Fast Tripling In Genus 2 Hyperelliptic Curves

Lindner, Sebastian

http://hdl.handle.net/11023/1723
master thesis

University of Calgary graduate students retain copyright ownership and moral rights for their thesis. You may use this material in any way that is permitted by the Copyright Act or through licensing that has been assigned to the document. For uses that are not allowable under copyright legislation or licensing, you are required to seek permission.

Downloaded from PRISM: https://prism.ucalgary.ca
UNIVERSITY OF CALGARY

FAST DIVISOR TRIPLING IN GENUS 2 HYPERELLIPTIC CURVES

by

SEBASTIAN LINDNER

A THESIS
SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE
DEGREE OF MASTERS OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

CALGARY, ALBERTA
SEPTEMBER, 2014

© SEBASTIAN LINDNER 2014
Abstract

We describe explicit formulas for tripling divisor classes on imaginary hyperelliptic curves given in Weierstrass form over arbitrary finite fields. Formulas are presented for both affine and projective coordinates, for divisor classes whose representations have extra field elements, and for simplified forms of the curve equation. By combining ideas from the algebraic and geometric methods we obtain savings compared to previous methods. The same methods are applied to addition and doubling, resulting in the fastest known formulas to-date.
Acknowledgements

I would like to thank my brother, my mom and most of all my wife Larisa for all the support, care and patience that I have been blessed with over the last couple years. I would not have been able accomplish all I have with out. I would like to also thank my supervisor Dr. Jacobson. I appreciate all the time and effort that he has committed accelerating my growth as a student and scholar.

I have learned more about hard work, patience and dealing with disappointments (along with their silver linings) than I ever had expected. Thank you to all.
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>i</td>
</tr>
<tr>
<td>Acknowledgements</td>
<td>ii</td>
</tr>
<tr>
<td>Table of Contents</td>
<td>iii</td>
</tr>
<tr>
<td>List of Tables</td>
<td>v</td>
</tr>
<tr>
<td>1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Motivation</td>
<td>2</td>
</tr>
<tr>
<td>1.2 Summary of Research Contribution</td>
<td>3</td>
</tr>
<tr>
<td>1.3 Contributions</td>
<td>4</td>
</tr>
<tr>
<td>1.3.1 Divisor Addition and Doubling</td>
<td>4</td>
</tr>
<tr>
<td>1.3.2 Divisor Tripling</td>
<td>7</td>
</tr>
<tr>
<td>1.4 Organization of Thesis</td>
<td>8</td>
</tr>
<tr>
<td>2 Mathematical Background</td>
<td>10</td>
</tr>
<tr>
<td>2.1 Hyperelliptic Curves</td>
<td>11</td>
</tr>
<tr>
<td>2.2 Polynomial and Rational Functions</td>
<td>13</td>
</tr>
<tr>
<td>2.3 Zeros and Poles</td>
<td>15</td>
</tr>
<tr>
<td>2.4 Divisors</td>
<td>18</td>
</tr>
<tr>
<td>2.5 The Jacobian $J$ of a Curve</td>
<td>19</td>
</tr>
<tr>
<td>2.6 Representing Divisors</td>
<td>20</td>
</tr>
<tr>
<td>2.6.1 Reduced Divisors</td>
<td>20</td>
</tr>
<tr>
<td>2.6.2 Mumford Representation</td>
<td>21</td>
</tr>
<tr>
<td>2.6.3 Cantor’s Algorithm</td>
<td>23</td>
</tr>
<tr>
<td>3 Generic Methods to Composition</td>
<td>25</td>
</tr>
<tr>
<td>3.1 The Frequent Case</td>
<td>26</td>
</tr>
<tr>
<td>3.2 From Cantor to Harley</td>
<td>28</td>
</tr>
<tr>
<td>3.2.1 Addition in the Frequent Case</td>
<td>28</td>
</tr>
<tr>
<td>3.2.2 Doubling in the Frequent Case</td>
<td>30</td>
</tr>
<tr>
<td>3.3 Geometric Method</td>
<td>31</td>
</tr>
<tr>
<td>3.3.1 Group Law Algorithm in the Frequent Case</td>
<td>32</td>
</tr>
<tr>
<td>3.3.2 Geometrically Computing Composition</td>
<td>33</td>
</tr>
<tr>
<td>3.3.3 Equivalence of Algebraic and Geometric Group Laws</td>
<td>36</td>
</tr>
<tr>
<td>3.3.4 Summary</td>
<td>39</td>
</tr>
<tr>
<td>4 Explicit Formulae</td>
<td>40</td>
</tr>
<tr>
<td>4.1 From Polynomial Algorithm to Explicit Formula</td>
<td>41</td>
</tr>
<tr>
<td>4.1.1 Representing Polynomial Arithmetic as Explicit Formulae</td>
<td>41</td>
</tr>
<tr>
<td>4.1.2 Curve Simplifications</td>
<td>42</td>
</tr>
<tr>
<td>4.2 Techniques to Improve Explicit Formulae</td>
<td>44</td>
</tr>
<tr>
<td>4.2.1 Karatsuba Multiplication</td>
<td>44</td>
</tr>
<tr>
<td>4.2.2 Karatsuba Reduction</td>
<td>45</td>
</tr>
<tr>
<td>4.2.3 Efficient Exact Division</td>
<td>45</td>
</tr>
<tr>
<td>4.2.4 Using ’Almost Inverse’</td>
<td>46</td>
</tr>
<tr>
<td>4.2.5 Montgomery’s Trick of Simultaneous Inversion</td>
<td>48</td>
</tr>
<tr>
<td>4.2.6 Reordering of the Normalization Step</td>
<td>49</td>
</tr>
</tbody>
</table>
4.2.7 Alternate Approach to Computing $s$ in Harley Composition ........................................... 49
4.2.8 Improved Solution to System of Equations ................................................................. 50
4.2.9 Omitting $l'$ and the Alternate Computation of $v''$ ..................................................... 52
4.2.10 Equating Coefficients .................................................................................................. 53
4.2.11 Geometric Linear System Simplification ...................................................................... 54

4.3 Projective Coordinates ...................................................................................................... 55
4.3.1 Weight Complexity Reduction ...................................................................................... 56
4.3.2 Computation of $u''$ modulo $u$ .................................................................................. 56

5 Divisor Doubling and Addition ............................................................................................. 59
5.1 Affine Formulae .................................................................................................................. 61
5.1.1 Addition ......................................................................................................................... 62
5.1.2 Doubling ......................................................................................................................... 71
5.2 Projective Formulae ............................................................................................................ 78
5.2.1 Addition ......................................................................................................................... 82
5.2.2 Mixed Addition .............................................................................................................. 88
5.2.3 Double ............................................................................................................................ 95

5.3 Alternate Representations of Divisor Classes ................................................................... 96
5.3.1 Semi-Affine Setting ....................................................................................................... 96
5.3.2 Geometric Setting ......................................................................................................... 100
5.3.3 New-coordinates Setting ............................................................................................... 101

5.4 Comparisons ..................................................................................................................... 101
5.5 Assessment for Non-adjacent Form Algorithms ............................................................... 102

6 Divisor Tripling ..................................................................................................................... 109
6.1 Specialized Double and Add ............................................................................................. 110
6.1.1 Omit Computation of $v''$ ............................................................................................. 112
6.1.2 Two Inversion ................................................................................................................ 113
6.1.3 One Inversion ............................................................................................................... 123
6.1.4 Projective ....................................................................................................................... 134
6.1.5 Comparisons .................................................................................................................. 151

6.2 Assessment for Double-base Algorithms ....................................................................... 151

6.3 Dedicated Tripling Methods ............................................................................................ 154
6.3.1 Geometric ...................................................................................................................... 154
6.3.2 NUCUBE ...................................................................................................................... 158
6.3.3 Dedicated Tripling Summary ....................................................................................... 161

6.4 Summary ........................................................................................................................... 161

7 Conclusion ............................................................................................................................ 162

7.1 Future Work ....................................................................................................................... 165

Bibliography ............................................................................................................................ 166

A ............................................................................................................................................... 171
A.1 6GE Explicit Formulae in Even Characteristic ................................................................. 171
A.2 Maple Code ........................................................................................................................ 175
## List of Tables

<table>
<thead>
<tr>
<th>Table Number</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>Arbitrary Affine Addition Formula</td>
<td>68</td>
</tr>
<tr>
<td>5.2</td>
<td>Even Characteristic Affine Addition Formula</td>
<td>69</td>
</tr>
<tr>
<td>5.3</td>
<td>Odd Characteristic Affine Addition Formula</td>
<td>70</td>
</tr>
<tr>
<td>5.4</td>
<td>Arbitrary Affine Doubling Formula</td>
<td>79</td>
</tr>
<tr>
<td>5.5</td>
<td>Even Characteristic Affine Doubling Formula</td>
<td>80</td>
</tr>
<tr>
<td>5.6</td>
<td>Odd Characteristic Affine Doubling Formula</td>
<td>81</td>
</tr>
<tr>
<td>5.7</td>
<td>Arbitrary Projective Addition Formula</td>
<td>89</td>
</tr>
<tr>
<td>5.8</td>
<td>Even Characteristic Projective Addition Formula</td>
<td>90</td>
</tr>
<tr>
<td>5.9</td>
<td>Odd Characteristic Projective Addition Formula</td>
<td>91</td>
</tr>
<tr>
<td>5.10</td>
<td>Arbitrary Mixed Addition Formula</td>
<td>92</td>
</tr>
<tr>
<td>5.11</td>
<td>Even Characteristic Mixed Addition Formula</td>
<td>93</td>
</tr>
<tr>
<td>5.12</td>
<td>Odd Characteristic Mixed Addition Formula</td>
<td>94</td>
</tr>
<tr>
<td>5.13</td>
<td>Arbitrary Projective Doubling Formula</td>
<td>97</td>
</tr>
<tr>
<td>5.14</td>
<td>Even Characteristic Projective Doubling Formula</td>
<td>98</td>
</tr>
<tr>
<td>5.15</td>
<td>Odd Characteristic Projective Doubling Formula</td>
<td>99</td>
</tr>
<tr>
<td>5.16</td>
<td>Comparisons of our Even Characteristic formulae</td>
<td>102</td>
</tr>
<tr>
<td>5.17</td>
<td>Comparisons of our Odd Characteristic formulae</td>
<td>103</td>
</tr>
<tr>
<td>5.18</td>
<td>Even Characteristic Affine Doubling $h(x) = x^{39}$</td>
<td>104</td>
</tr>
<tr>
<td>5.19</td>
<td>Odd Characteristic 6SA (Mixed) Addition Formula</td>
<td>105</td>
</tr>
<tr>
<td>5.20</td>
<td>Odd Characteristic 6SA Doubling Formula</td>
<td>106</td>
</tr>
<tr>
<td>5.21</td>
<td>Odd Characteristic 8 Coordinate Mixed Addition Formula</td>
<td>107</td>
</tr>
<tr>
<td>5.22</td>
<td>Odd Characteristic 8 Coordinate Doubling Formula</td>
<td>108</td>
</tr>
<tr>
<td>6.1</td>
<td>Arbitrary 2 Inversion Tripling Formula</td>
<td>120</td>
</tr>
<tr>
<td>6.2</td>
<td>Even Characteristic 2 Inversion Tripling Formula</td>
<td>121</td>
</tr>
<tr>
<td>6.3</td>
<td>Odd Characteristic 2 Inversion Tripling Formula</td>
<td>122</td>
</tr>
<tr>
<td>6.4</td>
<td>Arbitrary 1 Inversion Tripling Formula</td>
<td>131</td>
</tr>
<tr>
<td>6.5</td>
<td>Even Characteristic 1 Inversion Tripling Formula</td>
<td>132</td>
</tr>
<tr>
<td>6.6</td>
<td>Odd Characteristic 1 Inversion Tripling Formula</td>
<td>133</td>
</tr>
<tr>
<td>6.7</td>
<td>Arbitrary Projective Tripling Formula</td>
<td>148</td>
</tr>
<tr>
<td>6.8</td>
<td>Even Characteristic Projective Tripling Formula</td>
<td>149</td>
</tr>
<tr>
<td>6.9</td>
<td>Odd Characteristic Projective Tripling Formula</td>
<td>150</td>
</tr>
<tr>
<td>6.10</td>
<td>Even Characteristic Tripling Formula Comparisons.</td>
<td>152</td>
</tr>
<tr>
<td>6.11</td>
<td>Odd Characteristic Tripling Formula Comparisons.</td>
<td>153</td>
</tr>
<tr>
<td>6.12</td>
<td>Odd Characteristic 6SA 1 Inversion Tripling Formula</td>
<td>155</td>
</tr>
<tr>
<td>6.13</td>
<td>Odd Characteristic 8 Coordinate Tripling Formula</td>
<td>156</td>
</tr>
<tr>
<td>7.1</td>
<td>Recommendations for Non-Adjacent Form Implementation</td>
<td>163</td>
</tr>
<tr>
<td>7.2</td>
<td>Recommendations for Double-Base Implementation</td>
<td>164</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table Number</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.1</td>
<td>6GE Even Char Addition Formula</td>
<td>172</td>
</tr>
<tr>
<td>A.2</td>
<td>6GE Even Char Doubling Formula</td>
<td>173</td>
</tr>
<tr>
<td>A.3</td>
<td>6GE Even Char Tripling Formula</td>
<td>174</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction
1.1 Motivation

During the early history of cryptography, two parties would rely upon a key that they would exchange between themselves by means of a secure but non-cryptographic method. This key could then be used to exchange encrypted messages. The obvious limitations of “classic” cryptography is the need for a shared key to be established prior to secure communication. In 1976, a public-key cryptosystem known as the Diffie-Hellman key exchange was published [10] where a method for public-key agreement was described. This was the first published practical method for establishing a shared secret-key without using a prior shared secret.

Since the 1970’s, a large variety of encryption, digital signature, key agreement, and other techniques have been developed in the field of public-key cryptography. Given a finite group, the discrete logarithm problem is finding an integer $k$ solving the equation $b^k = g$ where $b$ and $g$ are elements of the group. The introduction of elliptic curve cryptography [25, 32] in the mid-1980’s yielded new public-key cryptosystems based on the discrete logarithm problem. Compared with traditional groups like $\mathbb{F}_q^*$, elliptic curves have the advantage of achieving the same level of conjectured security with a much smaller group size [29]. In 1988, Koblitz [26] generalized this idea by proposing Jacobians of hyperelliptic curves of arbitrary genus as a way to construct Abelian groups suitable for cryptography. Genus two hyperelliptic curves can achieve groups of the same size and security as elliptic curves but defined over finite fields of size roughly half the bits. Genus three and higher hyperelliptic curves succumb to specialized index calculus attacks that make those settings less secure [11, 17, 36], thus we only pay attention to the genus two setting.

As an example, we roughly describe the hyperelliptic curve Diffie-Hellman protocol: Suppose Alice wants to establish a shared key with Bob. Let $D$ be a divisor in the pre-agreed setting. Each party must have a key pair suitable for hyperelliptic curve cryptography, consisting of a private key $s$ (a randomly selected integer) and a public key $Q$ (where $Q = sD$). Let Alice’s key pair be $(s_A, Q_A)$ and Bob’s key pair be $(s_B, Q_B)$. After
an exchange of the respective $Q_i$ divisors, Alice computes the divisor $D' = s_A Q_B$. Bob computes $D' = s_B Q_A$. The shared secret $D'$ calculated by both parties is equal, because $s_A Q_B = s_A s_B D = s_B s_A D = s_B Q_A$.

