_B]P$ as the shared secret key.

An intruder (Eve) who copies the transmitted data only has $E(\mathbb{F}_q), P, Q_A, Q_B$ which is not enough to feasibly compute $[m_A m_B]P$. To perform an algebraic attack, Eve needs to solve one of the elliptic curve discrete logarithm problem instances $Q_A = [m_A]P$ or $Q_B = [m_B]P$ for m_A and m_B, respectively. Note that this is the best known approach but it is not known to be equivalent to ECDLP.

B.1.3 ElGamal cryptosystem

The original work of ElGamal [ElG85] is built on the use of the cyclic group $\mathbb{F}_p^*$. One can replace this group by other groups in which group operations are *easy* to apply and discrete logarithms are *hard* to compute. Candidates other than the group of points on an elliptic curve are

- the class group of an imaginary quadratic field [BW88],
- the group of units in $\mathbb{Z}/\mathbb{Z}_n$ for composite n [McC88],
- the group of nonsingular matrices over a finite field [OVS84], and
- the Jacobian of an hyperelliptic curve [Kob89].

The elliptic curve analogues of ElGamal encryption and decryption schemes are given in Algorithm B.1.3 and Algorithm B.1.4 respectively.

Algorithm B.1.3: Elliptic Curve ElGamal Encryption.

input : All necessary parameters for $P \in E(\mathbb{F}_q)$, public key Q, and plaintext M.
output : Ciphertext C_0, C_1.

1 Embed M into a point $T \in E(\mathbb{F}_q)$.
2 Select a random m, $0 < m < |\langle P \rangle|$.
3 Compute $C_0 = [m]P$.
4 Compute $C_1 = T + [m]Q$.
5 **return** (C_0, C_1).

Algorithm B.1.4: Elliptic Curve ElGamal Decryption.

input : All necessary parameters for $P \in E(\mathbb{F}_q)$, private key k, and ciphertext C_0, C_1.
output : Plaintext M.

1 Compute $T = C_1 - [k]C_0$.
2 Lookup M from T.
3 **return** (M).

Like the Diffie-Hellman key exchange scheme, the security of the elliptic curve ElGamal cryptosystem is also based on the hardness of the ECDLP. See Section B.2.4 for further discussion.

B.1.4 Elliptic curve digital signature algorithm

The definitions of a digital signature and other related concepts are given by Menezes [MOV96] in detail. The elliptic curve analogues of ElGamal digital signature generation and verification schemes are given in Algorithm B.1.5 and Algorithm B.1.6 respectively. Variations of these algorithms are standardized in ANSI X9.62, FIPS 186-2, IEEE 1363-2000, and ISO/IEC 15946-2.

Algorithm B.1.5: Elliptic Curve Digital Signature Generation.

input : All necessary parameters for $P \in E(\mathbb{F}_q)$, private key k, message M, a suitable hash function H.
output : Signature (s_0, s_1).

1 Select a random m, $0 < m < |\langle P \rangle|$.
2 Compute $[m]P$ and treat the x-coordinate as an integer i_m.
3 Set $s_0 \equiv i_m \pmod{|\langle P \rangle|}$. If $s_0 = 0$ go to step 1.
4 Compute $s_1 = k^{-1}(H(M) + ks_0) \pmod{|\langle P \rangle|}$. If $s_1 = 0$ go to step 1.
5 **return** (s_0, s_1).

Algorithm B.1.6: Elliptic Curve Digital Signature Verification.

input : All necessary parameters for $P \in E(\mathbb{F}_q)$, public key Q, signature (s_0, s_1), the message M, the hash function H.

output : r ={True, False} for the acceptance or the rejection of (s_0, s_1), respectively.

```
1  Set r =False.
2  if 0 < s_0, s_1 < |⟨P⟩| is satisfied then
3      Compute t_0 = s_1^{-1} s_0 (mod |⟨P⟩|), t_1 = s_1^{-1} H(M) (mod |⟨P⟩|).
4      Compute T = [t_0]P + [t_1]Q.
5      if T ≠ O then
6          Treat the x-coordinate of T as an integer i_T.
7          if s_0 ≡ i_T (mod |⟨P⟩|) then
8              r =True.
9          end
10     end
11 end
12 return r.
```

The selected hash function H is assumed to be cryptographically secure (preimage resistant and collision resistant).

B.2 Discrete Logarithms

The complexity of computing logarithms in an arbitrary group, G, depends on the structural properties of G. In particular, computing logarithms takes only linear time in some groups (e.g. $G = \mathbb{Z}_p^+$). Nechaev [Nec94] and Shoup [Sho97] introduced the generic group model for proving lower bounds on the discrete logarithm problem. They concluded that the discrete logarithm problem requires $O(\sqrt{p})$ group operations.

The security of ECC is based on the intractability of Elliptic Curve Discrete Logarithm Problem (ECDLP). It is believed that computing logarithms in the group generated by a rational point on an elliptic curve over a finite field is highly intractable under some reasonable assumptions. These situations are going to be made clear throughout the section.

Given $P \in E(K)$ and $Q \in \langle P \rangle$, the problem is to find the smallest positive value of $m \in \mathbb{Z}$ such that $[m]P = Q$. This can be carried out naïvely by trying every possible value for $0 < m < \#E(K)$. This method is called exhaustive search which obviously requires $O(n)$ steps with $n = |\langle P \rangle|$. For commonly used curves, $n = 2^{160} \approx 10^{48}$, thus exhaustive search is clearly infeasible.

B.2.1 Growth of order

Let $\epsilon > 0$ such that ϵ does not depend on n. An algorithm which halts in $O(\epsilon^n)$ steps is of exponential complexity[5]. It is convenient to form categories for the classification of algorithms as follows. Note c is a constant number dependent on the details of the algorithm.

$$\begin{aligned} \text{Linear} &\iff O(1) \\ \text{Polynomial} &\iff O(n) \\ \text{Superpolynomial} &\iff O((\log n)^{c \log\log\log n}) \\ \text{Subexponential} &\iff O(n^{c\sqrt{\log\log n/\log n}}) \\ \text{Exponential} &\iff O(\epsilon^n) \end{aligned}$$

These categories were developed largely by Knuth [Knu97] and are contained in his books on computer science. Subexponential time algorithms are at the heart of cryptanalytic interest. Therefore, its definition[6] is extended as in (B.1).

$$L_n(\gamma, c) \stackrel{\text{def}}{=} e^{c\sqrt{(\ln n)^{\gamma}(\ln\ln n)^{1-\gamma}}} \tag{B.1}$$

In the following sections, the algorithms are discussed in elliptic curve discrete logarithm notation instead of their classical interpretations.

B.2.2 Shanks' baby-step/giant-step attack

Let $k = \lceil\sqrt{n}\rceil$. The logarithm m can be written uniquely in the form $m = kj + i$ where $0 \le m < n$ and $0 \le i, j < k$. Then, $Q - [i]P = [kj]P$. The left side of this equation is called the *baby-steps* and the right side is called the *giant-steps*. This scheme provides a space-time trade off. This idea belongs to Shanks according to Odlyzko [Odl85]. Excluding the precomputation overheads, the algorithm needs $O(\sqrt{n})$ time and $O(\sqrt{n})$ space. The space requirement of this algorithm makes it impractical provided that n is sufficiently large, say 2^{160}.

B.2.3 Random walks and Pollard's rho attack

Algorithms based on random walks remedy the space requirement of Shank's algorithm preserving the worst-case running time. It should be noted that these algorithms are probabilistic and have a (*practically negligible*) probability of failure. Pollard [Pol78] introduced algorithms achieving these criteria. Pollard's rho method depends on finding two integer pairs (a, b) and (c, d) such that $[a]P + [b]Q = [c]P + [d]Q$ so that the desired logarithm corresponds to $l = (a - c)(d - b)^{-1} \pmod n$. To satisfy the asymptotic bound requirements, the variables a, b, c, d must be selected at random. This is computationally modeled with iterating functions. If P_0 is the base point and f is the iterating function, then the sequence is defined by

[5] Big-Oh notation is convenient to be used for brevity. Other notations can be followed to make more precise time and space estimations.

[6] Notice that $n^{c\sqrt{\log\log n/\log n}} = e^{c\sqrt{\log n \log\log n}}$.

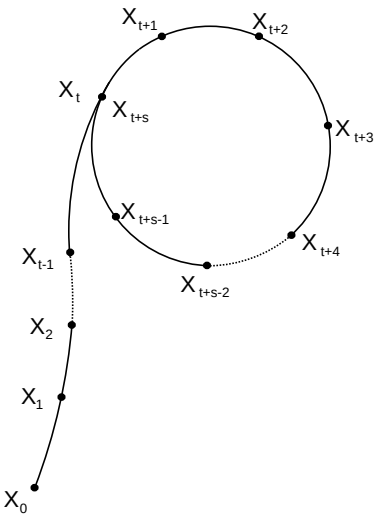

Figure B.1: Pollard's rho illustration: a rho-like shape where $t \approx \sqrt{\pi n/8}$ is the tail length and $s \approx \sqrt{\pi n/8}$ is the cycle length.

$P_i = f(P_{i-1})$ with $0 < i$. The points on the sequence collide after approximately $\sqrt{\pi n/2}$ iterations. This situation is depicted in Figure B.1.

Flajolet and Odlyzko [FO90] showed how the characteristics of random functions, their expected tail length and the expected cycle length of sequences can be computed for random functions.

Pollard's iterating function defines a partition with three cosets. Depending on the experimental data, Teske [Tes98], [Tes01] pointed out that Pollard's iterating function does not satisfy optimal random characteristics. Teske introduced a new iterating function using multi-cosets which are close models of a random function.

Brent [Bre80] replaced Floyd's cycle in Pollard's rho algorithm with a new strategy which provides a constant speed-up of 3/2. An asymptotically faster method was proposed by Sedgewick and Szymanski [SS79] but this method requires impractical amounts of memory.

Van Oorschot and Wiener [vOW99] developed the parallelized version of Pollard's rho algorithm that achieves M times speed-up over M processors. The use of automorphisms was discovered by Gallant *et al.* [GLV00] and Wiener and Zuccherato [WZ99]. The use of automorphisms was generalized to hyperelliptic curves by Duursma *et al.* [DGM99]. The notion of distinguished points was suggested by Silverman as a contribution to the ANSI X9F1 working group in 1997 (as reported in [HMV03]).

B.2.4 Pohlig-Hellman attack

Based on Pohlig and Hellman's [PH78] approach, it can be to shown that the ECDLP is solved more easily when the logarithms are computed in the subgroups of $\langle P \rangle$. Let $p_1, p_2, \ldots, p_i$ the prime divisors of n, the order of P. The algorithm is based on solving the ECDLP in smaller subgroups of order p_j by a square root attack (i.e. Pollard's rho method), then using this result to solve the same logarithm in the target group $\langle P \rangle$. In the final step, the solutions with respect

to different prime power order subgroups are merged by the Chinese remainder theorem. The algorithm is very efficient when the group order is smooth. Therefore, the group order must contain at least one large prime factor to avoid this attack. If $t = \max(p_0, p_1, \ldots, p_k)$, then the algorithm takes $O(\sqrt{p})$ group operations to halt. Thus, one should not use a composite integer which is a product of *small* primes for $\#E(\mathbb{F}_q)$ to prevent the Pohlig-Hellman attack. $\#E(\mathbb{F}_q)$ can be obtained by a deterministic polynomial time algorithm of Schoof [Sch85], [Sch95]. (see also [LL03] and [Dew98]).

B.2.5 Isomorphism attacks

Before Menezes *et al.* [MVO91] introduced the MOV attack, ECDLP was considered an intractable problem in all cases. Thus, it was believed that any attack on any specific instance of the problem was fully exponential. The MOV attack is based on the Weil pairing [Wei29]. If $\gcd(n, q) = 1$ where n is the the prime order of $P \in E(\mathbb{F}_q)$ and k is the multiplicative order of q modulo n, then $E(\mathbb{F}^*_{q^k})$ has a unique subgroup G of order n. The Weil pairing constructs an isomorphism from $\langle P \rangle$ to G. This pairing is defined when $n \nmid (q-1)$. Using this idea, Menezes *et al.* [MVO91] were able to reduce the elliptic curve discrete logarithm problem on a curve E over a finite field $\mathbb{F}_q$ to the discrete logarithm problem on a suitable extension field $\mathbb{F}_{q^k}$ of $\mathbb{F}_q$. The integer k is called the embedding degree. Their attack applies to supersingular curves which are now avoided in ECC standards. Frey and Rück [FR94] described a generalized method of this approach based on the Weil descent.

The independent results of Satoh and Araki [SA98], Semaev [Sem98], and Smart [Sma99] showed that anomalous curves are vulnerable to a kind of isomorphism attack where it is possible to solve ECDLP in linear time. Rück [Rüc99] extended Semaev's approach to the Jacobians of arbitrary curves.

The Tate pairing attacks construct an isomorphism between $\langle P \rangle$ and G without the restriction of $n \nmid (q-1)$ in the Weil pairing. Balasubramanian and Koblitz [BK98] proved that a large proportion of elliptic curves of prime order defined over prime fields have a large embedding degree. Thus, they are immune to pairing attacks.

Galbraith [Gal01] and Rubin and Silverberg [RS02] showed the upper bounds for k for supersingular abelian varieties of dimension g over finite fields. Frey [Fre01] explained why Jacobian varieties of hyperelliptic curves of genus ≤ 4 are candidates for cryptographically good abelian varieties. Silverman [Sil88], [Sil90], [Sil97] showed efficient computation algorithms for point heights.

Frey pointed to the use of the Weil descent for attacking elliptic curve cryptosystems in his public talk in the ECC'98 conference. Galbraith and Smart [GS99] provided a concrete explanation of the Weil descent methodology. Gaudry *et al.* [GHS02b] proposed the GHS attack which utilizes the Weil descent attack and makes some specific family of curves, widely in use, inappropriate for cryptographic use. A detailed analysis of the GHS attack is given in [MQ01]. The GHS attack was further extended by Hess [Hes03], [Hes04]. Menezes and Teske [MT06] analyzed characteristic two finite fields $\mathbb{F}_{2^n}$ for weakness under the generalized GHS attack of Hess. It was claimed that the fields $\mathbb{F}_{q^7}$ are potentially partially weak and the other fields $\mathbb{F}_{2^N}$ where $3, 5, 6, 7$ or $8 \nmid N$ are not weak under Hess' generalized GHS attack.

Maurer *et al.* [MMT01] analyzed the GHS attack with possible isomorphisms for elliptic curves over $\mathbb{F}_{2^k}$ having composite k, $160 \leq k \leq 600$ and determined the curves vulnerable to the GHS attack. Menezes *et al.* [MTW04] showed that the fields $\mathbb{F}_{2^{5k}}$, $37 \leq k \leq 120$ are vulnerable to the GHS attack. Galbraith *et al.* [GHS02a] extended Weil descent over isogenous curves. Arita [Ari00] and Arita *et al.* [AMNS06] provided evidence that there are cases when curves over finite fields $\mathbb{F}_{3^m}$ may also be vulnerable to the Weil descent attack. Diem [Die01], [Die03] has shown that the GHS attack can be extended to elliptic curves over $\mathbb{F}_{p^m}$ for $p = 3, 5, 7$.

B.2.6 Index calculus method

Index calculus is a subexponential technique for solving discrete logarithms in some groups. The method is attributed to Adleman in a detailed review by Schirokauer *et al.* [SWD96].

Miller [Mil86] showed that elliptic curve cryptosystems are immune to index calculus attacks. In this sense, the same level of security is believed to be provided with smaller key sizes. This property of Elliptic Curve Cryptography makes it one of the most promising asymmetric solutions among numerous alternatives. Miller's comments were supported by Silverman and Suzuki [SS98] and by Huang *et al.* [HKT00]. Enge and Gaudry [EG02] and Gaudry [Gau00] proposed subexponential-time index calculus variants for groups of known order in which a smoothness concept is available.

Silverman [Sil00] introduced the xedni calculus attack. Jacobson *et al.* [JKS$^+$00] proved that xedni calculus is ineffective asymptotically and also provided convincing experimental evidence that it is extremely inefficient for primes p of the sizes used in cryptography.

Gaudry [Gau04] proposed an index calculus algorithm which does not make use of any embedding into the Jacobian of a well-suited curve. The algorithm applies to general abelian varieties. Gaudry investigated the algorithm on the Weil restriction of elliptic curves and hyperelliptic curves over small degree extension fields. Gaudry showed that the attack can solve all ECDLPs defined over $\mathbb{F}_{q^3}$ in time $O(q^{10/7})$, with a small constant and an ECDLP over $\mathbb{F}_{q^4}$ or a genus 2 problem over $\mathbb{F}_{p^2}$ in $O(q^{14/9})$ time with a larger constant.

B.2.7 Fixed versus random curves

The task of searching for secure curves consists of several inspections which are related to known attacks. (see Blake [BSS99, VI.5]) Generating random curves for cryptographic use is assumed to be superior to using fixed curves in the context of security. However, it takes a computational effort bounded by $O(\log^8 p)$, where p is the characteristic of the underlying finite field, to generate a cryptographically secure random curve. Therefore, some standards (i.e. ANSI X9.63, FIPS 186-2) enforce the use of fixed curves[7] which are said to be immune to known attacks. Fixed curves have many users with each user associated to a unique ECDLP. Given that a fixed curve will be in the public domain for extensive periods of time and consequently a target for attackers, and the fact that there will be multiple users, a natural question to ask is: "How hard is it to solve subsequent ECDLPs after the first one is solved?". This is known as Multi-ECDLP. It is proved by Kuhn and Struik [KS01] that the best strategy for solving k

[7]These curves are defined over specific fields in which arithmetic is performed faster than arbitrary fields. NIST also recommends fixed parameters in selecting random curves.

ECDLPs is to devote all efforts to solving a randomly selected instance. It is claimed by Li *et al.* [LLX05] that the knowledge gained through computing many logarithms does not make it easier to find any other logarithm. This lead to the conclusion that it is safe for many users to share the same curve having different private keys. A security comparison of fixed versus random elliptic curves over $\mathbb{F}_p$ is provided by Hitchcock *et al.* [HMCD04]. The security analysis is performed under the assumption that Pollard's rho method is the fastest method for solving ECDLP in accordance with real life scenarios. The conclusion of [HMCD04] is that adding four bits to the order of a fixed curve avoids general software attacks and an additional six bits avoids attacks on curves with special properties, giving security similar to that of a randomly generated curve.

B.3 Cryptographic pairings

Bilinear pairings have found many applications in cryptography, such as the identity-based encryption scheme of Boneh and Franklin [BF03], the one-round tripartite key agreement scheme of Joux [Jou04] and the short signature scheme of Boneh *et al.*, see [BLS04c]. Galbraith gives a comprehensive survey on cryptographic pairings, see [Gal05].

Let $\mathbb{F}_q$ be a finite field with $q = p^n$ elements where $p \geq 5$ is prime and let E be an elliptic curve defined over $\mathbb{F}_q$. Let $\mathcal{O}$ denote the identity on E. Let r be a large prime that is coprime to q such that $r|\#E(\mathbb{F}_q)$ and let k be the embedding degree of E with respect to r. For practical purposes only $k > 1$ is considered. The base field is denoted by $\mathbb{F}_q$ and the extension field is denoted by $\mathbb{F}_{q^k}$. Let $f_{i,P} \in \mathbb{F}_q(E)$ be a function with divisor $\mathrm{div}(f_{i,P}) = i(P) - ([i]P) - (i-1)(\mathcal{O})$.

The Tate pairing Choose a point $P \in E(\mathbb{F}_q)[r]$. This implies $\mathrm{div}(f_{r,P}) = r(P) - r(\mathcal{O})$. Let $Q \in E(\mathbb{F}_{q^k})/rE(\mathbb{F}_{q^k})$ and let μ_r denote the group of r-th roots of unity in $\mathbb{F}_{q^k}^*$. The reduced Tate pairing e_r [BKLS02] is defined as

$$e_r \colon (P, Q) \mapsto f_{r,P}(Q)^{(q^k-1)/r} \in \mu_r.$$

Miller's algorithm [Mil04] computes the paired value iteratively by taking advantage of the fact that $f_{i+j,P}$ can be written as $f_{i+j,P} = f_i \cdot f_j \cdot l/v$, where l and v are the lines used in the computation of $[i]P + [j]P = [i+j]P$. That is, l is the line that intersects E at $[i]P$, $[j]P$ and $-[i+j]P$, and v is the vertical line that intersects E at both $[i+j]P$ and $-[i+j]P$. This enables the computation of the function $f_{2i,P}$ from $f_{i,P}$ directly by evaluating the lines that are used in point doubling of P. Similarly, function $f_{i+1,P}$ can be computed from $f_{i,P}$ so that $f_{r,P}$ can be computed in $\log_2 r$ steps, as summarised in Algorithm B.3.1.

There are many other optimizations which speed up the computation of the Miller loop (i.e. the loop in Algorithm B.3.1) in certain settings, including the denominator elimination technique [BKLS02], the use of efficiently computable endomorphisms [Sco05], [GS08], and loop shortening techniques [BGHS04], [HSV06], [BGHS07], [MKHO07], [ZZH07], [LLP08], [Ver08].

Algorithm B.3.1: Miller's algorithm

input : $P \in E(\mathbb{F}_{q^k})[r]$, $Q \in E(\mathbb{F}_{q^k})$, $r = (r_{m-1} \dots r_1 r_0)_2$ with $r_{m-1} = 1$.

output: $f_{r,P}(Q) \leftarrow f_{\text{var}}$.

$R \leftarrow P$, $f_{\text{var}} \leftarrow 1$.
for $i = m - 2$ **to** 0 **do**
    Compute lines l_{dbl} and v_{dbl} for doubling R.
    $R \leftarrow 2R$.
    $f_{\text{var}} \leftarrow f_{\text{var}}^2 \cdot l_{\text{dbl}}(Q)/v_{\text{dbl}}(Q)$.
    if $r_i = 1$ **then**
        Compute lines l_{add} and v_{add} for adding R and P.
        $R \leftarrow R + P$.
        $f_{\text{var}} \leftarrow f_{\text{var}} \cdot l_{\text{add}}(Q)/v_{\text{add}}(Q)$.
    end
end
return f_{var}.

Appendix C

Computer algebra scripts

This appendix verifies all formulae used in this thesis. Each section contains Maple scripts which check the correctness of birational maps and affine/projective formulae, algorithms, and register allocations.

C.1 Short Weierstrass form

```
> C:=(x,y)->(y^2-(x^3+a*x+b)):
> a1:=0: a3:=0: a2:=0: a4:=a: a6:=b:
> W:=(u,v)->(v^2+a1*u*v+a3*v-(u^3+a2*u^2+a4*u+a6)):
> CtoW:=(x,y)->(x,y):
> WtoC:=(u,v)->(u,v):
> simplify([W(CtoW(x1,y1))],[C(x1,y1)]); #Check CtoW.
> simplify([C(WtoC(u1,v1))],[W(u1,v1)]); #Check WtoC.
> simplify([(x1,y1)-WtoC(CtoW(x1,y1))],[C(x1,y1)]); #Check CtoW(WtoC).
> simplify([(u1,v1)-CtoW(WtoC(u1,v1))],[W(u1,v1)]); #Check WtoC(CtoW).
> ut,vt:=CtoW(x1,y1): simplify([(x1,-y1)-WtoC(ut,-vt-a1*ut-a3)],[C(x1,y1)]); #Check the negation.

> # Doubling formulae.
> unassign('x1','y1'): u1,v1:=CtoW(x1,y1):
> L:=(3*u1^2+2*a2*u1+a4-a1*v1)/(2*v1+a1*u1+a3): u3:=L^2+a1*L-a2-2*u1: v3:=L*(u1-u3)-v1-a1*u3-a3:
> simplify([W(u3,v3)],[C(x1,y1)]); x3std,y3std:=WtoC(u3,v3):
>
> x3:=x1*(mu-mu^2)+a*sigma: simplify([x3std-x3],[C(x1,y1),mu=(y1+3*c)/(2*y1),sigma=(a-3*x1^2)/(2*y1)^2,delta=
    ((3*x1*(y1-3*c)*(y1+3*c)-a*(9*x1^2+a))/(2*y1)^3),c^2-b]);
> x3:=x1/4-(9*b*x1)/(4*y1^2)-a*(3*x1^2-a)/(4*y1^2): simplify([x3std-x3],[C(x1,y1)]);
> x3:=(3*x1^2+a)^2/(2*y1)^2-2*x1: simplify([x3std-x3],[C(x1,y1)]);
> y3:=(y1-c)*mu^3+a*delta-c: simplify([y3std-y3],[C(x1,y1),mu=(y1+3*c)/(2*y1),sigma=(a-3*x1^2)/(2*y1)^2,delta
    =((3*x1*(y1-3*c)*(y1+3*c)-a*(9*x1^2+a))/(2*y1)^3),c^2-b]);
> y3:=y1/2-(3*b+2*a*x1)/(2*y1)-x3*(3*x1^2+a)/(2*y1): simplify([y3std-y3],[C(x1,y1)]);
> y3:=(3*x1^2+a)/(2*y1)*(x1-x3)-y1: simplify([y3std-y3],[C(x1,y1)]);

> # Addition formulae.
> unassign('x1','y1','x2','y2'): u1,v1:=CtoW(x1,y1): u2,v2:=CtoW(x2,y2):
> L:=(v2-v1)/(u2-u1): u3:=L^2+a1*L-a2-u1-u2: v3:=L*(u1-u3)-v1-a1*u3-a3:
> simplify([W(u3,v3)],[C(x1,y1),C(x2,y2)]); x3std,y3std:=WtoC(u3,v3):
>
> x3:=(x1^2+x1*x2+x2^2+a)^2/(y1+y2)^2-x1-x2: simplify([x3std-x3],[C(x1,y1),C(x2,y2)]);
> x3:=(y1-y2)^2/(x1-x2)^2-x1-x2: simplify([x3std-x3],[C(x1,y1),C(x2,y2)]);
> x3:=(x1^2+x1*x2+x2^2+a)^2/(y1+y2)^2-x1-x2: simplify([x3std-x3],[C(x1,y1),C(x2,y2)]);
```

```
> x3:=((x1+x2)*(x1*x2+a)-2*(y1*y2-b))/(x1-x2)^2: simplify([x3std-x3],[C(x1,y1),C(x2,y2)]);
> y3:=1/2*((x1^2+x1*x2+x2^2+a)/(y1+y2)*(x1+x2-2*x3)-y1-y2): simplify([y3std-y3],[C(x1,y1),C(x2,y2)]);
> y3:=(y1-y2)/(x1-x2)*(x1-x3)-y1: simplify([y3std-y3],[C(x1,y1),C(x2,y2)]);
> y3:=1/2*((x1^2+x1*x2+x2^2+a)/(y1+y2)*(x1+x2-2*x3)-y1-y2): simplify([y3std-y3],[C(x1,y1),C(x2,y2)]);
> y3:=(y1-y2)/(x1-x2)*(x1-x3)-y1: simplify([y3std-y3],[C(x1,y1),C(x2,y2)]);
```

DBL_S_a00.

```
> x1:=X1/Z1: y1:=Y1/Z1:
> x3:=x1/4-(9*b*x1)/(4*y1^2)-a*(3*x1^2-a)/(4*y1^2):
> y3:=y1/2-(3*b+2*a*x1)/(2*y1)-x3*(3*x1^2+a)/(2*y1):
>
> DBL_S_a00:=proc(X1,Y1,Z1) local X3,Y3,Z3:
>     X3:=2*Y1*Z1*((3*X1^2+a*Z1^2)^2-8*Y1^2*X1*Z1):
>     Y3:=(3*X1^2+a*Z1^2)*(12*Y1^2*X1*Z1-(3*X1^2+a*Z1^2)^2)-8*Y1^4*Z1^2:
>     Z3:=8*Y1^3*Z1^3:
>     return X3,Y3,Z3:
> end proc:
> X3,Y3,Z3:=DBL_S_a00(X1,Y1,Z1):
> simplify([x3-X3/Z3,y3-Y3/Z3],[(C(x1,y1))]); #Check.
>
```

DBL_S_a01, $5\mathbf{M} + 6\mathbf{S} + 1\mathbf{D} + 11\mathbf{a}$.

```
> DBL_S_a01_opr:=proc(X1,Y1,Z1) local X3,Y3,Z3,A,B,C,D,E,F,G,H:
>     A:=2*Y1*Z1: B:=Z1^2: C:=X1^2: D:=3*C+a*B: E:=Y1*A: F:=E^2: G:=(X1+E)^2-C-F: H:=D^2-2*G: X3:=H*A: Y3:=D*
    (G-H)-2*F: Z3:=A*A^2:
>     return X3,Y3,Z3:
> end proc:
> X3,Y3,Z3:=DBL_S_a01_opr(X1,Y1,Z1):
> simplify([x3-X3/Z3,y3-Y3/Z3],[(C(x1,y1))]); #Check.
>
```

DBL_S_a02, $7\mathbf{M} + 3\mathbf{S} + 10\mathbf{a}$, assumes $a = -3$.

```
> DBL_S_a02_opr:=proc(X1,Y1,Z1) local X3,Y3,Z3,A,B,C,D,E,F,G:
>     A:=3*(X1-Z1)*(X1+Z1): B:=2*Y1*Z1: C:=B^2: D:=Y1*B: E:=D^2: F:=2*X1*D: G:=A^2-2*F: X3:=B*G: Y3:=A*(F-G)-
    2*E: Z3:=B*C:
>     return X3,Y3,Z3:
> end proc:
> X3,Y3,Z3:=DBL_S_a02_opr(X1,Y1,Z1):
> simplify([x3-X3/Z3,y3-Y3/Z3],[(C(x1,y1)),a+3]); #Check.
>
```

ADD_S_a00.

```
> x1:=X1/Z1: y1:=Y1/Z1: x2:=X2/Z2: y2:=Y2/Z2:
> x3:=(x1^2+x1*x2+x2^2+a)^2/(y1+y2)^2-x1-x2:
> y3:=1/2*((x1^2+x1*x2+x2^2+a)/(y1+y2)*(x1+x2-2*x3)-y1-y2):
>
> ADD_S_a00:=proc(X1,Y1,Z1,X2,Y2,Z2) local X3,Y3,Z3:
>     X3:=(X1*Z2-Z1*X2)*(Z1*Z2*(Y1*Z2-Z1*Y2)^2-(X1*Z2+Z1*X2)*(X1*Z2-Z1*X2)^2):
>     Y3:=(Y1*Z2-Z1*Y2)*((2*X1*Z2+Z1*X2)*(X1*Z2-Z1*X2)^2-Z1*Z2*(Y1*Z2-Z1*Y2)^2)-Y1*Z2*(X1*Z2-Z1*X2)^3:
>     Z3:=Z1*Z2*(X1*Z2-Z1*X2)^3:
>     return X3,Y3,Z3:
> end proc:
> X3,Y3,Z3:=ADD_S_a00(X1,Y1,Z1,X2,Y2,Z2):
> simplify([x3-X3/Z3,y3-Y3/Z3],[(C(x1,y1)),(C(x2,y2))]); #Check.
>
```

ADD_S_a01, $12\mathbf{M} + 2\mathbf{S} + 7\mathbf{a}$.

```
> ADD_S_a01_opr:=proc(X1,Y1,Z1,X2,Y2,Z2) local X3,Y3,Z3,A,B,C,D,E,F,G,H,J:
>     A:=Z1*Z2: B:=X1*Z2: C:=Y1*Z2: D:=B-Z1*X2: E:=C-Z1*Y2: F:=D^2: G:=D*F: H:=F*B: J:=E^2*A+G-2*H: X3:=D*J:
    Y3:=E*(H-J)-G*C: Z3:=A*G:
>     return X3,Y3,Z3:
> end proc:
> X3,Y3,Z3:=ADD_S_a01_opr(X1,Y1,Z1,X2,Y2,Z2):
> simplify([x3-X3/Z3,y3-Y3/Z3],[(C(x1,y1)),(C(x2,y2))]); #Check.
>
```