The most important and expensive operation in (hyper)elliptic curve cryptography is scalar multiplication by an integer $k$ i.e., computing a scalar multiple $kP$ of a point $P$ on the points group or $kD$ of a divisor class $D$ on the Jacobian. Scalar multiplication can be performed using a binary double-and-add algorithm given a binary representation of the scalar $k$. Since inverting an element of the Jacobian is cheap, the non-adjacent form algorithm is generally used where on average $\log k$ doubles and $(\log k/3)$ additions instead of $(\log k/2)$ additions are required to compute the scalar multiple. Recent work has shown that the use of double-base algorithms, in which the scalar is represented via sums and differences of mixed powers of 2 and 3, can accelerate scalar multiplication if a fast tripling operation is available [6, 37, 11].

The fastest way to implement Jacobian arithmetic in genus two is using so-called explicit formulae, in which the group operations are described in terms of finite field operations involving the Mumford representation of the divisor classes. Explicit formulae for doubling and addition have been described in some detail (see, for example, [27, 40, 9]). To our knowledge the only work on tripling formulae is due to Balamohan [2], but this work only considered odd characteristic finite fields.

1.2 Summary of Research Contribution

In this paper we propose new explicit formulae for tripling divisor classes on genus two imaginary hyperelliptic curves given in Weierstrass form over arbitrary finite fields. We present formulae for both affine and projective coordinates, how our contributions adapt to special cases where extra field elements are used to describe divisor classes, and cases where simplifications of the curve equation are applied.
There are two main methods for deriving explicit formulae for the group law in the Jacobian: the algebraic method \cite{27} based on Harley's formulation of Cantor's algorithm, and the geometric method using interpolation of points \cite{9}. Previous works have focused on one or the other. By exploring both methods, and combining the best aspects of each in the most beneficial way, we produced tripling formulae that require fewer operations than any other over both even and odd characteristic fields. By applying these ideas to doubling and addition, we also found new formulae that require fewer operations than any other, resulting in the fastest known formulae to date for general Weierstrass form curves over even and odd characteristic fields. We note that the fastest tripling formulae came from specialized double-and-add methods, we have also explored the dedicated tripling technique NUCUBE \cite{22} and dedicated geometric tripling in our exposition.

We stress that our formulae are for arbitrary genus two imaginary hyperelliptic curves given in Weierstrass form; as such, we do not compare to special classes of curves such as those using Kummer surfaces \cite{18}, nor do we claim any speed records. The intention is to demonstrate that efficient tripling formulae exist in the hope that they can be used directly in, or generalized to, specialized settings such as those used in state-of-the-art implementations such as \cite{3}.

1.3 Contributions

The contributions of this work are given below, organized into two parts: divisor tripling and divisor addition and doubling.

1.3.1 Divisor Addition and Doubling

We have produced faster explicit formulae in both even and odd characteristic for the projective and affine settings. We augmented the Harley method with ideas from the geometric method and other algebraic tricks. We present exactly what contributions we made to achieve
Affine Setting
Both Characteristics:

- Use system of equations to solve for certain sub-expressions in the Harley method of composition (Section 4.2.7). This technique saves one square for doubling in even characteristic.

- Reuse computations from setting up the system of equations, resulting in the omission of subsequent computations (Section 6.1.1). This technique saves one multiplication for doubling and addition in both characteristics.

Odd Characteristic:

- Adapt a solution for system of equations trick from the geometric method to solve the system of equations. (Section 4.2.8). This technique saves one multiplication for doubling and addition each.

Even Characteristic:

- Reuse computations from setting up the system to compute the output polynomial differently (Section 5.1.1). This technique trades one multiplication for one square.

Projective Setting
Odd Characteristic:

- Adapt a solution for system of equations trick from the geometric method to solve the system of equations setup for a sub-expression (Section 4.2.8). This technique saves one multiplication for doubling and addition each.

Even Characteristic:
Create all new projective formulae adapting some ideas from the odd characteristic only work by Balamohan [2] (Sections 5.2 and 5.2.3). These formulae are faster than any other saving three squares in doubling, one multiplication and two squares in addition and two squares in mixed addition.

Alternate Representations

Our contributions to the the regular affine and projective settings adapt to other alternate settings where auxiliary coordinates are used to represent divisor classes. Alternate affine settings are the “semi-affine” and “geometric” settings both requiring 2 auxiliary coordinates. The alternate projective setting is the “new-coordinates” setting requiring 3 auxiliary coordinates (Section 5.3). We note that our formulae and their adaptations are fastest in all cases but one, even characteristic affine doubling and addition.

Odd Characteristic:

- The solution for system of equations trick (Section 4.2.7) adapts exactly the same way to the “semi-affine” setting saving one multiplication for doubling and addition each.

- The solution for system of equations trick (Section 4.2.7) also adapts exactly the same way to the projective “new-coordinates” setting saving one multiplication for doubling and addition each, creating the fastest formulae for that setting overall.

Even Characteristic:

- Create all new affine six coordinate “geometric” setting addition and doubling formulae adapting the odd characteristic only work of Costello [9] to arbitrary fields. The six coordinate “geometric” affine setting produces the fastest doubling and addition formulae in the affine setting over all in even characteristic.
The methodology is found in Section 3.3, an explanation of the six coordinate setting in Section 5.3, and the formulae are presented in Appendix A.1.

1.3.2 Divisor Tripling

In odd characteristic for the projective and affine settings, we improved previous state of the art tripling formulae requiring less field operations than any other to date. In even characteristic we created whole new tripling formulae adapting some of the techniques used previously by Balamohan in [2] to work over even characteristic fields, greatly reducing the number of field operations required relative to implementing a double then an addition.

Affine Setting

We present what contributions we made in the affine setting organized by the number of inversions required to compute the tripling.

Two Inversion:

- In odd characteristic we adapt a solution for system of equations trick from the geometric method to solve the system of equations setup for a certain sub-expression (Section 4.2.8). This technique saves two multiplications.
- In even characteristic we adapt some techniques from the odd characteristic tripling formulae in [2], producing new formulae.

One Inversion:

- In both characteristics we augment the two inversion formulae using a system of equations to different sub-expressions than in the doubling and addition formulae. Our implementation not only creates the fastest one inversion formulae for the affine setting, but is also faster than both the “geometric” and “semi-affine” six coordinate representation settings.
Projective Setting

We present what contributions we made in the projective setting.

- In odd characteristic we adapt a solution for system of equations trick from the geometric method to solve the system of equations setup for a certain sub-expression (Section 4.2.8). This technique saves two multiplications.

- In even characteristic we adapt some techniques from the odd characteristic tripling formulae in [2], creating all new formulae.

Alternate Representations

Our contributions to the the regular affine and projective settings adapt to other alternate settings where auxiliary coordinates are used to represent divisor classes.

Odd Characteristic:

- The solution for system of equations trick (Section 4.2.8) adapts exactly the same way to the affine “semi-affine” setting saving two multiplications.

- The solution for system of equations trick (Section 4.2.8) also adapts exactly the same way to the projective “new-coordinates” setting saving two multiplications.

Even Characteristic:

- Create affine six coordinate “geometric” setting two inversion tripling formulae using a double and an add together. The six coordinate “geometric” affine setting produces the fastest two inversion tripling formulae in the affine setting over all for even characteristic.

1.4 Organization of Thesis

The remainder of this thesis is organized as follows:
In Chapter 2 we introduce the mathematical background of hyperelliptic curves, restricting to the material necessary to understand this thesis. We describe basic definitions and properties of hyperelliptic curves, divisors, the Jacobian and its elements. We concluded the chapter with an exposition of efficient representation and computation of elements of the Jacobian using Mumford representation and Cantor’s Algorithm.

In Chapter 3 we describe the two main methods to greatly reduce the computational complexity of computing the group operation over the Jacobian. We first introduce the frequent case that input divisor classes can be in, then describe Harley’s method and the geometric method to computation of the group operation at the polynomial arithmetic level.

In Chapter 4 we describe the process of converting a polynomial algorithm to an explicit formula by presenting a summary of the process, simplifications to the hyperelliptic curve, and all algebraic tricks used to reduce the complexity of our formulae.

In Chapter 5 we present our novel addition and doubling formulae in affine and projective settings. We compare our formulae to previous best and adapt our contributions to alternate representations of divisor classes. We summarize with a table of all costs.

In Chapter 6 we present our novel tripling formulae in affine and projective settings. We also describe dedicated tripling methods that proved to be more complex than the double-and-add formulae. We compare our formulae to previous best and adapt our contributions to alternate representations of divisor classes. We summarize with a table of all costs.

In Chapter 7 we describe the status of our work, provide a summary and give considerations for future work.
Chapter 2

Mathematical Background
In this chapter we present an elementary introduction to the theory of hyperelliptic curves over finite fields of arbitrary characteristic, corresponding divisors and their properties, the Jacobian of a curve and computing the group operation in the Jacobian. We restrict attention to material that is relevant for this work.

Hyperelliptic curves are a special class of algebraic curves and can be viewed as a generalization of elliptic curves. The idea that groups formed from hyperelliptic curves can be a suitable setting for public hyperelliptic curve cryptography was first introduced by Koblitz [26]. Most of the material presented in this chapter was taken from [30]. For an introduction to algebraic geometry the reader is referred to [14].

2.1 Hyperelliptic Curves

In this section we present the main definitions and properties of hyperelliptic curves. We begin with Definition 2.1.1 describing a hyperelliptic curve over a finite field.

Definition 2.1.1. [30, Definition 1] Let $\mathbb{F}$ be a finite field and let $\overline{\mathbb{F}}$ be the algebraic closure of $\mathbb{F}$. A hyperelliptic curve $C$ of genus $g$ over $\mathbb{F}$ ($g \geq 1$) is an equation of the form

$$C : y^2 + h(x)y = f(x) \text{ in } \mathbb{F}[x, y],$$

(2.1)

where $h(x) \in \mathbb{F}[x]$ is a polynomial of degree at most $g$, $f(x) \in \mathbb{F}[x]$ is a monic polynomial of degree $2g + 1$, and there are no solutions $(x, y) \in \overline{\mathbb{F}} \times \overline{\mathbb{F}}$ which simultaneously satisfy the equation $y^2 + h(x)y = f(x)$ and the partial derivative equations $2y + h(x) = 0$ and $h'(x)y - f'(x) = 0$.

A singular point on a curve $C$ is a solution $(x, y) \in \overline{\mathbb{F}} \times \overline{\mathbb{F}}$ which simultaneously satisfies the equations $y^2 + h(x)y = f(x)$ and the partial derivative equations $2y + h(x) = 0$ and $h'(x)y - f'(x) = 0$. From Definition 2.1.1 we see that hyperelliptic curves do not have any singular points. In Definition 2.1.2 $C$ is described as a set of points over the extension field $K$ of $\mathbb{F}$.
Definition 2.1.2. [30, Definition 3] Let \( \mathbb{K} \) be an extension field of \( \mathbb{F} \). The set of \( \mathbb{K} \)-rational points on \( C \), denoted \( C(\mathbb{K}) \), is the set of all points \( P = (a, b) \in \mathbb{K} \times \mathbb{K} \) that satisfy equation 2.1 together with a special point at infinity denoted \( \infty \). The set of points \( C(\mathbb{K}) \) well be denoted by \( C \). The points in \( C \) other than \( \infty \) are called finite points.

Note that the point at infinity lies in the projective plane \( \mathbb{P}^2(\mathbb{F}) \). It is the only projective point lying on the line at infinity that satisfies the equation 2.1. If \( g \geq 2 \), then \( \infty \) is a singular (projective) point (which is allowed since \( \infty \) is not in \( \mathbb{F} \times \mathbb{F} \)). Finally we describe the opposite of a point \( P \) on \( C \) in Definition 2.1.3.

Definition 2.1.3. [30, Definition 5] Let \( P = (a, b) \) be a finite point on a hyperelliptic curve \( C \). The opposite point of \( P \) is the point \( \tilde{P} = (a, -b - h(a)) \). The opposite of \( \infty \) is defined as \( \tilde{\infty} = \infty \) itself. If a finite point \( P \) satisfies \( P = \tilde{P} \), then the point is said to be special; otherwise the point is ordinary.

Figure 2.1: The hyperelliptic curve \( C : y^2 = x^5 - 2x^4 - 7x^3 + 8x^2 + 12x \) over the reals.
Curves over finite fields cannot be easily graphed because they are comprised of discrete points, so in Figure 2.1, we present an example of a hyperelliptic curve over the real numbers.

2.2 Polynomial and Rational Functions

In this section we introduce the properties of polynomials and rational functions when they are viewed as functions on a hyperelliptic curve. This machinery is necessary for defining divisors in Section 2.4. We work up to a definition of a function field of \( C \) in Definition 2.2.5, starting with a description of coordinate rings of \( C \) in Definition 2.2.1 and Lemma 2.2.2.

**Definition 2.2.1.** [30, Definition 8] The coordinate ring of \( C \) over \( \mathbb{F} \), denoted \( \mathbb{F}[C] \), is the quotient ring

\[
\mathbb{F}[C] = \mathbb{F}[x, y]/(y^2 + h(x)y - f(x)),
\]

where \((y^2 - h(x)y - f(x))\) denotes the ideal in \( \mathbb{F}[x, y] \) generated by the polynomial \( y^2 + h(x)y - f(x) \). Similarly, the coordinate ring of \( C \) over \( \bar{\mathbb{F}} \) is defined as

\[
\bar{\mathbb{F}}[C] = \bar{\mathbb{F}}[x, y]/(y^2 + h(x)y - f(x)).
\]

An element of \( \bar{\mathbb{F}}[C] \) is called a polynomial function on \( C \).

**Lemma 2.2.2.** [30, Lemma 9] The polynomial \( r(x, y) = y^2 + h(x)y - f(x) \) is irreducible over \( \mathbb{F} \), hence \( \mathbb{F} \) is an integral domain.

The proof of Lemma 2.2.2 is given in [30, Lemma 9]. Note that every polynomial function \( G(x, y) \in \mathbb{F}[C] \) is represented as \( G(x, y) = u(x) - v(x)y \), where \( u(x), v(x) \in \mathbb{F} \), are unique.

**Definition 2.2.3.** [30, Definition 10] Let \( G(x, y) = u(x) - v(x)y \) be a polynomial function in \( \mathbb{F}[C] \). The conjugate of \( G(x, y) \) is defined to be the polynomial function \( \overline{G}(x, y) = u(x) + v(x)(h(x) + y) \).
The norm is used to correlate questions about polynomial functions in two variables into easier questions about polynomials in a single variable.

**Definition 2.2.4.** [30, Definition 11] Let \( G(x, y) = u(x) - b(x)y \) be a polynomial function in \( \mathbb{F}[C] \). The norm of \( G \) is the polynomial function \( N(G) = GG' \).

We present a description of a function field and rational functions in Definition 2.2.5. Rational functions are the functions used to describe divisors.

**Definition 2.2.5.** [30, Definition 13] The function field \( \mathbb{F}(C) \) of \( C \) over \( \mathbb{F} \) is the field of fractions of \( \mathbb{F}[C] \). Similarly, the function field \( \mathbb{F}(C) \) of \( C \) over \( \mathbb{F} \) is the field of fractions of \( \mathbb{F}[C] \). The elements of \( \mathbb{F}(C) \) are called rational functions on \( C \).

We point out that \( \mathbb{F}[C] \) is a subring of \( \mathbb{F}(C) \) and that every polynomial function is also a rational function. The following definition defines the value of a rational function at a finite point.