UADD_S_a00.

```
> x1:=X1/Z1: y1:=Y1/Z1: x2:=X2/Z2: y2:=Y2/Z2:
> x3:=(x1^2+x1*x2+x2^2+a)^2/(y1+y2)^2-x1-x2:
> y3:=1/2*((x1^2+x1*x2+x2^2+a)/(y1+y2)*(x1+x2-2*x3)-y1-y2):
>
> UADD_S_a00:=proc(X1,Y1,Z1,X2,Y2,Z2) local X3,Y3,Z3:
>     X3:=Z1*Z2*(Y1*Z2+Z1*Y2)*(2*((X1*Z2+Z1*X2)^2-Z1*Z2*(X1*X2-a*Z1*Z2))^2-2*Z1*Z2*(X1*Z2+Z1*X2)*(Y1*Z2+Z1*Y2
    )^2):
>     Y3:=((X1*Z2+Z1*X2)^2-Z1*Z2*(X1*X2-a*Z1*Z2))*(3*Z1*Z2*(X1*Z2+Z1*X2)*(Y1*Z2+Z1*Y2)^2-2*((X1*Z2+Z1*X2)^2-Z
    1*Z2*(X1*X2-a*Z1*Z2))^2)-Z1^2*Z2^2*(Y1*Z2+Z1*Y2)^4:
>     Z3:=2*Z1^3*Z2^3*(Y1*Z2+Z1*Y2)^3:
>     return X3,Y3,Z3:
> end proc:
> X3,Y3,Z3:=UADD_S_a00(X1,Y1,Z1,X2,Y2,Z2):
> simplify([x3-X3/Z3,y3-Y3/Z3],[(C(x1,y1)),(C(x2,y2))]); #Check.
>
```

UADD_S_a01, $11\mathbf{M} + 5\mathbf{S} + 1\mathbf{D} + 16\mathbf{a}$.

```
> UADD_S_a01_opr:=proc(X1,Y1,Z1,X2,Y2,Z2) local X3,Y3,Z3,A,B,C,D,E,F,G,H,J,K,L:
>     A:=X1*X2: B:=Z1*Z2: C:=(X1+Z1)*(X2+Z2)-A-B: D:=Y1*Z2+Z1*Y2: E:=B*D: F:=E*D: G:=C^2: H:=F^2: J:=G-B*(A-a
    *B): K:=((C+F)^2-G-H)/2: L:=2*(J^2-K): X3:=E*L: Y3:=J*(K-L)-H: Z3:=2*E*E^2:
>     return X3,Y3,Z3:
> end proc:
> X3,Y3,Z3:=UADD_S_a01_opr(X1,Y1,Z1,X2,Y2,Z2):
> simplify([x3-X3/Z3,y3-Y3/Z3],[(C(x1,y1)),(C(x2,y2))]); #Check.
>
```

DBL_Sw_a00.

```
> x1:=X1/Z1^2: y1:=Y1/Z1^3:
> x3:=x1/4-(9*b*x1)/(4*y1^2)-a*(3*x1^2-a)/(4*y1^2):
> y3:=y1/2-(3*b+2*a*x1)/(2*y1)-x3*(3*x1^2+a)/(2*y1):
>
> DBL_Sw_a00:=proc(X1,Y1,Z1) local X3,Y3,Z3:
>     X3:=(3*X1^2+a*Z1^4)^2-8*X1*Y1^2:
>     Y3:=(3*X1^2+a*Z1^4)*(4*X1*Y1^2-X3)-8*Y1^4:
>     Z3:=2*Y1*Z1:
>     return X3,Y3,Z3:
> end proc:
> X3,Y3,Z3:=DBL_Sw_a00(X1,Y1,Z1):
> simplify([x3-X3/Z3^2,y3-Y3/Z3^3],[(C(x1,y1))]); #Check.
>
```

DBL_Sw_a01, $1\mathbf{M} + 8\mathbf{S} + 1\mathbf{D} + 14\mathbf{a}$.

```
> DBL_Sw_a01_opr:=proc(X1,Y1,Z1) local X3,Y3,Z3,A,B,C,D,E,F:
>     A:=Z1^2: B:=Y1^2: C:=X1^2: D:=B^2: E:=2*((X1+B)^2-C-D): F:=3*C+a*A^2: X3:=F^2-2*E: Y3:=F*(E-X3)-8*D: Z3
    :=(Y1+Z1)^2-B-A:
>     return X3,Y3,Z3:
> end proc:
> X3,Y3,Z3:=DBL_Sw_a01_opr(X1,Y1,Z1):
> simplify([x3-X3/Z3^2,y3-Y3/Z3^3],[(C(x1,y1))]); #Check.
>
```

DBL_Sw_a02, $3\mathbf{M} + 5\mathbf{S} + 12\mathbf{a}$, assumes $a = -3$.

```
> DBL_Sw_a02_opr:=proc(X1,Y1,Z1) local X3,Y3,Z3,A,B,C,D:
>     A:=Z1^2: B:=Y1^2: C:=X1*B: D:= 3*(X1-A)*(X1+A): X3:=D^2-8*C: Y3:=D*(4*C-X3)-8*B^2: Z3:=(Y1+Z1)^2-B-A:
>     return X3,Y3,Z3:
> end proc:
> X3,Y3,Z3:=DBL_Sw_a02_opr(X1,Y1,Z1):
> simplify([x3-X3/Z3^2,y3-Y3/Z3^3],[(C(x1,y1)),a+3]); #Check.
>
```

ADD_Sw_a00.

```
> x1:=X1/Z1^2: y1:=Y1/Z1^3: x2:=X2/Z2^2: y2:=Y2/Z2^3:
> x3:=(x1^2+x1*x2+x2^2+a)^2/(y1+y2)^2-x1-x2:
> y3:=1/2*((x1^2+x1*x2+x2^2+a)/(y1+y2)*(x1+x2-2*x3)-y1-y2):
>
> ADD_Sw_a00:=proc(X1,Y1,Z1,X2,Y2,Z2) local X3,Y3,Z3:
>     X3:=(Y1*Z2^3-Z1^3*Y2)^2-(X1*Z2^2+Z1^2*X2)*(X1*Z2^2-Z1^2*X2)^2:
>     Y3:=(Y1*Z2^3-Z1^3*Y2)*(Z1^2*X2*(X1*Z2^2-Z1^2*X2)^2-X3)-Z1^3*Y2*(X1*Z2^2-Z1^2*X2)^3:
>     Z3:=Z1*Z2*(X1*Z2^2-Z1^2*X2):
>     return X3,Y3,Z3:
> end proc:
> X3,Y3,Z3:=ADD_Sw_a00(X1,Y1,Z1,X2,Y2,Z2):
> simplify([x3-X3/Z3^2,y3-Y3/Z3^3],[(C(x1,y1)),(C(x2,y2))]); #Check.
>
```

ADD_Sw_a01, $11\mathbf{M} + 5\mathbf{S} + 11\mathbf{a}$.

```
> ADD_Sw_a01_opr:=proc(X1,Y1,Z1,X2,Y2,Z2) local X3,Y3,Z3,U1,V1,U2,V2,A,B,C,D,E,F,G:
>     U2:=Z2^2: V2:=U2*Z2: U1:=Z1^2: V1:=U1*Z1: A:=U1*X2: B:=V1*Y2: C:=X1*U2-A: D:=Y1*V2-B: E:=C^2: F:=C*E: G
    :=A*E: X3:=D^2-F-2*G: Y3:=D*(G-X3)-B*F: Z3:=C*((Z1+Z2)^2-U1-U2)/2:
>     return X3,Y3,Z3:
> end proc:
> X3,Y3,Z3:=ADD_Sw_a01_opr(X1,Y1,Z1,X2,Y2,Z2):
> simplify([x3-X3/Z3^2,y3-Y3/Z3^3],[(C(x1,y1)),(C(x2,y2))]); #Check.
>
```

UADD_Sw_a00.

```
> x1:=X1/Z1^2: y1:=Y1/Z1^3: x2:=X2/Z2^2: y2:=Y2/Z2^3:
> x3:=(x1^2+x1*x2+x2^2+a)^2/(y1+y2)^2-x1-x2:
> y3:=1/2*((x1^2+x1*x2+x2^2+a)/(y1+y2)*(x1+x2-2*x3)-y1-y2):
>
> UADD_Sw_a00:=proc(X1,Y1,Z1,X2,Y2,Z2) local X3,Y3,Z3:
>     X3:=((X1*Z2^2+Z1^2*X2)^2-Z1^2*Z2^2*(X1*X2-a*Z1^2*Z2^2))^2-(X1*Z2^2+Z1^2*X2)*(Y1*Z2^3+Z1^3*Y2)^2:
>     Y3:=(((X1*Z2^2+Z1^2*X2)^2-Z1^2*Z2^2*(X1*X2-a*Z1^2*Z2^2))*((X1*Z2^2+Z1^2*X2)*(Y1*Z2^3+Z1^3*Y2)^2-2*X3)-(
    Y1*Z2^3+Z1^3*Y2)^4)/2:
>     Z3:=Z1*Z2*(Y1*Z2^3+Z1^3*Y2):
>     return X3,Y3,Z3:
> end proc:
> X3,Y3,Z3:=UADD_Sw_a00(X1,Y1,Z1,X2,Y2,Z2):
> simplify([x3-X3/Z3^2,y3-Y3/Z3^3],[(C(x1,y1)),(C(x2,y2))]); #Check.
>
```

UADD_Sw_a01, $8\mathbf{M} + 10\mathbf{S} + 1\mathbf{D} + 24\mathbf{a}$.

```
> UADD_Sw_a01_opr:=proc(X1,Y1,Z1,X2,Y2,Z2) local X3,Y3,Z3,U1,V1,U2,V2,A,B,C,D,E,F,G,H,J,K:
>     U2:=Z2^2: V2:=U2*Z2: U1:=Z1^2: V1:=U1*Z1: A:=X1*X2: B:=((Z1+Z2)^2-U1-U2)/2: C:=B^2: D:=(X1+U1)*(X2+U2)-
    A-C: E:=Y1*V2+V1*Y2: F:=E^2: G:=D^2: H:=F^2: J:=G-C*(A-a*C): K:=((F+D)^2-G-H)/2: X3:=J^2-K: Y3:=(J*(K-2*X3)-H)
    /2: Z3:=((B+E)^2-C-F)/2:
>     return X3,Y3,Z3:
> end proc:
> X3,Y3,Z3:=UADD_Sw_a01_opr(X1,Y1,Z1,X2,Y2,Z2):
> simplify([x3-X3/Z3^2,y3-Y3/Z3^3],[(C(x1,y1)),(C(x2,y2))]); #Check.
>
```

C.2 Extended Jacobi quartic form

```
> C:=(x,y)->(y^2-(d*x^4+2*a*x^2+1)):
> a1:=0: a3:=0: a6:=0: a2:=-4*a: a4:=4*(a^2-d):
> W:=(u,v)->(v^2+a1*u*v+a3*v-(u^3+a2*u^2+a4*u+a6)):
> CtoW:=(x,y)->((2*y+2)/x^2+2*a,(4*y+4)/x^3+4*a/x):
> WtoC:=(u,v)->(2*u/v,(2*u-4*a)*u^2/v^2-1):
> simplify([W(CtoW(x1,y1))],[C(x1,y1)]); #Check CtoW.
> simplify([C(WtoC(u1,v1))],[W(u1,v1)]); #Check WtoC.
> simplify([(x1,y1)-WtoC(CtoW(x1,y1))],[C(x1,y1)]); #Check CtoW(WtoC).
> simplify([(u1,v1)-CtoW(WtoC(u1,v1))],[W(u1,v1)]); #Check WtoC(CtoW).
> ut,vt:=CtoW(x1,y1): simplify([(-x1,y1)-WtoC(ut,-vt-a1*ut-a3)],[C(x1,y1)]); #Check the negation.

> # Doubling formulae.
> unassign('x1','y1'): u1,v1:=CtoW(x1,y1):
> L:=(3*u1^2+2*a2*u1+a4-a1*v1)/(2*v1+a1*u1+a3): u3:=L^2+a1*L-a2-2*u1: v3:=L*(u1-u3)-v1-a1*u3-a3:
> simplify([W(u3,v3)],[C(x1,y1)]); x3std,y3std:=WtoC(u3,v3):
>
> x3:=(2*y1/(2-y1^2+2*a*x1^2))*x1: simplify([x3std-x3],[C(x1,y1)]);
> x3:=2*x1*y1/(1-d*x1^4): simplify([x3std-x3],[C(x1,y1)]);
> y3:=(2*y1/(2-y1^2+2*a*x1^2))*((2*y1/(2-y1^2+2*a*x1^2))-y1)-1: simplify([y3std-y3],[C(x1,y1)]);
> y3:=2*y1^4/(1-d*x1^4)^2-a*x3^2-1: simplify([y3std-y3],[C(x1,y1)]);
> y3:=2*y1^2*(d*x1^4+2*x1^2+1)/(1-d*x1^4)^2-x3^2-1: simplify([y3std-y3],[C(x1,y1)]);

> # Addition formulae.
> unassign('x1','y1','x2','y2'): u1,v1:=CtoW(x1,y1): u2,v2:=CtoW(x2,y2):
> L:=(v2-v1)/(u2-u1): u3:=L^2+a1*L-a2-u1-u2: v3:=L*(u1-u3)-v1-a1*u3-a3:
> simplify([W(u3,v3)],[C(x1,y1),C(x2,y2)]); x3std,y3std:=WtoC(u3,v3):
>
> x3:=(x1^2-x2^2)/(x1*y2-y1*x2): simplify([x3std-x3],[C(x1,y1),C(x2,y2)]);
> x3:=(x1*y2+y1*x2)/(1-d*x1^2*x2^2): simplify([x3std-x3],[C(x1,y1),C(x2,y2)]);
> y3:=(y1*y2+2*a*x1*x2+s*x1^2+s*x2^2)/(1-s*x1*x2)^2-s*x3^2: simplify([y3std-y3],[C(x1,y1),C(x2,y2),d-s^2]);
> y3:=((y1*y2+2*a*x1*x2)*(1+d*x1^2*x2^2)+2*d*x1*x2*(x1^2+x2^2))/(1-d*x1^2*x2^2)^2: simplify([y3std-y3],[C(x1,
    y1),C(x2,y2)]);
> y3:=((x1^2-x2^2)^2-(x1*y2-y1*x2)*(x1^3*y2-y1*x2^3))/(x1*x2*(x1*y2-y1*x2)^2): simplify([y3std-y3],[C(x1,y1),
    C(x2,y2)]);
> y3:=(x1-x2)*(1+s*x1*x2)*(y1+y2+s*x1^2*y2+s*y1*x2^2)/((x1*y2-y1*x2)*(1-d*x1^2*x2^2))-s*x3^2-1: simplify([y3s
    td-y3],[C(x1,y1),C(x2,y2),d-s^2]);
> y3:=(1+s*x1*x2)*(x1*y1-y2*x2+s*x1^3*y2-s*y1*x2^3)/((x1*y2-y1*x2)*(1-d*x1^2*x2^2))-s*x3^2: simplify([y3std-y
    3],[C(x1,y1),C(x2,y2),d-s^2]);
> y3:=(x1-x2)*(y1+y2+d*x1*x2*(x1^2*y2+y1*x2^2))/((x1*y2-y1*x2)*(1-d*x1^2*x2^2))-1: simplify([y3std-y3],[C(x1,
    y1),C(x2,y2)]);
> y3:=(2*(x1*y1-x2*y2)-(x1*y2-y1*x2)*(y1*y2+2*a*x1*x2))/((x1*y2-y1*x2)*(1-d*x1^2*x2^2)): simplify([y3std-y3],
    [C(x1,y1),C(x2,y2)]);
> y3:=(x1-x2)^2/(x1*y2-y1*x2)^2*(y1*y2-2*a*x1*x2+1+d*x1^2*x2^2)-1: simplify([y3std-y3],[C(x1,y1),C(x2,y2)]);
> y3:=((x1^2+x2^2)*(y1*y2-2*a*x1*x2)-2*x1*x2*(1+d*x1^2*x2^2))/(x1*y2-y1*x2)^2: simplify([y3std-y3],[C(x1,y1),
    C(x2,y2)]);
```

DBL_Q_a00.

```
> x1:=X1/Z1: y1:=Y1/Z1:
> x3:=(2*y1/(2-y1^2+2*a*x1^2))*x1:
> y3:=(2*y1/(2-y1^2+2*a*x1^2))*((2*y1/(2-y1^2+2*a*x1^2))-y1)-1:
>
> DBL_Q_a00:=proc(X1,Y1,Z1) local X3,Y3,Z3:
>     X3:=2*X1*Y1*(2*Z1^2-Y1^2+2*a*X1^2):
>     Y3:=2*Y1^2*(Y1^2-2*a*X1^2)-(2*Z1^2-Y1^2+2*a*X1^2)^2:
>     Z3:=(2*Z1^2-Y1^2+2*a*X1^2)^2:
>     return X3,Y3,Z3:
> end proc:
> X3,Y3,Z3:=DBL_Q_a00(X1,Y1,Z1):
> simplify([x3-X3/Z3,y3-Y3/Z3],[(C(x1,y1))]); #Check.
>
```

DBL_Q_a01, $2\mathbf{M} + 5\mathbf{S} + 7\mathbf{a}$, assumes $a = -1/2$.

```
> DBL_Q_a01_opr:=proc(X1,Y1,Z1) local X3,Y3,Z3,A,B,C,D,E,F:
>     A:=2*Z1^2: B:=X1^2: C:=Y1^2: D:=B+C: E:=(X1+Y1)^2-D: F:=A-D: Z3:=F^2: X3:=E*F: Y3:=2*C*D-Z3:
>     return X3,Y3,Z3:
> end proc:
> X3,Y3,Z3:=DBL_Q_a01_opr(X1,Y1,Z1):
> simplify([x3-X3/Z3,y3-Y3/Z3],[(C(x1,y1)),a+1/2]); #Check.
>
> DBL_Q_a01_reg:=proc(X1,Y1,Z1) local X3,Y3,Z3,t1:
>     t1:=X1+Y1: X3:=X1^2: Y3:=Y1^2: Z3:=Z1^2: t1:=t1^2: X3:=X3+Y3: t1:=t1-X3: Z3:=2*Z3: Y3:=X3*Y3: Y3:=2*Y3:
    Z3:=Z3-X3: X3:=t1*Z3: Z3:=Z3^2: Y3:=Y3-Z3:
>     return X3,Y3,Z3:
> end proc:
> X3,Y3,Z3:=DBL_Q_a01_reg(X1,Y1,Z1):
> simplify([x3-X3/Z3,y3-Y3/Z3],[(C(x1,y1)),a+1/2]); #Check.
>
```

DBL_Q_a02, $3\mathbf{M} + 4\mathbf{S} + 4\mathbf{a}$, assumes $a = -1/2$.

```
> DBL_Q_a02_opr:=proc(X1,Y1,Z1) local X3,Y3,Z3,A,B,C,D,E,F:
>     A:=Z1^2: B:=X1^2: C:=Y1^2: D:=(C+B)/2: E:=A-D: Z3:=E^2: X3:=X1*Y1*E: Y3:=C*D-Z3:
>     return X3,Y3,Z3:
> end proc:
> X3,Y3,Z3:=DBL_Q_a02_opr(X1,Y1,Z1):
> simplify([x3-X3/Z3,y3-Y3/Z3],[(C(x1,y1)),a+1/2]); #Check.
>
> DBL_Q_a02_reg:=proc(X1,Y1,Z1) local X3,Y3,Z3,t1:
>     t1:=X1*Y1: X3:=X1^2: Y3:=Y1^2: Z3:=Z1^2: X3:=X3+Y3: X3:=X3/2: Y3:=Y3*X3: X3:=Z3-X3: Z3:=X3^2: Y3:=Y3-Z3
    : X3:=t1*X3:
>     return X3,Y3,Z3:
> end proc:
> X3,Y3,Z3:=DBL_Q_a02_reg(X1,Y1,Z1):
> simplify([x3-X3/Z3,y3-Y3/Z3],[(C(x1,y1)),a+1/2]); #Check.
>
```

DBL_Q_a03, $2\mathbf{M} + 5\mathbf{S} + 1\mathbf{D} + 8\mathbf{a}$, assumes $k = -2a$.

```
> DBL_Q_a03_opr:=proc(X1,Y1,Z1) local X3,Y3,Z3,A,B,C,D,E,F,G,H:
>     A:=2*Z1^2: B:=X1^2: C:=Y1^2: D:=B+C: E:=(X1+Y1)^2-D: F:=2*a*B: G:=C-F: H:=A-G: Z3:=H^2: X3:=E*H: Y3:=2*
    C*G-Z3:
>     return X3,Y3,Z3:
> end proc:
> X3,Y3,Z3:=DBL_Q_a03_opr(X1,Y1,Z1):
> simplify([x3-X3/Z3,y3-Y3/Z3],[(C(x1,y1)),k+2*a]); #Check.
>
> DBL_Q_a03_reg:=proc(X1,Y1,Z1) local X3,Y3,Z3,t1:
>     t1:=X1+Y1: X3:=X1^2: Y3:=Y1^2: Z3:=Z1^2: t1:=t1^2: t1:=t1-X3: t1:=t1-Y3: X3:=k*X3: X3:=Y3+X3: Z3:=2*Z3:
    Y3:=X3*Y3: Y3:=2*Y3: Z3:=Z3-X3: X3:=t1*Z3: Z3:=Z3^2: Y3:=Y3-Z3:
>     return X3,Y3,Z3:
> end proc:
> X3,Y3,Z3:=DBL_Q_a03_reg(X1,Y1,Z1):
> simplify([x3-X3/Z3,y3-Y3/Z3],[(C(x1,y1)),k+2*a]); #Check.
>
```

DBL_Q_a04, $3\mathbf{M} + 4\mathbf{S} + 1\mathbf{D} + 4\mathbf{a}$, assumes $k = -2a$.

```
> DBL_Q_a04_opr:=proc(X1,Y1,Z1) local X3,Y3,Z3,A,B,C,D,E,F,G,H:
>     A:=Z1^2: B:=X1^2: C:=Y1^2: D:=(C-2*a*B)/2: E:=A-D: Z3:=E^2: X3:=X1*Y1*E: Y3:=C*D-Z3:
>     return X3,Y3,Z3:
> end proc:
> X3,Y3,Z3:=DBL_Q_a04_opr(X1,Y1,Z1):
> simplify([x3-X3/Z3,y3-Y3/Z3],[(C(x1,y1)),k+2*a]); #Check.
>
```

```
> DBL_Q_a04_reg:=proc(X1,Y1,Z1) local X3,Y3,Z3,t1:
>     t1:=X1*Y1: X3:=X1^2: Y3:=Y1^2: Z3:=Z1^2: X3:=k*X3: X3:=X3+Y3: X3:=X3/2: Y3:=Y3*X3: X3:=Z3-X3: Z3:=X3^2:
    Y3:=Y3-Z3: X3:=t1*X3:
>     return X3,Y3,Z3:
> end proc:
> X3,Y3,Z3:=DBL_Q_a04_reg(X1,Y1,Z1):
> simplify([x3-X3/Z3,y3-Y3/Z3],[(C(x1,y1)),k+2*a]); #Check.
>
```

ADD_Q_a00.

```
> x1:=X1/Z1: y1:=Y1/Z1: x2:=X2/Z2: y2:=Y2/Z2:
> x3:=(x1*y2+y1*x2)/(1-d*x1^2*x2^2):
> y3:=((y1*y2+2*a*x1*x2)*(1+d*x1^2*x2^2)+2*d*x1*x2*(x1^2+x2^2))/(1-d*x1^2*x2^2)^2:
>
> ADD_Q_a00:=proc(X1,Y1,Z1,X2,Y2,Z2) local X3,Y3,Z3:
>     X3:=(X1*Y2-Y1*X2)*(X1^2*Z2^2-Z1^2*X2^2):
>     Y3:=(Y1*Y2-2*a*X1*X2)*(X1^2*Z2^2+Z1^2*X2^2)-2*X1*X2*(Z1^2*Z2^2+d*X1^2*X2^2):
>     Z3:=Z1*Z2*(X1*Y2-Y1*X2)^2:
>     return X3,Y3,Z3:
> end proc:
> X3,Y3,Z3:=ADD_Q_a00(X1,Y1,Z1,X2,Y2,Z2):
> simplify([x3-X3/Z3,y3-Y3/Z3],[(C(x1,y1)),(C(x2,y2))]); #Check.
>
```

ADD_Q_a01, $10\mathbf{M} + 5\mathbf{S} + 2\mathbf{D} + 10\mathbf{a}$.

```
> ADD_Q_a01_opr:=proc(X1,Y1,Z1,X2,Y2,Z2) local X3,Y3,Z3,A,B,C,D,E,F,G,H,J,K,L:
>     A:=Z1*Z2: B:=Y1*Y2: C:=X1*X2: D:=(X1-Y1)*(X2+Y2)+B-C: F:=(X1*Z2)^2: G:=(Z1*X2)^2: X3:=D*(F-G): Y3:=(B-(
    2*a)*C)*(F+G)-2*C*(A^2+d*C^2): Z3:=A*D^2:
>     return X3,Y3,Z3:
> end proc:
> X3,Y3,Z3:=ADD_Q_a01_opr(X1,Y1,Z1,X2,Y2,Z2):
> simplify([x3-X3/Z3,y3-Y3/Z3],[(C(x1,y1)),(C(x2,y2))]); #Check.
>
```

UADD_Q_a00.

```
> x1:=X1/Z1: y1:=Y1/Z1: x2:=X2/Z2: y2:=Y2/Z2:
> x3:=(x1*y2+y1*x2)/(1-d*x1^2*x2^2):
> y3:=((y1*y2+2*a*x1*x2)*(1+d*x1^2*x2^2)+2*d*x1*x2*(x1^2+x2^2))/(1-d*x1^2*x2^2)^2:
>
> UADD_Q_a00:=proc(X1,Y1,Z1,X2,Y2,Z2) local X3,Y3,Z3:
>     X3:=Z1*Z2*(X1*Y2+Y1*X2)*(Z1^2*Z2^2-d*X1^2*X2^2):
>     Y3:=Z1*Z2*(Y1*Y2+2*a*X1*X2)*(Z1^2*Z2^2+d*X1^2*X2^2)+2*d*X1*X2*Z1*Z2*(X1^2*Z2^2+Z1^2*X2^2):
>     Z3:=(Z1^2*Z2^2-d*X1^2*X2^2)^2:
>     return X3,Y3,Z3:
> end proc:
> X3,Y3,Z3:=UADD_Q_a00(X1,Y1,Z1,X2,Y2,Z2):
> simplify([x3-X3/Z3,y3-Y3/Z3],[(C(x1,y1)),(C(x2,y2))]); #Check.
>
```

UADD_Q_a01, $10\mathbf{M} + 7\mathbf{S} + 3\mathbf{D} + 17\mathbf{a}$.

```
> UADD_Q_a01_opr:=proc(X1,Y1,Z1,X2,Y2,Z2) local X3,Y3,Z3,A,B,C,D,E,F,G,H,J,K,L,M,N,P:
>     A:=Z1*Z2: B:=Y1*Y2: C:=X1*X2: D:=(X1+Y1)*(X2+Y2)-B-C: F:=(X1*Z2)^2: G:=(Z1*X2)^2: H:=A^2: J:=C^2: K:=d*
    J: L:=H-K: M:=L^2: N:=(A+L)^2-H-M: P:=(A+C)^2-H-J: X3:=D*N: Y3:=2*A*(B+(2*a)*C)*(H+K)+(2*d)*P*(F+G): Z3:=2*M:
>     return X3,Y3,Z3:
> end proc:
> X3,Y3,Z3:=UADD_Q_a01_opr(X1,Y1,Z1,X2,Y2,Z2):
> simplify([x3-X3/Z3,y3-Y3/Z3],[(C(x1,y1)),(C(x2,y2))]); #Check.
>
```

UADD_Q_a02, $12\mathbf{M} + 5\mathbf{S} + 3\mathbf{D} + 9\mathbf{a}$.

```
> UADD_Q_a02_opr:=proc(X1,Y1,Z1,X2,Y2,Z2) local X3,Y3,Z3,A,B,C,D,E,F,G,H,J,K,L,M:
>     A:=X1*Z2: B:=Y1*Z2: C:=Z1*X2: D:=Z1*Y2: E:=A*C: F:=B*D: G:=(A+B)*(C+D)-E-F: H:=Z1*Z2: J:=X1*X2: K:=H^2:
     L:=d*J^2: M:=K-L: X3:=G*M: Z3:=M^2: Y3:=(K+L)*(F+(2*a)*E)+(2*d)*E*(A^2+C^2):
>     return X3,Y3,Z3:
> end proc:
> X3,Y3,Z3:=UADD_Q_a02_opr(X1,Y1,Z1,X2,Y2,Z2):
> simplify([x3-X3/Z3,y3-Y3/Z3],[(C(x1,y1)),(C(x2,y2))]); #Check.
>
```

DBL_Qe_a00.

```
> x1:=X1/Z1: y1:=Y1/Z1: T1:=X1^2/Z1:
> x3:=(2*y1/(2-y1^2+2*a*x1^2))*x1:
> y3:=(2*y1/(2-y1^2+2*a*x1^2))*((2*y1/(2-y1^2+2*a*x1^2))-y1)-1:
>
> DBL_Qe_a00:=proc(X1,Y1,Z1,T1) local X3,Y3,Z3,T3:
>     X3:=2*X1*Y1*(2*Z1^2-Y1^2+2*a*X1^2):
>     Y3:=2*Y1^2*(Y1^2-2*a*X1^2)-(2*Z1^2-Y1^2+2*a*X1^2)^2:
>     T3:=(2*X1*Y1)^2:
>     Z3:=(2*Z1^2-Y1^2+2*a*X1^2)^2:
>     return X3,Y3,Z3,T3:
> end proc:
> X3,Y3,Z3,T3:=DBL_Qe_a00(X1,Y1,Z1,T1):
> simplify([x3-X3/Z3,y3-Y3/Z3,T3-X3^2/Z3],[(C(x1,y1))]); #Check.
>
```

DBL_Qe_a01, $8\mathbf{S} + 13\mathbf{a}$, assumes $a = -1/2$.

```
> DBL_Qe_a01_reg:=proc(X1,Y1,Z1,T1) local X3,Y3,Z3,T3,t1:
>     T3:=X1+Y1: X3:=X1^2: Y3:=Y1^2: Z3:=Z1^2: T3:=T3^2: X3:=X3+Y3: T3:=T3-X3: Z3:=2*Z3: Z3:=Z3-X3: X3:=T3+Z3
    : T3:=T3^2: Z3:=Z3^2: X3:=X3^2: X3:=X3-T3: X3:=X3-Z3: Z3:=2*Z3: Y3:=2*Y3: Y3:=Y3^2: Y3:=Y3+T3: Y3:=Y3-Z3: T3:=
    2*T3:
>     return X3,Y3,Z3,T3:
> end proc:
> X3,Y3,Z3,T3:=DBL_Qe_a01_reg(X1,Y1,Z1,T1):
> simplify([x3-X3/Z3,y3-Y3/Z3,T3-X3^2/Z3],[(C(x1,y1)),a+1/2]); #Check.
>
```

DBL_Qe_a02, $1\mathbf{M} + 7\mathbf{S} + 9\mathbf{a}$, assumes $a = -1/2$.

```
> DBL_Qe_a02_reg:=proc(X1,Y1,Z1,T1) local X3,Y3,Z3,T3,t1:
>     T3:=X1+Y1: X3:=X1^2: Y3:=Y1^2: Z3:=Z1^2: T3:=T3^2: X3:=X3+Y3: T3:=T3-X3: Z3:=2*Z3: Z3:=Z3-X3: X3:=T3*Z3
    : Z3:=Z3^2: T3:=T3^2: Y3:=2*Y3: Y3:=Y3^2: Y3:=Y3+T3: Y3:=Y3/2: Y3:=Y3-Z3:
>     return X3,Y3,Z3,T3:
> end proc:
> X3,Y3,Z3,T3:=DBL_Qe_a02_reg(X1,Y1,Z1,T1):
> simplify([x3-X3/Z3,y3-Y3/Z3,T3-X3^2/Z3],[(C(x1,y1)),a+1/2]); #Check.
>
```