**Definition 2.2.6.** [30, Definition 14] Let \( R \in \mathbb{F}(C) \), and let \( P \in C \), \( P \neq \infty \). Then \( R \) is said to be defined at \( P \) if there exist polynomial functions \( G, H \in \mathbb{F}[C] \) such that \( R = G/H \) and \( H(P) \neq 0 \); if no such \( G, H \in \mathbb{F}[C] \) exist, then \( R \) is not defined at \( P \). If \( R \) is defined at \( P \), the value of \( R \) at \( P \) is defined to be \( R(P) = G(P)/H(P) \).

Notice that the well-defined value \( R(P) \) is independent of the choice of \( G \) and \( H \). Definition 2.2.7 describes the degree of the polynomial function which is necessary for describing properties of divisors. Afterwards, properties of the degree are stated in Lemma 2.2.8.

**Definition 2.2.7.** [30, Definition 15] Let \( G(x, y) = u(x) - v(x)y \) be a non-zero polynomial function in \( \mathbb{F}[C] \). The degree of \( G \) is defined to be

\[
\deg(G) = \max[2 \deg_x(u), 2g + 1 + 2 \deg_x(v)].
\]
Lemma 2.2.8. \cite{30} Lemma 16] Let $G, H \in \mathbb{F}[C]$.

1. $\deg(G) = \deg_x(N(G))$.

2. $\deg(GH) = \deg(G) + \deg(H)$.

3. $\deg(G) = \deg(G^\prime)$

Finally the value of rational functions at $\infty$ is defined in Definition 2.2.9.

Definition 2.2.9. \cite{30} Definition 17] Let $R = G/H \in \mathbb{F}(C)$ be a rational function.

1. If $\deg(G) < \deg(H)$ then the value of $R$ at $\infty$ is defined to be $R(\infty) = 0$.

2. If $\deg(G) > \deg(H)$ then $R$ is not defined at $\infty$.

3. If $\deg(G) = \deg(H)$ then $R(\infty)$ is defined by the ratio of the leading coefficients (with respect to the deg function) of $G$ and $H$.

All of the definitions covered in this section are required to describe divisors over finite fields. Next we present zeros and poles of rational functions over function fields to complete the picture for describing divisors.

2.3 Zeros and Poles

In this section we describe the orders of zeros and poles of rational functions as well as the notion of a uniformizing parameter. Understanding orders of rational functions is necessary for defining divisors in Section 2.4. Proofs of all lemmas and theorems presented in this section can be found in \cite{30} Chapter 4.

Definition 2.3.1. \cite{30} Definition 18] Let $R \in \mathbb{F}(C)^* \text{ where } \mathbb{F}(C)^* = \mathbb{F}(C) \setminus \{0\}$ and $P \in C$. If $R(P) = 0$ then $R$ is said to have a zero at $P$. If $R$ is not defined at $P$ then $R$ is said to have a pole at $P$, in which case we write $R(P) = \infty$.  

15
Lemma 2.3.2. [30, Lemma 19] Let $G \in \mathbb{F}[C]^*$ where $\mathbb{F}[C]^* = \mathbb{F}[C] \setminus \{0\}$ and $P \in C$. If $G(P) = 0$ then $\overline{G}(\tilde{P}) = 0$.

Lemmas 2.3.3, 2.3.4 and 2.3.5 are used in Theorem 2.3.6 that introduces the existence of uniformizing parameters. Notice that Lemma 2.3.3 is not the same as Lemma 2.3.2 because a point is special when $G(P) = 0$ and $G(\tilde{P}) = 0$, not $\overline{G}(\tilde{P})$.

Lemma 2.3.3. [30, Lemma 20] Let $P = (a, b)$ be a point of $C$. Suppose that $G = u(x) - v(x)y \in \mathbb{F}[C]^*$ has a zero at $P$ and that $a$ is not a root of both $u(x)$ and $v(x)$. Then $\overline{G}(P) = 0$ if and only if $P$ is a special point.

Lemma 2.3.4. [30, Lemma 21] Let $P = (a, b)$ be a non-special point of $C$ and let $G = u(x) - v(x)y \in \mathbb{F}[C]^*$. Suppose that $G(P) = 0$ and $a$ is not a root of both $u(x)$ and $v(x)$. Then $G$ can be written in the form $(x - a)^sS(x, y)$, where $s$ is the highest power of $(x - a)$ which divides $N(G)$, and $S(x, y) \in \mathbb{F}(C)$ has neither a zero nor a pole at $P$.

Lemma 2.3.5. [30, Lemma 22] Let $P = (a, b)$ be a special point of $C$. Then $(x - a)$ can be written in the form $(y - b)^2S(x, y)$, where $S(x, y) \in \mathbb{F}(C)$ has neither a zero nor a pole at $P$.

Theorem 2.3.6 describes a uniformizing parameter for a point $P \in C$ as a polynomial function. The previous three Lemmas cover all possible cases for a point. Lemma 2.3.3 points out that if a polynomial function that is zero at $P$, and its conjugate is also zero at $P$, then that point must be special. This distinguishes how special points act in terms of the unifying parameter. For a point $P$, Lemmas 2.3.4 and 2.3.5 describe the existence of polynomials of a certain form related to the point based on $P$ being special or not.

Theorem 2.3.6. [30, Theorem 23] Let $P \in C$. Then there exists a function $U \in \mathbb{F}(C)$ with $U(P) = 0$ such that the following property holds: for each polynomial function $G \in \mathbb{F}[C]^*$, there exists an integer $d$ and function $S \in \mathbb{F}(C)$ such that $S(P) \neq 0, \infty$ and $G = U^dS$. 

16
Furthermore, the number $d$ does not depend on the choice of $U$. The function $U$ is called a uniformizing parameter for $P$.

The order of a polynomial function at a point is important for describing divisors. Using Theorem 2.3.6, the order of a polynomial function at a point is described in Definition 2.3.7 and properties of the order are given in Lemma 2.3.8.

**Definition 2.3.7.** [30, Definition 24] Let $G \in \mathbb{F}(C)^*$ and $P \in C$. Let $U \in \mathbb{F}(C)$ be the uniformizing parameter for $P$, and write $G = U^dS$ where $S \in \mathbb{F}(C)$, $S(P) \neq 0, \infty$. The order of $G$ at $P$ is defined to be $\text{ord}_P(G) = d$.

**Lemma 2.3.8.** [30, Lemma 25] Let $G_1, G_2 \in \mathbb{F}(C)^*$ and $P \in C$, and let $\text{ord}_P(G_1) = r_1$, $\text{ord}_P(G_2) = r_1$.

1. $\text{ord}_P(G_1G_2) = \text{ord}_P(G_1) + \text{ord}_P(G_2)$.

2. Suppose that $G_1 \neq -G_2$. If $r_1 \neq r_2$ then $\text{ord}_P(G_1 + G_2) = \min(r_1, r_2)$. If $r_1 = r_2$ then $\text{ord}_P(G_1 + G_2) \geq \min(r_1, r_2)$.

Next we present a generalization of Lemma 2.3.2 that extends properties of conjugates of polynomial functions to use orders, given in Lemma 2.3.9.

**Lemma 2.3.9.** [30, Lemma 28] Let $G \in \mathbb{F}[C]^*$ and $P \in C$. Then $\text{ord}_P(G) = \text{ord}_P(\tilde{P})(G)$.

We describe an important result about orders and poles of functions in the coordinate ring $\mathbb{F}[C]^*$ in Theorem 2.3.10.

**Theorem 2.3.10.** [30, Theorem 29] Let $G \in \mathbb{F}[C]^*$. Then $G$ has a finite number of zeros and poles. Moreover,

$$\sum_{P \in C} \text{ord}_P(G) = 0.$$ 

Finally we extend the notion of order to rational points in 2.3.11.
Definition 2.3.11. [30. Definition 30] Let $R = G/H \in \mathbb{F}(C)^*$ and $P \in C$. The order of $R$ at $P$ is defined to be $\text{ord}_P(R) = \text{ord}_P(G) - \text{ord}_P(H)$. 

Intuitively the order of a function $f$ at a point $P$ is the measure of the multiplicity of the intersection of the curve $C$ with $f = 0$. We now have the machinery needed to present divisors and their basic properties.

2.4 Divisors

In this section we describe the concept of divisors and their basic properties.

Definition 2.4.1. [30. Definition 31] A divisor is a formal sum of $C(\mathbb{F})$-points,

$$D = \sum_{P_i \in C} m_i P_i \quad m_i \in \mathbb{Z}$$

where only a finite number of the $m_i$ are non-zero. The degree of $D$, denote $\text{deg} D$, is the integer $\sum_{P \in C} m_P$. The order of $D$ at $P$ is the integer $m_P = \text{ord}_P(D)$.

We describe two properties of divisors, their weight and support in definitions 2.4.2 and 2.4.3.

Definition 2.4.2. [30. Definition 31] The number of points of a divisor is called the weight of the divisor.

Definition 2.4.3. [30. Definition 37] Let $D = \sum_{P_i \in C} m_i P_i$ be a divisor. The support of $D$ is the set

$$\text{supp}(D) = \{P \in C | m_i \neq 0\}.$$ 

The set of all divisors, denoted by $\mathbb{D}$, forms an additive abelian group under the addition rule

$$\sum_{P_i \in C} m_i P_i + \sum_{P_i \in C} n_i P_i = \sum_{P_i \in C} (m_i + n_i) P_i.$$
Let $D^0$ denote the subgroup of $D$ consisting of degree 0 divisors only. The representation of divisors using polynomials in Section 2.6.2 requires a description of a greatest common divisor for divisors given in Definition 2.4.4

**Definition 2.4.4.** [30, Definition 32] Let $D_1 = \sum_{P \in C} m_P P$ and $D_1 = \sum_{P \in C} n_P P$ be two divisors. The greatest common divisor of $D_1$ and $D_2$ is defined to be

$$\gcd(D_1, D_2) = \sum_{P \in C} \min(m_P, n_P) P - (\sum_{P \in C} \min(m_P, n_P)) \infty$$

2.5 The Jacobian $\mathcal{J}$ of a Curve

In this section we describe the Jacobian of a curve $C$. We begin with pointing out that a whole class of divisors arise from rational functions $R \in \mathbb{F}(C)$ called principal divisors as described in Definitions 2.5.1 and 2.5.2.

**Definition 2.5.1.** [30, Definition 33] Let $R \in \mathbb{F}(C)^*$. The divisor of $R$ is

$$\text{div}(R) = \sum_{P \in C} (\text{ord}_P R) P.$$ 

Notice that by Theorem 2.3.10 the divisor of a rational function is a finite formal sum and has degree 0, so for all $R \in \mathbb{F}(C)$, $\text{div}(R) \in D^0$.

**Definition 2.5.2.** [30, Definition 36] A divisor $D \in D^0$ is called a principal divisor if $D = \text{div}(R)$ for some rational function $R \in \mathbb{F}(C)^*$. The set of all principal divisors is denoted as $\mathcal{P}$ and $\mathcal{P}$ is a subset of $D^0$.

With $\mathcal{P}$ defined we can now describe the Jacobian as given in Definition 2.5.3

**Definition 2.5.3.** [30, Definition 36] The quotient group

$$\mathcal{J} = \frac{D^0}{\mathcal{P}}$$
is called the Jacobian of the curve $C$. If $D_1, D_2 \in \mathbb{D}^0$ and $D_1 - D_2 \in \mathbb{P}$ then we write $D_1 \approx D_2$. In that case, $D_1$ and $D_2$ are said to be equivalent divisors.

A convenient set of coset representatives and algorithms to perform operations exists for doing computations on $\mathbb{J}$. In Section 2.6.2 a unique representation of elements in the Jacobian called Mumford representation is described. Composition rules for the group operation are stated in Section 2.6.3 using Cantor’s algorithm.

2.6 Representing Divisors

In this section we describe how to uniquely represent divisor classes in the Jacobian. We start with describing a unique representation of a divisor class using reduced divisors, then describe a corresponding representation using polynomials called the Mumford representation of a divisor class.

2.6.1 Reduced Divisors

To define a reduced divisor we first start with the definition of a semi-reduced divisor in Definition 2.6.1

**Definition 2.6.1.** [30, Definition 38] A semi-reduced divisor is a divisor of the form

$$D_{sr} = \sum_{P_i \in C} m_i P_i - \left( \sum_{P_i \in C} m_i \right) \infty$$

where each $m_i \geq 0$ and the $P_i \in C \setminus \{\infty\}$ are finite points such that when $P_i \in \text{supp}(D)$ then $\tilde{P}_i \notin \text{supp}(D)$, unless $P_i = \tilde{P}_i$, in which case $m_i = 1$.

Lemma 2.6.2 shows that for each element $D \in \mathbb{D}^0$ there exists a semi-reduced divisor $D_{sr}$ equivalent to $D$. A proof for Lemma 2.6.2 can be found in [30, Lemma 39].

**Lemma 2.6.2.** [30, Lemma 39] For each divisor $D \in \mathbb{D}^0$ there exists a semi-reduced divisor $D_{sr} \in \mathbb{D}^0$ such that $D \approx D_{sr}$. 

20
Note that semi-reduced divisors are not unique in their equivalence class with respect to $\mathcal{J}$. In the case of hyperelliptic curves, a stronger result using Riemann-Roch theorem [14] exists stating that every element of $\mathcal{J}$ can be uniquely represented by a “reduced” divisor. This is important because in Section 2.6.3 computations on the group operation of $\mathcal{J}$ are done on the representative divisors of the divisor classes, and so the representations must be unique. See Definition 2.6.3 and Theorem 2.6.4.

**Definition 2.6.3.** [30, Definition 44] Let $D = \sum_{P_i \in C} m_i P_i - (\sum_{P_i \in C} m_i) \infty$ be a semi-reduced divisor. If $\sum_{P_i \in C} m_i \leq g$ (where $g$ is the genus of $C$) then $D$ is called a reduced divisor. A divisor $D = \sum_{P_i \in C} m_i P_i - (\sum_{P_i \in C} m_i) \infty \in \mathcal{D}^0$ is reduced if:

1. All of the $m_i$ are non-negative, and if $P_i$ is equal to its opposite then $m_i \leq 1$.
2. If $P_i \neq \bar{P}_i$, then $P_i$ and $\bar{P}_i$ do not both occur in the sum.
3. $\sum_{P_i \in C} m_i \leq g$.

**Theorem 2.6.4.** [30, Theorem 47] For each divisor $D \in \mathcal{D}^0$ there exists a unique reduced divisor $D_r$ such that $D \approx D_r$.

A proof of the Theorem 2.6.4 can be found in [30, Theorem 47].

2.6.2 Mumford Representation

In this section we describe the Mumford representation for divisor classes [33]. Two polynomials with certain properties can be used to describe a semi-reduced divisor, reducing divisor arithmetic to polynomial arithmetic which is better suited for implementations. We begin with Lemma 2.6.5 stating the existence of a $v(x)$ polynomial required, then describe that every semi-reduced divisor can be represented by two polynomials in Theorem 2.6.6. The proof of both can be found in [30, Theorem 42].
Lemma 2.6.5. [30, Lemma 41] Let $P = (a, b)$, be an non-special point of $C$. Then for each $k \geq 1$, there exists a unique polynomial $v_k \in \mathbb{F}[x]$ such that:

1. $\deg_x v_k < k$;
2. $v_k(a) = b$;
3. $v_k^2(x) + v_k(x)h(x) \equiv f(x) \pmod{(x-a)^k}$.

Theorem 2.6.6. [30, Theorem 42] Let $D_{sr} = \sum_{P_i \in C} m_i P_i - (\sum_{P_i \in C} m_i)\infty$ be a semi-reduced divisor where $P_i = (a_i, b_i)$. Let $u(x) = \prod (x-a_i)^{m_i}$ (monic). There exists a unique polynomial $v(x)$ satisfying:

1. $\deg_x v < \deg_x u$;
2. $v(a_i) = b_i$ for all $i$ for which $m_i \neq 0$;
3. $u(x)v(x)^2 + v(x)h(x) - f(x)$.

The Mumford representation of a semi-reduced divisor $D$ is $\gcd(\text{div}(u(x)), \text{div}(v(x) - y))$. For short we denote $D$ by $[u, v]$. Lemma 2.6.7 establishes the correspondence between Mumford representation and semi-reduced divisors. Proof of Lemma 2.6.7 can be found in [30, Lemma 43].

Lemma 2.6.7. [30, Lemma 43] Let $u(x), v(x) \in \mathbb{F}[x]$ such that $\deg_x u < \deg_x v$. Then $u|(v^2 + vh - f)$ if and only if $D = [u, v]$ is semi-reduced.

Recall that in $\mathcal{J}$ every equivalence class can be represented by a semi-reduced divisor. By Definition 2.6.3, reduced divisors are just semi-reduced divisors where $\deg_x u \leq g$ where $g$ is the genus of $C$. We denote equivalence classes, i.e. elements of $\mathcal{J}$, by $[D] = [u, v]$ where $D$ is the reduced divisor representing the class. The inverse of $[D]$ is represented by $[u, -h - v]$, where the second polynomial is given modulo $u$ if necessary. To establish that we are in fact working with a group we denote the neutral element to be $[D_0] = [1, 0]$.  

22
Reduced divisors are necessary for computation because the degree of the representative polynomials is kept small and representation of the corresponding divisor class is unique. We next describe how to compute the group operation in Section 2.6.3.

2.6.3 Cantor’s Algorithm

In this section we describe an algorithm that computes the addition of divisor classes in \( \mathbb{J} \) called Cantor’s algorithm, see [4]. Cantor only described his algorithm over odd characteristic fields; for an extension to arbitrary fields see [26]. Recall that an element of \( \mathbb{J} \) is an equivalence class represented by a reduced divisor \([D] = [u, v]\) using Mumford representation. Equating the linear factors of the polynomial \( u \) to 0 corresponds to the \( x \) coordinates of the points in the support of \( D \). The negative of the polynomial \( v \) interpolates said points in the support of \( D \). The algorithm takes an input of two reduced divisors \( D_1 \) and \( D_2 \) (representing divisor classes \([D_1]\) and \([D_2]\)) and produces a reduced divisor \( D' \) representing the composed equivalence class \([D_1 + D_2] = [u'', v'']\) where the polynomials \( u'' \) and \( v'' \) have the same relation to \([D_1 + D_2]\). Note that semi-reduced divisors can be given as input, but to minimize computational complexity, reduced divisors are required. The composition step adds the two divisor class representatives together but produces a semi-reduced output divisor. The reduction algorithm “reduces” the semi-reduced divisor to a unique equivalent reduced divisor representative. Generally both the composition Algorithm 2.1 and reduction Algorithm 2.2 are presented as one algorithm where the reduction Algorithm 2.2 is applied right after the composition Algorithm 2.1.

**Algorithm 2.1** Cantor’s Addition Algorithm in \( \mathbb{J} \) (Composition)

**Input:** \( D_1 = [u_1, v_1], \ D_2 = [u_2, v_2], \ C : y^2 + h(x)y = f(x). \)

**Output:** \( D = [u, v] \) semi-reduced with \( D \equiv D_1 + D_2 \)

1: Compute \( d_1 = \gcd(u_1, u_2) = e_1u_1 + e_2u_2; \)
2: Compute \( d = \gcd(d_1, v_1 + v_2 + h) = c_1d_1 + c_2(v_1 + v_2 + h); \)
3: Let \( s_1 = c_1e_1, \ s_2 = c_1e_2, \ s_3 = c_2; \)
4: \( u = \frac{u_1u_2}{d^2}; \)
5: \( v = \frac{s_1u_1v_2 + s_2u_2v_1 + s_3(v_1v_2 + f)}{d} \)
Algorithm 2.2 Cantor’s Reduction Algorithm in $\mathbb{J}$ (Reduction)

**Input:** $D = [u, v]$ semi-reduced.

**Output:** $D' = [u', v'] = D + D'$ reduced with $D \equiv D'$.

1: Let $u' = \frac{f-vh-v^2}{u}$;
2: Let $v' = (-h-v) \pmod{u'}$;
3: If $\deg u' > g$ put $u = u'$, $v = v'$;
   
   Goto Step 1;
4: Make $u'$ monic.

Theorems 2.6.8 and 2.6.9 state that the composition and reduction algorithms described work and produce the correct output. Proofs of both theorems are given in [30, Theorem 49] and [30, Theorem 51].

**Theorem 2.6.8.** [30, Theorem 49] Let $D_1 = [u_1, v_1]$ and $D_2 = [u_2, v_2]$ be semi-reduced divisors. Let $u$ and $v$ be defined as in Step 4 and 5 of Algorithm 2.1. Then $D = [u, v]$ is a semi-reduced divisor and $D \approx D_1 + D_2$.

**Theorem 2.6.9.** [30, Theorem 51] Let $D = [u, v]$ be a semi-reduced divisor. Then the divisor $D' = [u', v']$ returned by Algorithm 2.2 is reduced and $D \approx D'$.

These algorithms provide a universal way for computing arithmetic in $\mathbb{J}$ which applies to any genus and characteristic. However, in a straight forward implementation several unneeded computations are done. In the next chapter we describe different methods and specializations for reducing the complexity of these algorithms.
Chapter 3

Generic Methods to Composition
In this chapter we describe the different generic methods one can use to compute divisor class composition in genus 2. There are two main methods; Harley’s improvements to Cantor’s algorithm and the geometric method. Harley’s method is a specialization of Cantor’s algorithm. Harley noticed that breaking up the computations into certain sub-expressions allowed for reuse of those sub-expressions, saving on operations overall [20]. The geometric method uses the fact that divisors can be viewed as points on the curve. A polynomial that interpolates the points in the support of the input divisor classes is found. The interpolating polynomial then also intersects the curve at two more points, yielding the support of the composed divisor class. We will focus on the structural details of composition using the two methods working with generic polynomial arithmetic.

Both methods depend on the recognition of a frequent case that the divisor classes to be composed can take. Composition using both methods is greatly simplified when the divisor classes are in the frequent case. A division by zero is produced during the composition algorithms for the frequent case using both methods if the input divisor classes are not in the frequent case. The frequent case occurs so frequently in cryptographic practice [34] that we account for the odd chance that input divisors are not in the frequent case by branching to a generic Cantor’s algorithm. The resulting loss in efficiency is considered insignificant.

We first discuss the frequent cases divisor classes can be in when composing. We then give an exposition of Harley’s method and the geometric method, followed by a summary of how the two methods relate to each other.

3.1 The Frequent Case

The number of field operations required to compose divisor classes can be reduced by distinguishing between possible cases according to the properties of the input. Using a generic algorithm for all cases of input forces unneeded computations most of the time, so we want to distinguish between frequent and infrequent cases. First we distinguish between the cases
where the input divisors are the same or different, denoted by doubling and adding respectively. Second, it is helpful to know the weights of the input divisor classes: in the frequent case the weight is always two.

The input to Cantor’s composition algorithm is two reduced divisors representing divisor classes in \( \mathbb{J} \), each represented by two polynomials \([u, v]\) and \([u', v']\) (Section 2.6.2). Recall that the weight of a reduced divisor is defined as the number of distinct finite points in its support, which can also be determined by the degree of \( u \). Cantor’s algorithm allows for arbitrary inputs, but as mentioned before, we almost always see just one case in practice. Two normally distributed polynomials have a linear factor in common with probability of about \( 1/q \) when working over \( \mathbb{F}_q \), a finite field determined by a power of a prime \( q \). In cryptographic applications this happens very rarely, for example, no common factors with probability \( P \approx 1 - 2^{-128} \) when the base field has order \( 2^{128} \). The same reasoning explains why we almost always have \( \deg u = \deg u' = 2 \), i.e.; the divisor has a weight of two. From now on we will call this the frequent case for both doubling and addition.

**Frequent case of addition**

The frequent case of adding divisors occurs if and only if \( D = [u, v] \) and \( D' = [u', v'] \) are both of weight two and \( \gcd(u, u') = 1 \). This means that there are no points in common between the divisors.

**Frequent case of doubling**

The frequent case of doubling a divisor occurs if and only if \( D = [u, v] \) is of weight two and \( \gcd(u, \tilde{h}) = 1 \) with \( \tilde{h} = h + 2v \), where \( h \) comes from the definition of the curve. This means that both points represented by the divisor are not opposites of each other, in a sense the same as having no points in common.

To circumvent problems in implementation for the odd chance that the input divisor
classes are not in the frequent case, checks can be built into the formulae that when failed, branch to Cantor’s algorithm. The checks are trivial, only requiring a check to see if certain intermediate values are zero during computation. Since this happens so rarely, there is very little effect on the overall efficiency when using these formulae for scalar multiplication of divisors.

3.2 From Cantor to Harley

In this section we present simplifications for computing Cantor’s algorithm (Section 2.6.3) that arose due to Harley [20, 19]. We follow closely from [27, 41].

Consider a reduced divisor \( D = [u, v] \) representing \([D]\). Harley noticed that \( v \) is a square root of \( f \) modulo \( u \), and suggested that one can double the multiplicity of all points in the support of \( D \) by using a Newton iteration to compute a square root modulo \( u^2 \). Thus, the general idea to doubling a divisor class is:

1. Newton iteration: set \( U = u^2 \), and \( V = v + [(f - hv - v^2)/(h + 2v)] \) (mod \( U \)),

2. Get “other” roots: \( u' = (f - hV - V^2)/U \),

3. Make \( u' \) monic,

4. Reduce \( V \): \( v' = -V \) (mod \( u' \)).

The general idea for addition is similar to doubling although the Newton iteration is replaced by a Chinese remainder calculation. We start with an exposition of addition in the frequent case.

3.2.1 Addition in the Frequent Case

For addition, the two divisors \( D = [u, v] \), and \( D' = [u', v'] \), to be composed consist of four points different from one another and from one another’s negative. The results from using
Cantor’s composition algorithm are $U = uu'$ and a polynomial $V$ of degree $\leq 3$ satisfying $U|V^2 + V h - f$. Since the inputs $u$ and $u'$ also have the properties $u|v^2 + vh - f$ and $u'|v^2 + v'h - f$, we notice that $v''n - hv'' - f = uu'k$, $v^2 - hv - f = um$ and $v^2 - hv' - f = u'n$ for some polynomials $k, m, n$. Thus, $V$ can be obtained using the Chinese Remainder Theorem

$$V \equiv v \pmod{u},$$
$$V \equiv v' \pmod{u'}.$$  

At this point $D''_{sr} = [V, U]$ corresponds to a representation of $D''$ the output divisor that is semi-reduced only (not reduced). A reduction needs to be applied to acquire the proper reduced Mumford representation of $[D'']$. From Cantor’s reduction step

\[
\begin{align*}
    u'' &= \frac{f - V h - V^2}{U} \text{ made monic}, \\
    v'' &= (-h - V) \pmod{u''}. \tag{3.1}
\end{align*}
\]

To optimize these formulae, we do not follow the above literally. By applying Garner’s Algorithm for Chinese Remainder Theorem \cite{31} we obtain :

\[
v'' = -h - \left[ \left( \frac{v' - v}{u} \pmod{u'} \right) u + v \right] \pmod{u''}
\]

from (3.1). The reduction of $U$ can be optimized by reusing expressions. Let $s \equiv (v' - v)/u \pmod{u'}$. Then

\[
\begin{align*}
    u'' &= \frac{f - h V - V^2}{uu'} \\
    &= \frac{1}{uu'}(f - h(su + v) - (su + v)^2) \\
    &= \frac{1}{uu'}(f - hsu - hv - s^2u^2 - 2vsu - v^2) \\
    &= \frac{1}{uu'}[(f - hv - v^2) - s(hu + su^2 + 2vu)] \\
    &= \frac{1}{u'} \left[ \frac{(f - hv - v^2)}{u} - s(h + su + 2v) \right].
\end{align*}
\]

The computations are split into sub-expressions that can be reused when computing both $v''$ and $u''$. Algorithm \cite{31} gives a summary of all the steps.
### Algorithm 3.1 Harley Frequent Case for Addition

**Input:** $D = [u, v], D' = [u', v']$

**Output:** $D'' = [u'', v''] = D + D'$

1: $k = \frac{(f - hv - v^2)}{u}$

2: $s \equiv \frac{(v' - v)}{u} \pmod{u'}$

3: $l = su$

4: $u'' = \frac{(k - s(h + l + 2v))}{u}$ made monic

5: $v'' \equiv -h - (l + v) \pmod{u''}$

### 3.2.2 Doubling in the Frequent Case

For doubling, the divisor $D = [u, v]$ representing $[D]$ has two points in its support that are not opposites of each other. The results from composition using Cantor’s algorithm are $U = u^2$, and a polynomial $V$ of degree $\leq 3$ satisfying $U|V^2 + Vh - f$. Notice now, $V^2 + Vh - f = u^2m$ for some polynomial $m$. So

$$V^2 + Vh - f \equiv f - Vh - V^2 \equiv 0 \pmod{u^2}. \quad (3.2)$$

Newton’s method for root approximation is used in numerical analysis to approximate roots of functions. Given a function $g$ and its derivative $g'$ over the reals, better approximations of roots can be taken by starting at a guess $x_0$ and using the iteration

$$x_{n+1} = x_n + \frac{g(x_n)}{g'(x_n)}.$$

This idea adapts to our discrete setting, but rather than giving approximations there is an equality. Put $g(t) = f - t^2 - th$. Then $V$ can be seen as a root of $g(t)$ modulo $U = u^2$ and $v$ as a root of $g(v)$ modulo $u$ by equation 3.2 that results in the equality

$$V \equiv v - \frac{g(v)}{g'(v)} \pmod{U}$$

$$\equiv v - \left(\frac{v^2 + hv - f}{h + 2v}\right) \pmod{U}$$

$$\equiv v + \left(\frac{f - hv - v^2}{h + 2v}\right) \pmod{U}.$$
The reduction step is still left: again we have

\[ u'' = \frac{f - Vh - V^2}{uu'} \] made monic,

\[ v'' = (-h - V) \pmod{u''}. \]

To optimize these formulae, we do not follow this literally. By taking the path to compute \( V \) above, some pre-computed quantities can be reused when computing \( u'' \). Let \( s \equiv k/(h + 2v) \mod u \), where \( k = (f - hv - v^2)/u \). Since \( V^2 + Vh - f \equiv 0 \mod U \) implies \( V^2 + Vh - f \equiv 0 \mod u \), we get \( V = v + su \), resulting in

\[ u'' = \frac{f - hV - V^2}{uu} \]
\[ = \frac{1}{uu} (f - h(v + su) - (v + su)^2) \]
\[ = \frac{1}{uu} (f - hv - hsu - v^2 - 2vsu - s^2u^2) \]
\[ = \frac{1}{u} \left( \frac{f - hv - v^2}{u} - hs - 2vs - s^2u \right) \]
\[ = s^2 - s(h + 2v) - k \cdot \frac{u}{u}. \]

Algorithm 3.2 results from splitting the above expression into sub-expressions.