DBL_Qe_a03, $2\mathbf{M} + 6\mathbf{S} + 6\mathbf{a}$, assumes $a = -1/2$.

```
> DBL_Qe_a03_reg:=proc(X1,Y1,Z1,T1) local X3,Y3,Z3,T3,t1:
>     T3:=X1*Y1: X3:=X1^2: Y3:=Y1^2: Z3:=Z1^2: X3:=X3+Y3: X3:=X3/2: Z3:=Z3-X3: X3:=T3*Z3: Z3:=Z3^2: T3:=T3^2:
     Y3:=Y3^2: Y3:=Y3+T3: Y3:=Y3/2: Y3:=Y3-Z3:
>     return X3,Y3,Z3,T3:
> end proc:
> X3,Y3,Z3,T3:=DBL_Qe_a03_reg(X1,Y1,Z1,T1):
> simplify([x3-X3/Z3,y3-Y3/Z3,T3-X3^2/Z3],[(C(x1,y1)),a+1/2]); #Check.
>
```

DBL_Qe_a04, $3\mathbf{M} + 5\mathbf{S} + 4\mathbf{a}$, assumes $a = -1/2$.

```
> DBL_Qe_a04_reg:=proc(X1,Y1,Z1,T1) local X3,Y3,Z3,T3,t1:
>     T3:=X1*Y1: X3:=X1^2: Y3:=Y1^2: Z3:=Z1^2: X3:=X3+Y3: X3:=X3/2: Y3:=Y3*X3: X3:=Z3-X3: Z3:=X3^2: Y3:=Y3-Z3
    : X3:=X3*T3: T3:=T3^2:
>     return X3,Y3,Z3,T3:
> end proc:
> X3,Y3,Z3,T3:=DBL_Qe_a04_reg(X1,Y1,Z1,T1):
> simplify([x3-X3/Z3,y3-Y3/Z3,T3-X3^2/Z3],[(C(x1,y1)),a+1/2]); #Check.
>
```

DBL_Qe_a05, $8\mathbf{S} + 2\mathbf{D} + 14\mathbf{a}$, assumes $k = -2a$.

```
> DBL_Qe_a05_opr:=proc(X1,Y1,Z1,T1) local X3,Y3,Z3,T3,A,B,C,D,E,F,G,H:
>     A:=Z1^2: B:=Y1^2: C:=X1^2: D:=B+C: E:=(X1+Y1)^2-D: F:=2*A-B+(2*a)*C: G:=E^2: H:=F^2: X3:=(E+F)^2-G-H: T
    3:=2*G: Z3:=2*H: Y3:=(2*B)^2-(2*a)*G-Z3:
>     return X3,Y3,Z3,T3:
> end proc:
> X3,Y3,Z3,T3:=DBL_Qe_a05_opr(X1,Y1,Z1,T1):
> simplify([x3-X3/Z3,y3-Y3/Z3,T3-X3^2/Z3],[(C(x1,y1)),k+2*a]); #Check.
>
> DBL_Qe_a05_reg:=proc(X1,Y1,Z1,T1) local X3,Y3,Z3,T3,t1:
>     T3:=X1+Y1: X3:=X1^2: Y3:=Y1^2: Z3:=Z1^2: T3:=T3^2: T3:=T3-X3: t1:=T3-Y3: X3:=k*X3: X3:=X3+Y3: Z3:=2*Z3:
     Z3:=Z3-X3: T3:=t1^2: X3:=t1+Z3: X3:=X3^2: Z3:=Z3^2: X3:=X3-T3: X3:=X3-Z3: Z3:=2*Z3: t1:=k*T3: T3:=2*T3: Y3:=2
    *Y3: Y3:=Y3^2: Y3:=Y3+t1: Y3:=Y3-Z3:
>     return X3,Y3,Z3,T3:
> end proc:
> X3,Y3,Z3,T3:=DBL_Qe_a05_reg(X1,Y1,Z1,T1):
> simplify([x3-X3/Z3,y3-Y3/Z3,T3-X3^2/Z3],[(C(x1,y1)),k+2*a]); #Check.
>
```

DBL_Qe_a06, $1\mathbf{M} + 7\mathbf{S} + 1\mathbf{D} + 12\mathbf{a}$, assumes $k = -2a$.

```
> DBL_Qe_a06_reg:=proc(X1,Y1,Z1,T1) local X3,Y3,Z3,T3,t1:
>     t1:=X1+Y1: X3:=X1^2: Y3:=Y1^2: Z3:=Z1^2: t1:=t1^2: t1:=t1-X3: t1:=t1-Y3: X3:=k*X3: X3:=X3+Y3: Z3:=2*Z3:
     Y3:=X3*Y3: Z3:=Z3-X3: T3:=t1^2: X3:=t1+Z3: Z3:=Z3^2: Y3:=2*Y3: Y3:=Y3-Z3: X3:=X3^2: X3:=X3-T3: X3:=X3-Z3: X3:
    =X3/2:
>     return X3,Y3,Z3,T3:
> end proc:
> X3,Y3,Z3,T3:=DBL_Qe_a06_reg(X1,Y1,Z1,T1):
> simplify([x3-X3/Z3,y3-Y3/Z3,T3-X3^2/Z3],[(C(x1,y1)),k+2*a]); #Check.
>
```

DBL_Qe_a07, $1\mathbf{M} + 7\mathbf{S} + 2\mathbf{D} + 10\mathbf{a}$, assumes $k = -2a$.

```
> DBL_Qe_a07_reg:=proc(X1,Y1,Z1,T1) local X3,Y3,Z3,T3,t1:
>     t1:=X1+Y1: X3:=X1^2: Y3:=Y1^2: Z3:=Z1^2: t1:=t1^2: t1:=t1-X3: t1:=t1-Y3: X3:=k*X3: X3:=X3+Y3: Z3:=2*Z3:
     Z3:=Z3-X3: X3:=t1*Z3: T3:=t1^2: Z3:=Z3^2: Y3:=2*Y3: Y3:=Y3^2: t1:=k*T3: Y3:=Y3+t1: Y3:=Y3/2: Y3:=Y3-Z3:
>     return X3,Y3,Z3,T3:
> end proc:
> X3,Y3,Z3,T3:=DBL_Qe_a07_reg(X1,Y1,Z1,T1):
> simplify([x3-X3/Z3,y3-Y3/Z3,T3-X3^2/Z3],[(C(x1,y1)),k+2*a]); #Check.
>
```

DBL_Qe_a08, $2\mathbf{M} + 6\mathbf{S} + 1\mathbf{D} + 8\mathbf{a}$, assumes $k = -2a$.

```
> DBL_Qe_a08_reg:=proc(X1,Y1,Z1,T1) local X3,Y3,Z3,T3,t1:
>     T3:=X1+Y1: X3:=X1^2: Y3:=Y1^2: Z3:=Z1^2: T3:=T3^2: T3:=T3-X3: T3:=T3-Y3: X3:=k*X3: X3:=Y3+X3: Z3:=2*Z3:
     Y3:=X3*Y3: Y3:=2*Y3: Z3:=Z3-X3: X3:=T3*Z3: Z3:=Z3^2: Y3:=Y3-Z3: T3:=T3^2:
>     return X3,Y3,Z3,T3:
> end proc:
> X3,Y3,Z3,T3:=DBL_Qe_a08_reg(X1,Y1,Z1,T1):
> simplify([x3-X3/Z3,y3-Y3/Z3,T3-X3^2/Z3],[(C(x1,y1)),k+2*a]); #Check.
>
```

DBL_Qe_a09, $2\mathbf{M} + 6\mathbf{S} + 2\mathbf{D} + 6\mathbf{a}$, assumes $k = -2a$.

```
> DBL_Qe_a09_reg:=proc(X1,Y1,Z1,T1) local X3,Y3,Z3,T3,t1:
>     t1:=X1*Y1: X3:=X1^2: Y3:=Y1^2: Z3:=Z1^2: X3:=k*X3: X3:=X3+Y3: X3:=X3/2: Z3:=Z3-X3: X3:=t1*Z3: Z3:=Z3^2:
    T3:=t1^2: Y3:=Y3^2: t1:=k*T3: Y3:=Y3+t1: Y3:=Y3/2: Y3:=Y3-Z3:
>     return X3,Y3,Z3,T3:
> end proc:
> X3,Y3,Z3,T3:=DBL_Qe_a09_reg(X1,Y1,Z1,T1):
> simplify([x3-X3/Z3,y3-Y3/Z3,T3-X3^2/Z3],[(C(x1,y1)),k+2*a]); #Check.
>
```

DBL_Qe_a10, $3\mathbf{M} + 5\mathbf{S} + 1\mathbf{D} + 4\mathbf{a}$, assumes $k = -2a$.

```
> DBL_Qe_a10_opr:=proc(X1,Y1,Z1,T1) local X3,Y3,Z3,T3,A,B,C,D,E,F:
>     A:=Z1^2: B:=Y1^2: C:=X1^2: D:=X1*Y1: E:=(B-(2*a)*C)/2: F:=A-E: T3:=D^2: Z3:=F^2: X3:=D*F: Y3:=B*E-Z3:
>     return X3,Y3,Z3,T3:
> end proc:
> X3,Y3,Z3,T3:=DBL_Qe_a10_opr(X1,Y1,Z1,T1):
> simplify([x3-X3/Z3,y3-Y3/Z3,T3-X3^2/Z3],[(C(x1,y1)),k+2*a]); #Check.
>
> DBL_Qe_a10_reg:=proc(X1,Y1,Z1,T1) local X3,Y3,Z3,T3,t1:
>     T3:=X1*Y1: X3:=X1^2:Y3:=Y1^2: Z3:=Z1^2: X3:=k*X3: X3:=X3+Y3: X3:=X3/2: Y3:=Y3*X3: X3:=Z3-X3: Z3:=X3^2:
    Y3:=Y3-Z3: X3:=T3*X3: T3:=T3^2:
>     return X3,Y3,Z3,T3:
> end proc:
> X3,Y3,Z3,T3:=DBL_Qe_a10_reg(X1,Y1,Z1,T1):
> simplify([x3-X3/Z3,y3-Y3/Z3,T3-X3^2/Z3],[(C(x1,y1)),k+2*a]); #Check.
>
```

ADD_Qe_a00.

```
> x1:=X1/Z1: y1:=Y1/Z1: T1:=X1^2/Z1: x2:=X2/Z2: y2:=Y2/Z2: T2:=X2^2/Z2:
> x3:=(x1*y2+y1*x2)/(1-d*x1^2*x2^2):
> y3:=((y1*y2+2*a*x1*x2)*(1+d*x1^2*x2^2)+2*d*x1*x2*(x1^2+x2^2))/(1-d*x1^2*x2^2)^2:
>
> ADD_Qe_a00:=proc(X1,Y1,Z1,T1,X2,Y2,Z2,T2) local X3,Y3,Z3,T3:
>     X3:=(X1*Y2-Y1*X2)*(T1*Z2-Z1*T2):
>     Y3:=(Y1*Y2-2*a*X1*X2)*(T1*Z2+Z1*T2)-2*X1*X2*(Z1*Z2+d*T1*T2):
>     T3:=(T1*Z2-Z1*T2)^2:
>     Z3:=(X1*Y2-Y1*X2)^2:
>     return X3,Y3,Z3,T3:
> end proc:
> X3,Y3,Z3,T3:=ADD_Qe_a00(X1,Y1,Z1,T1,X2,Y2,Z2,T2):
> simplify([x3-X3/Z3,y3-Y3/Z3,T3-X3^2/Z3],[(C(x1,y1)),(C(x2,y2))]); #Check.
>
```

ADD_Qe_b00.

```
> x1:=X1/Z1: y1:=Y1/Z1: T1:=X1^2/Z1: x2:=X2/Z2: y2:=Y2/Z2: T2:=X2^2/Z2:
> x3:=(x1*y2+y1*x2)/(1-d*x1^2*x2^2):
> y3:=((y1*y2+2*a*x1*x2)*(1+d*x1^2*x2^2)+2*d*x1*x2*(x1^2+x2^2))/(1-d*x1^2*x2^2)^2:
>
> ADD_Qe_b00:=proc(X1,Y1,Z1,T1,X2,Y2,Z2,T2) local X3,Y3,Z3,T3:
>     X3:=(X1*Y2-Y1*X2)*(T1*Z2-Z1*T2):
>     Y3:=(T1*Z2+Z1*T2-2*X1*X2)*(Y1*Y2-2*a*X1*X2+Z1*Z2+d*T1*T2)-(X1*Y2-Y1*X2)^2:
>     T3:=(T1*Z2-Z1*T2)^2:
>     Z3:=(X1*Y2-Y1*X2)^2:
>     return X3,Y3,Z3,T3:
> end proc:
> X3,Y3,Z3,T3:=ADD_Qe_b00(X1,Y1,Z1,T1,X2,Y2,Z2,T2):
> simplify([x3-X3/Z3,y3-Y3/Z3,T3-X3^2/Z3],[(C(x1,y1)),(C(x2,y2))]); #Check.
>
```

ADD_Qe_b01, $7\mathbf{M} + 3\mathbf{S} + 2\mathbf{D} + 19\mathbf{a}$, assumes $a = -1/2$.

```
> ADD_Qe_b01_reg:=proc(X1,Y1,Z1,T1,X2,Y2,Z2,T2) local X3,Y3,Z3,T3,t1,t2,t3:
>     t1:=T1+Z1: t2:=d*T2: t2:=t2+Z2: t1:=t1*t2: t2:=Z1*T2: T3:=T1*Z2: t1:=t1-T3: t3:=d*t2: t1:=t1-t3: t3:=T3
    +t2: T3:=T3-t2: Z3:=X1-Y1: t2:=X2+Y2: Z3:=Z3*t2: t2:=X1*X2: Y3:=Y1*Y2: Z3:=Z3-t2: Z3:=Z3+Y3: Y3:=Y3+t1: t1:=2*
    t2: t1:=t3-t1: Y3:=Y3+t2: Y3:=t1*Y3: X3:=Z3+T3: X3:=X3^2: Z3:=Z3^2: Y3:=Y3-Z3: X3:=X3-Z3: T3:=T3^2: X3:=X3-T3:
     X3:=X3/2:
>     return X3,Y3,Z3,T3:
> end proc:
> X3,Y3,Z3,T3:=ADD_Qe_b01_reg(X1,Y1,Z1,T1,X2,Y2,Z2,T2):
> simplify([x3-X3/Z3,y3-Y3/Z3,T3-X3^2/Z3],[(C(x1,y1)),(C(x2,y2)),a+1/2]); #Check.
>
```

ADD_Qe_b02, $8\mathbf{M} + 2\mathbf{S} + 2\mathbf{D} + 15\mathbf{a}$, assumes $a = -1/2$.

```
> ADD_Qe_b02_reg:=proc(X1,Y1,Z1,T1,X2,Y2,Z2,T2) local X3,Y3,Z3,T3,t1,t2,t3:
>     t1:=T1+Z1: t2:=d*T2: t2:=t2+Z2: t1:=t1*t2: t2:=Z1*T2: T3:=T1*Z2: t1:=t1-T3: t3:=d*t2: t1:=t1-t3: t3:=T3
    +t2: T3:=T3-t2: Z3:=X1-Y1: t2:=X2+Y2: Z3:=Z3*t2: t2:=X1*X2: Y3:=Y1*Y2: Z3:=Z3-t2: Z3:=Z3+Y3: Y3:=Y3+t1: t1:=2*
    t2: t1:=t3-t1: Y3:=Y3+t2: Y3:=t1*Y3: X3:=Z3*T3: Z3:=Z3^2: Y3:=Y3-Z3: T3:=T3^2:
>     return X3,Y3,Z3,T3:
> end proc:
> X3,Y3,Z3,T3:=ADD_Qe_b02_reg(X1,Y1,Z1,T1,X2,Y2,Z2,T2):
> simplify([x3-X3/Z3,y3-Y3/Z3,T3-X3^2/Z3],[(C(x1,y1)),(C(x2,y2)),a+1/2]); #Check.
>
```

ADD_Qe_b03, $7\mathbf{M} + 3\mathbf{S} + 3\mathbf{D} + 19\mathbf{a}$, assumes $k = -2a$.

```
> ADD_Qe_b03_opr:=proc(X1,Y1,Z1,T1,X2,Y2,Z2,T2) local X3,Y3,Z3,T3,A,B,C,D,E,F:
>     A:=T1*Z2: B:=Z1*T2: C:=X1*X2: D:=Y1*Y2: E:=(X1-Y1)*(X2+Y2)-C+D: F:=A-B: Z3:=E^2: T3:=F^2: X3:=((E+F)^2-
    T3-Z3)/2: Y3:=(A+B-2*C)*(D-(2*a)*C+(Z1+T1)*(Z2+d*T2)-A-d*B)-Z3:
>     return X3,Y3,Z3,T3:
> end proc:
> X3,Y3,Z3,T3:=ADD_Qe_b03_opr(X1,Y1,Z1,T1,X2,Y2,Z2,T2):
> simplify([x3-X3/Z3,y3-Y3/Z3,T3-X3^2/Z3],[(C(x1,y1)),(C(x2,y2)),k+2*a]); #Check.
>
> ADD_Qe_b03_reg:=proc(X1,Y1,Z1,T1,X2,Y2,Z2,T2) local X3,Y3,Z3,T3,t1,t2,t3:
>     t1:=T1+Z1: t2:=d*T2: t2:=t2+Z2: t1:=t1*t2: t2:=Z1*T2: T3:=T1*Z2: t1:=t1-T3: t3:=d*t2: t1:=t1-t3: t3:=T3
    +t2: T3:=T3-t2: Z3:=X1-Y1: t2:=X2+Y2: Z3:=Z3*t2: t2:=X1*X2: Y3:=Y1*Y2: Z3:=Z3-t2: Z3:=Z3+Y3: Y3:=Y3+t1: t1:=2*
    t2: t1:=t3-t1: t2:=k*t2: Y3:=Y3+t2: Y3:=t1*Y3: X3:=Z3+T3: X3:=X3^2: Z3:=Z3^2: Y3:=Y3-Z3: X3:=X3-Z3: T3:=T3^2:
    X3:=X3-T3: X3:=X3/2:
>     return X3,Y3,Z3,T3:
> end proc:
> X3,Y3,Z3,T3:=ADD_Qe_b03_reg(X1,Y1,Z1,T1,X2,Y2,Z2,T2):
> simplify([x3-X3/Z3,y3-Y3/Z3,T3-X3^2/Z3],[(C(x1,y1)),(C(x2,y2)),k+2*a]); #Check.
>
```

ADD_Qe_b04, $8\mathbf{M} + 2\mathbf{S} + 3\mathbf{D} + 15\mathbf{a}$, assumes $k = -2a$.

```
> ADD_Qe_b04_opr:=proc(X1,Y1,Z1,T1,X2,Y2,Z2,T2) local X3,Y3,Z3,T3,A,B,C,D,E,F:
>     A:=T1*Z2: B:=Z1*T2: C:=X1*X2: D:=Y1*Y2: E:=(X1-Y1)*(X2+Y2)-C+D: F:=A-B: Z3:=E^2: T3:=F^2: X3:=E*F: Y3:=
    (A+B-2*C)*(D-(2*a)*C+(Z1+T1)*(Z2+d*T2)-A-d*B)-Z3:
>     return X3,Y3,Z3,T3:
> end proc:
> X3,Y3,Z3,T3:=ADD_Qe_b04_opr(X1,Y1,Z1,T1,X2,Y2,Z2,T2):
> simplify([x3-X3/Z3,y3-Y3/Z3,T3-X3^2/Z3],[(C(x1,y1)),(C(x2,y2)),k+2*a]); #Check.
>
> ADD_Qe_b04_reg:=proc(X1,Y1,Z1,T1,X2,Y2,Z2,T2) local X3,Y3,Z3,T3,t1,t2,t3:
>     t1:=T1+Z1: t2:=d*T2: t2:=t2+Z2: t1:=t1*t2: t2:=Z1*T2: T3:=T1*Z2: t1:=t1-T3: t3:=d*t2: t1:=t1-t3: t3:=T3
    +t2: T3:=T3-t2: Z3:=X1-Y1: t2:=X2+Y2: Z3:=Z3*t2: t2:=X1*X2: Y3:=Y1*Y2: Z3:=Z3-t2: Z3:=Z3+Y3: Y3:=Y3+t1: t1:=2*
    t2: t1:=t3-t1: t2:=k*t2: Y3:=Y3+t2: Y3:=t1*Y3: X3:=Z3*T3: Z3:=Z3^2: Y3:=Y3-Z3: T3:=T3^2:
>     return X3,Y3,Z3,T3:
> end proc:
> X3,Y3,Z3,T3:=ADD_Qe_b04_reg(X1,Y1,Z1,T1,X2,Y2,Z2,T2):
> simplify([x3-X3/Z3,y3-Y3/Z3,T3-X3^2/Z3],[(C(x1,y1)),(C(x2,y2)),k+2*a]); #Check.
>
```

UADD_Qe_a00.

```
> x1:=X1/Z1: y1:=Y1/Z1: T1:=X1^2/Z1: x2:=X2/Z2: y2:=Y2/Z2: T2:=X2^2/Z2:
> x3:=(x1*y2+y1*x2)/(1-d*x1^2*x2^2):
> y3:=((y1*y2+2*a*x1*x2)*(1+d*x1^2*x2^2)+2*d*x1*x2*(x1^2+x2^2))/(1-d*x1^2*x2^2)^2:
>
> UADD_Qe_a00:=proc(X1,Y1,Z1,T1,X2,Y2,Z2,T2) local X3,Y3,Z3,T3:
>     X3:=(X1*Y2+Y1*X2)*(Z1*Z2-d*T1*T2):
>     Y3:=(Y1*Y2+2*a*X1*X2)*(Z1*Z2+d*T1*T2)+2*d*X1*X2*(T1*Z2+Z1*T2):
>     T3:=(X1*Y2+Y1*X2)^2:
>     Z3:=(Z1*Z2-d*T1*T2)^2:
>     return X3,Y3,Z3,T3:
> end proc:
> X3,Y3,Z3,T3:=UADD_Qe_a00(X1,Y1,Z1,T1,X2,Y2,Z2,T2):
> simplify([x3-X3/Z3,y3-Y3/Z3,T3-X3^2/Z3],[(C(x1,y1)),(C(x2,y2))]); #Check.
>
```

UADD_Qe_a01, $8\mathbf{M} + 3\mathbf{S} + 2\mathbf{D} + 17\mathbf{a}$, assumes $a = -1/2, h = 2d$.

```
> UADD_Qe_a01_reg:=proc(X1,Y1,Z1,T1,X2,Y2,Z2,T2) local X3,Y3,Z3,T3,t1,t2,t3:
>     t1:=T1+Z1: t2:=T2+Z2: T3:=T1*T2: Z3:=Z1*Z2: t1:=t1*t2: t1:=t1-T3: t1:=t1-Z3: T3:=d*T3: t2:=Z3+T3: Z3:=Z
    3-T3: T3:=X1+Y1: t3:=X2+Y2: X3:=X1*X2: Y3:=Y1*Y2: T3:=T3*t3: T3:=T3-X3: T3:=T3-Y3: t1:=h*t1: t1:=t1*X3: Y3:=Y3
    -X3: Y3:=Y3*t2: Y3:=Y3+t1: X3:=Z3+T3: X3:=X3^2: T3:=T3^2: Z3:=Z3^2: X3:=X3-T3: X3:=X3-Z3: X3:=X3/2:
>     return X3,Y3,Z3,T3:
> end proc:
> X3,Y3,Z3,T3:=UADD_Qe_a01_reg(X1,Y1,Z1,T1,X2,Y2,Z2,T2):
> simplify([x3-X3/Z3,y3-Y3/Z3,T3-X3^2/Z3],[(C(x1,y1)),(C(x2,y2)),a+1/2, h-2*d]); #Check.
>
```

UADD_Qe_a02, $9\mathbf{M} + 2\mathbf{S} + 2\mathbf{D} + 13\mathbf{a}$, assumes $a = -1/2, h = 2d$.

```
> UADD_Qe_a02_reg:=proc(X1,Y1,Z1,T1,X2,Y2,Z2,T2) local X3,Y3,Z3,T3,t1,t2,t3:
>     t1:=T1+Z1: t2:=T2+Z2: T3:=T1*T2: Z3:=Z1*Z2: t1:=t1*t2: t1:=t1-T3: t1:=t1-Z3: T3:=d*T3: t2:=Z3+T3: Z3:=Z
    3-T3: T3:=X1+Y1: t3:=X2+Y2: X3:=X1*X2: Y3:=Y1*Y2: T3:=T3*t3: T3:=T3-X3: T3:=T3-Y3: t1:=h*t1: t1:=t1*X3: Y3:=Y3
    -X3: Y3:=Y3*t2: Y3:=Y3+t1: X3:=Z3*T3: T3:=T3^2: Z3:=Z3^2:
>     return X3,Y3,Z3,T3:
> end proc:
> X3,Y3,Z3,T3:=UADD_Qe_a02_reg(X1,Y1,Z1,T1,X2,Y2,Z2,T2):
> simplify([x3-X3/Z3,y3-Y3/Z3,T3-X3^2/Z3],[(C(x1,y1)),(C(x2,y2)),a+1/2, h-2*d]); #Check.
>
```

UADD_Qe_a03, $8\mathbf{M} + 3\mathbf{S} + 3\mathbf{D} + 17\mathbf{a}$, assumes $k = -2a, h = 2d$.

```
> UADD_Qe_a03_opr:=proc(X1,Y1,Z1,T1,X2,Y2,Z2,T2) local X3,Y3,Z3,T3,A,B,C,D,E,F,G:
>     A:=Z1*Z2: B:=T1*T2: C:=X1*X2: D:=Y1*Y2: E:=d*B: F:=(X1+Y1)*(X2+Y2)-C-D: G:=A-E: Z3:=G^2: T3:=F^2: X3:=(
    (F+G)^2-T3-Z3)/2: Y3:=(D+(2*a)*C)*(A+E)+(2*d)*C*((T1+Z1)*(T2+Z2)-A-B):
>     return X3,Y3,Z3,T3:
> end proc:
> X3,Y3,Z3,T3:=UADD_Qe_a03_opr(X1,Y1,Z1,T1,X2,Y2,Z2,T2):
> simplify([x3-X3/Z3,y3-Y3/Z3,T3-X3^2/Z3],[(C(x1,y1)),(C(x2,y2)),k+2*a, h-2*d]); #Check.
>
> UADD_Qe_a03_reg:=proc(X1,Y1,Z1,T1,X2,Y2,Z2,T2) local X3,Y3,Z3,T3,t1,t2,t3:
>     t1:=T1+Z1: t2:=T2+Z2: T3:=T1*T2: Z3:=Z1*Z2: t1:=t1*t2:  t1:=t1-T3: t1:=t1-Z3: T3:=d*T3: t2:=Z3+T3: Z3:=
    Z3-T3: T3:=X1+Y1: t3:=X2+Y2: X3:=X1*X2: Y3:=Y1*Y2: T3:=T3*t3: T3:=T3-X3: T3:=T3-Y3: t1:=h*t1: t1:=t1*X3: X3:=k
    *X3: Y3:=Y3-X3: Y3:=Y3*t2: Y3:=Y3+t1: X3:=Z3+T3: X3:=X3^2: T3:=T3^2: Z3:=Z3^2: X3:=X3-T3: X3:=X3-Z3: X3:=X3/2:
>     return X3,Y3,Z3,T3:
> end proc:
> X3,Y3,Z3,T3:=UADD_Qe_a03_reg(X1,Y1,Z1,T1,X2,Y2,Z2,T2):
> simplify([x3-X3/Z3,y3-Y3/Z3,T3-X3^2/Z3],[(C(x1,y1)),(C(x2,y2)),k+2*a, h-2*d]); #Check.
>
```

UADD_Qe_a04, $9\mathbf{M} + 2\mathbf{S} + 3\mathbf{D} + 13\mathbf{a}$, assumes $k = -2a, h = 2d$.

```
> UADD_Qe_a04_opr:=proc(X1,Y1,Z1,T1,X2,Y2,Z2,T2) local X3,Y3,Z3,T3,A,B,C,D,E,F,G:
>     A:=Z1*Z2: B:=T1*T2: C:=X1*X2: D:=Y1*Y2: E:=d*B: F:=(X1+Y1)*(X2+Y2)-C-D: G:=A-E: Z3:=G^2: T3:=F^2: X3:=F
    *G: Y3:=(D+(2*a)*C)*(A+E)+(2*d)*C*((T1+Z1)*(T2+Z2)-A-B):
>     return X3,Y3,Z3,T3:
> end proc:
> X3,Y3,Z3,T3:=UADD_Qe_a04_opr(X1,Y1,Z1,T1,X2,Y2,Z2,T2):
> simplify([x3-X3/Z3,y3-Y3/Z3,T3-X3^2/Z3],[(C(x1,y1)),(C(x2,y2)),k+2*a, h-2*d]); #Check.
>
> UADD_Qe_a04_reg:=proc(X1,Y1,Z1,T1,X2,Y2,Z2,T2) local X3,Y3,Z3,T3,t1,t2,t3:
>     t1:=T1+Z1: t2:=T2+Z2: T3:=T1*T2: Z3:=Z1*Z2: t1:=t1*t2: t1:=t1-T3: t1:=t1-Z3: T3:=d*T3: t2:=Z3+T3: Z3:=Z
    3-T3: T3:=X1+Y1: t3:=X2+Y2: X3:=X1*X2: Y3:=Y1*Y2: T3:=T3*t3: T3:=T3-X3: T3:=T3-Y3: t1:=d*t1: t1:=t1*X3: t1:=2*
    t1: X3:=k*X3: Y3:=Y3-X3: Y3:=Y3*t2: Y3:=Y3+t1: X3:=Z3*T3: T3:=T3^2: Z3:=Z3^2:
>     return X3,Y3,Z3,T3:
> end proc:
> X3,Y3,Z3,T3:=UADD_Qe_a04_reg(X1,Y1,Z1,T1,X2,Y2,Z2,T2):
> simplify([x3-X3/Z3,y3-Y3/Z3,T3-X3^2/Z3],[(C(x1,y1)),(C(x2,y2)),k+2*a, h-2*d]); #Check.
>
```

UADD_Qe_b00.

```
> x1:=X1/Z1: y1:=Y1/Z1: T1:=X1^2/Z1: x2:=X2/Z2: y2:=Y2/Z2: T2:=X2^2/Z2:
> x3:=(x1*y2+y1*x2)/(1-d*x1^2*x2^2):
> y3:=((y1*y2+2*a*x1*x2)*(1+d*x1^2*x2^2)+2*d*x1*x2*(x1^2+x2^2))/(1-d*x1^2*x2^2)^2:
>
> UADD_Qe_b00:=proc(X1,Y1,Z1,T1,X2,Y2,Z2,T2) local X3,Y3,Z3,T3:
>     X3:=(X1*Y2+Y1*X2)*(Z1*Z2-d*T1*T2):
>     Y3:=(Z1*Z2+d*T1*T2+2*s*X1*X2)*(Y1*Y2+2*a*X1*X2+s*T1*Z2+s*Z1*T2)-s*(X1*Y2+Y1*X2)^2:
>     T3:=(X1*Y2+Y1*X2)^2:
>     Z3:=(Z1*Z2-d*T1*T2)^2:
>     return X3,Y3,Z3,T3:
> end proc:
> X3,Y3,Z3,T3:=UADD_Qe_b00(X1,Y1,Z1,T1,X2,Y2,Z2,T2):
> simplify([x3-X3/Z3,y3-Y3/Z3,T3-X3^2/Z3],[(C(x1,y1)),(C(x2,y2)),d-s^2]); #Check.
>
```