**Algorithm 3.2** Harley Frequent Case for Doubling

**Input:** \( D = [u, v] \)

**Output:** \( D' = [u', v'] = 2D \)

1: \( k = \frac{(f - hv - v^2)}{u} \)
2: \( s \equiv \frac{k}{(h + 2v)} \pmod u \)
3: \( l = su \)
4: \( u' = s^2 - \frac{s(h + 2v) - k}{u} \) made monic
5: \( v' \equiv -h - \frac{l + v}{u''} \pmod{u''} \)

### 3.3 Geometric Method

In this section we cover the geometric method for composition of two divisor classes in the Jacobian of a hyperelliptic curve. Analogous to the elliptic curve chord and tangent
method, points in the support of two divisor classes can be interpolated resulting in new points corresponding to the support of the composed divisor. The main difference between this method and using Cantor’s algorithm is that the geometric method uses linear algebra, i.e., solving a system of linear equations, to compute \( V \). We start with some background on how this method works for divisors in Mumford representation, then given an exposition of computing double and add. Finally we show that the algebraic and geometric methods for divisor class composition are equivalent. The work here mainly comes from [9, 28]. The authors only treat the case of odd characteristic; we extend these ideas to the arbitrary case, covering even and odd characteristic implementations of doubling and adding in the geometric setting.

3.3.1 Group Law Algorithm in the Frequent Case

The authors in [5] describe the geometric group law for divisors defined over odd characteristic fields. We expand those ideas to describe the group law for composing over arbitrary characteristic fields, only for the frequent case. Denote the input divisors by \( D = P_1 + P_2 \) and \( D' = Q_1 + Q_2 \), where \( P_i \) and \( Q_i \) are points in their support on the hyperelliptic curve \( C \). In the frequent case for addition, all four points are distinct as mentioned before. These four points uniquely determine a cubic polynomial \( y = l'(x) \), that interpolates them. In the frequent case for doubling i.e.; \( D' = D \), the two points are not equal to each other’s negative as before, and we take tangents of multiplicity two at the points to produce the cubic polynomial \( y = l'(x) \).

Using the definition of the hyperelliptic curve, the intersection of the cubic \( y = l' \) with the curve by substituting \( y = l'(x) \) into the equation of the curve yields

\[
l'^2(x) + l'(x)h(x) = f(x),
\]

a degree six polynomial whose roots represent intersection points between \( l'(x) \) and \( C \). There are six solutions, four of which come from the points we started with. Let \( R_1 = (x_5, y_5) \) and
$R_2 = (x_6, y_6)$ be the other two points, and define

$$D'' = D' + D$$

where $D'' = -R_1 + -R_2$. This definition comes from the identity

$$D + D' - D'' = \infty.$$

where $\infty$ is the point at infinity in our setting.

### 3.3.2 Geometrically Computing Composition

Next we describe the steps taken to compute composition of two divisors using the geometric method. We split the exposition into the two main cases of adding and doubling. As before we assume the input divisors are in Mumford representation. We start with an exposition of adding in the frequent case.

#### Addition in the Frequent Case

Given two divisors $D = [u, v]$ and $D' = [u', v']$ representing $[D]$ and $[D']$, we want to compute $D + D' = D'' = [u'', v'']$. The composition step in addition for $D$ and $D'$ involves building a four by four linear system of equations that we solve for the coefficients of $l'$. Let $l' = l'_3 x^3 + l'_2 x^2 + l'_1 x + l'_0$ be the desired polynomial that interpolates the four non-trivial points in the supports of $D$ and $D'$. Consider the divisor class $D$. Since we let $l'(x) = y$, we get that $v(x) - l'(x) = 0$ for the $x$ components of the non-trivial points in the support of $D$.

To produce equations linear in the coefficients of $l'(x)$, we reduce modulo $u$ to create an equation of degree one relating the coefficients of $l'$. The equality

$$0 \equiv l'_3 x^3 + l'_2 x^2 + l'_1 x + l'_0 - v_1 x - v_0 \pmod{x^2 + u_1 x + u_0},$$

$$\equiv (l'_3 u_1^2 - u_0) - l'_2 u_1 + l'_1 - v_1) x + (l'_3 u_1 u_0 - l'_2 u_0 + l'_0 - v_0) \pmod{x^2 + u_1 x + u_0},$$

$$= a_1 x + a_0,$$

(3.3)
provides two equations, \((a_1 = 0 \text{ and } a_0 = 0)\) relating the four coefficients of \(l'\) linearly. A similar construction is used with \(D'\) to produce two more linear equations identically. The resulting four by four linear system of equations is

\[
\begin{pmatrix}
1 & 0 & -u_0 & u_1u_0 \\
0 & 1 & -u_1 & u_1^2 - u_0 \\
1 & 0 & -u'_0 & u'_1u'_0 \\
0 & 1 & -u'_1 & u'_1^2 - u'_0
\end{pmatrix} \times
\begin{pmatrix}
l'_0 \\
l'_1 \\
l'_2 \\
l'_3
\end{pmatrix} =
\begin{pmatrix}
v_0 \\
v_1 \\
v'_0 \\
v'_1
\end{pmatrix}. 
\tag{3.4}
\]

Then \(l' = V\) interpolates all four points. Recall that \(U = uu'\) is the polynomial that corresponds to all four points in the support of \(D_{sr}\), where \(D_{sr} = [U, V]\) is the representation of the divisor \(D'' = [u'', v'']\) before a reduction is taken.

The next step is to reduce \(D_{sr}\) to \(D''\). We use Cantor’s reduction step (Algorithm 2.2) here, giving the expressions

\[
u'' = \frac{l'^2 + hl' - f}{uu'}, \text{ made monic}
\]
\[
v'' \equiv -h - l' \pmod{u''}.
\]

The entire algorithm is given in Algorithm 3.3

**Algorithm 3.3 Geometric Frequent Case for Adding**

**Input:** \(D = [u, v], D' = [u', v']\)

**Output:** \(D'' = [u'', v''] = D + D'\)

1: Compute \(l'\):
2: Construct polynomial equations through (3.3) using \(u, v\)
3: Construct polynomial equations through (3.3) using \(u', v'\)
4: Solve linear system (3.4) for \(l' = l'_3x^3 + l'_2x^2 + l'_1x + l'_0\)
5: \(u'' = \frac{l'^2 + hl' - f}{uu'}\), made monic,
6: \(v'' \equiv -h - l' \pmod{u''}\)

Doubling in the Frequent Case

Given a divisor \(D = [u, v]\), we want to compute \(2D = D' = [u', v']\). As in the addition case, a linear system of equations is used to solve for the coefficients of \(l'\). Equation 3.3 immediately
provides two linear equations. There are two possible approaches for obtaining the other two equations. The first is matching derivatives

\[ \frac{dy}{dx} = \frac{dl'}{dx} \]

point by point on the curve. This approach ensures multiplicity is accounted for and gives rise to two linear equations. The other way is to reduce the substitution of \( l' = y \) into the curve \( C \) modulo \( u^2 \) (reduction modulo \( u^2 \) instead of \( u \) ensures the zeros have multiplicity two) and then linearize using the equations that arise from substituting \( v = y \) into \( C \) and reducing modulo \( u \). This approach results in four more equations. The second approach produces the simplest equations, and thus is used in this exposition. Maple code producing all equations and linearizing them is given in Appendix A.2. Here we present the smallest equations,

\begin{align*}
a_3 &= l'_3(-3h_1u_0 - 6u_0v_1) + l'_2(h_2u_1^2 + 2h_2u_0 - 2u_1v_1 + h_0 + 2v_0) \\
&\quad + l'_1(2h_1 + 4v_1 - h_1u_1 - 2h_2u_1) + l'_0(-3h_2) \\
&\quad + 2h_2u_1v_1 - 2u_1^3 - h_1v_1 + 4h_2v_0 - 2u_1u_0 - 3v_1^2 - f_2, \\
&\quad (3.5) \\
a_4 &= l'_3(h_2u_0 - 2h_1u_1 - 4u_1v_1 + h_0 + 2v_0) + l'_2(h_2u_1 + h_1 + 2v_1) \\
&\quad + l'_1(-2h_2) + 3h_2v_1 - 3u_1^2 + 2u_0 - f_3, \\
&\quad (3.6) \\
\end{align*}

where \( a_3 = a_4 = 0 \).

Let

\[ XY = Z \]

be the four by four system of equations using the simplest equations above and from equation 3.3. The rows of \( X \) are given as

\[ \begin{align*}
(1, 0, -u_0, u_1u_0); \\
(0, 1, -u_1, u_1^2 - u_0); \\
(-3h_2, 2h_1+4v_1-h_1u_1-2h_2u_1, h_2u_1^2+2h_2u_0-2u_1v_1+h_0+2v_0, -3h_1u_0-6u_0v_1); \\
\end{align*} \]
Let \( Y = (l'_0, l'_1, l'_2, l'_3)^T \) and the rows of \( Z \) are

\[
(-v_0);
\]

\[
(-v_1);
\]

\[
(2h_2u_1v_1 - 2u_1^2 - h_1v_1 + 4h_2v_0 - 2u_1u_0 - 3v_1^2 - f_2);
\]

\[
(3h_2v_1 - 3u_1^2 + 2u_0 - f_3).
\]

Solving for \( Y \) produces \( l' = V \), the polynomial that interpolates all four points. Recall that \( U = uu' \) is the polynomial that corresponds to all four points in the support of \( D'_{sr} \), where \( D'_{sr} = [U, V] \) is the semi-reduced representation of the divisor \( D' = [u', v'] \) before a reduction is taken.

The next step is to reduce \( D'_{sr} \) to \( D' \). We use Cantor’s reduction step (Algorithm 2.2) here, giving the expressions

\[
u' = \frac{l'^2 + hl' - f}{u'^2}, \text{ made monic}
\]

\[
v' \equiv \frac{-h - l'}{u''} \mod u''.
\]

The entire algorithm is given in Algorithm 3.4.

**Algorithm 3.4 Geometric Frequent Case for Doubling**

**Input:** \( D = [u, v] \)

**Output:** \( D' = [u', v'] = 2D \)

1: Compute \( l' \):

2: Construct polynomial equations from equation (3.3) using \( v, u \)

3: Solve linear system for \( l' \) using equations (3.5) and (3.6) constructed in Step 2

4: \( u' = \frac{l'^2 + hl' - f}{u'^2} \) made monic,

5: \( v' \equiv \frac{-h - l'}{u''} \mod u'' 

3.3.3 Equivalence of Algebraic and Geometric Group Laws

In this section we explain the equivalence of algebraic and geometric group laws in the frequent case for the composition of two divisor classes [28]. The equivalence comes from \( l'(x) \) being equal to the the semi-reduced \( V \) output of Cantor’s algorithm i.e.; \( l + v \) taking
the Harley route. The steps in the reduction are the same for both methods, but we explore the geometric meaning behind it.

Let \( D = P_1 + P_2 \), \( D' = Q_1 + Q_2 \) where \( P_1 = (x_1, y_1) \), \( P_2 = (x_2, y_2) \), \( Q_1 = (x_3, y_3) \) and \( Q_2 = (x_4, y_4) \). Let \( l'(x) \) be the cubic polynomial that interpolates the four points, and let the points \( R_1, R_2 \) be the other two points of intersection between \( l'(x) \) and the curve. If \( R_1 = (x_5, y_5) \) and \( R_2 = (x_6, y_6) \), then

\[
\frac{d}{dx}(x - x_5)(x - x_6),
\]

and \( v''(x) \) equal to the line passing through the points \( -R_1 = (x_5, -y_5) \) and \( -R_2 = (x_6, -y_6) \) corresponds to the divisor \( D'' = [u'', v''] \) using Mumford’s representation (Section 2.6.2).

Recall the steps of Cantor’s composition algorithm. Let \([u, v]\) and \([u', v']\) be the Mumford representations of \( D \) and \( D' \) respectively, so \( u = (x - x_1)(x - x_2) \) and \( u' = (x - x_3)(x - x_4) \). Applying the frequent case to Cantor’s algorithm (Section 2.1) gives \( d = \gcd(u, u', (v + v' + h)) = 1 = s_1u + s_2u' + s_3(v + v' + h) \). One of the values \( s_i \) can always be set to zero when \( d = 1 \). Let \( s_3 = 0 \) for addition and \( s_2 = 0 \) for doubling. The rest of the \( s_i \) have degree one. Cantor’s algorithm outputs an intermediate semi-reduced representation \( D_{sr} = [U, V] \) where

\[
U = uu',
\]

\[
V = s_1uv' + s_2u'v \pmod{U},
\]

for addition, or

\[
U = u^2,
\]

\[
V = s_1uv' + s_3(2v + f) \pmod{U},
\]

for doubling. The polynomial \( U \) has degree four and the polynomial \( V \) has degree three. At this point the following relation holds,

\[
V = l'(x).
\]

37
The proof in the addition case is given in [28], the proof for doubling is similar. We summarize the proof for addition here.

Let \( \tilde{v} = s_1 uv' + s_2 u'v \), where \( \tilde{v} \) is \( V \) without the modular reduction in equation 3.8. The idea is to show that \( l' - \tilde{v} \) and \( U \) are constant multiple of each other. Plugging in \( x_i \) where \( i = 1, 2 \), results in

\[
u(x_i) = 0 \text{ and } v(x_i) = y_i,
\]

so

\[
\tilde{v}(x_i) = s_2(x_i)u'(x_i)y_i.
\]

Notice

\[
d = 1 = s_1u + s_2u' \implies s_2(x_i)u'(x_i) = 1,
\]

so then \( \tilde{v}(x_i) = y_i \). A similar argument holds for \( i = 3, 4 \). Now \( l' - \tilde{v} \) and \( U \) both have degree four and have zeros at \( x_i = 1, 2, 3, 4 \), so one is a constant multiple of the other. Thus \( V \equiv l'(x) \pmod{U} \). Notice \( l' \) is cubic, \( U \) is quartic and \( V = \tilde{v} \pmod{U} \), so \( V = l' \).

Next we look at reduction. In Cantor’s reduction step,

\[
u'' = \frac{f - Vh - V^2}{uu'},
\]

producing

\[
uu'u'' = f - Vh - V^2 = f - l'h - l'^2,
\]

by equation 3.9. Recall that \( u \), and \( u' \) make up the \( (x - x_1)(x - x_2)(x - x_3)(x - x_4) \) portion of \( f - l'h - l'^2 \), so \( u'' \) must be a constant multiple of \( (x - x_5)(x - x_6) \). The next step of Cantor’s reduction is

\[
v'' = -h - V \pmod{u''}.
\]

Recall \( l'(x) \) passes through \( R_1 \) and \( R_2 \), so \( l'(x_5) = y_5 \) and \( l'(x_6) = y_6 \). Negating and reducing modulo \( u'' \) makes \( v''(x_5) = -y_5 \) and \( v''(x_6) = -y_6 \), since \( u'' \) is zero at those points. We see that \( v'' \) has degree one, so it is a line passing through \( -R_1 \) and \( -R_1 \) as required. The output
of Cantor’s algorithm corresponds to the divisor obtained when using the geometric version of the group law.

3.3.4 Summary

There are a few points to make about the differences and similarities of both methods to computing composition of two divisors. Although the geometric method is motivated quite differently than the algebraic method, both end up having very similar computations.

We emphasize that from Sections 3.2.1 and 3.2.2 where we describe the steps for adding and doubling divisors using Cantor’s algorithm, \( su + v = l + v = V = l' \) where \( l' \) is the interpolating polynomial from the geometric method. \( V \) is the intermediate polynomial needed to compute the reduced output divisor when composing two divisors using Mumford’s representation. The geometric method computes \( l' \) in a different way than \( l + v \) but after those intermediate values are found identical techniques can be used to compute the reduced output divisor.
Chapter 4

Explicit Formulae
In this chapter we describe how to derive explicit formulae based on the generic algorithms presented in the previous chapter for the affine and projective settings. Explicit formulae are an efficient way to compose two divisor classes, as they directly deal with operations over the base field instead of using polynomial arithmetic. As a result, algebraic tricks, shortcuts and the omission of unused coefficients can be implemented to reduce the number of operations required.