UADD_Qe_b01, $7\mathbf{M} + 3\mathbf{S} + 1\mathbf{D} + 18\mathbf{a}$, assumes $k = -2a, d = 1$.

```
> UADD_Qe_b01_reg:=proc(X1,Y1,Z1,T1,X2,Y2,Z2,T2) local X3,Y3,Z3,T3,t1,t2,t3:
>     t1:=T1+Z1: t2:=T2+Z2: T3:=T1*T2: Z3:=Z1*Z2: t1:=t1*t2: t2:=Z3+T3: Z3:=Z3-T3: t1:=t1-t2: T3:=X1+Y1: t3:=
    X2+Y2: X3:=X1*X2: Y3:=Y1*Y2: T3:=T3*t3: t3:=T3-X3: t3:=t3-Y3: T3:=t3^2: Y3:=Y3+t1: t1:=2*X3: t1:=t1+t2: t2:=k*
    X3: Y3:=Y3-t2: Y3:=Y3*t1: Y3:=Y3-T3: X3:=t3+Z3: X3:=X3^2: Z3:=Z3^2: X3:=X3-Z3: X3:=X3-T3: X3:=X3/2:
>     return X3,Y3,Z3,T3:
> end proc:
> X3,Y3,Z3,T3:=UADD_Qe_b01_reg(X1,Y1,Z1,T1,X2,Y2,Z2,T2):
> simplify([x3-X3/Z3,y3-Y3/Z3,T3-X3^2/Z3],[(C(x1,y1)),(C(x2,y2)),k+2*a,s-1,d-1]); #Check.
>
```

UADD_Qe_b02, $8\mathbf{M} + 2\mathbf{S} + 1\mathbf{D} + 14\mathbf{a}$, assumes $k = -2a, d = 1$.

```
> UADD_Qe_b02_reg:=proc(X1,Y1,Z1,T1,X2,Y2,Z2,T2) local X3,Y3,Z3,T3,t1,t2,t3:
>     t1:=T1+Z1: t2:=T2+Z2: T3:=T1*T2: Z3:=Z1*Z2: t1:=t1*t2: t2:=Z3+T3: Z3:=Z3-T3: t1:=t1-t2: T3:=X1+Y1: t3:=
    X2+Y2: X3:=X1*X2: Y3:=Y1*Y2: T3:=T3*t3: t3:=T3-X3: t3:=t3-Y3: T3:=t3^2: Y3:=Y3+t1: t1:=2*X3: t1:=t1+t2: t2:=k*
    X3: Y3:=Y3-t2: Y3:=Y3*t1: Y3:=Y3-T3: X3:=t3*Z3: Z3:=Z3^2:
>     return X3,Y3,Z3,T3:
> end proc:
> X3,Y3,Z3,T3:=UADD_Qe_b02_reg(X1,Y1,Z1,T1,X2,Y2,Z2,T2):
> simplify([x3-X3/Z3,y3-Y3/Z3,T3-X3^2/Z3],[(C(x1,y1)),(C(x2,y2)),k+2*a,s-1,d-1]); #Check.
>
```

UADD_Qe_b03, $7\mathbf{M} + 3\mathbf{S} + 5\mathbf{D} + 18\mathbf{a}$, assumes $k = -2a, d = s^2$.

```
> UADD_Qe_b03_opr:=proc(X1,Y1,Z1,T1,X2,Y2,Z2,T2) local X3,Y3,Z3,T3,A,B,C,D,E,F,G:
>     A:=Z1*Z2: B:=T1*T2: C:=X1*X2: D:=Y1*Y2: E:=d*B: F:=(X1+Y1)*(X2+Y2)-C-D: G:=A-E: Z3:=G^2: T3:=F^2: X3:=(
```

```
    (F+G)^2-T3-Z3)/2: Y3:=(A+E+(2*s)*C)*(D+(2*a)*C+s*((T1+Z1)*(T2+Z2)-A-B))-s*T3:
>     return X3,Y3,Z3,T3:
> end proc:
> X3,Y3,Z3,T3:=UADD_Qe_b03_opr(X1,Y1,Z1,T1,X2,Y2,Z2,T2):
> simplify([x3-X3/Z3,y3-Y3/Z3,T3-X3^2/Z3],[(C(x1,y1)),(C(x2,y2)),k+2*a,d-s^2]); #Check.
>
> UADD_Qe_b03_reg:=proc(X1,Y1,Z1,T1,X2,Y2,Z2,T2) local X3,Y3,Z3,T3,t1,t2,t3:
>     t1:=s*T1: t2:=s*T2: T3:=t1*t2: t1:=t1+Z1: t2:=t2+Z2: Z3:=Z1*Z2: t1:=t1*t2: t2:=Z3+T3: Z3:=Z3-T3: t1:=t1
    -t2: T3:=X1+Y1: t3:=X2+Y2: X3:=X1*X2: Y3:=Y1*Y2: T3:=T3*t3: T3:=T3-X3: t3:=T3-Y3: Y3:=Y3+t1: T3:=t3^2: t1:=2*s
    : t1:=t1*X3: t1:=t1+t2: t2:=k*X3: Y3:=Y3-t2: Y3:=Y3*t1: t1:=s*T3: Y3:=Y3-t1: X3:=t3+Z3: X3:=X3^2: Z3:=Z3^2: X3
    :=X3-Z3: X3:=X3-T3: X3:=X3/2:
>     return X3,Y3,Z3,T3:
> end proc:
> X3,Y3,Z3,T3:=UADD_Qe_b03_reg(X1,Y1,Z1,T1,X2,Y2,Z2,T2):
> simplify([x3-X3/Z3,y3-Y3/Z3,T3-X3^2/Z3],[(C(x1,y1)),(C(x2,y2)),k+2*a,d-s^2]); #Check.
>
```

UADD_Qe_b04, $8\mathbf{M} + 2\mathbf{S} + 5\mathbf{D} + 14\mathbf{a}$, assumes $k = -2a, d = s^2$.

```
> UADD_Qe_b04_opr:=proc(X1,Y1,Z1,T1,X2,Y2,Z2,T2) local X3,Y3,Z3,T3,A,B,C,D,E,F,G:
>     A:=Z1*Z2: B:=T1*T2: C:=X1*X2: D:=Y1*Y2: E:=d*B: F:=(X1+Y1)*(X2+Y2)-C-D: G:=A-E: Z3:=G^2: T3:=F^2: X3:=F
    *G: Y3:=(A+E+(2*s)*C)*(D+(2*a)*C+s*((T1+Z1)*(T2+Z2)-A-B))-s*T3:
>     return X3,Y3,Z3,T3:
> end proc:
> X3,Y3,Z3,T3:=UADD_Qe_b04_opr(X1,Y1,Z1,T1,X2,Y2,Z2,T2):
> simplify([x3-X3/Z3,y3-Y3/Z3,T3-X3^2/Z3],[(C(x1,y1)),(C(x2,y2)),k+2*a,d-s^2]); #Check.
>
> UADD_Qe_b04_reg:=proc(X1,Y1,Z1,T1,X2,Y2,Z2,T2) local X3,Y3,Z3,T3,t1,t2,t3:
>     t1:=s*T1: t2:=s*T2: T3:=t1*t2: t1:=t1+Z1: t2:=t2+Z2: Z3:=Z1*Z2: t1:=t1*t2: t2:=Z3+T3: Z3:=Z3-T3: t1:=t1
    -t2: T3:=X1+Y1: t3:=X2+Y2: X3:=X1*X2: Y3:=Y1*Y2: T3:=T3*t3: T3:=T3-X3: t3:=T3-Y3: Y3:=Y3+t1: T3:=t3^2: t1:=2*s
    : t1:=t1*X3: t1:=t1+t2: t2:=k*X3: Y3:=Y3-t2: Y3:=Y3*t1: t1:=s*T3: Y3:=Y3-t1: X3:=t3*Z3: Z3:=Z3^2:
>     return X3,Y3,Z3,T3:
> end proc:
> X3,Y3,Z3,T3:=UADD_Qe_b04_reg(X1,Y1,Z1,T1,X2,Y2,Z2,T2):
> simplify([x3-X3/Z3,y3-Y3/Z3,T3-X3^2/Z3],[(C(x1,y1)),(C(x2,y2)),k+2*a,d-s^2]); #Check.
>
```

C.3 Twisted Hessian form

```
> C:=(x,y)->(a*x^3+y^3+1-d*x*y):
> a1:=-3^3*d/(27*a-d^3): a3:=-3^8/(27*a-d^3)^2: a2:=-3^6*d^2/(27*a-d^3)^2: a4:=-3^11*d/(27*a-d^3)^3: a6:=-3^1
    5/(27*a-d^3)^4:
> W:=(u,v)->(v^2+a1*u*v+a3*v-(u^3+a2*u^2+a4*u+a6)):
> CtoW:=(x,y)->(-3^5*x/((27*a-d^3)*(d*x+3*y+3)),3^9*y/((27*a-d^3)^2*(d*x+3*y+3))):
> WtoC:=(u,v)->(81*(27*a-d^3)*u/(729*v*a^2-54*v*a*d^3+v*d^6-729*d*u*a+27*d^4*u-6561),-(27*a-d^3)^2*v/(729*v*a
    ^2-54*v*a*d^3+v*d^6-729*d*u*a+27*d^4*u-6561)):
> simplify([W(CtoW(x1,y1))],[C(x1,y1)]); #Check CtoW.
> simplify([C(WtoC(u1,v1))],[W(u1,v1)]); #Check WtoC.
> simplify([(x1,y1)-WtoC(CtoW(x1,y1))],[C(x1,y1)]); #Check CtoW(WtoC).
> simplify([(u1,v1)-CtoW(WtoC(u1,v1))],[W(u1,v1)]); #Check WtoC(CtoW).
> ut,vt:=CtoW(x1,y1): simplify([(x1/y1,1/y1)-WtoC(ut,-vt-a1*ut-a3)],[C(x1,y1)]); #Check the negation.

> # Doubling formulae.
> unassign('x1','y1'): u1,v1:=CtoW(x1,y1):
> L:=(3*u1^2+2*a2*u1+a4-a1*v1)/(2*v1+a1*u1+a3): u3:=L^2+a1*L-a2-2*u1: v3:=L*(u1-u3)-v1-a1*u3-a3:
> simplify([W(u3,v3)],[C(x1,y1)]); x3std,y3std:=WtoC(u3,v3):
>
> x3:=(x1-y1^3*x1)/(a*y1*x1^3-y1): simplify([x3std-x3],[C(x1,y1)]);
> y3:=(y1^3-a*x1^3)/(a*y1*x1^3-y1): simplify([y3std-y3],[C(x1,y1)]);

> # Addition formulae.
> unassign('x1','y1','x2','y2'): u1,v1:=CtoW(x1,y1): u2,v2:=CtoW(x2,y2):
```

```
> L:=(v2-v1)/(u2-u1): u3:=L^2+a1*L-a2-u1-u2: v3:=L*(u1-u3)-v1-a1*u3-a3:
> simplify([W(u3,v3)],[C(x1,y1),C(x2,y2)]); x3std,y3std:=WtoC(u3,v3):
>
> x3:=(y1*x2^2-x1^2*y2)/(x1*y2^2-y1^2*x2): simplify([x3std-x3],[C(x1,y1),C(x2,y2)]);
> x3:=(x1+x2-y1*y2*(x1*y2+y1*x2))/(a*x1*x2*(x1*y2+y1*x2)-y1-y2): simplify([x3std-x3],[C(x1,y1),C(x2,y2)]);
> x3:=(x1-y1^2*x2*y2)/(a*x1*y1*x2^2-y2): simplify([x3std-x3],[C(x1,y1),C(x2,y2)]);
> y3:=(x1*y1-x2*y2)/(x1*y2^2-y1^2*x2): simplify([y3std-y3],[C(x1,y1),C(x2,y2)]);
> y3:=(y1*y2*(y1+y2)-a*x1*x2*(x1+x2))/(a*x1*x2*(x1*y2+y1*x2)-y1-y2): simplify([y3std-y3],[C(x1,y1),C(x2,y2)])
    ;
> y3:=(y1*y2^2-a*x1^2*x2)/(a*x1*y1*x2^2-y2): simplify([y3std-y3],[C(x1,y1),C(x2,y2)]);
```

DBL_H_a00.

```
> x1:=X1/Z1: y1:=Y1/Z1:
> x3:=(x1-y1^3*x1)/(a*y1*x1^3-y1):
> y3:=(y1^3-a*x1^3)/(a*y1*x1^3-y1):
>
> DBL_H_a00:=proc(X1,Y1,Z1) local X3,Y3,Z3:
>     X3:=X1*(Z1^3-Y1^3):
>     Y3:=Z1*(Y1^3-a*X1^3):
>     Z3:=Y1*(a*X1^3-Z1^3):
>     return X3,Y3,Z3:
> end proc:
> X3,Y3,Z3:=DBL_H_a00(X1,Y1,Z1):
> simplify([x3-X3/Z3,y3-Y3/Z3],[(C(x1,y1))]); #Check.
>
```

DBL_H_a01, $6\mathbf{M} + 3\mathbf{S} + 1\mathbf{D} + 3\mathbf{a}$.

```
> DBL_H_a01_opr:=proc(X1,Y1,Z1) local X3,Y3,Z3,A,B,C,D,E,F,G:
>     A:=X1^2: B:=Y1^2: C:=Z1^2: D:=X1*A: E:=Y1*B: F:=Z1*C: G:=a*D: X3:=X1*(E-F): Y3:=Z1*(G-E): Z3:=Y1*(F-G):
>     return X3,Y3,Z3:
> end proc:
> X3,Y3,Z3:=DBL_H_a01_opr(X1,Y1,Z1):
> simplify([x3-X3/Z3,y3-Y3/Z3],[(C(x1,y1))]); #Check.
>
> DBL_H_a01_reg:=proc(X1,Y1,Z1) local X3,Y3,Z3,t1,t2,t3,t4,t:
>     t1:=Z1^2: t1:=Z1*t1: t2:=Y1^2: t2:=Y1*t2: t3:=X1^2: t3:=X1*t3: t3:=a*t3: t4:=t2-t1: X3:=X1*t4: t2:=t3-t
    2: Z3:=Z1*t2: t1:=t1-t3: Y3:=Y1*t1: t:=Y3: Y3:=Z3: Z3:=t:
>     return X3,Y3,Z3:
> end proc:
> X3,Y3,Z3:=DBL_H_a01_reg(X1,Y1,Z1):
> simplify([x3-X3/Z3,y3-Y3/Z3],[(C(x1,y1))]); #Check.
>
```

DBL_H_a02, $3\mathbf{M} + 6\mathbf{S} + 18\mathbf{a}$, assumes $a = 1$.

```
> DBL_H_a02_opr:=proc(X1,Y1,Z1) local X3,Y3,Z3,A,B,C,D,E,F,G,H,J:
>     A:=X1^2: B:=Y1^2: C:=Z1^2: D:=A+B: E:=A+C: F:=B+C: G:=((X1+Y1)^2-D)/2: H:=((X1+Z1)^2-E)/2: J:=((Y1+Z1)^
    2-F)/2: X3:=(H-G)*(F+J): Y3:=(J-H)*(D+G): Z3:=(G-J)*(E+H):
>     return X3,Y3,Z3:
> end proc:
> X3,Y3,Z3:=DBL_H_a02_opr(X1,Y1,Z1):
> simplify([x3-X3/Z3,y3-Y3/Z3],[(C(x1,y1)),a-1]); #Check.
>
> DBL_H_a02_reg:=proc(X1,Y1,Z1) local X3,Y3,Z3,t1,t2,t3,t4,t5,t6:
>     t1:=X1^2: t2:=Y1^2: t3:=Z1^2: t4:=t1+t2: t5:=t1+t3: t6:=t2+t3: t1:=X1+Y1: t1:=t1^2: t1:=t1-t4: t1:=t1/2
    : t2:=X1+Z1: t2:=t2^2: t2:=t2-t5: t2:=t2/2: t3:=Y1+Z1: t3:=t3^2: t3:=t3-t6: t3:=t3/2: X3:=t2-t1: Y3:=t3-t2: Z3
    :=t1-t3: t3:=t6+t3: X3:=X3*t3: t1:=t4+t1: Y3:=Y3*t1: t2:=t5+t2: Z3:=Z3*t2:
>     return X3,Y3,Z3:
> end proc:
> X3,Y3,Z3:=DBL_H_a02_reg(X1,Y1,Z1):
> simplify([x3-X3/Z3,y3-Y3/Z3],[(C(x1,y1)),a-1]); #Check.
>
```

DBL_H_a03, $7\mathbf{M} + 1\mathbf{S} + 8\mathbf{a}$, assumes $a = 1$.

```
> DBL_H_a03_opr:=proc(X1,Y1,Z1) local X3,Y3,Z3,A,B,C:
>     A:=Y1^2: B:=(Z1-Y1)*(A+(Y1+Z1)*Z1): C:=(Y1-X1)*(X1*(X1+Y1)+A): X3:=X1*B: Y3:=Z1*C: Z3:=Y1*(-B-C):
>     return X3,Y3,Z3:
> end proc:
> X3,Y3,Z3:=DBL_H_a03_opr(X1,Y1,Z1):
> simplify([x3-X3/Z3,y3-Y3/Z3],[(C(x1,y1)),a-1]); #Check.
>
> DBL_H_a03_reg:=proc(X1,Y1,Z1) local X3,Y3,Z3,t1,t2,t3,t:
>     t1:=Y1^2: t2:=Y1+Z1: t2:=t2*Z1: t2:=t1+t2: t3:=Z1-Y1: t2:=t3*t2: t3:=X1+Y1: t3:=X1*t3: t3:=t3+t1: t1:=Y
    1-X1: t1:=t1*t3: X3:=X1*t2: Z3:=Z1*t1: t1:=-t1: t1:=t1-t2: Y3:=Y1*t1: t:=Y3: Y3:=Z3: Z3:=t:
>     return X3,Y3,Z3:
> end proc:
> X3,Y3,Z3:=DBL_H_a03_reg(X1,Y1,Z1):
> simplify([x3-X3/Z3,y3-Y3/Z3],[(C(x1,y1)),a-1]); #Check.
>
```

ADD_H_a00.

```
> x1:=X1/Z1: y1:=Y1/Z1: x2:=X2/Z2: y2:=Y2/Z2:
> x3:=(x1-y1^2*x2*y2)/(a*x1*y1*x2^2-y2):
> y3:=(y1*y2^2-a*x1^2*x2)/(a*x1*y1*x2^2-y2):
>
> ADD_H_a00:=proc(X1,Y1,Z1,X2,Y2,Z2) local X3,Y3,Z3:
>     X3:=X1^2*Y2*Z2-Y1*Z1*X2^2:
>     Y3:=Z1^2*X2*Y2-X1*Y1*Z2^2:
>     Z3:=Y1^2*X2*Z2-X1*Z1*Y2^2:
>     return X3,Y3,Z3:
> end proc:
> X3,Y3,Z3:=ADD_H_a00(X1,Y1,Z1,X2,Y2,Z2):
> simplify([x3-X3/Z3,y3-Y3/Z3],[(C(x1,y1)),(C(x2,y2))]); #Check.
>
```

ADD_H_a01, $12\mathbf{M} + 3\mathbf{a}$.

```
> ADD_H_a01_opr:=proc(X1,Y1,Z1,X2,Y2,Z2) local X3,Y3,Z3,A,B,C,D,E,F:
>     A:=X1*Z2: B:=Y1*Z2: C:=Z1*X2: D:=Z1*Y2: E:=X1*Y2: F:=Y1*X2: X3:=E*A-F*C: Y3:=C*D-A*B: Z3:=F*B-E*D:
>     return X3,Y3,Z3:
> end proc:
> X3,Y3,Z3:=ADD_H_a01_opr(X1,Y1,Z1,X2,Y2,Z2):
> simplify([x3-X3/Z3,y3-Y3/Z3],[(C(x1,y1)),(C(x2,y2))]); #Check.
>
> ADD_H_a01_reg:=proc(X1,Y1,Z1,X2,Y2,Z2) local X3,Y3,Z3,t1,t2,t3,t4,t5,t6:
>     t1:=X1*Z2: t2:=Y1*Z2: t3:=Z1*X2: t4:=Z1*Y2: t5:=X1*Y2: t6:=Y1*X2: X3:=t5*t1: Y3:=t3*t4: Z3:=t6*t2: t3:=
    t6*t3: X3:=X3-t3: t1:=t1*t2: Y3:=Y3-t1: t4:=t5*t4: Z3:=Z3-t4:
>     return X3,Y3,Z3:
> end proc:
> X3,Y3,Z3:=ADD_H_a01_reg(X1,Y1,Z1,X2,Y2,Z2):
> simplify([x3-X3/Z3,y3-Y3/Z3],[(C(x1,y1)),(C(x2,y2))]); #Check.
>
```

ADD_H_a02, $11\mathbf{M} + 17\mathbf{a}$.

```
> ADD_H_a02_opr:=proc(X1,Y1,Z1,X2,Y2,Z2) local X3,Y3,Z3,A,B,C,D,E,F,G,H,J,K:
>     A:=X1*Z2: B:=Y1*Z2: C:=Z1*X2: D:=Z1*Y2: E:=X1*Y2: F:=Y1*X2: G:=(C+E)*(A-F): H:=(C-E)*(A+F): J:=(C+B)*(A
    -D): K:=(C-B)*(A+D): X3:=G-H: Y3:=K-J: Z3:=J+K-G-H-2*(E-B)*(F+D):
>     return X3,Y3,Z3:
> end proc:
> X3,Y3,Z3:=ADD_H_a02_opr(X1,Y1,Z1,X2,Y2,Z2):
> simplify([x3-X3/Z3,y3-Y3/Z3],[(C(x1,y1)),(C(x2,y2))]); #Check.
>
> ADD_H_a02_reg:=proc(X1,Y1,Z1,X2,Y2,Z2) local X3,Y3,Z3,t1,t2,t3,t4,t5,t6:
>     t1:=X1*Z2: t2:=Y1*Z2: t3:=Z1*X2: t4:=Z1*Y2: t5:=X1*Y2: t6:=Y1*X2: X3:=t3+t5: Y3:=t1-t6: X3:=X3*Y3: Y3:=
```

```
    t5-t2: Z3:=t6+t4: Z3:=Y3*Z3: Z3:=2*Z3: Z3:=X3+Z3: t5:=t3-t5: t6:=t1+t6: t5:=t5*t6: X3:=X3-t5: t6:=t3+t2: Y3:=t
    1-t4: Y3:=t6*Y3: t2:=t3-t2: t1:=t1+t4: t1:=t2*t1: t2:=t1-t5: t2:=Y3+t2: Z3:=t2-Z3: Y3:=t1-Y3:
>     return X3,Y3,Z3:
> end proc:
> X3,Y3,Z3:=ADD_H_a02_reg(X1,Y1,Z1,X2,Y2,Z2):
> simplify([x3-X3/Z3,y3-Y3/Z3],[(C(x1,y1)),(C(x2,y2))]); #Check.
>
```

UADD_H_a00.

```
> x1:=X1/Z1: y1:=Y1/Z1: x2:=X2/Z2: y2:=Y2/Z2:
> x3:=(x1-y1^2*x2*y2)/(a*x1*y1*x2^2-y2):
> y3:=(y1*y2^2-a*x1^2*x2)/(a*x1*y1*x2^2-y2):
>
> UADD_H_a00:=proc(X1,Y1,Z1,X2,Y2,Z2) local X3,Y3,Z3:
>     X3:=X1*Z1*Z2^2-Y1^2*X2*Y2:
>     Y3:=Y1*Z1*Y2^2-a*X1^2*X2*Z2:
>     Z3:=a*X1*Y1*X2^2-Z1^2*Z2*Y2:
>     return X3,Y3,Z3:
> end proc:
> X3,Y3,Z3:=UADD_H_a00(X1,Y1,Z1,X2,Y2,Z2):
> simplify([x3-X3/Z3,y3-Y3/Z3],[(C(x1,y1)),(C(x2,y2))]); #Check.
>
```

UADD_H_a01, $12\mathbf{M} + 3\mathbf{a}$.

```
> UADD_H_a01_opr:=proc(X1,Y1,Z1,X2,Y2,Z2) local X3,Y3,Z3,A,B,C,D,E,F:
>     A:=X1*Z2: B:=Z1*Z2: C:=Y1*X2: D:=Y1*Y2: E:=Z1*Y2: F:=a*X1*X2: X3:=A*B-C*D: Y3:=D*E-F*A: Z3:=F*C-B*E:
>     return X3,Y3,Z3:
> end proc:
> X3,Y3,Z3:=UADD_H_a01_opr(X1,Y1,Z1,X2,Y2,Z2):
> simplify([x3-X3/Z3,y3-Y3/Z3],[(C(x1,y1)),(C(x2,y2))]); #Check.
>
> UADD_H_a01_reg:=proc(X1,Y1,Z1,X2,Y2,Z2) local X3,Y3,Z3,t1,t2,t3,t4,t5,t6:
>     t1:=X1*Z2: t2:=Z1*Z2: t3:=Y1*X2: t4:=Y1*Y2: t5:=Z1*Y2: t6:=X1*X2: t6:=a*t6: X3:=t1*t2: Y3:=t4*t5: Z3:=t
    6*t3: t3:=t3*t4: t1:=t6*t1: t2:=t2*t5: X3:=X3-t3: Y3:=Y3-t1: Z3:=Z3-t2:
>     return X3,Y3,Z3:
> end proc:
> X3,Y3,Z3:=UADD_H_a01_reg(X1,Y1,Z1,X2,Y2,Z2):
> simplify([x3-X3/Z3,y3-Y3/Z3],[(C(x1,y1)),(C(x2,y2))]); #Check.
>
```

UADD_H_a02, $11\mathbf{M} + 17\mathbf{a}$.

```
> UADD_H_a02_opr:=proc(X1,Y1,Z1,X2,Y2,Z2) local X3,Y3,Z3,A,B,C,D,E,F,G,H,J,K:
>     A:=X1*Z2: B:=Z1*Z2: C:=Y1*X2: D:=Y1*Y2: E:=Z1*Y2: F:=a*X1*X2: G:=(D+B)*(A-C): H:=(D-B)*(A+C): J:=(D+F)*
    (A-E): K:=(D-F)*(A+E): X3:=G-H: Y3:=K-J: Z3:=J+K-G-H-2*(B-F)*(C+E):
>     return X3,Y3,Z3:
> end proc:
> X3,Y3,Z3:=UADD_H_a02_opr(X1,Y1,Z1,X2,Y2,Z2):
> simplify([x3-X3/Z3,y3-Y3/Z3],[(C(x1,y1)),(C(x2,y2))]); #Check.
>
> UADD_H_a02_reg:=proc(X1,Y1,Z1,X2,Y2,Z2) local X3,Y3,Z3,t1,t2,t3,t4,t5,t6:
>     t1:=X1*Z2: t2:=Z1*Z2: t3:=Y1*X2: t4:=Y1*Y2: Z3:=Z1*Y2: X3:=X1*X2: X3:=a*X3: Y3:=t2-X3: Y3:=2*Y3: t5:=t3
    +Z3: Y3:=Y3*t5: t5:=t4+t2: t2:=t4-t2: t6:=t1-t3: t3:=t1+t3: t2:=t2*t3: t3:=t5*t6: t5:=t4+X3: t4:=t4-X3: X3:=t3
    -t2: Y3:=Y3+t2: t2:=t1-Z3: t2:=t5*t2: Y3:=Y3-t2: t1:=t1+Z3: t1:=t4*t1: Y3:=t1-Y3: Z3:=Y3-t3: Y3:=t1-t2:
>     return X3,Y3,Z3:
> end proc:
> X3,Y3,Z3:=UADD_H_a02_reg(X1,Y1,Z1,X2,Y2,Z2):
> simplify([x3-X3/Z3,y3-Y3/Z3],[(C(x1,y1)),(C(x2,y2))]); #Check.
>
```

DBL_He_a00.

```
> x1:=X1/Z1: y1:=Y1/Z1: R1:=X1*Y1/Z1: S1:=X1^2/Z1: T1:=Y1^2/Z1:
> x3:=(x1-y1^3*x1)/(a*y1*x1^3-y1):
> y3:=(y1^3-a*x1^3)/(a*y1*x1^3-y1):
>
> DBL_He_a00:=proc(X1,Y1,Z1,R1,S1,T1) local X3,Y3,Z3,R3,S3,T3:
>     X3:=(X1*Z1-R1*T1)*(a*R1*S1-Y1*Z1):
>     Y3:=(Y1*T1-a*X1*S1)*(a*R1*S1-Y1*Z1):
>     R3:=(X1*Z1-R1*T1)*(Y1*T1-a*X1*S1):
>     S3:=(X1*Z1-R1*T1)^2:
>     T3:=(Y1*T1-a*X1*S1)^2:
>     Z3:=(a*R1*S1-Y1*Z1)^2:
>     return X3,Y3,Z3,R3,S3,T3:
> end proc:
> X3,Y3,Z3,R3,S3,T3:=DBL_He_a00(X1,Y1,Z1,R1,S1,T1):
> simplify([x3-X3/Z3,y3-Y3/Z3,R3-X3*Y3/Z3,S3-X3^2/Z3,T3-Y3^2/Z3],[(C(x1,y1))]); #Check.
>
```

DBL_He_a01, $9\mathbf{M} + 3\mathbf{S} + 1\mathbf{D} + 3\mathbf{a}$.

```
> DBL_He_a01_opr:=proc(X1,Y1,Z1,R1,S1,T1) local X3,Y3,Z3,R3,S3,T3,A,B,C,D:
>     A:=a*S1: B:=X1*Z1-T1*R1: C:=Y1*T1-A*X1: D:=R1*A-Z1*Y1: X3:=B*D: Y3:=C*D: R3:=B*C: S3:=B^2: T3:=C^2: Z3:
    =D^2:
>     return X3,Y3,Z3,R3,S3,T3:
> end proc:
> X3,Y3,Z3,R3,S3,T3:=DBL_He_a01_opr(X1,Y1,Z1,R1,S1,T1):
> simplify([x3-X3/Z3,y3-Y3/Z3,R3-X3*Y3/Z3,S3-X3^2/Z3,T3-Y3^2/Z3],[(C(x1,y1))]); #Check.
>
> DBL_He_a01_reg:=proc(X1,Y1,Z1,R1,S1,T1) local X3,Y3,Z3,R3,S3,T3,t1:
>     t1:=X1*Z1: S3:=a*S1: X3:=S3*X1: S3:=R1*S3: R3:=T1*R1: T3:=Y1*T1: Y3:=Z1*Y1: R3:=t1-R3: t1:=T3-X3: Y3:=S
    3-Y3: S3:=R3^2: T3:=t1^2: Z3:=Y3^2: X3:=R3*Y3: Y3:=t1*Y3: R3:=R3*t1:
>     return X3,Y3,Z3,R3,S3,T3:
> end proc:
> X3,Y3,Z3,R3,S3,T3:=DBL_He_a01_reg(X1,Y1,Z1,R1,S1,T1):
> simplify([x3-X3/Z3,y3-Y3/Z3,R3-X3*Y3/Z3,S3-X3^2/Z3,T3-Y3^2/Z3],[(C(x1,y1))]); #Check.
>
```