We first present an overview of the process for converting a polynomial algorithm to an explicit formula. We give an exposition of curve simplifications, algebraic techniques and shortcuts we used to create our formulae. Finally we give an exposition of projective coordinates and techniques used to improve computations in that setting.

4.1 From Polynomial Algorithm to Explicit Formula

In this section we describe the process of converting a polynomial algorithm to an explicit formula for divisor class doubling, adding and tripling. There are three main aspects: representing polynomials, simplification of the hyperelliptic curve and techniques that reduce the number of operations. We follow closely with work presented in [27].

4.1.1 Representing Polynomial Arithmetic as Explicit Formulae

A polynomial algorithm directly deals with polynomials; the input and output are polynomials and each step multiplies, adds, divides, polynomials together. The first step for converting a polynomial algorithm to an explicit formula is to consider polynomials as sets of their respective coefficients, i.e., elements of the base field the polynomial is defined over. We can convert a polynomial operation to operations on field elements by considering what happens to the coefficients of the polynomial during the polynomial operation. For example, consider Harley doubling (Algorithm 3.2) from the previous chapter. The input is two polynomials \(u = x^2 + u_1 x + u_0\) and \(v = v_1 x + v_0\) that represent a reduced divisor and all the
steps consist of polynomial operations. We consider \( u \) and \( v \) as the set \([u_1, v_1, u_0, v_0]\) instead, and for example, we compute Step 3

\[
l_3x^3 + l_2x^2 + l_1x + l_0 = (s_1x + s_0)(x^2 + u_1x + u_0)
\]

in the algorithm with the formula

\[
\begin{align*}
l_3 &= s_1; \\
l_2 &= s_0 + u_1s_1; \\
l_1 &= s_1u_0 + s_0u_1; \\
l_0 &= s_0u_0.
\end{align*}
\]

Every step in every algorithm we covered in the previous chapter can be given as an explicit formula like this. To give an explicit formula for an entire algorithm, we sequentially place the converted steps in order. Explicit formulae allow for optimization at the field element level.

4.1.2 Curve Simplifications

Many steps of the polynomial algorithms presented in the previous chapter involve computations with \( f \) and \( h \), so explicit formulae heavily depend on the equation of the hyperelliptic curve. In this section we consider what assumptions and simplifications we can make about the curve itself. We first recall isomorphisms that transform a curve into another isomorphic curve with a less complex equation [7].

Recall that a hyperelliptic curve can be given as

\[ C : y^2 + h(x)y = f(x) \]

where \( h(x) \) is at most degree two, and \( f(x) \) is at most degree five. An isomorphic transformation of \( C \) is defined as a transformation of the variables \( x \) and \( y \) (change of variables) to other expressions involving \( x \) and \( y \) [7]. What isomorphic transformations can be applied
depends on the characteristic of the base field the curve is defined over. We now give an exposition of what isomorphic transformations we can take over odd and even characteristic fields as described in [27].

Odd Characteristic
In the odd characteristic case, applying the substitution \( y \rightarrow y - h(x)/2 \) invokes the transformation

\[
y^2 + h(x)y = f(x) \rightarrow (y - h(x)/2)^2 + h(x)(y - h(x)/2) = f(x)
\]

\[
\Rightarrow y^2 - 2h(x)y/2 + (h(x))^2/4 + h(x)y - 2(h(x))^2/4 = f(x)
\]

\[
\Rightarrow y^2 = f(x)'
\]

where \( f' = f + (h(x))^2/2 \). Since this transformation is always possible, \( h = 0 \) can be assumed.

If the characteristic of the base field is not five, the transformation \( x \rightarrow x - f_4/5 \) that takes \( f(x) \rightarrow f(x)' \) to get

\[
f(x)' = x^5 + f_3'x^3 + f_2'x^2 + f_1'x + f_0'
\]

can be achieved.

Even Characteristic
In even characteristic \( h \) must be non-zero because division by two is not possible and the transformation

\[
y \rightarrow y - \frac{h(x)}{2}
\]

can not be made. For our genus two setting, \( h \) non-constant is guaranteed because otherwise the curve is singular. As the formulae for doubling heavily rely on the coefficients of the curve that are non-zero, we consider the following transformations:

\[
y \rightarrow h_2^3y + f_3h_2x + \frac{f_3(f_3 + h_1h_2 + f_4h_2^2 + f_2h_2^2)}{h_2^3}, \quad x \rightarrow h_2^3x + f_4,
\]

where dividing the equation by \( h_2^{10} \) results in \( h_2 = 1 \) and \( f_4 = f_3 = f_2 = 0 \).
With these isomorphic transformations the number of field operations needed in the explicit formulae for algorithms from the previous chapter are reduced. Many of the coefficients in the polynomials representing the curve can be assumed to be zero or one, and in summary \( h_2 = h_1 = h_0 = f_4 = 0 \) in odd characteristic and \( f_4 = f_3 = f_2 = 0, h_2 = 1 \) in even characteristic. Next we look at techniques to further reduce field operations.

### 4.2 Techniques to Improve Explicit Formulae

In this section we list the main techniques used to improve the efficiency of our explicit formulae. These techniques are used in the doubling and adding formulae, as well as throughout our tripling formulae. Most of these techniques are found in literature: see [27, 41] for relevant expositions, or the texts [38, 7] for more algebraic expositions.

#### 4.2.1 Karatsuba Multiplication

In 1962 Karatsuba introduced a novel algorithm for multiplication of polynomials [24]. Compared to the schoolbook method, the Karatsuba method saves field multiplications at the cost of extra field additions. Finding the product of two polynomials occurs regularly in divisor class composition using Mumford representation, so field operations are reduced by taking advantage of this technique. Here we give an example of multiplying two linear polynomials.

Given two polynomials \( A(x) = a_1 x + a_0 \) and \( B(x) = b_1 x + b_0 \), let \( D_0 = a_0 b_0 \), \( D_1 = a_1 b_1 \), and \( D_{0,1} = (a_0 + a_1)(b_0 + b_1) \). The product of \( A(x) \) and \( B(x) \) can be given as

\[
C(x) = D_1 x^2 + (D_{0,1} - D_1 - D_0) x + D_0.
\]

This works because

\[
(D_{0,1} - D_1 - D_0) = (a_0 + a_1)(b_0 + b_1) - a_1 b_1 - a_0 b_0
\]

\[
= a_0 b_0 + a_0 b_1 + a_1 b_0 + a_1 b_1 - a_1 b_1 - a_0 b_0
\]

\[
= a_0 b_1 + a_1 b_0,
\]
as required by the schoolbook method. In this case one multiplication is traded for three additions. This technique can be applied, often more than once, to multiplying polynomials of any degree. This technique is used every time there is multiplication of polynomials throughout our formulae.

4.2.2 Karatsuba Reduction

The technique introduced by Karatsuba [24] is also used to compute polynomial modulo reduction efficiently. If the modulus is monic, and there is a one degree difference between the modulus and quotient polynomials, one field multiplication can be saved over the schoolbook method. To reduce a polynomial of degree three modulo a polynomial of degree two as is required in the geometric and Harley composition algorithms,

\[ ax^3 + bx^2 + cx + d \equiv (c - (i + j)(a + (b - ia)) + ia + j(b - ia))x + d - j(b - ia) \mod x^2 + ix + j \]

is used requiring only three multiplications instead of four. This technique is used when computing reducing \( V \mod U \) in all our formulae.

4.2.3 Efficient Exact Division

As described in [38] for example, the quotient of two polynomials can be computed more efficiently by observing that computing the quotient of two polynomials of degree \( d_1 \) and \( d_2 \) with \( d_1 > d_2 \) only depends on the \( d_1 - d_2 + 1 \) highest coefficients of the dividend and divisor. So all the coefficients of the polynomials do not need to be considered. This observation is a perfect example of why creating explicit formulas of field operations is more efficient than polynomial arithmetic, not all coefficients need to computed in every step.

For example, when computing the reduced \( u' \) output for Harley addition (Algorithm 3.1), the quotient of a monic degree three polynomial \( k = (f - v_2h - v_2^2)/u_2 \) and \( u_2 \) a monic degree two polynomial need to be computed. We only need the highest \( 3 - 2 + 1 = 2 \) coefficients of \( k \) to compute the quotient, so

\[ k = x^3 + (f_4 - u_{21})x^2 + cx + d \]

for some constants \( c, d \) that
are not required.

4.2.4 Using 'Almost Inverse'

At certain points in the Harley addition and doubling algorithms (Algorithms 3.1 and 3.2), the inverses of polynomials modulo other polynomials are taken. Let $A(x)$ and $B(x)$ be two polynomials. In [38], the authors note that finding $A(x)^{-1} \mod B(x)$ can be achieved by computing the resultant $r = \text{res}(A(x), B(x))$ through the determinant of a Sylvester matrix $M$. Once the determinant is found, Cramer’s rule is used to solve for the coefficients. For our formulae the division by $r$ is not computed leaving the result as $r/A(x)$. The extra copy of $r$ is removed later on through Montgomery’s inversion trick (Section 4.2.5). The other benefit of using this method is that if $r$ is non-zero then we know that there are no common factors between the two polynomials as described in [38].

We go on to describe the process: let $A(x) = a_1x + a_0$ and $B(x) = x^2 + b_1x + b_0$, where $\gcd(A(x), B(x)) = 1$. The usual way to compute the inverse of $A(x)$ modulo $B(x)$ is to use the extended Euclidean algorithm to solve

$$S(x)B(x) + T(x)A(x) = 1$$

for some polynomials $T(x) = t_1x + t_0, S(x) = s_0$, then $A^{-1}(x) = T(x) \mod B(x))$. Instead the resultant of $A(x)$ and $B(x)$ is computed. Let $M$ be the Sylvester matrix whose entries are coefficients of the two polynomials with dimensions $\deg A + \deg B \times \deg A + \deg B$. If
deg $A = n$ and deg $B = m$ then in general we have

$$M = \begin{pmatrix}
  b_n & a_m \\
  b_{n-1} & b_n & a_{m-1} & a_m \\
  : & : & \ddots & \ddots \\
  : & : & b_n & a_1 \\
  : & : & b_{n-1} & a_0 \\
  : & : & \vdots & a_0 & a_m \\
  b_0 & \vdots & \vdots & \ddots & \vdots \\
  b_0 & \vdots & \vdots & \ddots & \vdots \\
  \vdots & \vdots & \vdots & \ddots & \vdots \\
  b_0 & \vdots & \vdots & \ddots & a_0
\end{pmatrix}.$$ 

Creating the Sylvester matrix using $A$ and $B$ results in:

$$M = \begin{pmatrix}
  1 & a_1 & 0 \\
  b_1 & a_0 & a_1 \\
  b_0 & 0 & a_0
\end{pmatrix},$$

where $r = \det(M)$. Solving

$$\begin{pmatrix}
  1 & a_1 & 0 \\
  b_1 & a_0 & a_1 \\
  b_0 & 0 & a_0
\end{pmatrix} \cdot \begin{pmatrix}
  s_0 \\
  t_1 \\
  t_0
\end{pmatrix} = \begin{pmatrix}
  0 \\
  0 \\
  1
\end{pmatrix},$$

yields $t_1, t_0, s_0$ the Bezout coefficients one would acquire from using the extend Euclidean algorithm.

The most efficient way to find solutions to a system of equations for two by two systems is applying Cramer’s rule. The values of $t_1, t_0$ can be found as,

$$t_1 = \frac{\det M'}{\det M}, \text{ and } t_0 = \frac{\det M''}{\det M},$$
where

\[
M' = \begin{pmatrix} 1 & 0 & 0 \\ b_1 & 0 & a_1 \\ b_0 & 1 & a_0 \end{pmatrix}, \quad \text{and} \quad M'' = \begin{pmatrix} 1 & a_1 & 0 \\ b_1 & a_0 & 0 \\ b_0 & 0 & 1 \end{pmatrix}.
\]

Instead of taking the inverse of \(\det(M) = r\) and multiplying through to solve for \(t_1\) and \(t_0\), we let

\[
t'_1 = \det(M') \quad \text{and} \quad t'_0 = \det(M''),
\]

where \(t'_1 = rt_1\) and \(t'_0 = rt_0\), and so the almost inverse is \(T'(x) = r/A(x) \mod B(x)\). The only computations needed are finding the determinant of \(M\), since computing the determinant of \(M'\) and \(M''\) reuses computations.

In summary we find the determinant of \(M\) to compute \(r\), resulting in

\[
r = b_0a_1 + a_0(a_0 - b_1a_1).
\]

We then solve for

\[
t'_1 = t_1r = -a_1,
\]
\[
t'_0 = t_0r = a_0 - a_1b_1,
\]

the same way. So we have \(T' = t'_1x + t'_0 = rT\) the almost inverse of \(A(x) \mod B(x)\).

4.2.5 Montgomery’s Trick of Simultaneous Inversion

The idea of Montgomery is similar to that of Karatsuba; trading costly inversions for cheaper operations, e.g., multiplications. Throughout the explicit formulae, we come across places in the algorithm that need inversions. Instead of computing inverses as needed, we leave a 'copy' of the value to be inverted in the intermediate operands until we get to a point where all inverses can be found.

For example, in Harley addition and doubling (Algorithms 3.1 and 3.2) we compute an almost inverse (Section 4.2.4). In both algorithms the values \(r\) and \(s_1\) need to be inverted,
but $s_1$ is computed later on. So the almost inverse (Section 4.2.4) is computed, leaving a copy of $r$ in the numerator. Adjustments for the extra copy of $r$ are computed after the inversion. When the inversion does take place, the values that need to be inverted are multiplied together and the inverse of the product is taken resulting in,

$$w = \frac{1}{s_1 \cdot r}.$$ 

The individual inverses can be extracted via multiplications,

$$\frac{1}{r} = w \cdot s_1, \text{ and } \frac{1}{s_1} = w \cdot r.$$ 

The extra cost of a few multiplications is often more efficient than doing one extra inversion, so there are savings overall.

4.2.6 Reordering of the Normalization Step

At the end of composition in Harley addition and doubling (Algorithms 3.1 and 3.2) the $u''$ polynomial needs to be normalized making it monic. Reordering where the normalization happens i.e.; normalizing polynomials that $u''$ is composed of before computing $u''$ has been shown to save one multiplication in our setting [35]. Notice that the highest degree coefficient of $u'$ solely comes from the highest term coefficient of the intermediate polynomial $s$. By normalizing $s$, $u'$ is guaranteed to be monic. The obvious benefits are that working with monic polynomials uses fewer field operations, but the cost is adjusted for the extra 'copy' of $1/s_1$ in the $s$ polynomial, the inverse of the highest term coefficient of $s$.

4.2.7 Alternate Approach to Computing $s$ in Harley Composition

In the Harley implementation of doubling and adding (Algorithms 3.1 and 3.2), a resultant is computed and used to create an almost inverse, then $s$ is computed through Karatsuba multiplication. Instead the author of [2] sets up a system of equations for the coefficients of $s$ and then uses Cramer’s rule to solve for $s$. Using this method results in no savings over the
almost inverse technique. Note that this is not the same as the geometric approach, for the interpolating polynomial is not being solved for, only one of the sub-expressions in Harley’s method.