DBL_He_a02, $5\mathbf{M} + 6\mathbf{S} + 1\mathbf{D} + 29\mathbf{a}$.

```
> DBL_He_a02_opr:=proc(X1,Y1,Z1,R1,S1,T1) local X3,Y3,Z3,R3,S3,T3,A,B,C,D,E,F,G,H:
>     A:=a*S1: B:=(T1+Z1)*(X1-R1): C:=(T1-Z1)*(X1+R1): D:=(T1+A)*(X1-Y1): E:=(T1-A)*(X1+Y1): F:=B-C: G:=E-D:
    H:=D+E-B-C-2*(Z1-A)*(R1+Y1): S3:=F^2: T3:=G^2: Z3:=H^2: X3:=((F+H)^2-S3-Z3)/2: Y3:=((G+H)^2-T3-Z3)/2: R3:=((F+
    G)^2-S3-T3)/2:
>     return X3,Y3,Z3,R3,S3,T3:
> end proc:
> X3,Y3,Z3,R3,S3,T3:=DBL_He_a02_opr(X1,Y1,Z1,R1,S1,T1):
> simplify([x3-X3/Z3,y3-Y3/Z3,R3-X3*Y3/Z3,S3-X3^2/Z3,T3-Y3^2/Z3],[(C(x1,y1))]); #Check.
>
> DBL_He_a02_reg:=proc(X1,Y1,Z1,R1,S1,T1) local X3,Y3,Z3,R3,S3,T3,t1,t2,t3:
>     S3:=a*S1: t1:=Z1-S3: t1:=2*t1: t2:=R1+Y1: t1:=t1*t2: t2:=X1-Y1: Y3:=X1+Y1: t3:=T1+S3: t2:=t3*t2: t1:=t2
    -t1: t3:=T1-S3: t3:=t3*Y3: t1:=t1+t3: t2:=t3-t2: t3:=T1+Z1: Y3:=X1-R1: t3:=t3*Y3: t1:=t1-t3: Z3:=T1-Z1: X3:=X1
    +R1: X3:=Z3*X3: t3:=t3-X3: t1:=t1-X3: S3:=t3^2: T3:=t2^2: Z3:=t1^2: X3:=t3+t1: X3:=X3^2: X3:=X3-S3: X3:=X3-Z3:
     X3:=X3/2: Y3:=t2+t1: Y3:=Y3^2: Y3:=Y3-T3: Y3:=Y3-Z3: Y3:=Y3/2: R3:=t3+t2: R3:=R3^2: R3:=R3-S3: R3:=R3-T3: R3:
    =R3/2:
>     return X3,Y3,Z3,R3,S3,T3:
> end proc:
> X3,Y3,Z3,R3,S3,T3:=DBL_He_a02_reg(X1,Y1,Z1,R1,S1,T1):
> simplify([x3-X3/Z3,y3-Y3/Z3,R3-X3*Y3/Z3,S3-X3^2/Z3,T3-Y3^2/Z3],[(C(x1,y1))]); #Check.
>
```

ADD_He_a00.

```
> x1:=X1/Z1: y1:=Y1/Z1: R1:=X1*Y1/Z1: S1:=X1^2/Z1: T1:=Y1^2/Z1: x2:=X2/Z2: y2:=Y2/Z2: R2:=X2*Y2/Z2: S2:=X2^2/
    Z2: T2:=Y2^2/Z2:
```

```
> x3:=(x1-y1^2*x2*y2)/(a*x1*y1*x2^2-y2):
> y3:=(y1*y2^2-a*x1^2*x2)/(a*x1*y1*x2^2-y2):
>
> ADD_He_a00:=proc(X1,Y1,Z1,R1,S1,T1,X2,Y2,Z2,R2,S2,T2) local X3,Y3,Z3,R3,S3,T3:
>     X3:=(S1*Y2-Y1*S2)*(T1*X2-X1*T2):
>     Y3:=(Z1*R2-R1*Z2)*(T1*X2-X1*T2):
>     R3:=(S1*Y2-Y1*S2)*(Z1*R2-R1*Z2):
>     S3:=(S1*Y2-Y1*S2)^2:
>     T3:=(Z1*R2-R1*Z2)^2:
>     Z3:=(T1*X2-X1*T2)^2:
>     return X3,Y3,Z3,R3,S3,T3:
> end proc:
> X3,Y3,Z3,R3,S3,T3:=ADD_He_a00(X1,Y1,Z1,R1,S1,T1,X2,Y2,Z2,R2,S2,T2):
> simplify([x3-X3/Z3,y3-Y3/Z3,R3-X3*Y3/Z3,S3-X3^2/Z3,T3-Y3^2/Z3],[(C(x1,y1)),(C(x2,y2))]); #Check.
>
```

ADD_He_a01, $9\mathbf{M} + 3\mathbf{S} + 3\mathbf{a}$.

```
> ADD_He_a01_opr:=proc(X1,Y1,Z1,R1,S1,T1,X2,Y2,Z2,R2,S2,T2) local X3,Y3,Z3,R3,S3,T3,A,B,C:
>     A:=S1*Y2-Y1*S2: B:=Z1*R2-R1*Z2: C:=T1*X2-X1*T2: X3:=A*C: Y3:=B*C: R3:=A*B: S3:=A^2: T3:=B^2: Z3:=C^2:
>     return X3,Y3,Z3,R3,S3,T3:
> end proc:
> X3,Y3,Z3,R3,S3,T3:=ADD_He_a01_opr(X1,Y1,Z1,R1,S1,T1,X2,Y2,Z2,R2,S2,T2):
> simplify([x3-X3/Z3,y3-Y3/Z3,R3-X3*Y3/Z3,S3-X3^2/Z3,T3-Y3^2/Z3],[(C(x1,y1)),(C(x2,y2))]); #Check.
>
> ADD_He_a01_reg:=proc(X1,Y1,Z1,R1,S1,T1,X2,Y2,Z2,R2,S2,T2) local X3,Y3,Z3,R3,S3,T3,t1:
>     t1:=R1*Z2: Z3:=Z1*R2: Z3:=Z3-t1: t1:=Y1*S2: S3:=S1*Y2: S3:=S3-t1: t1:=T1*X2: X3:=X1*T2: t1:=t1-X3: X3:=
    S3*t1: Y3:=Z3*t1: R3:=S3*Z3: S3:=S3^2: T3:=Z3^2: Z3:=t1^2:
>     return X3,Y3,Z3,R3,S3,T3:
> end proc:
> X3,Y3,Z3,R3,S3,T3:=ADD_He_a01_reg(X1,Y1,Z1,R1,S1,T1,X2,Y2,Z2,R2,S2,T2):
> simplify([x3-X3/Z3,y3-Y3/Z3,R3-X3*Y3/Z3,S3-X3^2/Z3,T3-Y3^2/Z3],[(C(x1,y1)),(C(x2,y2))]); #Check.
>
```

ADD_He_a02, $6\mathbf{M} + 6\mathbf{S} + 15\mathbf{a}$.

```
> ADD_He_a02_opr:=proc(X1,Y1,Z1,R1,S1,T1,X2,Y2,Z2,R2,S2,T2) local X3,Y3,Z3,R3,S3,T3,A,B,C:
>     A:=S1*Y2-Y1*S2: B:=Z1*R2-R1*Z2: C:=T1*X2-X1*T2: S3:=A^2: T3:=B^2: Z3:=C^2: X3:=((A+C)^2-S3-Z3)/2: Y3:=(
    (B+C)^2-T3-Z3)/2: R3:=((A+B)^2-S3-T3)/2:
>     return X3,Y3,Z3,R3,S3,T3:
> end proc:
> X3,Y3,Z3,R3,S3,T3:=ADD_He_a02_opr(X1,Y1,Z1,R1,S1,T1,X2,Y2,Z2,R2,S2,T2):
> simplify([x3-X3/Z3,y3-Y3/Z3,R3-X3*Y3/Z3,S3-X3^2/Z3,T3-Y3^2/Z3],[(C(x1,y1)),(C(x2,y2))]); #Check.
>
> ADD_He_a02_reg:=proc(X1,Y1,Z1,R1,S1,T1,X2,Y2,Z2,R2,S2,T2) local X3,Y3,Z3,R3,S3,T3,t1:
>     t1:=R1*Z2: R3:=Z1*R2: R3:=R3-t1: t1:=X1*T2: X3:=T1*X2: X3:=X3-t1: t1:=Y1*S2: Y3:=S1*Y2: t1:=Y3-t1: S3:=
    t1^2: T3:=R3^2: Z3:=X3^2: Y3:=R3+X3: Y3:=Y3^2: Y3:=Y3-T3: Y3:=Y3-Z3: Y3:=Y3/2: X3:=t1+X3: X3:=X3^2: X3:=X3-S3:
     X3:=X3-Z3: X3:=X3/2: R3:=t1+R3: R3:=R3^2: R3:=R3-S3: R3:=R3-T3: R3:=R3/2:
>     return X3,Y3,Z3,R3,S3,T3:
> end proc:
> X3,Y3,Z3,R3,S3,T3:=ADD_He_a02_reg(X1,Y1,Z1,R1,S1,T1,X2,Y2,Z2,R2,S2,T2):
> simplify([x3-X3/Z3,y3-Y3/Z3,R3-X3*Y3/Z3,S3-X3^2/Z3,T3-Y3^2/Z3],[(C(x1,y1)),(C(x2,y2))]); #Check.
>
```

UADD_He_a00.

```
> x1:=X1/Z1: y1:=Y1/Z1: R1:=X1*Y1/Z1: S1:=X1^2/Z1: T1:=Y1^2/Z1: x2:=X2/Z2: y2:=Y2/Z2: R2:=X2*Y2/Z2: S2:=X2^2/
    Z2: T2:=Y2^2/Z2:
> x3:=(x1-y1^2*x2*y2)/(a*x1*y1*x2^2-y2):
> y3:=(y1*y2^2-a*x1^2*x2)/(a*x1*y1*x2^2-y2):
>
> UADD_He_a00:=proc(X1,Y1,Z1,R1,S1,T1,X2,Y2,Z2,R2,S2,T2) local X3,Y3,Z3,R3,S3,T3:
>     X3:=(X1*Z2-T1*R2)*(a*R1*S2-Z1*Y2):
```

```
>     Y3:=(Y1*T2-a*S1*X2)*(a*R1*S2-Z1*Y2):
>     R3:=(X1*Z2-T1*R2)*(Y1*T2-a*S1*X2):
>     S3:=(X1*Z2-T1*R2)^2:
>     T3:=(Y1*T2-a*S1*X2)^2:
>     Z3:=(a*R1*S2-Z1*Y2)^2:
>     return X3,Y3,Z3,R3,S3,T3:
> end proc:
> X3,Y3,Z3,R3,S3,T3:=UADD_He_a00(X1,Y1,Z1,R1,S1,T1,X2,Y2,Z2,R2,S2,T2):
> simplify([x3-X3/Z3,y3-Y3/Z3,R3-X3*Y3/Z3,S3-X3^2/Z3,T3-Y3^2/Z3],[(C(x1,y1)),(C(x2,y2))]); #Check.
>
```

UADD_He_a01, $9\mathbf{M} + 3\mathbf{S} + 2\mathbf{D} + 3\mathbf{a}$.

```
> UADD_He_a01_opr:=proc(X1,Y1,Z1,R1,S1,T1,X2,Y2,Z2,R2,S2,T2) local X3,Y3,Z3,R3,S3,T3,A,B,C:
>     A:=X1*Z2-T1*R2: B:=Y1*T2-a*S1*X2: C:=a*R1*S2-Z1*Y2: X3:=A*C: Y3:=B*C: R3:=A*B: S3:=A^2: T3:=B^2: Z3:=C^
    2:
>     return X3,Y3,Z3,R3,S3,T3:
> end proc:
> X3,Y3,Z3,R3,S3,T3:=UADD_He_a01_opr(X1,Y1,Z1,R1,S1,T1,X2,Y2,Z2,R2,S2,T2):
> simplify([x3-X3/Z3,y3-Y3/Z3,R3-X3*Y3/Z3,S3-X3^2/Z3,T3-Y3^2/Z3],[(C(x1,y1)),(C(x2,y2))]); #Check.
>
> UADD_He_a01_reg:=proc(X1,Y1,Z1,R1,S1,T1,X2,Y2,Z2,R2,S2,T2) local X3,Y3,Z3,R3,S3,T3,t1:
>     t1:=X1*Z2: Z3:=Z1*Y2: Y3:=Y1*T2: T3:=T1*R2: R3:=R1*S2: S3:=S1*X2: S3:=a*S3: R3:=a*R3: t1:=t1-T3: Y3:=Y3
    -S3: Z3:=R3-Z3: X3:=t1*Z3: R3:=t1*Y3: S3:=t1^2: T3:=Y3^2: Y3:=Y3*Z3: Z3:=Z3^2:
>     return X3,Y3,Z3,R3,S3,T3:
> end proc:
> X3,Y3,Z3,R3,S3,T3:=UADD_He_a01_reg(X1,Y1,Z1,R1,S1,T1,X2,Y2,Z2,R2,S2,T2):
> simplify([x3-X3/Z3,y3-Y3/Z3,R3-X3*Y3/Z3,S3-X3^2/Z3,T3-Y3^2/Z3],[(C(x1,y1)),(C(x2,y2))]); #Check.
>
```

UADD_He_a02, $6\mathbf{M} + 6\mathbf{S} + 2\mathbf{D} + 15\mathbf{a}$.

```
> UADD_He_a02_opr:=proc(X1,Y1,Z1,R1,S1,T1,X2,Y2,Z2,R2,S2,T2) local X3,Y3,Z3,R3,S3,T3,A,B,C:
>     A:=X1*Z2-T1*R2: B:=Y1*T2-a*S1*X2: C:=a*R1*S2-Z1*Y2: S3:=A^2: T3:=B^2: Z3:=C^2: X3:=((A+C)^2-S3-Z3)/2: Y
    3:=((B+C)^2-T3-Z3)/2: R3:=((A+B)^2-S3-T3)/2:
>     return X3,Y3,Z3,R3,S3,T3:
> end proc:
> X3,Y3,Z3,R3,S3,T3:=UADD_He_a02_opr(X1,Y1,Z1,R1,S1,T1,X2,Y2,Z2,R2,S2,T2):
> simplify([x3-X3/Z3,y3-Y3/Z3,R3-X3*Y3/Z3,S3-X3^2/Z3,T3-Y3^2/Z3],[(C(x1,y1)),(C(x2,y2))]); #Check.
>
> UADD_He_a02_reg:=proc(X1,Y1,Z1,R1,S1,T1,X2,Y2,Z2,R2,S2,T2) local X3,Y3,Z3,R3,S3,T3,t1:
>     t1:=X1*Z2: Z3:=Z1*Y2: Y3:=Y1*T2: T3:=T1*R2: R3:=R1*S2: S3:=S1*X2: S3:=a*S3: R3:=a*R3: t1:=t1-T3: Y3:=Y3
    -S3: X3:=R3-Z3: S3:=t1^2: T3:=Y3^2: Z3:=X3^2: R3:=t1+Y3: R3:=R3^2: R3:=R3-S3: R3:=R3-T3: R3:=R3/2: Y3:=Y3+X3:
    Y3:=Y3^2: Y3:=Y3-T3: Y3:=Y3-Z3: Y3:=Y3/2: X3:=t1+X3: X3:=X3^2: X3:=X3-S3: X3:=X3-Z3: X3:=X3/2:
>     return X3,Y3,Z3,R3,S3,T3:
> end proc:
> X3,Y3,Z3,R3,S3,T3:=UADD_He_a02_reg(X1,Y1,Z1,R1,S1,T1,X2,Y2,Z2,R2,S2,T2):
> simplify([x3-X3/Z3,y3-Y3/Z3,R3-X3*Y3/Z3,S3-X3^2/Z3,T3-Y3^2/Z3],[(C(x1,y1)),(C(x2,y2))]); #Check.
>
```

C.4 Twisted Edwards form

```
> C:=(x,y)->(a*x^2+y^2-(1+d*x^2*y^2)):
> a1:=0: a3:=0: a6:=0: a2:=2*(a+d): a4:=(a-d)^2:
> W:=(u,v)->(v^2+a1*u*v+a3*v-(u^3+a2*u^2+a4*u+a6)):
> CtoW:=(x,y)->((1+y)^2*(1-d*x^2)/x^2,2*(2-(a+d)*x^2+2*(1-d*x^2)*y)/x^3):
> WtoC:=(u,v)->(2*u/v,(u-a+d)/(u+a-d)):
> simplify([W(CtoW(x1,y1))],[C(x1,y1)]); #Check CtoW.
> simplify([C(WtoC(u1,v1))],[W(u1,v1)]); #Check WtoC.
> simplify([(x1,y1)-WtoC(CtoW(x1,y1))],[C(x1,y1)]); #Check CtoW(WtoC).
> simplify([(u1,v1)-CtoW(WtoC(u1,v1))],[W(u1,v1)]); #Check WtoC(CtoW).
> ut,vt:=CtoW(x1,y1): simplify([(-x1,y1)-WtoC(ut,-vt-a1*ut-a3)],[C(x1,y1)]); #Check the negation.
```

```
> # Doubling formulae.
> unassign('x1','y1'): u1,v1:=CtoW(x1,y1):
> L:=(3*u1^2+2*a2*u1+a4-a1*v1)/(2*v1+a1*u1+a3): u3:=L^2+a1*L-a2-2*u1: v3:=L*(u1-u3)-v1-a1*u3-a3:
> simplify([W(u3,v3)],[C(x1,y1)]); x3std,y3std:=WtoC(u3,v3):
>
> x3:=2*x1*y1/(y1^2+a*x1^2): simplify([x3std-x3],[C(x1,y1)]);
> x3:=2*x1*y1/(1+d*x1^2*y1^2): simplify([x3std-x3],[C(x1,y1)]);
> y3:=(y1^2-a*x1^2)/(2-y1^2-a*x1^2): simplify([y3std-y3],[C(x1,y1)]);
> y3:=(y1^2-a*x1^2)/(1-d*x1^2*y1^2): simplify([y3std-y3],[C(x1,y1)]);

> # Addition formulae.
> unassign('x1','y1','x2','y2'): u1,v1:=CtoW(x1,y1): u2,v2:=CtoW(x2,y2):
> L:=(v2-v1)/(u2-u1): u3:=L^2+a1*L-a2-u1-u2: v3:=L*(u1-u3)-v1-a1*u3-a3:
> simplify([W(u3,v3)],[C(x1,y1),C(x2,y2)]); x3std,y3std:=WtoC(u3,v3):
>
> x3:=(x1*y1+x2*y2)/(y1*y2+a*x1*x2): simplify([x3std-x3],[C(x1,y1),C(x2,y2)]);
> x3:=(e*(y1*x1+y2*x2)+f*(x1*y2+y1*x2))/(e*(y1*y2+a*x1*x2)+f*(1+d*x1*x2*y1*y2)): simplify([x3std-x3],[C(x1,y1
    ),C(x2,y2)]);
> x3:=(x1*y2+y1*x2)/(1+d*x1*x2*y1*y2): simplify([x3std-x3],[C(x1,y1),C(x2,y2)]);
> y3:=(x1*y1-x2*y2)/(x1*y2-y1*x2): simplify([y3std-y3],[C(x1,y1),C(x2,y2)]);
> y3:=(g*(y1*x1-y2*x2)+h*(y1*y2-a*x1*x2))/(g*(x1*y2-y1*x2)+h*(1-d*x1*x2*y1*y2)): simplify([y3std-y3],[C(x1,y1
    ),C(x2,y2)]);
> y3:=(y1*y2-a*x1*x2)/(1-d*x1*x2*y1*y2): simplify([y3std-y3],[C(x1,y1),C(x2,y2)]);
```

DBL_E_a00.

```
> x1:=X1/Z1: y1:=Y1/Z1:
> x3:=2*x1*y1/(y1^2+a*x1^2):
> y3:=(y1^2-a*x1^2)/(2-y1^2-a*x1^2):
>
> DBL_E_a00:=proc(X1,Y1,Z1) local X3,Y3,Z3:
>     X3:=2*X1*Y1*(2*Z1^2-Y1^2-a*X1^2):
>     Y3:=(Y1^2-a*X1^2)*(Y1^2+a*X1^2):
>     Z3:=(Y1^2+a*X1^2)*(2*Z1^2-Y1^2-a*X1^2):
>     return X3,Y3,Z3:
> end proc:
> X3,Y3,Z3:=DBL_E_a00(X1,Y1,Z1):
> simplify([x3-X3/Z3,y3-Y3/Z3],[(C(x1,y1))]); #Check.
>
```

DBL_E_a01, $3\mathbf{M} + 4\mathbf{S} + 1\mathbf{D} + 7\mathbf{a}$.

```
> DBL_E_a01_opr:=proc(X1,Y1,Z1) local X3,Y3,Z3,B,C,D,E,F,G,H,J:
>     B:=(X1+Y1)^2: C:=X1^2: D:=Y1^2: E:=a*C: F:=E+D: H:=Z1^2: J:=F-2*H: X3:=(B-C-D)*J: Y3:=F*(E-D): Z3:=F*J:
>     return X3,Y3,Z3:
> end proc:
> X3,Y3,Z3:=DBL_E_a01_opr(X1,Y1,Z1):
> simplify([x3-X3/Z3,y3-Y3/Z3],[(C(x1,y1))]); #Check.
>
```

ADD_E_a00.

```
> x1:=X1/Z1: y1:=Y1/Z1: x2:=X2/Z2: y2:=Y2/Z2:
> x3:=(x1*y2+y1*x2)/(1+d*x1*x2*y1*y2):
> y3:=(y1*y2-a*x1*x2)/(1-d*x1*x2*y1*y2):
>
> ADD_E_a00:=proc(X1,Y1,Z1,X2,Y2,Z2) local X3,Y3,Z3:
>     X3:=(X1*Y2-Y1*X2)*(X1*Y1*Z2^2+X2*Y2*Z1^2):
>     Y3:=(Y1*Y2+a*X1*X2)*(X1*Y1*Z2^2-X2*Y2*Z1^2):
>     Z3:=Z1*Z2*(X1*Y2-Y1*X2)*(Y1*Y2+a*X1*X2):
>     return X3,Y3,Z3:
> end proc:
> X3,Y3,Z3:=ADD_E_a00(X1,Y1,Z1,X2,Y2,Z2):
> simplify([x3-X3/Z3,y3-Y3/Z3],[(C(x1,y1)),(C(x2,y2))]); #Check.
>
```

ADD_E_a01, $11\mathbf{M} + 2\mathbf{D} + 9\mathbf{a}$.

```
> ADD_E_a01_opr:=proc(X1,Y1,Z1,X2,Y2,Z2) local X3,Y3,Z3,A,B,C,D,E,F,G,H,J,K:
>     A:=X1*Z2: B:=Y1*Z2: C:=Z1*X2: D:=Z1*Y2: E:=A*B: F:=C*D: G:=E+F: H:=E-F: J:=(A-C)*(B+D)-H: K:=(A+D)*(B+a
    *C)-E-a*F: X3:=G*J: Y3:=H*K: Z3:=J*K:
>     return X3,Y3,Z3:
> end proc:
> X3,Y3,Z3:=ADD_E_a01_opr(X1,Y1,Z1,X2,Y2,Z2):
> simplify([x3-X3/Z3,y3-Y3/Z3],[(C(x1,y1)),(C(x2,y2))]); #Check.
>
```

UADD_E_a00.

```
> x1:=X1/Z1: y1:=Y1/Z1: x2:=X2/Z2: y2:=Y2/Z2:
> x3:=(x1*y2+y1*x2)/(1+d*x1*x2*y1*y2):
> y3:=(y1*y2-a*x1*x2)/(1-d*x1*x2*y1*y2):
>
> UADD_E_a00:=proc(X1,Y1,Z1,X2,Y2,Z2) local X3,Y3,Z3:
>     X3:=Z1*Z2*(X1*Y2+Y1*X2)*(Z1^2*Z2^2-d*X1*X2*Y1*Y2):
>     Y3:=Z1*Z2*(Y1*Y2-a*X1*X2)*(Z1^2*Z2^2+d*X1*X2*Y1*Y2):
>     Z3:=(Z1^2*Z2^2-d*X1*X2*Y1*Y2)*(Z1^2*Z2^2+d*X1*X2*Y1*Y2):
>     return X3,Y3,Z3:
> end proc:
> X3,Y3,Z3:=UADD_E_a00(X1,Y1,Z1,X2,Y2,Z2):
> simplify([x3-X3/Z3,y3-Y3/Z3],[(C(x1,y1)),(C(x2,y2))]); #Check.
>
```

UADD_E_a01, $10\mathbf{M} + 1\mathbf{S} + 2\mathbf{D} + 7\mathbf{a}$.

```
> UADD_E_a01_opr:=proc(X1,Y1,Z1,X2,Y2,Z2) local X3,Y3,Z3,A,B,C,D,E,F,G:
>     A:=Z1*Z2: B:=A^2: C:=X1*X2: D:=Y1*Y2: E:=d*C*D: F:=B-E: G:=B+E: X3:=A*F*((X1+Y1)*(X2+Y2)-C-D): Y3:=A*G*
    (D-a*C): Z3:=F*G:
>     return X3,Y3,Z3:
> end proc:
> X3,Y3,Z3:=UADD_E_a01_opr(X1,Y1,Z1,X2,Y2,Z2):
> simplify([x3-X3/Z3,y3-Y3/Z3],[(C(x1,y1)),(C(x2,y2))]); #Check.
>
```

DBL_E_a01, $3\mathbf{M} + 4\mathbf{S} + 2\mathbf{D} + 6\mathbf{a}$.

```
> DBL_E_a01_opr:=proc(X1,Y1,Z1) local X3,Y3,Z3,A,B,C,D,E,U:
>     A:=X1^2: B:=Y1^2: U:=a*B: C:=A+U: D:=A-U: E:=(X1+Y1)^2-A-B: X3:=C*D: Y3:=E*(C-2*d*Z1^2): Z3:=D*E:
>     return X3,Y3,Z3:
> end proc:
> X3,Y3,Z3:=DBL_E_a01_opr(X1,Y1,Z1):
> simplify([x3-Z3/X3,y3-Z3/Y3],[(C(x1,y1))]); #Check.
>
```

DBL_Ei_a00.

```
> x1:=Z1/X1: y1:=Z1/Y1:
> x3:=2*x1*y1/(y1^2+a*x1^2):
> y3:=(y1^2-a*x1^2)/(2-y1^2-a*x1^2):
>
> DBL_Ei_a00:=proc(X1,Y1,Z1) local X3,Y3,Z3:
>     X3:=(X1^2-a*Y1^2)*(X1^2+a*Y1^2):
>     Y3:=2*X1*Y1*(X1^2+a*Y1^2-2*d*Z1^2):
>     Z3:=2*X1*Y1*(X1^2-a*Y1^2):
>     return X3,Y3,Z3:
> end proc:
> X3,Y3,Z3:=DBL_Ei_a00(X1,Y1,Z1):
> simplify([x3-Z3/X3,y3-Z3/Y3],[(C(x1,y1))]); #Check.
>
```

ADD_Ei_a00.

```
> x1:=Z1/X1: y1:=Z1/Y1: x2:=Z2/X2: y2:=Z2/Y2:
> x3:=(x1*y2+y1*x2)/(1+d*x1*x2*y1*y2):
> y3:=(y1*y2-a*x1*x2)/(1-d*x1*x2*y1*y2):
>
> ADD_Ei_a00:=proc(X1,Y1,Z1,X2,Y2,Z2) local X3,Y3,Z3:
>     X3:=Z1*Z2*(X1*X2+a*Y1*Y2)*(X1*Y1*Z2^2-Z1^2*X2*Y2):
>     Y3:=Z1*Z2*(X1*Y2-Y1*X2)*(X1*Y1*Z2^2+Z1^2*X2*Y2):
>     Z3:=(X1*Y1*Z2^2-Z1^2*X2*Y2)*(X1*Y1*Z2^2+Z1^2*X2*Y2):
>     return X3,Y3,Z3:
> end proc:
> X3,Y3,Z3:=ADD_Ei_a00(X1,Y1,Z1,X2,Y2,Z2):
> simplify([x3-Z3/X3,y3-Z3/Y3],[(C(x1,y1)),(C(x2,y2))]); #Check.
>
```

ADD_Ei_a01, $11\mathbf{M} + 2\mathbf{D} + 9\mathbf{a}$.

```
> ADD_Ei_a01_opr:=proc(X1,Y1,Z1,X2,Y2,Z2) local X3,Y3,Z3,A,B,C,D,E,F,G,H,J,K:
>     A:=X1*Z2: B:=Y1*Z2: C:=Z1*X2: D:=Z1*Y2: E:=A*B: F:=C*D: G:=E+F: H:=E-F: X3 := ((A+a*D)*(B+C)-E-a*F)*H:
    Y3 := ((A-C)*(B+D)-H)*G: Z3 := G*H:
>     return X3,Y3,Z3:
> end proc:
> X3,Y3,Z3:=ADD_Ei_a01_opr(X1,Y1,Z1,X2,Y2,Z2):
> simplify([x3-Z3/X3,y3-Z3/Y3],[(C(x1,y1)),(C(x2,y2))]); #Check.
>
```

UADD_Ei_a00.

```
> x1:=Z1/X1: y1:=Z1/Y1: x2:=Z2/X2: y2:=Z2/Y2:
> x3:=(x1*y2+y1*x2)/(1+d*x1*x2*y1*y2):
> y3:=(y1*y2-a*x1*x2)/(1-d*x1*x2*y1*y2):
>
> UADD_Ei_a00:=proc(X1,Y1,Z1,X2,Y2,Z2) local X3,Y3,Z3:
>     X3:=(X1*X2-a*Y1*Y2)*(X1*Y1*X2*Y2+d*Z1^2*Z2^2):
>     Y3:=(X1*Y2+Y1*X2)*(X1*Y1*X2*Y2-d*Z1^2*Z2^2):
>     Z3:=Z1*Z2*(X1*Y2+Y1*X2)*(X1*X2-a*Y1*Y2):
>     return X3,Y3,Z3:
> end proc:
> X3,Y3,Z3:=UADD_Ei_a00(X1,Y1,Z1,X2,Y2,Z2):
> simplify([x3-Z3/X3,y3-Z3/Y3],[(C(x1,y1)),(C(x2,y2))]); #Check.
>
```

UADD_Ei_a01, $9\mathbf{M} + 1\mathbf{S} + 2\mathbf{D} + 7\mathbf{a}$.

```
> UADD_Ei_a01_opr:=proc(X1,Y1,Z1,X2,Y2,Z2) local X3,Y3,Z3,A,B,C,D,E,F,H:
>     A:=Z1*Z2: B:=d*A^2: C:=X1*X2: D:=Y1*Y2: E:=C*D: F:=(X1+Y1)*(X2+Y2)-C-D: H:=C-a*D: X3:=(E+B)*H: Y3:=(E-B
    )*F: Z3:=A*H*F:
>     return X3,Y3,Z3:
> end proc:
> X3,Y3,Z3:=UADD_Ei_a01_opr(X1,Y1,Z1,X2,Y2,Z2):
> simplify([x3-Z3/X3,y3-Z3/Y3],[(C(x1,y1)),(C(x2,y2))]); #Check.
>
```

DBL_Ee_a00.

```
> x1:=X1/Z1: y1:=Y1/Z1: T1:=X1*Y1/Z1:
> x3:=2*x1*y1/(y1^2+a*x1^2):
> y3:=(y1^2-a*x1^2)/(2-y1^2-a*x1^2):
>
> DBL_Ee_a00:=proc(X1,Y1,T1,Z1) local X3,Y3,T3,Z3:
>     X3:=2*X1*Y1*(2*Z1^2-Y1^2-a*X1^2):
>     Y3:=(Y1^2-a*X1^2)*(Y1^2+a*X1^2):
>     T3:=2*X1*Y1*(Y1^2-a*X1^2):
```

```
>     Z3:=(Y1^2+a*X1^2)*(2*Z1^2-Y1^2-a*X1^2):
>     return X3,Y3,T3,Z3:
> end proc:
> X3,Y3,T3,Z3:=DBL_Ee_a00(X1,Y1,T1,Z1):
> simplify([x3-X3/Z3,y3-Y3/Z3,T3-X3*Y3/Z3],[(C(x1,y1))]); #Check.
>
```

DBL_Ee_a01, $4\mathbf{M} + 4\mathbf{S} + 1\mathbf{D} + 7\mathbf{a}$.

```
> DBL_Ee_a01_opr:=proc(X1,Y1,T1,Z1) local X3,Y3,T3,Z3,A,B,C,D,E,F,G,H,J:
>     A:=X1^2: B:=Y1^2: C:=2*Z1^2: D:=A+B: E:=(X1+Y1)^2-D: F:=a*A: G:=B+F: H:=B-F: J:=C-G: X3:=E*J: Y3:=H*G:
    T3:=E*H: Z3:=G*J:
>     return X3,Y3,T3,Z3:
> end proc:
> X3,Y3,T3,Z3:=DBL_Ee_a01_opr(X1,Y1,T1,Z1):
> simplify([x3-X3/Z3,y3-Y3/Z3,T3-X3*Y3/Z3],[(C(x1,y1))]); #Check.
>
```

ADD_Ee_a00.

```
> x1:=X1/Z1: y1:=Y1/Z1: T1:=X1*Y1/Z1: x2:=X2/Z2: y2:=Y2/Z2: T2:=X2*Y2/Z2:
> x3:=(x1*y2+y1*x2)/(1+d*x1*x2*y1*y2):
> y3:=(y1*y2-a*x1*x2)/(1-d*x1*x2*y1*y2):
>
> ADD_Ee_a00:=proc(X1,Y1,T1,Z1,X2,Y2,T2,Z2) local X3,Y3,T3,Z3:
>     X3:=(X1*Y2-Y1*X2)*(T1*Z2+Z1*T2):
>     Y3:=(Y1*Y2+a*X1*X2)*(T1*Z2-Z1*T2):
>     T3:=(T1*Z2-Z1*T2)*(T1*Z2+Z1*T2):
>     Z3:=(X1*Y2-Y1*X2)*(Y1*Y2+a*X1*X2):
>     return X3,Y3,T3,Z3:
> end proc:
> X3,Y3,T3,Z3:=ADD_Ee_a00(X1,Y1,T1,Z1,X2,Y2,T2,Z2):
> simplify([x3-X3/Z3,y3-Y3/Z3,T3-X3*Y3/Z3],[(C(x1,y1)),(C(x2,y2))]); #Check.
>
```

ADD_Ee_a01, $9\mathbf{M} + 1\mathbf{D} + 7\mathbf{a}$.

```
> ADD_Ee_a01_opr:=proc(X1,Y1,T1,Z1,X2,Y2,T2,Z2) local X3,Y3,T3,Z3,A,B,C,D,E,F,G,H:
>     A:=X1*X2: B:=Y1*Y2: C:=Z1*T2: D:=T1*Z2: E:=D+C: F:=(X1-Y1)*(X2+Y2)+B-A: G:=B+a*A: H:=D-C: X3:=E*F: Y3:=
    G*H: Z3:=F*G: T3:=E*H:
>     return X3,Y3,T3,Z3:
> end proc:
> X3,Y3,T3,Z3:=ADD_Ee_a01_opr(X1,Y1,T1,Z1,X2,Y2,T2,Z2):
> simplify([x3-X3/Z3,y3-Y3/Z3,T3-X3*Y3/Z3],[(C(x1,y1)),(C(x2,y2))]); #Check.
>
```

ADD_Ee_a02, $8\mathbf{M} + 10\mathbf{a}$, assumes $a = -1$.

```
> ADD_Ee_a02_opr:=proc(X1,Y1,T1,Z1,X2,Y2,T2,Z2) local X3,Y3,T3,Z3,A,B,C,D,E,F,G,H:
>     A:=(Y1-X1)*(Y2+X2): B:=(Y1+X1)*(Y2-X2): C:=2*Z1*T2: D:=2*T1*Z2: E:=D+C: F:=B-A: G:=B+A: H:=D-C: X3:=E*F
    : Y3:=G*H: T3:=E*H: Z3:=F*G:
>     return X3,Y3,T3,Z3:
> end proc:
> X3,Y3,T3,Z3:=ADD_Ee_a02_opr(X1,Y1,T1,Z1,X2,Y2,T2,Z2):
> simplify([x3-X3/Z3,y3-Y3/Z3,T3-X3*Y3/Z3],[(C(x1,y1)),(C(x2,y2)),a+1]); #Check.
>
```

UADD_Ee_a00.

```
> x1:=X1/Z1: y1:=Y1/Z1: T1:=X1*Y1/Z1: x2:=X2/Z2: y2:=Y2/Z2: T2:=X2*Y2/Z2:
> x3:=(x1*y2+y1*x2)/(1+d*x1*x2*y1*y2):
> y3:=(y1*y2-a*x1*x2)/(1-d*x1*x2*y1*y2):
>
```

```
> UADD_Ee_a00:=proc(X1,Y1,T1,Z1,X2,Y2,T2,Z2) local X3,Y3,T3,Z3:
>     X3:=(X1*Y2+Y1*X2)*(Z1*Z2-d*T1*T2):
>     Y3:=(Y1*Y2-a*X1*X2)*(Z1*Z2+d*T1*T2):
>     T3:=(X1*Y2+Y1*X2)*(Y1*Y2-a*X1*X2):
>     Z3:=(Z1*Z2-d*T1*T2)*(Z1*Z2+d*T1*T2):
>     return X3,Y3,T3,Z3:
> end proc:
> X3,Y3,T3,Z3:=UADD_Ee_a00(X1,Y1,T1,Z1,X2,Y2,T2,Z2):
> simplify([x3-X3/Z3,y3-Y3/Z3,T3-X3*Y3/Z3],[(C(x1,y1)),(C(x2,y2))]); #Check.
>
```

UADD_Ee_a01, $9\mathbf{M} + 2\mathbf{D} + 7\mathbf{a}$.

```
> UADD_Ee_a01_opr:=proc(X1,Y1,T1,Z1,X2,Y2,T2,Z2) local X3,Y3,T3,Z3,A,B,C,D,E,F,G,H:
>     A:=X1*X2: B:=Y1*Y2: C:=d*T1*T2: D:=Z1*Z2: E:=(X1+Y1)*(X2+Y2)-A-B: F:=D-C: G:=D+C: H:=B-a*A: X3:=E*F: Y3
    :=G*H: Z3:=F*G: T3:=E*H:
>     return X3,Y3,T3,Z3:
> end proc:
> X3,Y3,T3,Z3:=UADD_Ee_a01_opr(X1,Y1,T1,Z1,X2,Y2,T2,Z2):
> simplify([x3-X3/Z3,y3-Y3/Z3,T3-X3*Y3/Z3],[(C(x1,y1)),(C(x2,y2))]); #Check.
>
```

UADD_Ee_a02, $8\mathbf{M} + 1\mathbf{D} + 9\mathbf{a}$, assumes $a = -1$.

```
> UADD_Ee_a02_opr:=proc(X1,Y1,T1,Z1,X2,Y2,T2,Z2) local X3,Y3,T3,Z3,A,B,C,D,E,F,G,H:
>     A:=(Y1-X1)*(Y2-X2): B:=(Y1+X1)*(Y2+X2): C:=(2*d)*T1*T2: D:=2*Z1*Z2: E:=B-A: F:=D-C: G:=D+C: H:=B+A: X3:
    =E*F: Y3:=G*H: Z3:=F*G: T3:=E*H:
>     return X3,Y3,T3,Z3:
> end proc:
> X3,Y3,T3,Z3:=UADD_Ee_a02_opr(X1,Y1,T1,Z1,X2,Y2,T2,Z2):
> simplify([x3-X3/Z3,y3-Y3/Z3,T3-X3*Y3/Z3],[(C(x1,y1)),(C(x2,y2)),a+1]); #Check.
>
```

C.5 Twisted Jacobi intersection form

```
> C:=(s,c,d)->(b*s^2+c^2-1,a*s^2+d^2-1):
> a1:=0: a3:=0: a2:=-a-b: a4:=a*b: a6:=0:
> W:=(u,v)->(v^2+a1*u*v+a3*v-(u^3+a2*u^2+a4*u+a6)):
> CtoW:=(s,c,d)->((1+c)*(1+d)/s^2,-(1+c)*(1+d)*(c+d)/s^3):
> WtoC:=(u,v)->(2*v/(a*b-u^2),2*u*(b-u)/(a*b-u^2)-1,2*u*(a-u)/(a*b-u^2)-1):
> simplify([W(CtoW(s1,c1,d1))],[C(s1,c1,d1)]); #Check CtoW.
> simplify([C(WtoC(u1,v1))],[W(u1,v1)]); #Check WtoC.
> simplify([(s1,c1,d1)-WtoC(CtoW(s1,c1,d1))],[C(s1,c1,d1)]); #Check CtoW(WtoC).
> simplify([(u1,v1)-CtoW(WtoC(u1,v1))],[W(u1,v1)]); #Check WtoC(CtoW).
> ut,vt:=CtoW(s1,c1,d1): simplify([(-s1,c1,d1)-WtoC(ut,-vt-a1*ut-a3)],[C(s1,c1,d1)]); #Check the negation.

> # Doubling formulae.
> unassign('s1','c1','d1'): u1,v1:=CtoW(s1,c1,d1):
> L:=(3*u1^2+2*a2*u1+a4-a1*v1)/(2*v1+a1*u1+a3): u3:=L^2+a1*L-a2-2*u1: v3:=L*(u1-u3)-v1-a1*u3-a3:
> simplify([W(u3,v3)],[C(s1,c1,d1)]); s3std,c3std,d3std:=WtoC(u3,v3):
>
> s3:=2*s1*c1*d1/(d1^2+a*s1^2*c1^2): simplify([s3std-s3],[C(s1,c1,d1)]);
> s3:=2*s1*c1*d1/(c1^2+b*s1^2*d1^2): simplify([s3std-s3],[C(s1,c1,d1)]);
> s3:=2*s1*c1*d1/(c1^2+d1^2-c1^2*d1^2): simplify([s3std-s3],[C(s1,c1,d1)]);
> s3:=2*s1*c1*d1/(1-a*b*s1^4): simplify([s3std-s3],[C(s1,c1,d1)]);
> c3:=(2*(c1^4+b*s1^2*c1^2)-d1^2-a*s1^2*c1^2)/(d1^2+a*s1^2*c1^2): simplify([c3std-c3],[C(s1,c1,d1)]);
> c3:=(2*c1^2-d1^2-a*s1^2*c1^2)/(d1^2+a*s1^2*c1^2): simplify([c3std-c3],[C(s1,c1,d1)]);
> c3:=(c1^2-b*s1^2*d1^2)/(c1^2+b*s1^2*d1^2): simplify([c3std-c3],[C(s1,c1,d1)]);
> c3:=(c1^2-d1^2+c1^2*d1^2)/(c1^2+d1^2-c1^2*d1^2): simplify([c3std-c3],[C(s1,c1,d1)]);
> c3:=(c1^2-b*s1^2*d1^2)/(1-a*b*s1^4): simplify([c3std-c3],[C(s1,c1,d1)]);
> d3:=(d1^2-a*s1^2*c1^2)/(d1^2+a*s1^2*c1^2): simplify([d3std-d3],[C(s1,c1,d1)]);
> d3:=(2*d1^2-c1^2-b*s1^2*d1^2)/(c1^2+b*s1^2*d1^2): simplify([d3std-d3],[C(s1,c1,d1)]);
```

```
> d3:=(d1^2-c1^2+c1^2*d1^2)/(c1^2+d1^2-c1^2*d1^2): simplify([d3std-d3],[C(s1,c1,d1)]);
> d3:=(d1^2-a*s1^2*c1^2)/(1-a*b*s1^4): simplify([d3std-d3],[C(s1,c1,d1)]);
> d3:=(2*(d1^4+a*s1^2*d1^2)-c1^2-b*s1^2*d1^2)/(c1^2+b*s1^2*d1^2): simplify([d3std-d3],[C(s1,c1,d1)]);

> # Addition formulae.
> unassign('s1','c1','d1','s2','c2','d2'): u1,v1:=CtoW(s1,c1,d1): u2,v2:=CtoW(s2,c2,d2):
> L:=(v2-v1)/(u2-u1): u3:=L^2+a1*L-a2-u1-u2: v3:=L*(u1-u3)-v1-a1*u3-a3:
> simplify([W(u3,v3)],[C(s1,c1,d1),C(s2,c2,d2)]); s3std,c3std,d3std:=WtoC(u3,v3):
>
> s3:=(s1^2-s2^2)/(s1*c2*d2-c1*d1*s2): simplify([s3std-s3],[C(s1,c1,d1),C(s2,c2,d2)]);
> s3:=(s1*c2*d2+c1*d1*s2)/(c1^2+b*s1^2*d2^2): simplify([s3std-s3],[C(s1,c1,d1),C(s2,c2,d2)]);
> s3:=(s1*c2*d2+c1*d1*s2)/(1-a*b*s1^2*s2^2): simplify([s3std-s3],[C(s1,c1,d1),C(s2,c2,d2)]);
> c3:=(s1*c1*d2-d1*s2*c2)/(s1*c2*d2-c1*d1*s2): simplify([c3std-c3],[C(s1,c1,d1),C(s2,c2,d2)]);
> c3:=(c1*c2-b*s1*d1*s2*d2)/(c1^2+b*s1^2*d2^2): simplify([c3std-c3],[C(s1,c1,d1),C(s2,c2,d2)]);
> c3:=(c1*c2-b*s1*d1*s2*d2)/(1-a*b*s1^2*s2^2): simplify([c3std-c3],[C(s1,c1,d1),C(s2,c2,d2)]);
> d3:=(s1*d1*c2-c1*s2*d2)/(s1*c2*d2-c1*d1*s2): simplify([d3std-d3],[C(s1,c1,d1),C(s2,c2,d2)]);
> d3:=(d1*d2-a*s1*c1*s2*c2)/(c1^2+b*s1^2*d2^2): simplify([d3std-d3],[C(s1,c1,d1),C(s2,c2,d2)]);
> d3:=(d1*d2-a*s1*c1*s2*c2)/(1-a*b*s1^2*s2^2): simplify([d3std-d3],[C(s1,c1,d1),C(s2,c2,d2)]);
```

DBL_I_a00.

```
> s1:=S1/Z1: c1:=C1/Z1: d1:=D1/Z1:
> s3:=2*s1*c1*d1/(1-a*b*s1^4):
> c3:=(c1^2-b*s1^2*d1^2)/(1-a*b*s1^4):
> d3:=(d1^2-a*s1^2*c1^2)/(1-a*b*s1^4):
>
> DBL_I_a00:=proc(S1,C1,D1,Z1) local S3,C3,D3,Z3:
>     S3:=2*S1*C1*D1*Z1:
>     C3:=C1^2*Z1^2-b*S1^2*D1^2:
>     D3:=-C1^2*Z1^2-b*S1^2*D1^2+2*D1^2*Z1^2:
>     Z3:=C1^2*Z1^2+b*S1^2*D1^2:
>     return S3,C3,D3,Z3:
> end proc:
> S3,C3,D3,Z3:=DBL_I_a00(S1,C1,D1,Z1):
> simplify([s3-S3/Z3,c3-C3/Z3,d3-D3/Z3],[(C(s1,c1,d1))]); #Check.
>
```

DBL_I_a01, $3\mathbf{M} + 4\mathbf{S} + 1\mathbf{D} + 7\mathbf{a}$.

```
> DBL_I_a01_opr:=proc(S1,C1,D1,Z1) local S3,C3,D3,Z3,U1,V1,E,F,G,H:
>     U1:=S1*D1: V1:=C1*Z1: E:=D1*Z1: F:=U1^2: G:=V1^2: S3:=(U1+V1)^2-G-F: H:=b*F: C3:=G-H: Z3:=G+H: D3:=2*E^
    2-Z3:
>     return S3,C3,D3,Z3:
> end proc:
> S3,C3,D3,Z3:=DBL_I_a01_opr(S1,C1,D1,Z1):
> simplify([s3-S3/Z3,c3-C3/Z3,d3-D3/Z3],[(C(s1,c1,d1))]); #Check.
>
> DBL_I_a01_reg:=proc(S1,C1,D1,Z1) local S3,C3,D3,Z3,t1:
>     S3:=S1*D1: C3:=C1*Z1: D3:=D1*Z1: Z3:=S3+C3: S3:=S3^2: C3:=C3^2: D3:=D3^2: Z3:=Z3^2: D3:=2*D3: t1:=b*S3:
     S3:=S3+C3: S3:=Z3-S3: Z3:=t1+C3: C3:=C3-t1: D3:=D3-Z3:
>     return S3,C3,D3,Z3:
> end proc:
> S3,C3,D3,Z3:=DBL_I_a01_reg(S1,C1,D1,Z1):
> simplify([s3-S3/Z3,c3-C3/Z3,d3-D3/Z3],[(C(s1,c1,d1))]); #Check.
>
```

DBL_I_a02, $3\mathbf{M} + 4\mathbf{S} + 6\mathbf{a}$, assumes $b = 1$.

```
> DBL_I_a02_opr:=proc(S1,C1,D1,Z1) local S3,C3,D3,Z3,U1,V1,E,F,G,H:
>     U1:=S1*D1: V1:=C1*Z1: E:=D1*Z1: F:=U1^2: G:=V1^2: Z3:=G+F: S3:=(U1+V1)^2-Z3: C3:=G-F: D3:=2*E^2-Z3:
>     return S3,C3,D3,Z3:
> end proc:
> S3,C3,D3,Z3:=DBL_I_a02_opr(S1,C1,D1,Z1):
```

```
> simplify([s3-S3/Z3,c3-C3/Z3,d3-D3/Z3],[(C(s1,c1,d1)),b-1]); #Check.
>
> DBL_I_a02_reg:=proc(S1,C1,D1,Z1) local S3,C3,D3,Z3,t1:
>     S3:=S1*D1: C3:=C1*Z1: D3:=D1*Z1: t1:=S3+C3: t1:=t1^2: S3:=S3^2: C3:=C3^2: D3:=D3^2: Z3:=C3+S3: C3:=C3-S
    3: D3:=2*D3: D3:=D3-Z3: S3:=t1-Z3:
>     return S3,C3,D3,Z3:
> end proc:
> S3,C3,D3,Z3:=DBL_I_a02_reg(S1,C1,D1,Z1):
> simplify([s3-S3/Z3,c3-C3/Z3,d3-D3/Z3],[(C(s1,c1,d1)),b-1]); #Check.
>
```

DBL_I_b00.

```
> s1:=S1/Z1: c1:=C1/Z1: d1:=D1/Z1:
> s3:=2*s1*c1*d1/(1-a*b*s1^4):
> c3:=(c1^2-b*s1^2*d1^2)/(1-a*b*s1^4):
> d3:=(d1^2-a*s1^2*c1^2)/(1-a*b*s1^4):
>
> DBL_I_b00:=proc(S1,C1,D1,Z1) local S3,C3,D3,Z3:
>     S3:=2*S1*C1*D1*Z1:
>     C3:=C1^2*Z1^2-b*S1^2*D1^2:
>     D3:=-C1^2*Z1^2-b*S1^2*D1^2+2*a*S1^2*D1^2+2*D1^4:
>     Z3:=C1^2*Z1^2+b*S1^2*D1^2:
>     return S3,C3,D3,Z3:
> end proc:
> S3,C3,D3,Z3:=DBL_I_b00(S1,C1,D1,Z1):
> simplify([s3-S3/Z3,c3-C3/Z3,d3-D3/Z3],[(C(s1,c1,d1))]); #Check.
>
```

DBL_I_b01, $2\mathbf{M} + 5\mathbf{S} + 2\mathbf{D} + 8\mathbf{a}$.

```
> DBL_I_b01_opr:=proc(S1,C1,D1,Z1) local S3,C3,D3,Z3,U1,V1,F,G,H:
>     U1:=S1*D1: V1:=C1*Z1: F:=U1^2: G:=V1^2: H:=b*F: C3:=G-H: Z3:=G+H: S3:=(U1+V1)^2-F-G: D3:=2*(a*F+D1^4)-Z
    3:
>     return S3,C3,D3,Z3:
> end proc:
> S3,C3,D3,Z3:=DBL_I_b01_opr(S1,C1,D1,Z1):
> simplify([s3-S3/Z3,c3-C3/Z3,d3-D3/Z3],[(C(s1,c1,d1))]); #Check.
>
> DBL_I_b01_reg:=proc(S1,C1,D1,Z1) local S3,C3,D3,Z3,t1:
>     S3:=S1*D1: C3:=C1*Z1: D3:=D1^2: Z3:=S3+C3: S3:=S3^2: C3:=C3^2: D3:=D3^2: Z3:=Z3^2: t1:=a*S3: D3:=t1+D3:
     D3:=2*D3: t1:=b*S3: S3:=S3+C3: S3:=Z3-S3: Z3:=C3+t1: C3:=C3-t1: D3:=D3-Z3:
>     return S3,C3,D3,Z3:
> end proc:
> S3,C3,D3,Z3:=DBL_I_b01_reg(S1,C1,D1,Z1):
> simplify([s3-S3/Z3,c3-C3/Z3,d3-D3/Z3],[(C(s1,c1,d1))]); #Check.
>
```

DBL_I_b02, $2\mathbf{M} + 5\mathbf{S} + 1\mathbf{D} + 7\mathbf{a}$, assumes $b = 1$.

```
> DBL_I_b02_opr:=proc(S1,C1,D1,Z1) local S3,C3,D3,Z3,U1,V1,F,G:
>     U1:=S1*D1: V1:=C1*Z1: F:=U1^2: G:=V1^2: C3:=G-F: Z3:=G+F: S3:=(U1+V1)^2-Z3: D3:=2*(a*F+D1^4)-Z3:
>     return S3,C3,D3,Z3:
> end proc:
> S3,C3,D3,Z3:=DBL_I_b02_opr(S1,C1,D1,Z1):
> simplify([s3-S3/Z3,c3-C3/Z3,d3-D3/Z3],[(C(s1,c1,d1)),b-1]); #Check.
>
> DBL_I_b02_reg:=proc(S1,C1,D1,Z1) local S3,C3,D3,Z3,t1:
>     S3:=S1*D1: C3:=C1*Z1: D3:=D1^2: D3:=D3^2: t1:=S3^2: S3:=S3+C3: S3:=S3^2: C3:=C3^2: Z3:=C3+t1: S3:=S3-Z3
    : C3:=C3-t1: t1:=a*t1: D3:=D3+t1: D3:=2*D3: D3:=D3-Z3:
>     return S3,C3,D3,Z3:
> end proc:
> S3,C3,D3,Z3:=DBL_I_b02_reg(S1,C1,D1,Z1):
> simplify([s3-S3/Z3,c3-C3/Z3,d3-D3/Z3],[(C(s1,c1,d1)),b-1]); #Check.
>
```

ADD_I_a00.

```
> s1:=S1/Z1: c1:=C1/Z1: d1:=D1/Z1: s2:=S2/Z2: c2:=C2/Z2: d2:=D2/Z2:
> s3:=(s1*c2*d2+c1*d1*s2)/(1-a*b*s1^2*s2^2):
> c3:=(c1*c2-b*s1*d1*s2*d2)/(1-a*b*s1^2*s2^2):
> d3:=(d1*d2-a*s1*c1*s2*c2)/(1-a*b*s1^2*s2^2):
>
> ADD_I_a00:=proc(S1,C1,D1,Z1,S2,C2,D2,Z2) local S3,C3,D3,Z3:
>     S3:=S1^2*Z2^2-Z1^2*S2^2:
>     C3:=S1*C1*D2*Z2-D1*Z1*S2*C2:
>     D3:=S1*D1*C2*Z2-C1*Z1*S2*D2:
>     Z3:=S1*Z1*C2*D2-C1*D1*S2*Z2:
>     return S3,C3,D3,Z3:
> end proc:
> S3,C3,D3,Z3:=ADD_I_a00(S1,C1,D1,Z1,S2,C2,D2,Z2):
> simplify([s3-S3/Z3,c3-C3/Z3,d3-D3/Z3],[(C(s1,c1,d1)),(C(s2,c2,d2))]); #Check.
>
```

ADD_I_a01, $12\mathbf{M} + 11\mathbf{a}$.

```
> ADD_I_a01_opr:=proc(S1,C1,D1,Z1,S2,C2,D2,Z2) local S3,C3,D3,Z3,E,F,G,H,J,K,L,M,N,P:
>     E:=S1*Z2: F:=Z1*S2: G:=C1*D2: H:=D1*C2: J:=E-F: K:=E+F: L:=G-H: M:=G+H: N:=K*L: P:=J*M: S3:=J*K: C3:=(N
    +P)/2: D3:=(P-N)/2: Z3:=(D1*Z2+Z1*D2)*(S1*C2-C1*S2)-D3:
>     return S3,C3,D3,Z3:
> end proc:
> S3,C3,D3,Z3:=ADD_I_a01_opr(S1,C1,D1,Z1,S2,C2,D2,Z2):
> simplify([s3-S3/Z3,c3-C3/Z3,d3-D3/Z3],[(C(s1,c1,d1)),(C(s2,c2,d2))]); #Check.
>
```

ADD_I_b00.

```
> s1:=S1/Z1: c1:=C1/Z1: d1:=D1/Z1: s2:=S2/Z2: c2:=C2/Z2: d2:=D2/Z2:
> s3:=(s1*c2*d2+c1*d1*s2)/(1-a*b*s1^2*s2^2):
> c3:=(c1*c2-b*s1*d1*s2*d2)/(1-a*b*s1^2*s2^2):
> d3:=(d1*d2-a*s1*c1*s2*c2)/(1-a*b*s1^2*s2^2):
>
> ADD_I_b00:=proc(S1,C1,D1,Z1,S2,C2,D2,Z2) local S3,C3,D3,Z3:
>     S3:=(1/b)*(Z1^2*C2^2-C1^2*Z2^2):
>     C3:=S1*C1*D2*Z2-D1*Z1*S2*C2:
>     D3:=S1*D1*C2*Z2-C1*Z1*S2*D2:
>     Z3:=S1*Z1*C2*D2-C1*D1*S2*Z2:
>     return S3,C3,D3,Z3:
> end proc:
> S3,C3,D3,Z3:=ADD_I_b00(S1,C1,D1,Z1,S2,C2,D2,Z2):
> simplify([s3-S3/Z3,c3-C3/Z3,d3-D3/Z3],[(C(s1,c1,d1)),(C(s2,c2,d2))]); #Check.
>
```

ADD_I_b01, $12\mathbf{M} + 1\mathbf{D} + 11\mathbf{a}$.

```
> ADD_I_b01_opr:=proc(S1,C1,D1,Z1,S2,C2,D2,Z2) local S3,C3,D3,Z3,E,F,G,H,J,K,L,M,N,P:
>     E:=C1*Z2: F:=Z1*C2: G:=S1*D2: H:=D1*S2: J:=F-E: K:=F+E: L:=G-H: M:=G+H: N:=K*L: P:=J*M: S3:=(1/b)*J*K:
    C3:=(N-P)/2: Z3:=(N+P)/2: D3:=(S1*Z2-Z1*S2)*(C1*D2+D1*C2)-C3:
>     return S3,C3,D3,Z3:
> end proc:
> S3,C3,D3,Z3:=ADD_I_b01_opr(S1,C1,D1,Z1,S2,C2,D2,Z2):
> simplify([s3-S3/Z3,c3-C3/Z3,d3-D3/Z3],[(C(s1,c1,d1)),(C(s2,c2,d2))]); #Check.
>
```

ADD_I_b02, $12\mathbf{M} + 11\mathbf{a}$, assumes $b = 1$.

```
> ADD_I_b02_opr:=proc(S1,C1,D1,Z1,S2,C2,D2,Z2) local S3,C3,D3,Z3,E,F,G,H,J,K,L,M,N,P:
>     E:=C1*Z2: F:=Z1*C2: G:=S1*D2: H:=D1*S2: J:=F-E: K:=F+E: L:=G-H: M:=G+H: N:=K*L: P:=J*M: S3:=J*K: C3:=(N
    -P)/2: Z3:=(N+P)/2: D3:=(S1*Z2-Z1*S2)*(C1*D2+D1*C2)-C3:
>     return S3,C3,D3,Z3:
```

```
> end proc:
> S3,C3,D3,Z3:=ADD_I_b02_opr(S1,C1,D1,Z1,S2,C2,D2,Z2):
> simplify([s3-S3/Z3,c3-C3/Z3,d3-D3/Z3],[(C(s1,c1,d1)),(C(s2,c2,d2)),b-1]); #Check.
>
```

UADD_I_a00.

```
> s1:=S1/Z1: c1:=C1/Z1: d1:=D1/Z1: s2:=S2/Z2: c2:=C2/Z2: d2:=D2/Z2:
> s3:=(s1*c2*d2+c1*d1*s2)/(1-a*b*s1^2*s2^2):
> c3:=(c1*c2-b*s1*d1*s2*d2)/(1-a*b*s1^2*s2^2):
> d3:=(d1*d2-a*s1*c1*s2*c2)/(1-a*b*s1^2*s2^2):
>
> UADD_I_a00:=proc(S1,C1,D1,Z1,S2,C2,D2,Z2) local S3,C3,D3,Z3:
>     S3:=S1*Z1*C2*D2+C1*D1*S2*Z2:
>     C3:=C1*Z1*C2*Z2-b*S1*D1*S2*D2:
>     D3:=D1*Z1*D2*Z2-a*S1*C1*S2*C2:
>     Z3:=C1^2*Z2^2+b*S1^2*D2^2:
>     return S3,C3,D3,Z3:
> end proc:
> S3,C3,D3,Z3:=UADD_I_a00(S1,C1,D1,Z1,S2,C2,D2,Z2):
> simplify([s3-S3/Z3,c3-C3/Z3,d3-D3/Z3],[(C(s1,c1,d1)),(C(s2,c2,d2))]); #Check.
>
```

UADD_I_a01, $13\mathbf{M} + 2\mathbf{S} + 5\mathbf{D} + 13\mathbf{a}$.

```
> UADD_I_a01_opr:=proc(S1,C1,D1,Z1,S2,C2,D2,Z2) local S3,C3,D3,Z3,U1,U2,V1,V2,E,F,G,H,J,K:
>     E:=C1*Z2: F:=Z1*C2: G:=S1*D2: H:=D1*S2: J:=F*H: K:=E*G: S3:=(E+F)*(G+H)-J-K: C3:=(E-b*H)*(G+F)+b*J-K: D
    3:=(D1*Z1-a*S1*C1)*(S2*C2+D2*Z2)-J+a*K: Z3:=E^2+b*G^2:
>     return S3,C3,D3,Z3:
> end proc:
> S3,C3,D3,Z3:=UADD_I_a01_opr(S1,C1,D1,Z1,S2,C2,D2,Z2):
> simplify([s3-S3/Z3,c3-C3/Z3,d3-D3/Z3],[(C(s1,c1,d1)),(C(s2,c2,d2))]); #Check.
>
```

UADD_I_a02, $13\mathbf{M} + 1\mathbf{S} + 2\mathbf{D} + 15\mathbf{a}$, assumes $b = 1$.

```
> UADD_I_a02_opr:=proc(S1,C1,D1,Z1,S2,C2,D2,Z2) local S3,C3,D3,Z3,U1,U2,V1,V2,E,F,G,H,J,K,L,M,N,P:
>     U1:=S1*C1: V1:=D1*Z1: U2:=S2*C2: V2:=D2*Z2: E:=S1*D2: F:=C1*Z2: G:=D1*S2: H:=Z1*C2: J:=U1*V2: K:=V1*U2:
    S3:=(H+F)*(E+G)-J-K: C3:=(H+E)*(F-G)-J+K: D3:=(V1-a*U1)*(U2+V2)+a*J-K: Z3:=(H+G)^2-2*K:
>     return S3,C3,D3,Z3:
> end proc:
> S3,C3,D3,Z3:=UADD_I_a02_opr(S1,C1,D1,Z1,S2,C2,D2,Z2):
> simplify([s3-S3/Z3,c3-C3/Z3,d3-D3/Z3],[(C(s1,c1,d1)),(C(s2,c2,d2)),b-1]); #Check.
>
```