In doubling, to compute \( s = s_1 x + s_0 = (ax+b)/(cx+d) \mod x^2 + u_1 x + u_0 \), the polynomial representation

\[
a x + b = (c x + d)(s_1 x + s_0) - s_1 (c x^2 + c u_1 x + c u_0)
\]
is used. By equating coefficients the following linear equations arise:

\[
b = d s_0 - c u_0 s_1,
\]
\[
a = c s_0 + (d - c u_1) s_1.
\]

It takes two multiplications to compute the coefficients. Once the system is set up, it takes six multiplications using Cramer’s rule to solve the system and find the determinant. In Harley addition, the same technique is used but \( s = s_1 x + s_0 = (ax+b)/(x^2 + u'_1 x + u'_0) \mod x^2 + u_1 x + u_0 \) is computed instead. The polynomial representation is

\[
a x + b = s_1 x + s_0 (u' - u) - (u'_1 - u_1) u,
\]
and the linear equations are:

\[
b = (u'_1 - u_1)s_0 + (u'_0 - u_0 - u_1 (u'_1 - u_1)) s_1,
\]
\[
a = (u'_0 - u_0)s_0 + (u_0 (u_1 - u'_1)) s_1.
\]

The cost is two multiplications to compute the coefficients.

We note that in odd characteristic, an alternate technique to Cramer’s rule we call the “improved solution to a system of equations” can be applied saving one multiplication.

4.2.8 Improved Solution to System of Equations

Finding the solution to a linear two by two system of equations usually takes six multiplications using Cramer’s rule. In [21], the author introduces a trick to save one multiplication
at the cost of 13 additions, leaving an extra factor of 2 in the intermediate operands that is easily dealt with. We note that this trick cannot be applied in the even characteristic setting because there are factors of 2. This trick is applied in the geometric setting \[9\]; we introduce applying it with the use of computing $s$ with a linear two by two system of equations (Section 4.2.8) for a savings of one multiplication in odd characteristic.

Consider the following $2 \times 2$ system

$$\begin{pmatrix} a & b \\ c & e \end{pmatrix} \times \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} s \\ t \end{pmatrix},$$

that solves for $x$ and $y$. Note that if instead the system is set up as

$$\begin{pmatrix} m_4 & -m_2 \\ -m_3 & m_1 \end{pmatrix} \times \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} s \\ t \end{pmatrix},$$

and we can use the inverse of the matrix to obtain

$$\begin{pmatrix} m_1/d & m_2/d \\ m_3/d & m_4/d \end{pmatrix} \times \begin{pmatrix} s \\ t \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix},$$

where $m_1 = e$, $m_2 = -b$, $m_3 = -c$, $m_4 = a$ and $d = m_1m_4 - m_2m_3$. Then

$$x = s(m_1/d) + t(m_2/d)$$
$$y = s(m_3/d) + t(m_4/d).$$

Now let

$$k_1 = (m_2 - s)(t - m_1) = (tm_2 - m_2m_1 - st + sm_1),$$
$$k_2 = (-s - m_2)(t + m_1) = (-st - sm_1 - tm_2 - m_2m_1),$$
$$k_3 = (m_4 - s)(t - m_3) = (tm_4 - m_4m_3 - st + sm_3),$$
$$k_4 = (-s - m_4)(t + m_3) = (-st - sm_3 - tm_4 - m_4m_3).$$
Notice that,

\[ k_1 - k_2 = (tm_2 - m_2m_1 - st + sm_1) - (-st - sm_1 - tm_2 - m_2m_1) \]
\[ = 2(sm_1 + tm_2) \]
\[ = 2dx, \]

\[ k_3 - k_4 = (tm_4 - m_4m_3 - st + sm_3) - (-st - sm_3 - tm_4 - m_4m_3) \]
\[ = 2(sm_3 + tm_4) \]
\[ = 2dy. \]

Let \( x' = k_1 - k_2 = 2dx \), \( y' = k_3 - k_4 = 2dy \), and

\[ d' = k_3 + k_4 - k_1 - k_2 - 2(m_2 - m_4)(m_1 + m_3) \]
\[ = 2(m_4m_3 + m_2m_1) - 2(m_2m_1 + m_2m_3 - m_4m_1 - m_4m_3) \]
\[ = 2(m_4m_1 - m_2m_3) \]
\[ = 2d, \]

then \( x = x'/d' \) and \( y = y'/d' \).

4.2.9 Omitting \( l' \) and the Alternate Computation of \( v'' \)

Here we introduce a new technique that saves one multiplication in the Harley method. Let \( l' = s_1(l'') + v \) be the unreduced representation of the output \( v'' \). In the Harley method \( l'' = s''u \) is computed \[27\], where \( l''_1, l''_2 \) are used in the computation of the output \( u''_0 \) as well as in the computation of \( v'' \).

When using the system of equations method for computing \( s \) (Section 4.2.7), we can reuse computations from setting up the system of equations to simplify the equation for \( u''_0 \) and remove the need for the computation of \( l''_1 \) and \( l''_2 \). Now the only computation that relies on \( l'' \) is

\[ v'' = -h - l' = -h - s_1(l'') - v \mod u''. \]
From the Karastuba reduction equation in Section 4.2.2 we get,

\[ u''_1 = s_1(u''_1s''_0 + u''_1u_1 - u''_1s''_0u_1 + u''_0 - u_0) - v_1 - h_1 + h_2u''_1 \]
\[ = s_1(u''_1 - s''_0)(u_1 - u''_1) + u''_0 - u_0) - v_1 - h_1 + h_2u''_1 \]
\[ v''_0 = s_1(u''_0s''_0 + u''_1u_0 - u''_0u_1 - s''_0u_0) - v_0 - h_0 + h_2u''_0 \]
\[ = s_1(s''_0(u''_0 - u_0) + u''_0(u_1 - u''_1)) - v_0 - h_0 + h_2u''_0. \]

The resulting representation of the equation requires five multiplications to compute, and since we do not compute the two multiplications needed for \( l'' \) anymore, there is an overall reduction of one multiplication.

4.2.10 Equating Coefficients

One technique for computing polynomial operations that is sometimes advantageous over regular operations is equating coefficients between polynomial equalities. This technique can be used when computing \( u' \) and \( u'' \) in all doubling and addition algorithms respectively from the previous chapter.

For example consider \( u'' \) the output polynomial of both geometric and Harley addition algorithms. Mumford reduction (Section 2.6.2) describes the equality

\[ uu'u'' = \frac{l'^2 + hl' - f}{l'^2}. \]

Equating coefficients of \( x^5 \) and \( x^4 \) results in

\[ u'_1 + u_1 + u''_1 = \frac{2l'1l'3 + l'3h_2 - 1}{l'^2}, \]

and

\[ u'_0 + u_1u'_1 + u''_1(u'_1 + u_1) = \frac{2l'1l'3 + l'^2 + l'2h_2 + l'3h_1 - f_4}{l'^2}, \]

respectively. These expressions produce identical formulae to those in the Harley method without considering any sub-expressions. This technique is used in the geometric method [9], to compute \( u'' \) and \( u' \) in addition (Algorithm 3.3) and doubling (3.4).
4.2.11 Geometric Linear System Simplification

In Section [3.3.2] four by four linear systems of equations are set up to solve for \( l' \) the interpolating polynomial in doubling and addition. Simplifications to those systems can be obtained that result in smaller two by two linear systems that solve only for \( l'_2 \) and \( l'_3 \) [9]. Once these are solved, equation [3.3] can be used to with \( l'_2 \) and \( l'_3 \) to solve for \( l'_1 \) and \( l'_0 \). We first present simplifications for the addition system of equations, and then for doubling.

The Addition System

The original linear four by four system for computing the interpolating polynomial \( l' \) in addition is:

\[
\begin{pmatrix}
1 & 0 & -u_0 & u_1 u_0 \\
0 & 1 & -u_1 & u_1^2 - u_0 \\
1 & 0 & -u'_0 & u'_1 u'_0 \\
0 & 1 & -u'_1 & u'_1^2 - u'_0
\end{pmatrix}
\times
\begin{pmatrix}
l'_0 \\
l'_1 \\
l'_2 \\
l'_3
\end{pmatrix}
= \begin{pmatrix}
v_0 \\
v_1 \\
v'_0 \\
v'_1
\end{pmatrix}.
\]

(4.1)

Notice that the respective subtraction of row one and two from row three and four produces a two by two system that can be solved for \( l'_2 \) and \( l'_3 \) given as

\[
\begin{pmatrix}
u_0 - u'_0 & u'_1 u'_0 - u_1 u_0 \\
u_1 - u'_1 & (u'_1^2 - u'_0)(u_1^2 - u_0)
\end{pmatrix}
\times
\begin{pmatrix}
l'_2 \\
l'_3
\end{pmatrix}
= \begin{pmatrix}
v'_0 - v_0 \\
v'_1 - v_1
\end{pmatrix}.
\]

(4.2)

Equation [3.3] can be used to produce

\[
l'_1 = l'_2 u_1 - l'_3 (u'_1^2 - u_0) + v_1
\]

\[
l'_0 = l'_2 u_0 - l'_3 u_1 u_0 + v_0.
\]

The Doubling System of Equations

We refer to the original four by four system \( XY = Z \) (Equation [3.7]). The four by four system simplifies to a two by two system \( X'Y' = Z' \) by subtracting appropriate multiples of row one and two from row three. The rows of \( X' \) consist of

\[
(u_1 h_1 - h_2 (u_1^2 + u_0) + h_0 + 2u_1 v_1 + 2v_0, u_1^2 (2h_2 u_1 - 2h_1 - 4v_1) + u_0 (h_2 u_1 - h_1 - 2v_1)),
\]
\[(2v_1 - h_2u_1 + h_1, u_1(2h_2u_1 - 2h_1 - 4v_1) - h_2v_0 + h_0 + 2v_0),\]
the rows of \(Y'\) are \((l'_2), (l'_3)\), and the rows of \(Z'\) are
\[(2u_1(u_1^2 + u_0) - h_1v_1 - h_2v_0 - v_1^2 + f_2),\]
\[(3u_1^2 + f_3 - 2u_0 - h_2v_1).\]
After solving the above system for \(l'_2\) and \(l'_3\), the process of obtaining \(D'' = [2]D = [u'', v'']\) is identical to the case of addition.

### 4.3 Projective Coordinates

The projective setting is an augmentation of the regular setting that introduces one or more auxiliary elements or “coordinates” that are used to trade inversions for other field operations. This setting is beneficial in cases where the computation of an inversion requires a large number of field operations.

One inversion is required in the process of computing divisor class addition and doubling. Instead of following this path, an extra coordinate \(z\) is passed along with the affine input where the quintuple \([u_1, u_0, v_1, v_0, z]\) stands for \([u_1/z, u_0/z, v_1/z, v_0/z]\) in the projective setting. The extra coordinate keeps track of values that would have been inverted and multiplied through in the affine setting. Instead of taking an inverse in every composition step, the value to be inverted, denoted by \(z\), is computed and then inverted and multiplied through at the end of the entire scalar multiplication process.

Throughout our exposition of projective divisor class composition we refer to the ‘weight’ of a field element to mean the value that should be inverted and multiplied through to produce the true affine value of that element. For example consider \(s' = d's = s'_1x + s'_0\) produced in Step 3 of affine doubling in algorithm 5.2. Both \(s'_1\) and \(s'_0\) have a weight of \(d'\) with respect to the affine value of \(s_1\) and \(s_0\). Another example is \(s'' = s/s_1 = x + s''_0\) produced in Step 4 of affine doubling in algorithm 5.2. The weight of \(s''_0\) is \(1/s_1\) with respect to \(s_0\). Inverting \(1/s_1 = s_1\) and multiplying with \(s''_0\) produces the true affine value of \(s_0\).
4.3.1 Weight Complexity Reduction

In [2], the author uses a specialized version of the alternate system of equations computation of \( s \) (Section 4.2.7) to reduce the complexity of the weights of intermediate affine values relative to previous projective implementations in [27]. This technique is also used in affine one inversion divisor class tripling when pushing back the first inversion.

Instead of using a system of equations to solve for \( s \) given by the equations

\[
(x^2 + u_1 x + u_0)(s_1 x + s_0) = \tilde{v}_1 x + \tilde{v}_0 \mod x^2 + u_1' x + u_0'
\]

and

\[
k_1 x + k_0 = (s_1 x + s_0)(\tilde{v}_1 x + \tilde{v}_0) \mod x^2 + u_1 x + u_0
\]

for addition and doubling respectively (Section 4.2.7), the author solves for \( s_0'' = s_0/s_1 \) and \( c = 1/s_1 \) using

\[
(x^2 + u_1 x + u_0)(x + s_0'') = c(\tilde{v}_1 x + \tilde{v}_0) \mod x^2 + u_1' x + u_0'
\]

and

\[
c(k_1 x + k_0) = (x + s_0'')(\tilde{v}_1 x + \tilde{v}_0) \mod x^2 + u_1 x + u_0,
\]

resulting in the weights of computed \( u' \) for doubling and \( u'' \) for addition to include \( 1/s_1 \) instead of \( s_1 \). The resulting weight of \( u' \) and \( u'' \) does not include \( s_1 \), so the number of field operations needed to compute \( v' \) in doubling and \( v'' \) in addition, as well as the output weight \( z' \) and \( z'' \) is reduced.

4.3.2 Computation of \( u'' \) modulo \( u \)

Balamohan introduced a technique that computes \( u'' \), the output polynomial representing part of the output divisor \( D'' = [u'', v''] \), when tripling in the odd characteristic projective setting [2]. The polynomial \( u'' \) is computed modulo the input polynomial \( u \) producing an equality where coefficients are equated to create an explicit formula for \( u'' \). In this work we extend the technique to arbitrary fields. We describe how the equality is obtained.
When computing a double-then-add to produce a tripling, the Harley algorithms for addition and doubling (Algorithms 3.1 and 3.2) give rise to the equalities:

\[ s_1 uu' = k - s (su + \tilde{v}) \quad (4.3) \]

\[ \tilde{s}_1 u' u'' = k - \tilde{s} (\tilde{s}u' + \tilde{v}) \quad (4.4) \]

where,

- \( D = [u, v] \) is the divisor to be tripled.
- \( D' = [u', v'] = 2D. \)
- \( D'' = [u'', v''] = 3D. \)
- \( \hat{v} = 2v + h. \)
- \( k = \frac{f - hv - v^2}{u}. \)
- \( s = \frac{k}{v} \mod u. \)
- \( \tilde{s} = \frac{v' - v}{u} \mod u'. \)

Subtracting equation 4.3 from equation 4.4 results in

\[ \tilde{s}_1^2 u' u'' - s_1^2 uu' = s (su + \hat{v}) - \tilde{s} (\tilde{s}u' + \tilde{v}). \quad (4.5) \]

Notice that since \( \gcd(u, u') = 1 \) as required by the frequent case, (4.5) can be taken modulo \( u \) resulting in

\[ \tilde{s}_1^2 u'' = \frac{\hat{v}(s - \tilde{s})}{u'} \mod u. \quad (4.6) \]

Let \( \bar{s} = s - \tilde{s} \) and consider the equality (4.6) after applying a modular reduction by \( u \) on both sides,

\[ \bar{s}_1^2 (u'_1 - u_1)x + \bar{s}_1^2 (u''_0 - u_0) = (\hat{v}_1 \bar{s}_0 + \hat{v}_0 \bar{s}_1 - u'_1 \hat{v}_1 \bar{s}_1)x + (\bar{s}_0 \hat{v}_0 - u'_0 \hat{v}_1 \bar{s}_1). \quad (4.7) \]
Equating coefficients of both sides of the equality results in the explicit formulae:

\[ \tilde{s}^2_1 u_1'' = \hat{v}_1 \tilde{s}_0 + \hat{v}_0 \tilde{s}_1 - u'_1 \hat{v}_1 \tilde{s}_1 + s^2_1 u_1; \]
\[ \tilde{s}^2_1 u_0'' = \hat{v}_0 \tilde{s}_0 - u'_0 \hat{v}_1 \tilde{s}_1 + s^2_1 u_0. \]

Notice that the extra copies of \( \tilde{s}^2_1 \) are not computed and considered part of the weight resulting in the explicit formula:

\[ u_1'' = \hat{v}_1 \tilde{s}_0 + \hat{v}_0 \tilde{s}_1 - u'_1 \hat{v}_1 \tilde{s}_1 + s^2_1 u_1; \]
\[ u_0'' = \hat{v}_0 \tilde{s}_0 - u'_0 \hat{v}_1 \tilde{s}_1 + s^2_1 u_0, \]

where the \( u_i'' \) have a weight of \( \tilde{s}^2_1 \).