DBL_Imd_a00.

```
> s1:=S1/Z1: c1:=C1/Z1: d1:=D1/Z1: U1:=S1*D1: V1:=C1*Z1:
> s3:=2*s1*c1*d1/(1-a*b*s1^4):
> c3:=(c1^2-b*s1^2*d1^2)/(1-a*b*s1^4):
> d3:=(d1^2-a*s1^2*c1^2)/(1-a*b*s1^4):
>
> DBL_Imd_a00:=proc(S1,C1,D1,Z1,U1,V1) local S3,C3,D3,Z3,U3,V3:
>     S3:=2*S1*C1*D1*Z1:
>     C3:=C1^2*Z1^2-b*S1^2*D1^2:
>     D3:=-C1^2*Z1^2-b*S1^2*D1^2+2*D1^2*Z1^2:
>     Z3:=C1^2*Z1^2+b*S1^2*D1^2:
>     U3:=S3*D3:
>     V3:=C3*Z3:
>     return S3,C3,D3,Z3,U3,V3:
> end proc:
> S3,C3,D3,Z3,U3,V3:=DBL_Imd_a00(S1,C1,D1,Z1,U1,V1):
> simplify([s3-S3/Z3,c3-C3/Z3,d3-D3/Z3,U3-S3*D3,V3-C3*Z3],[(C(s1,c1,d1))]); #Check.
>
```

DBL_Imd_a01, $3\mathbf{M} + 4\mathbf{S} + 1\mathbf{D} + 7\mathbf{a}$.

```
> DBL_Imd_a01_opr:=proc(S1,C1,D1,Z1,U1,V1) local S3,C3,D3,Z3,U3,V3,E,F,G,H:
>     E:=D1*Z1: F:=U1^2: G:=V1^2: S3:=(U1+V1)^2-G-F: H:=b*F: C3:=G-H: Z3:=G+H: D3:=2*E^2-Z3: U3:=S3*D3: V3:=C
    3*Z3:
>     return S3,C3,D3,Z3,U3,V3:
> end proc:
> S3,C3,D3,Z3,U3,V3:=DBL_Imd_a01_opr(S1,C1,D1,Z1,U1,V1):
> simplify([s3-S3/Z3,c3-C3/Z3,d3-D3/Z3,U3-S3*D3,V3-C3*Z3],[(C(s1,c1,d1))]); #Check.
>
> DBL_Imd_a01_reg:=proc(S1,C1,D1,Z1,U1,V1) local S3,C3,D3,Z3,U3,V3,t1:
>     D3:=D1*Z1: Z3:=U1+V1: S3:=U1^2: C3:=V1^2: D3:=D3^2: Z3:=Z3^2: D3:=2*D3: t1:=b*S3: S3:=S3+C3: S3:=Z3-S3:
     Z3:=t1+C3: C3:=C3-t1: D3:=D3-Z3: U3:=S3*D3: V3:=C3*Z3:
>     return S3,C3,D3,Z3,U3,V3:
> end proc:
> S3,C3,D3,Z3,U3,V3:=DBL_Imd_a01_reg(S1,C1,D1,Z1,U1,V1):
> simplify([s3-S3/Z3,c3-C3/Z3,d3-D3/Z3,U3-S3*D3,V3-C3*Z3],[(C(s1,c1,d1))]); #Check.
>
```

DBL_Imd_a02, $3\mathbf{M} + 4\mathbf{S} + 6\mathbf{a}$, assumes $b = 1$.

```
> DBL_Imd_a02_opr:=proc(S1,C1,D1,Z1,U1,V1) local S3,C3,D3,Z3,U3,V3,E,F,G,H:
>     E:=D1*Z1: F:=U1^2: G:=V1^2: Z3:=G+F: S3:=(U1+V1)^2-Z3: C3:=G-F: D3:=2*E^2-Z3: U3:=S3*D3: V3:=C3*Z3:
>     return S3,C3,D3,Z3,U3,V3:
> end proc:
> S3,C3,D3,Z3,U3,V3:=DBL_Imd_a02_opr(S1,C1,D1,Z1,U1,V1):
> simplify([s3-S3/Z3,c3-C3/Z3,d3-D3/Z3,U3-S3*D3,V3-C3*Z3],[(C(s1,c1,d1)),b-1]); #Check.
>
> DBL_Imd_a02_reg:=proc(S1,C1,D1,Z1,U1,V1) local S3,C3,D3,Z3,U3,V3,t1:
>     D3:=D1*Z1: t1:=U1+V1: t1:=t1^2: S3:=U1^2: C3:=V1^2: D3:=D3^2: Z3:=C3+S3: C3:=C3-S3: D3:=2*D3: D3:=D3-Z3
    : S3:=t1-Z3: U3:=S3*D3: V3:=C3*Z3:
>     return S3,C3,D3,Z3,U3,V3:
> end proc:
> S3,C3,D3,Z3,U3,V3:=DBL_Imd_a02_reg(S1,C1,D1,Z1,U1,V1):
> simplify([s3-S3/Z3,c3-C3/Z3,d3-D3/Z3,U3-S3*D3,V3-C3*Z3],[(C(s1,c1,d1)),b-1]); #Check.
>
```

ADD_Imd_a00.

```
> s1:=S1/Z1: c1:=C1/Z1: d1:=D1/Z1: U1:=S1*D1: V1:=C1*Z1: s2:=S2/Z2: c2:=C2/Z2: d2:=D2/Z2: U2:=S2*D2: V2:=C2*Z
    2:
> s3:=(s1*c2*d2+c1*d1*s2)/(1-a*b*s1^2*s2^2):
> c3:=(c1*c2-b*s1*d1*s2*d2)/(1-a*b*s1^2*s2^2):
> d3:=(d1*d2-a*s1*c1*s2*c2)/(1-a*b*s1^2*s2^2):
>
> ADD_Imd_a00:=proc(S1,C1,D1,Z1,U1,V1,S2,C2,D2,Z2,U2,V2) local S3,C3,D3,Z3,U3,V3:
>     S3:=(1/b)*(Z1^2*C2^2-C1^2*Z2^2):
>     C3:=S1*C1*D2*Z2-D1*Z1*S2*C2:
>     D3:=S1*D1*C2*Z2-C1*Z1*S2*D2:
>     Z3:=S1*Z1*C2*D2-C1*D1*S2*Z2:
>     U3:=S3*D3:
>     V3:=C3*Z3:
>     return S3,C3,D3,Z3,U3,V3:
> end proc:
> S3,C3,D3,Z3,U3,V3:=ADD_Imd_a00(S1,C1,D1,Z1,U1,V1,S2,C2,D2,Z2,U2,V2):
> simplify([s3-S3/Z3,c3-C3/Z3,d3-D3/Z3,U3-S3*D3,V3-C3*Z3],[(C(s1,c1,d1)),(C(s2,c2,d2))]); #Check.
>
```

ADD_Imd_a01, $11\mathbf{M} + 1\mathbf{D} + 9\mathbf{a}$.

```
> ADD_Imd_a01_opr:=proc(S1,C1,D1,Z1,U1,V1,S2,C2,D2,Z2,U2,V2) local S3,C3,D3,Z3,U3,V3,E,F,G,H,J,K,L,M,N,P:
>     E:=C1*Z2: F:=Z1*C2: G:=S1*D2: H:=D1*S2: J:=F-E: K:=F+E: L:=G-H: M:=G+H: N:=K*L: P:=J*M: S3:=(1/b)*J*K:
    C3:=(N-P)/2: Z3:=(N+P)/2: D3:=U1*V2-V1*U2: U3:=S3*D3: V3:=C3*Z3:
>     return S3,C3,D3,Z3,U3,V3:
```

```
> end proc:
> S3,C3,D3,Z3,U3,V3:=ADD_Imd_a01_opr(S1,C1,D1,Z1,U1,V1,S2,C2,D2,Z2,U2,V2):
> simplify([s3-S3/Z3,c3-C3/Z3,d3-D3/Z3,U3-S3*D3,V3-C3*Z3],[(C(s1,c1,d1)),(C(s2,c2,d2))]); #Check.
>
```

ADD_Imd_a02, $11\mathbf{M} + 9\mathbf{a}$, assumes $b = 1$.

```
> ADD_Imd_a02_opr:=proc(S1,C1,D1,Z1,U1,V1,S2,C2,D2,Z2,U2,V2) local S3,C3,D3,Z3,U3,V3,E,F,G,H,J,K,L,M,N,P:
>     E:=C1*Z2: F:=Z1*C2: G:=S1*D2: H:=D1*S2: J:=F-E: K:=F+E: L:=G-H: M:=G+H: N:=K*L: P:=J*M: S3:=J*K: C3:=(N
    -P)/2: Z3:=(N+P)/2: D3:=U1*V2-V1*U2: U3:=S3*D3: V3:=C3*Z3:
>     return S3,C3,D3,Z3,U3,V3:
> end proc:
> S3,C3,D3,Z3,U3,V3:=ADD_Imd_a02_opr(S1,C1,D1,Z1,U1,V1,S2,C2,D2,Z2,U2,V2):
> simplify([s3-S3/Z3,c3-C3/Z3,d3-D3/Z3,U3-S3*D3,V3-C3*Z3],[(C(s1,c1,d1)),(C(s2,c2,d2)),b-1]); #Check.
>
> ADD_Imd_a02_reg:=proc(S1,C1,D1,Z1,U1,V1,S2,C2,D2,Z2,U2,V2) local S3,C3,D3,Z3,U3,V3,t1,t2:
>     t1:=C1*Z2: Z3:=Z1*C2: t2:=Z3-t1: C3:=Z3+t1: t1:=S1*D2: D3:=D1*S2: S3:=t1-D3: D3:=t1+D3: t1:=C3*S3: D3:=
    t2*D3: S3:=t2*C3: C3:=t1-D3: C3:=C3/2: Z3:=t1+D3: Z3:=Z3/2: t1:=U1*V2: t2:=V1*U2: D3:=t1-t2: U3:=S3*D3: V3:=C3
    *Z3:
>     return S3,C3,D3,Z3,U3,V3:
> end proc:
> S3,C3,D3,Z3,U3,V3:=ADD_Imd_a02_reg(S1,C1,D1,Z1,U1,V1,S2,C2,D2,Z2,U2,V2):
> simplify([s3-S3/Z3,c3-C3/Z3,d3-D3/Z3,U3-S3*D3,V3-C3*Z3],[(C(s1,c1,d1)),(C(s2,c2,d2)),b-1]); #Check.
>
```

ADD_Imd_a03, $10\mathbf{M} + 9\mathbf{a}$, assumes $b = 1, Z_2 = 1$.

```
> ADD_Imd_a03_opr:=proc(S1,C1,D1,Z1,U1,V1,S2,C2,D2,Z2,U2,V2) local S3,C3,D3,Z3,U3,V3,F,G,H,J,K,L,M,N,P:
>     F:=Z1*C2: G:=S1*D2: H:=D1*S2: J:=F-C1: K:=F+C1: L:=G-H: M:=G+H: N:=K*L: P:=J*M: S3:=J*K: C3:=(N-P)/2: Z
    3:=(N+P)/2: D3:=U1*V2-V1*U2: U3:=S3*D3: V3:=C3*Z3:
>     return S3,C3,D3,Z3,U3,V3:
> end proc:
> S3,C3,D3,Z3,U3,V3:=ADD_Imd_a03_opr(S1,C1,D1,Z1,U1,V1,S2,C2,D2,Z2,U2,V2):
> simplify([s3-S3/Z3,c3-C3/Z3,d3-D3/Z3,U3-S3*D3,V3-C3*Z3],[(C(s1,c1,d1)),(C(s2,c2,d2)),b-1,Z2-1]); #Check.
>
> ADD_Imd_a03_reg:=proc(S1,C1,D1,Z1,U1,V1,S2,C2,D2,Z2,U2,V2) local S3,C3,D3,Z3,U3,V3,t1,t2:
>     Z3:=Z1*C2: t2:=Z3-C1: C3:=Z3+C1: t1:=S1*D2: D3:=D1*S2: S3:=t1-D3: D3:=t1+D3: t1:=C3*S3: D3:=t2*D3: S3:=
    t2*C3: C3:=t1-D3: C3:=C3/2: Z3:=t1+D3: Z3:=Z3/2: t1:=U1*V2: t2:=V1*U2: D3:=t1-t2: U3:=S3*D3: V3:=C3*Z3:
>     return S3,C3,D3,Z3,U3,V3:
> end proc:
> S3,C3,D3,Z3,U3,V3:=ADD_Imd_a03_reg(S1,C1,D1,Z1,U1,V1,S2,C2,D2,Z2,U2,V2):
> simplify([s3-S3/Z3,c3-C3/Z3,d3-D3/Z3,U3-S3*D3,V3-C3*Z3],[(C(s1,c1,d1)),(C(s2,c2,d2)),b-1,Z2-1]); #Check.
>
```

UADD_Imu_a00.

```
> s1:=S1/Z1: c1:=C1/Z1: d1:=D1/Z1: U1:=S1*C1: V1:=D1*Z1: s2:=S2/Z2: c2:=C2/Z2: d2:=D2/Z2: U2:=S2*C2: V2:=D2*Z
    2:
> s3:=(s1*c2*d2+c1*d1*s2)/(1-a*b*s1^2*s2^2):
> c3:=(c1*c2-b*s1*d1*s2*d2)/(1-a*b*s1^2*s2^2):
> d3:=(d1*d2-a*s1*c1*s2*c2)/(1-a*b*s1^2*s2^2):
>
> UADD_Imu_a00:=proc(S1,C1,D1,Z1,U1,V1,S2,C2,D2,Z2,U2,V2) local S3,C3,D3,Z3,U3,V3:
>     S3:=S1*Z1*C2*D2+C1*D1*S2*Z2:
>     C3:=C1*Z1*C2*Z2-b*S1*D1*S2*D2:
>     D3:=D1*Z1*D2*Z2-a*S1*C1*S2*C2:
>     Z3:=C1^2*Z2^2+b*S1^2*D2^2:
>     U3:=S3*C3:
>     V3:=D3*Z3:
>     return S3,C3,D3,Z3,U3,V3:
> end proc:
> S3,C3,D3,Z3,U3,V3:=UADD_Imu_a00(S1,C1,D1,Z1,U1,V1,S2,C2,D2,Z2,U2,V2):
> simplify([s3-S3/Z3,c3-C3/Z3,d3-D3/Z3,U3-S3*C3,V3-D3*Z3],[(C(s1,c1,d1)),(C(s2,c2,d2))]); #Check.
>
```

UADD_Imu_a01, $11\mathbf{M} + 2\mathbf{S} + 5\mathbf{D} + 13\mathbf{a}$.

```
> UADD_Imu_a01_opr:=proc(S1,C1,D1,Z1,U1,V1,S2,C2,D2,Z2,U2,V2) local S3,C3,D3,Z3,U3,V3,E,F,G,H,J,K:
>     E:=C1*Z2: F:=Z1*C2: G:=S1*D2: H:=D1*S2: J:=F*H: K:=E*G: S3:=(E+F)*(G+H)-J-K: C3:=(E-b*H)*(G+F)+b*J-K: D
    3:=(V1-a*U1)*(U2+V2)-J+a*K: Z3:=E^2+b*G^2: U3:=S3*C3: V3:=D3*Z3:
>     return S3,C3,D3,Z3,U3,V3:
> end proc:
> S3,C3,D3,Z3,U3,V3:=UADD_Imu_a01_opr(S1,C1,D1,Z1,U1,V1,S2,C2,D2,Z2,U2,V2):
> simplify([s3-S3/Z3,c3-C3/Z3,d3-D3/Z3,U3-S3*C3,V3-D3*Z3],[(C(s1,c1,d1)),(C(s2,c2,d2))]); #Check.
>
```

UADD_Imu_a02, $11\mathbf{M} + 1\mathbf{S} + 2\mathbf{D} + 15\mathbf{a}$, assumes $b = 1$.

```
> UADD_Imu_a02_opr:=proc(S1,C1,D1,Z1,U1,V1,S2,C2,D2,Z2,U2,V2) local S3,C3,D3,Z3,U3,V3,E,F,G,H,J,K,L,M,N,P:
>     E:=S1*D2: F:=C1*Z2: G:=D1*S2: H:=Z1*C2: J:=U1*V2: K:=V1*U2: S3:=(H+F)*(E+G)-J-K: C3:=(H+E)*(F-G)-J+K: D
    3:=(V1-a*U1)*(U2+V2)+a*J-K: Z3:=(H+G)^2-2*K: U3:=S3*C3: V3:=D3*Z3:
>     return S3,C3,D3,Z3,U3,V3:
> end proc:
> S3,C3,D3,Z3,U3,V3:=UADD_Imu_a02_opr(S1,C1,D1,Z1,U1,V1,S2,C2,D2,Z2,U2,V2):
> simplify([s3-S3/Z3,c3-C3/Z3,d3-D3/Z3,U3-S3*C3,V3-D3*Z3],[(C(s1,c1,d1)),(C(s2,c2,d2)),b-1]); #Check.
>
```

C.6 Scripts for Chapter 7

This Maple script verifies that (7.3) and (7.4) commute with the original point doubling formulas.

```
> b:=c^2: W:=(x,y)->y^2-(x^3+a*x+b): #The short Weierstrass curve, W.
> L:=(3*x1^2+a)/(2*y1): x3:=L^2-2*x1: y3:=L*(x1-x3)-y1: #Double on W.
> mu:=(y1+3*c)/(2*y1): sigma:=(a-3*x1^2)/(2*y1)^2: #Double on W with new formulas.
> delta:=(3*x1*(y1-3*c)*(y1+3*c)-a*(9*x1^2+a))/(2*y1)^3: #Double on W with new formulas.
> x3new:=x1*(mu-mu^2)+ a*sigma: y3new:=(y1-c)*mu^3+a*delta-c: #Double on W with new formulas.
> simplify(x3-x3new,[W(x1,y1)]); simplify(y3-y3new,[W(x1,y1)]); #Check.
```

This Maple script verifies that (7.5), (7.6), (7.7), and (7.8) commute with the original doubling and addition formulas.

```
> Q:=(x,y)->y^2-(c*x^3+1): #The curve considered in this work, Q.
> W:=(u,v)->v^2-(u^3+c^2): #The short Weierstrass curve, W.
> QtoW:=(x,y)->c*x,(x,y)->c*y: #The map from Q to W.
> WtoQ:=(u,v)->u/c,(u,v)->v/c: #The map from W to Q.
> ##Verify the correctness of point additon formulas.
> u1,v1:=QtoW(x1,y1): u2,v2:=QtoW(x2,y2): #Map the points (x1,y1) and (x2,y2) on Q to W.
> L:=(v1-v2)/(u1-u2): u3:=L^2-u1-u2: v3:=L*(u1-u3)-v1: #Add on W with the original formulas.
> x3,y3:=WtoQ(u3,v3): #Map the sum (u3,v3) on W to Q.
> simplify(W(u3,v3),[Q(x1,y1),Q(x2,y2)]); #Check.
> Lnew:=(y1-y2)/(x1-x2): x3new:=c^(-1)*Lnew^2-x1-x2: y3new:=Lnew*(x1-x3)-y1: ##Add on Q.
> simplify(x3-x3new,[Q(x1,y1),Q(x2,y2)]); simplify(y3-y3new,[Q(x1,y1),Q(x2,y2)]); #Check.
> unassign('Lnew','L','u2','v2','u3','v3','x3','y3','x3new','y3new');
> ##Verify the correctness of point doubling formulas.
> L:=3*u1^2/(2*v1): u3:=L^2-2*u1: v3:=L*(u1-u3)-v1: #Double on W with the original formulas.
> x3,y3:=WtoQ(u3,v3): #Map the sum (u3,v3) on W to Q.
> simplify(W(u3,v3),[Q(x1,y1)]); #Check.
> mu:=(y1+3)/(2*y1): x3new:=x1*(mu-mu^2): y3new:=(y1-1)*mu^3-1: #Double on Q.
> simplify(x3-x3new,[Q(x1,y1)]); simplify(y3-y3new,[Q(x1,y1)]); #Check.
```

This Maple script verifies the correctness of (7.9), (7.10), and (7.11).

```
> Q:=(x,y)->y^2-(c*x^3+1): #The curve considered in this work, Q.
> W:=(u,v)->v^2-(u^3+c^2): #The short Weierstrass curve, W.
> QtoW:=(x,y)->c*x,(x,y)->c*y: #The maps from Q to W.
> WtoQ:=(u,v)->u/c,(u,v)->v/c: #The maps from W to Q.
> ##Verify the correctness of the line formulas for addition.
```

```
> u1,v1:=QtoW(x1,y1): u2,v2:=QtoW(x2,y2): uQ,vQ:=QtoW(xQ,yQ): ##(xi,yi) on Q to (ui,vi) on W.
> L:=(v1-v2)/(u1-u2): l:=L*(u1-uQ)+vQ-v1: v:=uQ-(L^2-u1-u2): #Compute the addition-line on W.
> Lnew:=(y1-y2)/(x1-x2): gadd:=c*(Lnew*(x2-xQ)-y2+yQ)/(c*(x1+x2+xQ)-Lnew^2): #New line on Q.
> simplify(l/v-gadd,[Q(x1,y1),Q(x2,y2),Q(xQ,yQ)]); #Check.
> ##Verify the correctness of the line formulas for doubling.
> L:=3*u1^2/(2*v1): l:=L*(u1-uQ)+vQ-v1: v:=uQ-(L^2-2*u1): #Compute the doubling-line on W.
> gdbl:=2*c*y1*(x1-xQ)^2/(x1^2*(3*c*xQ)-y1^2+3+2*y1*yQ): #New line on Q.
> simplify(l/v-gdbl,[Q(x1,y1),Q(xQ,yQ)]); #Check.
> ##Verify the correctness of the line formulas for the sum of negatives.
> l:=uQ-u1: v:=1: #The vertical line on W.
> gvert:=-c*(x1-xQ): #The new line on Q.
> simplify(l/v-gvert,[Q(x1,y1),Q(x2,y2),Q(xQ,yQ)]); #Check.
```

This Maple script verifies the correctness of (7.14) and (7.15).

```
> Q:=(X,Y,Z)->Y^2*Z-(c*X^3+Z^3): x1:=X1/Z1: y1:=Y1/Z1:
> x3:=x1*(y1^2-9)/(2*y1)^2: y3:=(y1-1)*(y1+3)^3/(2*y1)^3-1:
> Line:=x1^2*(3*c*xQ)-y1^2+3-2*y1*yQ:
> ##Point doubling formulas in homogenous projective coordinates.
> X3:=2*X1*Y1*(Y1^2-9*Z1^2):
> Y3:=(Y1-Z1)*(Y1+3*Z1)^3-8*Z1*Y1^3:
> Z3:=(2*Y1*Z1)*(2*Y1)^2:
> gDBL:=X1^2*(3*c*xQ)-Y1^2+3*Z1^2-2*Y1*Z1*yQ: #Line formulas.
> simplify(x3-X3/Z3,[Q(X1,Y1,Z1)]); simplify(y3-Y3/Z3,[Q(X1,Y1,Z1)]); #Check.
> factor(Line-gDBL/Z1^2); #Check.
```

This Maple script shows how to schedule operations for (7.14). The point doubling without line computation needs $4\mathbf{M} + 3\mathbf{S} + 0\mathbf{D}$.

```
> Q:=(X,Y,Z)->Y^2*Z-(c*X^3+Z^3):
> ##Point doubling formulas with register allocations.
> X3:=2*X1: X3:=X3*Y1: Z3:=3*Z1: t1:=Y1+Z3: t1:=t1^2: Y3:=Y1^2: Z3:=Z3^2: t2:=Y3-Z3:
> t2:=3*t2: X3:=X3*t2: t2:=t2+Z3: t2:=t2+Z3: Z3:=Y3+Z3: Z3:=t1-Z3: t2:=t2+Z3: Z3:=Y3*Z3:
> Z3:=4*Z3: Y3:=t1*t2: Y3:=Y3-Z3:
> simplify(Q(X3,Y3,Z3),[Q(X1,Y1,Z1)]); #Check.
```

This Maple script shows how to schedule operations for (7.14) and (7.15). Multiplication with `c1` or with y_Q counts as $(k/2)\mathbf{M}$. Assume that `c1` is precomputed. The point doubling with line computation needs $5\mathbf{M} + 5\mathbf{S}$ if $k = 2$ or more generally $(k + 3)\mathbf{M} + 5\mathbf{S}$ if k is even.

```
> Q:=(X,Y,Z)->Y^2*Z-(c*X^3+Z^3):
> Line:=X1^2*(3*c*xQ)-Y1^2+3*Z1^2-2*Y1*Z1*yQ:
> c1:=3*c*xQ: #Precomputed value.
> ##Point doubling formulas and line computation with register allocations.
> t1:=X1+Y1: t2:=Y1+Z1: t1:=t1^2: t2:=t2^2: X3:=X1^2: Y3:=Y1^2: Z3:=Z1^2: t1:=t1-X3:
> t1:=t1-Y3: t2:=t2-Y3: t2:=t2-Z3: Z3:=3*Z3: t3:=Y3-Z3: gDBL:=X3*c1-t3-t2*yQ:
> t3:=t3+t2: t4:=3*Z3: X3:=Y3-t4: X3:=t1*X3: t1:=3*t2: t2:=t1+t2:  Z3:=t2*Y3:
> Y3:=Y3+t4: t1:=t1+Y3: Y3:=t3*t1: Y3:=Y3-Z3:
simplify(Q(X3,Y3,Z3),[Q(X1,Y1,Z1)]); simplify(Line-gDBL); #Check.
```

This Maple script verifies the correctness of (7.16) and (7.17).

```
> Q1:=(X,Y,Z)->Y^2*Z-(c*X^3+Z^3): x1:=X1/Z1: y1:=Y1/Z1: x2:=X2/Z2: y2:=Y2/Z2:
> L:=(y1-y2)/(x1-x2): x3:=c^(-1)*L^2-x1-x2: y3:=L*(x1-x3)-y1:
> Line:=(y1-y2)*(x2-xQ)-(x1-x2)*(y2-yQ):
> ##Point addition formulas in homogenous projective coordinates.
> X3:=(X1*Z2-Z1*X2)*(Z1*Z2*(Y1*Z2-Z1*Y2)^2-c*(X1*Z2+Z1*X2)*(X1*Z2-Z1*X2)^2):
> Y3:=(Y1*Z2-Z1*Y2)*(c*(2*X1*Z2+Z1*X2)*(X1*Z2-Z1*X2)^2-Z1*Z2*(Y1*Z2-Z1*Y2)^2) -
    c*Y1*Z2*(X1*Z2-Z1*X2)^3:
> Z3:=c*Z1*Z2*(X1*Z2-Z1*X2)^3:
> gADD:=(Y1*Z2-Z1*Y2)*(X2-xQ*Z2)-(X1*Z2-Z1*X2)*Y2+(X1*Z2-Z1*X2)*Z2*yQ: #Line formulas.
> simplify(x3-X3/Z3,[Q1(X1,Y1,Z1),Q1(X2,Y2,Z2)]); #Check.
> simplify(y3-Y3/Z3,[Q1(X1,Y1,Z1),Q1(X2,Y2,Z2)]); factor(Line-gADD/Z1/Z2^2); #Check.
```

This Maple script shows how to schedule operations for (7.16) and (7.17) with $Z_2 = 1$.

```
> Z2:=1: Q:=(X,Y,Z)->Y^2*Z-(c*X^3+Z^3):
> Line:=(Y1*Z2-Z1*Y2)*(X2-xQ*Z2)-(X1*Z2-Z1*X2)*(Y2-yQ*Z2):
> c1:=X2-xQ: c2:=Y2-yQ: #Precomputed values.
> ##Point addition formulas and line computation with register allocations.
> t1:=Z1*X2: t1:=X1-t1: t2:=Z1*Y2: t2:=Y1-t2: gADD:=c1*t2-t1*Y2+t1*yQ:
> t3:=t1^2: t3:=c*t3: X3:=t3*X1: t3:=t1*t3: t4:=t2^2: t4:=t4*Z1: t4:=t3+t4:
> t4:=t4-X3: t4:=t4-X3: X3:=X3-t4: t2:=t2*X3: Y3:=t3*Y1: Y3:=t2-Y3: X3:=t1*t4: Z3:=Z1*t3:
> simplify(Q(X3,Y3,Z3),[Q(X1,Y1,Z1),Q(X2,Y2,Z2)]); simplify(Line-gADD); #Check.
```

This Maple script shows how to schedule operations for (7.16) and (7.17).

```
> Q:=(X,Y,Z)->Y^2*Z-(c*X^3+Z^3):
> Line:=(Y1*Z2-Z1*Y2)*(X2-xQ*Z2)-(X1*Z2-Z1*X2)*(Y2-yQ*Z2):
> c1:=X2-xQ*Z2: c2:=Y2-yQ*Z2: #Precomputed values.
> ##Point addition formulas and line computation with register allocations.
> t1:=Z1*X2: X3:=X1*Z2: t1:=X3-t1: t2:=Z1*Y2: Y3:=Y1*Z2: t2:=Y3-t2:
> gADD:=c1*t2-t1*Y2+t1*Z2*yQ:
> Z3:=Z1*Z2: t3:=t1^2: t3:=c*t3: X3:=t3*X3: t3:=t1*t3: t4:=t2^2: t4:=t4*Z3: t4:=t3+t4:
> t4:=t4-X3: t4:=t4-X3: X3:=X3-t4: t2:=t2*X3: Y3:=t3*Y3: Y3:=t2-Y3: X3:=t1*t4: Z3:=Z3*t3:
> simplify(Q(X3,Y3,Z3),[Q(X1,Y1,Z1),Q(X2,Y2,Z2)]); simplify(Line-gADD); #Check.
```

Bibliography

[AL96] W. Adams and P. Loustaunau, *An introduction to Gröbner bases*, American Mathematical Society, 1996.

[ALNR09] Christophe Arène, Tanja Lange, Michael Naehrig, and Christophe Ritzenthaler, *Faster pairing computation*, Cryptology ePrint Archive, 2009, `http://eprint.iacr.org/2009/155`.

[Ame01] American National Standards Institute (ANSI), *Public Key Cryptography for the Financial Services Industry, Key Agreement and Key Transport Using Elliptic Curve Cryptography*, 2001.

[Ame05] ———, *Public Key Cryptography for the Financial Services Industry, The Elliptic Curve Digital Signature Algorithm (ECDSA)*, 2005.

[AMNS06] Seigo Arita, Kazuto Matsuo, Koh-ichi Nagao, and Mahoro Shimura, *A Weil descent attack against elliptic curve cryptosystems over quartic extension fields*, IEICE Trans Fundamentals **E89-A** (2006), no. 5, 1246–1254.

[Ari00] Seigo Arita, *Weil descent of elliptic curves over finite fields of characteristic three*, ASIACRYPT 2000, LNCS, vol. 1976, Springer, Berlin / Heidelberg, 2000, pp. 248–258.

[Ava05] Roberto M. Avanzi, *A note on the signed sliding window integer recoding and its left-to-right analogue*, SAC 2004, LNCS, vol. 3357, Springer, 2005, pp. 130–143.

[BBJ+08] Daniel J. Bernstein, Peter Birkner, Marc Joye, Tanja Lange, and Christiane Peters, *Twisted Edwards curves*, AFRICACRYPT 2008, LNCS, vol. 5023, Springer, 2008, pp. 389–405.

[BBLP07] Daniel J. Bernstein, Peter Birkner, Tanja Lange, and Christiane Peters, *Optimizing double-base elliptic-curve single-scalar multiplication*, INDOCRYPT 2007, LNCS, vol. 4859, Springer, 2007, pp. 167–182.

[BBLP08] ———, *ECM using Edwards curves*, Cryptology ePrint Archive, Report 2008/016, 2008, `http://eprint.iacr.org/`.

[BCC+09] Daniel J. Bernstein, Hsueh-Chung Chen, Ming-Shing Chen, Chen-Mou Cheng, Chun-Hung Hsiao, Tanja Lange, Zong-Cing Lin, and Bo-Yin Yang, *The billion-mulmod-per-second PC*, Workshop record of SHARCS'09, 2009.

[BCP97] Wieb Bosma, John Cannon, and Catherine Playoust, *The MAGMA algebra system. I. The user language.*, Journal of Symbolic Computation **24** (1997), no. 3-4, 235–265.

[Ber06a] Daniel J. Bernstein, *Addition laws on elliptic curves*, 2006, `http://cr.yp.to/talks/2009.09.22/slides.pdf`.

[Ber06b] ———, *Curve25519: New Diffie-Hellman speed records*, PKC, LNCS, vol. 3958, Springer, 2006, pp. 207–228.

[BF03] Dan Boneh and Matthew K. Franklin, *Identity-based encryption from the Weil pairing*, SIAM J. Comput. **32** (2003), no. 3, 586–615.

[BGHS04] Paulo S. L. M. Barreto, Steven D. Galbraith, Colm Ó' Héigeartaigh, and Michael Scott, *Efficient pairing computation on supersingular abelian varieties*, Cryptology ePrint Archive, Report 2004/375, 2004, `http://eprint.iacr.org/2004/375`.

[BGHS07] ———, *Efficient pairing computation on supersingular abelian varieties*, Des. Codes Cryptography **42** (2007), no. 3, 239–271.

[BGM+93] Ian F. Balke, XuHong Gao, Ronald C. Mullin, Scott A. Vanstone, and Tomik Yaghoobian, *Applications of finite fields*, Kluwer Academic Publishers, Boston/Dordrecht/London, 1993.

[BJ02] Eric Brier and Marc Joye, *Weierstraß elliptic curves and side-channel attacks*, PKC 2002, LNCS, vol. 2274, Springer, 2002, pp. 335–345.

[BJ03a] Olivier Billet and Marc Joye, *The Jacobi model of an elliptic curve and side-channel analysis*, AAECC-15, LNCS, vol. 2643, Springer, 2003, pp. 34–42.

[BJ03b] Eric Brier and Marc Joye, *Fast point multiplication on elliptic curves through isogenies*, AAECC-15, LNCS, vol. 2643, Springer, 2003, pp. 43–50.

[BK98] R. Balasubramanian and Neal Koblitz, *The improbability that an elliptic curve has subexponential discrete log problem under the Menezes - Okamoto - Vanstone algorithm*, Journal of Cryptology **11** (1998), no. 2, 141–145.

[BKL09] Daniel J. Bernstein, David Kohel, and Tanja Lange, *Twisted Hessian curves*, Explicit-Formulas Database, 2009, `http://www.hyperelliptic.org/EFD/g1p/auto-twistedhessian.html`.

[BKLS02] Paulo S. L. M. Barreto, Hae Yong Kim, Ben Lynn, and Michael Scott, *Efficient algorithms for pairing-based cryptosystems*, CRYPTO 2002, LNCS, vol. 2442, Springer, 2002, pp. 354–369.

[BL95] Wieb Bosma and H. W. Lenstra Jr., *Complete systems of two addition laws for elliptic curves*, Journal of Number Theory **53** (1995), 229–240.

[BL07a] Daniel J. Bernstein and Tanja Lange, *Explicit-formulas database*, 2007, `http://www.hyperelliptic.org/EFD`.

[BL07b] ———, *Faster addition and doubling on elliptic curves*, ASIACRYPT 2007, LNCS, vol. 4833, Springer, 2007, pp. 29–50.

[BL07c] ———, *Inverted Edwards coordinates*, AAECC-17, LNCS, vol. 4851, Springer, 2007, pp. 20–27.

[BL08] ———, *Analysis and optimization of elliptic-curve single-scalar multiplication*, Finite Fields and Applications Fq8, Contemporary Mathematics, vol. 461, American Mathematical Society, 2008, pp. 1–18.

[BLR08] Daniel J. Bernstein, Tanja Lange, and Reza Rezaeian Farashahi, *Binary Edwards curves*, CHES 2008, LNCS, vol. 5154, Springer, 2008, pp. 244–265.

[BLS03] Paulo S. L. M. Barreto, Ben Lynn, and Michael Scott, *Constructing elliptic curves with prescribed embedding degrees*, Security in Communication Networks, LNCS, vol. 2576, Springer, 2003, pp. 257–267.

[BLS04a] ———, *Efficient implementation of pairing-based cryptosystems*, Journal of Cryptology **17** (2004), no. 4, 321–334.

[BLS04b] ———, *On the selection of pairing-friendly groups*, SAC 2003, LNCS, vol. 3006, Springer, 2004, pp. 17–25.

[BLS04c] Dan Boneh, Ben Lynn, and Hovav Shacham, *Short signatures from the Weil pairing*, Journal of Cryptology **17** (2004), no. 4, 297–319.

[BRCMC+09] Daniel J. Bernstein, Tien Ren Chen, Chen Mou Cheng, Tanja Lange, and Bo Yin Yang, *ECM on graphics cards*, EUROCRYPT 2009, LNCS, vol. 5479, Springer, 2009, pp. 483–501.

[Bre80] Richard P. Brent, *An improved Monte Carlo factorization algorithm*, BIT Numerical Mathematics **20** (1980), no. 2, 176–184.

[BS09] Naomi Benger and Michael Scott, *Constructing tower extensions for the implementation of pairing-based cryptography*, Cryptology ePrint Archive, Report 2009/556, 2009, `http://eprint.iacr.org/`.

[BSS99] I. F. Blake, G. Seroussi, and N. P. Smart, *Elliptic curves in cryptography*, Cambridge University Press, July 1999.

[Buc65] Bruno Buchberger, *An algorithm for finding the basis elements of the residue class ring of a zero dimensional polynomial ideal*, Ph.D. thesis, University of Innsbruck, 1965.

[BW88] Johannes Buchmann and Hugh C. Williams, *A key-exchange system based on imaginary quadratic fields*, Journal of Cryptology **1** (1988), no. 2, 107–118.

[BW93] Thomas Becker and Volker Weispfenning, *Gröbner bases*, A Computational Approach to Commutative Algebra Series: Graduate Texts in Mathematics, vol. 141, Springer, 1993.

[BW05] Friederike Brezing and Annegret Weng, *Elliptic curves suitable for pairing based cryptography*, Des. Codes Cryptography **37** (2005), no. 1, 133–141.

[BZ09] Richard Brent and Paul Zimmermann, *Modern computer arithmetic*, In preparation, November 2009, version 0.4, `http://www.loria.fr/~zimmerma/mca/mca-0.4.pdf`.

[CC86] David V. Chudnovsky and Gregory V. Chudnovsky, *Sequences of numbers generated by addition in formal groups and new primality and factorization tests*, Advances in Applied Mathematics **7** (1986), no. 4, 385–434.

[CF05] Henri Cohen and Gerhard Frey (eds.), *Handbook of elliptic and hyperelliptic curve cryptography*, CRC Press, 2005.

[CLN09] Craig Costello, Tanja Lange, and Michael Naehrig, *Faster pairing computations on curves with high-degree twists*, Cryptology ePrint Archive, 2009, `http://eprint.iacr.org/2009/615`.

[CLO07] David A. Cox, John B. Little, and Don O'Shea, *Ideals, varieties, and algorithms: An introduction to computational algebraic geometry and commutative algebra*, Undergraduate Texts in Mathematics, Springer, 3rd ed. 2007.

[CMO98] Henri Cohen, Atsuko Miyaji, and Takatoshi Ono, *Efficient elliptic curve exponentiation using mixed coordinates*, ASIACRYPT'98, LNCS, vol. 1514, Springer, 1998, pp. 51–65.

[Coh93] Henri Cohen, *A course in computational algebraic number theory*, Springer-Verlag New York, Inc., New York, NY, USA, 1993.

[CV09] Wouter Castryck and Frederik Vercauteren, *Toric forms of elliptic curves and their arithmetic*, preprint, 2009.

[Dew98] L. Dewaghe, *Remarks on the Schoof-Elkies-Atkin algorithm*, Mathematics of Computation **67** (1998), no. 223, 1247–1252.

[DGM99] I. Duursma, Pierrick Gaudry, and François Morain, *Speeding up the discrete log computation on curves with automorphisms*, ASIACRYPT'99 (London, UK), Springer-Verlag, 1999, pp. 103–121.

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

[DI06] Christophe Doche and Laurent Imbert, *Extended double-base number system with applications to elliptic curve cryptography.*, INDOCRYPT'06, LNCS, vol. 4329, Springer, 2006, pp. 335–348.

[Die01] Claus Diem, *A study on theoretical and practical aspects of Weil-restriction of varieties*, Ph.D. thesis, University of Essen, Germany, 2001.

[Die03] ———, *The GHS-attack in odd characteristic*, Journal of the Ramanujan Mathematical Society **18** (2003), 1–32.

[DIK06] Christophe Doche, Thomas Icart, and David R. Kohel, *Efficient scalar multiplication by isogeny decompositions*, PKC 2006, LNCS, vol. 3958, Springer, 2006, pp. 191–206.

[DL06] Wolfram Decker and Christoph Lossen, *Computing in algebraic geometry, a quick start using singular*, Algorithms and Computation in Mathematics, vol. 16, Springer, Berlin Heidelberg, 2006.

[dR94] Peter de Rooij, *Efficient exponentiation using precomputation and vector addition chains*, EUROCRYPT'94, 1994, pp. 389–399.

[DS08] M. Prem Das and Palash Sarkar, *Pairing computation on twisted Edwards form elliptic curves*, Pairing'08, Lecture Notes in Mathematics, vol. 5209, Springer, 2008, pp. 192–210.

[Duq07] Sylvain Duquesne, *Improving the arithmetic of elliptic curves in the Jacobi model*, Information Processing Letters **104** (2007), no. 3, 101–105.

[Edw07] Harold M. Edwards, *A normal form for elliptic curves*, Bulletin of the AMS **44** (2007), no. 3, 393–422.

[EG02] Andreas Enge and Pierrick Gaudry, *A general framework for subexponential discrete logarithm algorithms*, Acta Arithmetica (2002), no. 1, 83–103.

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

[ELM03] Kirsten Eisenträger, Kristin Lauter, and Peter L. Montgomery, *Fast elliptic curve arithmetic and improved Weil pairing evaluation*, CT-RSA 2003, LNCS, vol. 2612, Springer, 2003, pp. 343–354.

[FNW09] Rongquan Feng, Menglong Nie, and Hongfeng Wu, *Twisted jacobi intersections curves*, Cryptology ePrint Archive, Report 2009/597, 2009, `http://eprint.iacr.org/`.

[FO90] Philippe Flajolet and Andrew M. Odlyzko, *Random mapping statistics*, EUROCRYPT '89 (New York, USA), Springer-Verlag, 1990, pp. 329–354.

[FR94] Gerhard Frey and Hans-Georg Rück, *A remark concerning m-divisibility and the discrete logarithm in the divisor class group of curves*, Mathematics of Computation **62** (1994), no. 206, 865–874.

[Fra76] John B. Fraleigh, *A first course in abstract algebra (addison-wesley series in mathematics)*, Addison-Wesley Pub. Co, July 1976.

[Fre01] Gerhard Frey, *Applications of arithmetical geometry to cryptographic constructions*, Proceedings of the Fifth International Conference on Finite Fields and Applications (University of Augsburg, Germany), Springer Verlag, 2001, pp. 128–161.

[FST06] David Freeman, Michael Scott, and Edlyn Teske, *A taxonomy of pairing-friendly elliptic curves*, Cryptology ePrint Archive, Report 2006/372, 2006, `http://eprint.iacr.org/2006/372`.

[Ful69] William Fulton, *An introduction to algebraic geometry*, A. W. Benjamin Publishing Company, New York, 1969.

[Gal01] Steven D. Galbraith, *Supersingular curves in cryptography*, ASIACRYPT'01 (London, UK), Springer-Verlag, 2001, pp. 495–513.

[Gal05] ———, *Pairings*, London Mathematics Society Lecture Note Series, vol. 317, pp. 183–213, Cambridge University Press, 2005.

[Gau00] Pierrick Gaudry, *An algorithm for solving the discrete log problem on hyperelliptic curves*, EUROCRYPT'00 **1807** (2000), 19–34.

[Gau04] ———, *Index calculus for abelian varieties and the elliptic curve discrete logarithm problem*, Cryptology ePrint Archive, Report 2004/073, March 2004.

[Gau06] ———, *Variants of the Montgomery form based on theta functions*, 2006, Computational Challenges Arising in Algorithmic Number Theory and Cryptography Workshop.

[GHS02a] Steven D. Galbraith, Florian Hess, and Nigel P. Smart, *Extending the GHS Weil descent attack*, EUROCRYPT'02, LNCS, vol. 2332, Springer, Berlin / Heidelberg, 2002, pp. 29–44.

[GHS02b] Pierrick Gaudry, Florian Hess, and Nigel P. Smart, *Constructive and destructive facets of Weil descent on elliptic curves*, Journal of Cryptology: The journal of the International Association for Cryptologic Research **15** (2002), no. 1, 19–46.

[GL08] Pierrick Gaudry and David Lubicz, *The arithmetic of characteristic 2 Kummer surfaces*, Cryptology ePrint Archive, Report 2008/133, 2008, `http://eprint.iacr.org/`.

[GL09] ———, *The arithmetic of characteristic 2 Kummer surfaces and of elliptic Kummer lines*, Finite Fields and Their Applications **15** (2009), no. 2, 246–260.

[GLS09a] Steven D. Galbraith, Xibin Lin, and Michael Scott, *Endomorphisms for faster elliptic curve cryptography on a large class of curves*, EUROCRYPT 2009, LNCS, vol. 5479, Springer, 2009, pp. 518–535.

[GLS09b] ———, *Endomorphisms for faster elliptic curve cryptography on a large class of curves*, Cryptology ePrint Archive, Report 2008/194, extended version, 17-Sep-2009, 2009, http://eprint.iacr.org.

[GLV00] Robert Gallant, Robert Lambert, and Scott A. Vanstone, *Improving the parallelized Pollard lambda search on anomalous binary curves*, Mathematics of Computation **69** (2000), no. 232, 1699–1705.

[GLV01] Robert P. Gallant, Robert J. Lambert, and Scott A. Vanstone, *Faster point multiplication on elliptic curves with efficient endomorphisms*, CRYPTO'01, vol. 2139, Springer-Verlag, 2001, pp. 190–200.

[GM00] Steven D. Galbraith and James McKee, *The probability that the number of points on an elliptic curve over a finite field is prime*, J. London Math. Soc. **2** (2000), no. 62, 671–684.

[GPS09] G.-M. Greuel, G. Pfister, and H. Schönemann, SINGULAR *3-1-0 — A computer algebra system for polynomial computations*, 2009, `http://www.singular.uni-kl.de`.

[GS99] Steven D. Galbraith and Nigel P. Smart, *A cryptographic application of Weil descent*, Proceedings of the 7th IMA International Conference on Cryptography and Coding (London, UK), Springer-Verlag, 1999, pp. 191–200.

[GS08] Steven D. Galbraith and Michael Scott, *Exponentiation in pairing-friendly groups using homomorphisms*, Pairing 2008, LNCS, vol. 5209, Springer, 2008, pp. 211–224.

[GT07] Pierrick Gaudry and Emmanuel Thomé, *The mpFq library and implementing curve-based key exchanges, SPEED 2007, pp.49–64*, 2007, `http://www.loria.fr/~gaudry/publis/mpfq.pdf`.

[Har77] Robin Hartshorne, *Algebraic geometry*, Springer-Verlag, Inc., New York, 1977.

[Hes02] Florian Hess, *Computing Riemann-Roch spaces in algebraic function fields and related topics*, J. Symb. Comput. **33** (2002), no. 4, 425–445.

[Hes03] ———, *The GHS attack revisited*, International Conference on the Theory and Applications of Cryptographic Techniques-EUROCRPYT 2003, LNCS, vol. 2656, Springer, Berlin / Heidelberg, 2003, pp. 374–387.

[Hes04] ———, *Generalising the GHS attack on the elliptic curve discrete logarithm problem*, LMS Journal of Computation and Mathematics **7** (2004), 167–192.

[HKT00] Ming-Deh A. Huang, Ka Lam Kueh, and Ki-Seng Tan, *Lifting elliptic curves and solving the elliptic curve discrete logarithm problem*, Algorithmic Number Theory Symposium-ANTS'00, 2000, pp. 377–384.

[HMCD04] Yvonne R. Hitchcock, Paul Montague, Gary Carter, and Ed Dawson, *The efficiency of solving multiple discrete logarithm problems and the implications for the security of fixed elliptic curves*, International Journal of Information Security **3** (2004), no. 2, 86–98.

[HMV03] Darrel Hankerson, Alfred J. Menezes, and Scott A. Vanstone, *Guide to elliptic curve cryptography*, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2003.

[HNM98] Toshio Hasegawa, Junko Nakajima, and Mitsuru Matsui, *A practical implementation of elliptic curve cryptosystems over GF(p) on a 16-bit microcomputer*, PKC 2002, LNCS, vol. 1431, Springer, 1998, pp. 182–194.

[HSV06] Florian Hess, Nigel P. Smart, and Frederik Vercauteren, *The Eta pairing revisited*, IEEE Transactions on Information Theory **52** (2006), no. 10, 4595–4602.

[Hun74] Thomas W. Hungerford, *Algebra*, Springer, New York, 1974.

[IJ08] Sorina Ionica and Antoine Joux, *Another approach to pairing computation in Edwards coordinates*, Cryptology ePrint Archive, 2008, `http://eprint.iacr.org/2008/292`.

[Ins00] Institute of Electrical and Electronics Engineers (IEEE), *Standard Specifications For Public-Key Cryptography. IEEE P1363*, 2000.

[Int06] International Organization for Standards (ISO), *Information technology - Security techniques - Digital signatures with appendix - Part 3: Discrete logarithm based mechanisms*, 2006.

[Jac29] Carl Gustav Jakob Jacobi, *Fundamenta nova theoriae functionum ellipticarum*, Sumtibus Fratrum Borntræger, 1829.

[JKS+00] Michael J. Jacobson, Neal Koblitz, Joseph H. Silverman, Andreas Stein, and Edlyn Teske, *Analysis of the xedni calculus attack*, Designs, Codes and Cryptography **20** (2000), no. 1, 41–64.

[Jou04] Antoine Joux, *A one round protocol for tripartite Diffie-Hellman*, Journal of Cryptology **17** (2004), no. 4, 263–276.

[JQ01] Marc Joye and Jean Jacques Quisquater, *Hessian elliptic curves and side-channel attacks*, CHES 2001, vol. 2162, LNCS, no. Generators, Springer, 2001, pp. 402–410.

[JY03] Marc Joye and Sung-Ming Yen, *The Montgomery powering ladder*, CHES 2002, LNCS, vol. 2523, Springer, 2003, pp. 291–302.

[KM05] Neal Koblitz and Alfred Menezes, *Pairing-based cryptography at high security levels*, Cryptography and Coding, LNCS, vol. 3796, Springer, 2005, pp. 13–36.

[Knu97] Donald E. Knuth, *The art of computer programming, volume 2 (3rd ed.): Seminumerical algorithms*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.

[Kob87] Neal Koblitz, *Elliptic curve cryptosystems*, Mathematics of Computation **48** (1987), no. 177, 203–209.

[Kob89] ———, *Hyperelliptic cryptosystems*, Journal of Cryptology **1** (1989), no. 3, 139–150.

[KS01] Fabian Kuhn and René Struik, *Random walks revisited: Extensions of Pollard's rho algorithm for computing multiple discrete logarithms*, Revised Papers from the 8th Annual International Workshop on Selected Areas in Cryptography-SAC'01 (London, UK), Springer-Verlag, 2001, pp. 212–229.

[Len87] Hendrik W. Lenstra, *Factoring integers with elliptic curves*, The Annals of Mathematics **126** (1987), no. 3, 649–673.

[LG09] Patrick Longa and Catherine Gebotys, *Novel precomputation schemes for elliptic curve cryptosystems*, ACNS '09, to appear, LNCS, Springer, 2009.

[LL94] Chae Hoon Lim and Pil Joong Lee, *More flexible exponentiation with precomputation*, CRYPTO '94 (London, UK), Springer-Verlag, 1994, pp. 95–107.

[LL03] Reynald Lercier and David Lubicz, *Counting points on elliptic curves over finite fields of small characteristic in quasi quadratic time*, Advances in Cryptology-EUROCRPYT'03, International Conference on the Theory and Applications of Cryptographic Techniques **2656** (2003), 360–373.

[LLP08] Eunjeong Lee, Hyang-Sook Lee, and Cheol-Min Park, *Efficient and generalized pairing computation on abelian varieties*, Cryptology ePrint Archive, Report 2008/040, 2008, `http://eprint.iacr.org/2008/040`.

[LLX05] Jun Quan Li, Mu Lan Liu, and Liang Liang Xiao, *Solving the multi-discrete logarithm problems over a group of elliptic curves with prime order*, Acta Mathematica Sinica **21** (2005), no. 6, 1443–1450.

[LN96] Rudolf Lidl and Harald Niederreiter, *Finite fields (Encyclopedia of mathematics and its applications)*, Cambridge University Press, October 1996.

[LS01] Pierre Yvan Liardet and Nigel P. Smart, *Preventing SPA/DPA in ECC systems using the Jacobi form.*, CHES 2001, LNCS, vol. 2162, Springer, 2001, pp. 391–401.

[MAP08] *Maple 12, Waterloo Maple Inc.*, 2008, `http://www.maplesoft.com/`.

[McC88] K. S. McCurley, *A key distribution system equivalent to factoring*, Journal of Cryptology **1** (1988), no. 2, 95–105.

[Mil86] Victor S. Miller, *Use of elliptic curves in cryptography*, CRYPTO'85, LNCS, vol. 218, Springer, 1986, pp. 417–426.

[Mil04] ———, *The Weil pairing, and its efficient calculation*, Journal of Cryptology **17** (2004), no. 4, 235–261.

[MKHO07] Seiichi Matsuda, Naoki Kanayama, Florian Hess, and Eiji Okamoto, *Optimised versions of the Ate and twisted Ate pairings*, Cryptography and Coding, LNCS, vol. 4887, Springer, 2007, pp. 302–312.

[MM99] Henry McKean and Victor Moll, *Elliptic curves: Function theory, geometry, arithmetic*, Cambridge University Press, 1999.

[MMT01] Markus Maurer, Alfred J. Menezes, and Edlyn Teske, *Analysis of the GHS Weil descent attack on the ECDLP over characteristic two finite fields of composite degree*, INDOCRYPT'01 (London, UK), Springer-Verlag, 2001, pp. 195–213.

[Möl03] Bodo Möller, *Improved techniques for fast exponentiation*, ICISC 2002, LNCS, vol. 2587, Springer, 2003, pp. 298–312.

[Mon85] Peter L. Montgomery, *Modular multiplication without trial division*, Mathematics of Computation **44** (1985), no. 170, 519–521.

[Mon87] ———, *Speeding the Pollard and elliptic curve methods of factorization*, Mathematics of Computation **48** (1987), no. 177, 243–264.

[MOV96] Alfred J. Menezes, Paul C. Van Oorschot, and Scott A. Vanstone, *Handbook of applied cryptography*, CRC Press, Inc., Boca Raton, FL, USA, 1996.

[MP06] Michael Monagan and Roman Pearce, *Rational simplification modulo a polynomial ideal*, ISSAC'06, ACM, 2006, pp. 239–245.

[MQ01] Alfred J. Menezes and M. Qu, *Analysis of the weil descent attack of Gaudry, Hess and Smart*, Topics in Cryptology, CT-RSA 2001, LNCS, vol. 2020, Springer Verlag, 2001, pp. 308–318.

[MT06] Alfred J. Menezes and Edlyn Teske, *Cryptographic implications of Hess' generalized GHS attack*, Applicable Algebra in Engineering, Communication and Computing **16** (2006), no. 6, 439–460.

[MTW04] Alfred J. Menezes, Edlyn Teske, and Annegret Weng, *Weak fields for ECC*, Topics in Cryptology-CT-RSA'04, LNCS, vol. 2964, Springer Berlin / Heidelberg, 2004, pp. 366–386.

[Mus01] C. Musili, *Algebraic geometry for beginners*, Texts and Readings in Mathematics, no. 20, Hindustan Book Agency, 2001.

[MVO91] Alfred J. Menezes, Scott A. Vanstone, and Tatsuaki Okamoto, *Reducing elliptic curve logarithms to logarithms in a finite field*, Proceedings of the twenty-third annual ACM symposium on Theory of computing-STOC'91 (New York, NY, USA), ACM Press, 1991, pp. 80–89.

[Nat00] National Institute of Standards and Technology (NIST), *Digital signature standard (DSS). FIPS PUB 186-2*, 2000.

[Nec94] V. I. Nechaev, *Complexity of a determinate algorithm for the discrete logarithm*, Mathematical Notes **55** (1994), no. 2, 165–172.

[Odl85] Andrew M. Odlyzko, *Discrete logarithms in finite fields and their cryptographic significance*, EUROCRYPT'84 (New York, USA), Springer-Verlag New York, Inc., 1985, pp. 224–314.

[Ope99] Open Mobile Alliance (OMA), *Wireless Application Protocol - Wireless Transport Layer Security*, 1999.

[OVS84] R. W. K. Odoni, V. Varadharajan, and P. W. Sanders, *Public key distribution in matrix rings*, Electronics Letters **20** (1984), no. 9, 386–387.

[Pea05] Roman Pearce, *Rational expression simplification with side relations*, Master's thesis, Simon Fraser University, 2005.

[PH78] S. Pohlig and M. Hellman, *An improved algorithm for computing logarithms over $gf(p)$ and its cryptographic significance (corresp.)*, IEEE Transactions on Information Theory **24** (1978), 106–110.

[Pol78] John M. Pollard, *Monte Carlo methods for index computation* $(\mathrm{mod}\, p)$, Mathematics of Computation **32** (1978), no. 143, 918–924.

[RS02] Karl Rubin and Alice Silverberg, *Supersingular abelian varieties in cryptology*, CRYPTO'02, LNCS, vol. 2442, Springer, Berlin / Heidelberg, 2002, pp. 336–353.

[RS09] Reza Rezaeian Farashahi and Igor Shparlinski, *On the number of distinct elliptic curves in some families*, Designs, Codes and Cryptography (Online) (2009).

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman, *A method for obtaining digital signatures and public-key cryptosystems*, Communications of the ACM **21** (1978), 120–126.

[Rüc99] Hans-Georg Rück, *On the discrete logarithm in the divisor class group of curves*, Mathematics of Computation **68** (1999), no. 226, 805–806.

[SA98] T. Satoh and K. Araki, *Fermat quotients and the polynomial time discrete log algorithm for anomalous elliptic curves*, Commentarii Mathematici Universitatis Sancti Pauli **47** (1998), 81–92.

[Sch85] René J. Schoof, *Elliptic curves over finite fields and the computation of square roots* $\mathrm{mod}\, p$, Mathematics of Computation **44** (1985), no. 170, 483–494.

[Sch95] ———, *Counting points on elliptic curves over finite fields*, Journal de Théorie des Nombres de Bordeaux **7** (1995), 219–254.

[Sco04] Michael Scott, *Faster identity based encryption*, Electronics Letters **40** (2004), no. 14, 861–862.

[Sco05] ———, *Faster pairings using an elliptic curve with an efficient endomorphism*, INDOCRYPT'05, LNCS, vol. 3797, Springer, 2005, pp. 258–269.

[Sem98] I. A. Semaev, *Evaluation of discrete logarithms in a group of p-torsion points of an elliptic curve in characteristic p*, Mathematics of Computation **67** (1998), no. 221, 353–356.

[Sho97] Victor Shoup, *Lower bounds for discrete logarithms and related problems*, EUROCRYPT'97, vol. 1233, Springer Verlag, Berlin, 1997, pp. 256–266.

[Sil88] Joseph H. Silverman, *Computing heights on elliptic curves*, Mathematics of Computation **51** (1988), no. 183, 339–358.

[Sil90] ———, *The difference between the Weil height and the canonical height on elliptic curves*, Mathematics of Computation **55** (1990), no. 192, 723–743.

[Sil94] ———, *The arithmetic of elliptic curves*, Graduate Texts in Mathematics, vol. 106, Springer-Verlag, 1st ed. 1986. Corr. 3rd printing, 1994.

[Sil97] ———, *Computing canonical heights with little (or no) factorization*, Mathematics of Computation **66** (1997), no. 218, 787–805.

[Sil00] ———, *The xedni calculus and the elliptic curve discrete logarithm problem*, Designs, Codes and Cryptography **20** (2000), no. 1, 5–40.

[Sma99] Nigel P. Smart, *The discrete logarithm problem on elliptic curves of trace one*, Journal of Cryptology **12** (1999), no. 3, 193–196.

[Sma01] ———, *The Hessian form of an elliptic curve*, CHES 2001, LNCS, vol. 2162, Springer, 2001, pp. 118–125.

[SS79] Robert Sedgewick and Thomas G. Szymanski, *The complexity of finding periods*, STOC '79: Proceedings of the eleventh annual ACM symposium on Theory of computing (New York, NY, USA), ACM Press, 1979, pp. 74–80.

[SS98] Joseph H. Silverman and Joe Suzuki, *Elliptic curve discrete logarithms and the index calculus*, ASIACRYPT'98, LNCS, Springer-Verlag, 1998.

[Sti93] Henning Stichtenoth, *Algebraic function fields and codes*, Springer, 1993.

[SW03] Nigel P. Smart and E. J. Westwood, *Point multiplication on ordinary elliptic curves over fields of characteristic three.*, Applicable Algebra in Engineering, Communication and Computing **13** (2003), no. 6, 485–497.

[SWD96] Oliver Schirokauer, Damian Weber, and Thomas F. Denny, *Discrete logarithms: The effectiveness of the index calculus method*, Algorithmic Number Theory Symposium-ANTS, 1996, pp. 337–361.

[Tes98] Edlyn Teske, *Speeding up Pollard's rho method for computing discrete logarithms*, Algorithmic Number Theory, vol. 1423, 1998, pp. 541–554.

[Tes01] ———, *On random walks for Pollard's rho method*, Mathematics of Computation **70** (2001), 809–825.

[Ver08] Frederik Vercauteren, *Optimal pairings*, Cryptology ePrint Archive, Report 2008/096, 2008, `http://eprint.iacr.org/2008/096`.

[vH03] Mark van Hoeij, *An algorithm for computing the Weierstrass normal form*, ISSAC '95: International symposium on symbolic and algebraic computation, ACM, 2003, pp. 90–95.

[vOW99] Paul C. van Oorschot and Michael J. Wiener, *Parallel collision search with cryptanalytic applications*, Journal of Cryptology **12** (1999), no. 1, 1–28.

[Wag02] Samuel S. Wagstaff, *Cryptanalysis of number theoretic ciphers*, CRC Press Inc., October 2002.

[Was03] Lawrence C. Washington, *Elliptic curves: Number theory and cryptography*, CRC Press, 2003.

[Wei29] André Weil, *L'arithmétique sur les courbes algébriques*, Acta Mathematica **52** (1929), 281–315.

[WW27] E. T. Whittaker and G. N. Watson, *A course of modern analysis*, Cambridge University Press, 1927.

[WZ99] Michael J. Wiener and Robert J. Zuccherato, *Faster attacks on elliptic curve cryptosystems*, Proceedings of the Selected Areas in Cryptography-SAC'98 (London, UK), Springer-Verlag, 1999, pp. 190–200.

[Yui88] Noriko Yui, *Jacobi quartics, Legendre polynomials and formal groups*, Elliptic Curves and Modular Forms in Algebraic Topology, Lecture Notes in Mathematics, vol. 1326, Springer, 1988, pp. 182–215.

[ZZH07] Chang-An Zhao, Fangguo Zhang, and Jiwu Huang, *A note on the Ate pairing*, Cryptology ePrint Archive, Report 2007/247, 2007, `http://eprint.iacr.org/2007/247`.