In our tripling formulae, only \( u_0'' \) is computed using the above technique. The value \( u'_1'' \) is computed more efficiently by using the regular equating of coefficients technique described in projective addition (Step 5 of Section 5.2). The use of this technique to compute \( u'_0'' \) is not more efficient in the affine setting either, requiring \( 3M \). However, in the projective setting the weight computations simplify resulting in savings. Part of the savings come from noticing the value \( u_0 \) is adjusted to have the same weight as \( u'_0'' \) in the computation of \( u'_0'' \). The adjustment of \( u_0 \) is required for the computation of \( v'' \), resulting in a savings of \( 1M \).
Chapter 5

Divisor Doubling and Addition
In this chapter we describe our explicit formulae for adding and doubling in the affine and projective settings based on the generic algorithms presented in Chapter 3 and the techniques explored for creating explicit formulae in the previous chapter. We have produced explicit formulae that require the least number of field operations to date for both addition and doubling in projective and affine settings through combining the best techniques from the geometric and Harley methods for computing divisor class composition. All of our improvements also can be adapted to settings that use extra elements to describe divisor classes, producing the fastest formulae overall.

We give an exposition of each formula, then summarize the formula in a table and give a cost analysis. We present the formulas for arbitrary Weierstrass equations of the curve. We also distinguish between the even and odd cases for the characteristic of the base field the curve is defined over. The number of operations and the formulae themselves are not always the same in the different characteristic cases. The resulting formulae are presented in a table at the end of each case with counts for the number of field operations in even and odd characteristic.

The number of field multiplications and squares required to compute an explicit formula is the metric that we use for assessing efficiency. We do not take additions into consideration because they are comparably cheap. The cost of field multiplications, squares and inversions is variable on different implementations, so numerical cost analysis of explicit formulae is implementation dependent. Formula cost analysis is also characteristic dependent because in even characteristic squares and inversions are usually much cheaper to execute than in odd characteristic. With all of these factors aside, we can say that a reduction in field operations of an explicit formula directly corresponds to a reduction in the computational cost of the formula. When giving counts for the explicit formulae we denote a field inversion with $I$, a field multiplication with $M$, a curve coefficient multiplication with a $C$, and a field squaring with $S$. When implementing an explicit formula, curve coefficients are known beforehand
when the curve is chosen, so implementations can be optimized for any coefficients of \( f(x) \) and \( h(x) \) and these are considered to have no operation cost and are given with a separate count. Recall that in even characteristic \( f_4 = f_3 = f_2 = 0 \) and \( h_2 = 1 \). We keep track of the value \( h_2 \) in our formulae in case curve specializations where \( h_2 = 0 \) are considered, but in general we do not count multiplications by \( h_2 \) in our totals. In odd characteristic \( h(x) = 0 \) and \( f_4 = 0 \) as described in Section 4.1.2.

We first present our affine formulae by giving an overview of the algorithms and then presenting explicit formulae for addition and doubling. Afterwards we give an overview of the projective setting and present explicit formulae for addition, doubling and mixed addition. Finally we discuss other settings that use 6 and 8 elements to describe divisor classes in the affine and projective settings respectively.

5.1 Affine Formulae

In this section we give an exposition of our explicit formulae for the addition and doubling of divisor classes. By exploring all techniques used in literature for converting algebraic and geometric algorithms to explicit formulae and combining these techniques in a novel way, we found formulae that require fewer field operations in odd characteristic for adding and doubling in the affine setting. The key difference in our formulae is the use of a linear system of equations to solve for the sub-expression \( s \) instead of computing a resultant and then an almost inverse (Section 4.2.7).

The idea comes from the use of a system of equations to solve for the interpolating polynomial \( l' \) in the geometric method as given in Section 3.3 and has been used in [2]. In the even characteristic setting this technique requires the same number of field operations as the fastest known formulae [12], but in odd characteristic we introduce the use of the improved solution to a system of equations technique (Section 4.2.8) to trade one multiplication for thirteen additions as used in the geometric method [9]. An extra factor of two is produced.
in the denominator when solving the system, so we can not adapt this technique to the even characteristic case and instead we use Cramer’s rule. In the even characteristic setting, we save a squaring in doubling, and trade a multiplication for a square in addition.

Another benefit we found to this technique is that one of the pre-computations involved for setting up the system of equations is reused in the computation of $u''$. This allows us to trade a multiplication for a square in even characteristic as well as make the computation of $u''$ not need any values from the computation of $l$. We also found a saving of one multiplication when computing $v''$ by omitting the computation of $l$ in all cases as described in Section 4.2.9.

We start with an exposition of our fastest affine addition formula.

### 5.1.1 Addition

Our formula is an adaptation of the Harley method for divisor class addition (Algorithm 3.1). We compute the first sub-expression $s = (v - v')/u'$ by solving a system of equations for $s' = sd$, where $d$ is the determinant of the system as described in Section 4.2.7. Afterwards we compute monic $s'' = s/s_1 = x + s''_0$ using Montgomery’s simultaneous inversion trick (Section 4.2.5) to ensure the output $u''$ is monic since the highest degree coefficient of $u''$ is exactly $s_1^2$ as described in Section 4.2.6. Next we skip computation of $l'' = s''u$ because it is unneeded, the coefficients of $l''$ are broken down into coefficients of $s''$ and $u$ and reorganized to use the least number of operations in the computation of $u''$ (Section 4.2.9). Next we compute the output polynomial $u''$ using the coefficient equating technique (Section 4.2.10) to solve for $u''$ and then Karatsuba reduction (Section 4.2.2) to compute $v''$. A summary of this method is given in Algorithm 5.1, followed by a detailed description of how each step is converted to explicit formulae.
Algorithm 5.1 Summary of divisor class addition

**Input:** $D = [u_1, u_0, v_1, v_0], D' = [u'_1, u'_0, v'_1, v'_0]$ where,

- $u = x^2 + u_1 x + u_0$
- $v = v_1 x + v_0$
- $u' = x^2 + u'_1 x + u'_0$
- $v' = v'_1 x + v'_0$
- $f = x^5 + f_4 x^4 + f_3 x^3 + f_2 x^2 + f_1 x + f_0$
- $h = h x^2 + h_1 x + h_0$ (where $h = 1$)

**Output:** $D'' = [u'', u'_0, v'_1, v'_0] = D + D'$ where,

- $u'' = x^2 + u''_1 x + u''_0$
- $v'' = v''_1 x + v''_0$

1: Setup system of equations that solves for $s \equiv (v' - v)/u \mod u'$
2: Solve system of equations for $s'$ up to a factor of $d'$
3: Compute $s'' = s/s_1 = x + s''_0$ and $1/s_1, 1/s'_1, s_1$
4: Compute $u''$ by equating $x^4$ and $x^5$ coefficients of $uu'u'' = \frac{v^2 + h'' - l}{l_1^2}$ where $l' = (s_1)s''u + v$
5: Compute $v'' \equiv -h - l' \mod u''$

**Step 1:** Set up system of equations to solve for $s \equiv (v' - v)/u \mod u'$

From Step 2 of Harley Addition, (Algorithm 3.1),

$$s_1 x + s_0 = \frac{\tilde{v}_1 x + \tilde{v}_0}{x^2 + u_1 x + u_0} \mod x^2 + u'_1 x + u'_0,$$

where $\tilde{v} = \tilde{v}_1 x + \tilde{v}_0 = (v'_1 - v_1) x + (v'_0 - v_0)$. By multiplying $x^2 + u_1 x + u_0$ to both sides we get

$$(x^2 + u_1 x + u_0)(s_1 x + s_0) = \tilde{v}_1 x + \tilde{v}_0 \mod x^2 + u'_1 x + u'_0.$$

Equating coefficients results in the equations

$$\tilde{v}_0 = s_0(u_0 - u'_0) + s_1(u'_0(u'_1 - u_1)),
\tilde{v}_1 = s_0(-u'_1 - u_1) + s_1(u_0 - u'_0 + u'_1(u'_1 - u_1)),$$

which are linear in the coefficients of $s$. We can find $s = s_1 x + s_0$ by solving

$$M' \equiv \left( \begin{array}{cc} u_0 - u'_0 & u'_0(u'_1 - u_1) \\ -(u'_1 - u_1) & u_0 - u'_0 + u'_1(u'_1 - u_1) \end{array} \right) \times \left( \begin{array}{c} s_0 \\ s_1 \end{array} \right) = \left( \begin{array}{c} \tilde{v}_0 \\ \tilde{v}_1 \end{array} \right).$$
Manipulating this system using Cramer’s rule to create a new system that solves for \( s \), as done in Section 4.2.8, results in

\[
M = \begin{pmatrix}
\frac{u_0 - u'_0 + u'_1(u'_1 - u_1)}{\det M'} & -\frac{u'_0(u'_1 - u_1)}{\det M'} \\
\frac{(u'_1 - u_1)}{\det M'} & \frac{u_0 - u'_0}{\det M'}
\end{pmatrix} \times \begin{pmatrix}
\tilde{v}_0 \\
\tilde{v}_1
\end{pmatrix} = \begin{pmatrix}
s_0 \\
s_1
\end{pmatrix}.
\]

The explicit formula for this step is:

\[
m_3 = u'_1 - u_1; \quad m_4 = u_0 - u'_0; \quad m_1 = m_4 + u'_1m_3; \quad m_2 = -u'_0m_3;
\]

\[
\tilde{v}_0 = v'_0 - v_0; \quad \tilde{v}_1 = v'_1 - v_1;
\]

The cost of this step is \( 2M, 0S, 0C \) in both characteristics.

**Step 2: Solve system for** \( s' = sd' = s'_1x + s'_0 = d'(v' - v)/u \mod u' \)

In the odd characteristic setting, this step is where we see a novel reduction in the number of multiplications. We solve the system from step 1 by using either Cramer’s rule in even characteristic or the improved solution to a system of equations in odd (Section 4.2.8), resulting in \( s' = d's = s'_1x + s'_0 \) where \( d' = 2d \) in odd characteristic and \( d' = d \) in even. We note that \( d = \det M' \).

In general and in even characteristic, the explicit formula is:

\[
s'_0 = \tilde{v}_0m_1 + \tilde{v}_1m_2; \quad s'_1 = \tilde{v}_0m_3 + \tilde{v}_1m_4;
\]

\[
d' = m_4m_1 - m_2m_3.
\]

Specializing to odd characteristic the explicit formula is:

\[
w_0 = (m_2 - \tilde{v}_0)(\tilde{v}_1 - m_1); \quad w_1 = (-\tilde{v}_0 - m_2)(\tilde{v}_1 + m_1);
\]

\[
w_2 = (m_4 - \tilde{v}_0)(\tilde{v}_1 - m_3); \quad w_3 = (-\tilde{v}_0 - m_4)(\tilde{v}_1 + m_3);
\]

\[
s'_0 = w_0 - w_1; \quad s'_1 = w_2 - w_3;
\]

\[
d' = w_2 + w_3 - w_0 - w_1 - 2(m_2 - m_4)(m_1 + m_3).
\]

In cost of this step is \( 5M, 0S, 0C \) in odd characteristic and \( 6M, 0S, 0C \) in even characteristic.
Step 3: Compute $s'' = s/s_1 = x + s''_0$

Note that $s' = d's$ and $d'$ from the previous steps. Here Montgomery’s inversion trick (Section 4.2.5) is used to invert $d'$ and make $s_1$ monic at the same time. The explicit formula is:

\[
\begin{align*}
    w_1 &= 1/(d's_1) = (1/d'^2 s_1); \\
    w_2 &= w_1d' = (1/d's_1); \\
    w_3 &= w_2d' = (1/s_1); \\
    w_4 &= w_3^2 = (1/s_1^2); \\
    s_1 &= s'^2 w_1 = (s_1);
\end{align*}
\]

The polynomial $s'' = s/s_1 = x + s''_0$ and value $s_1$ were computed. The value $s_1$ is computed because it is needed to normalize parts of $v''$ in Step 5. The values $w_3$ and $w_4$ are multiplied with values in Step 4 because of the extra copies of $1/s_1$ that arise from using $s''$ instead of $s$. The cost of this step is $1I, 5M, 2S, 0C$ in both characteristics.

Step 4: Compute $u''$

The technique of equating coefficients to compute $u''$ as described in Section 4.2.10 is used here. Notice

\[
(x^2 + u''_1 x + u''_0) = \frac{(l'_3 x^3 + l''_2 x^2 + l'_1 x + l'_0)^2 + (h_2 x^2 + h_1 x + h_0)(l'_3 x^3 + l''_2 x^2 + l'_1 x + l'_0) - f(x)}{l'_3^2 (x^2 + u_1 x + u_0)(x^2 + u'_1 x + u'_0)}
\]

from Cantor’s reduction step and that $u''$ needs to be made monic. The above equation produces the equality

\[
(x^2 + u_1 x + u_0)(x^2 + u'_1 x + u'_0)(x^2 + u''_1 x + u''_0) = \frac{l'' + l'h - f}{l'_3^2}.
\]

Equating the coefficients of $x^5$ and $x^4$ results in

\[
u'_1 + u_1 + u''_1 = \frac{2l''_3 l''_2 + l''_3 h_2 - 1}{l'_3^2},
\]
and
\[ u''_0 + u_0 + u'_0 + u_1u'_1 + u''_1(u'_1 + u_1) = \frac{2l'_1l'_3 + l''_2 + h_2l'_2 + h_1l'_3 - f_4}{l'_3}, \]
respectively.

To compute \( u''_1 \) and \( u''_0 \), \( l' \) is not computed directly in our explicit formulae but is broken up into the additive parts of its coefficients, including the \( v \) portion to group values together in terms of the adjustments by \( w_3 = 1/s_1 \) and \( w_4 = 1/s_2 \) they need as described in Section 4.2.9. The polynomial \( s'' = s/s_1 \) was computed instead of \( s \), so \( l'' = s''u \) should be computed resulting in

\[ l'' = su(1/s_1) = s''u = (x + s''_0)(x^2 + u_1x + u_0) = x^3 + (s''_0 + u_1)x^2 + (s''_0u_1 + u_0)x + s''_0u_0 \]

where the coefficients have an extra copy of \( 1/s_1 \) in them relative to \( l' \), resulting in

\[ l' = s_1l'' + v = s_1x^3 + (s_1(s'' + u_1))x^2 + (s_1(s''_0u_1 + u_0) + v_1)x + (s_1(s''_0u_0) + v_0). \]

We account for the extra \( 1/l'_3^2 = 1/s_1^2 \) on the right hand side by adjusting with \( w_3 = 1/s_1 \) and \( w_4 = 1/s_2 \) accordingly, resulting in:

\[ u''_1 = -u' - u_1 + 2(s''_0 + u_1) + h_2w_3 - w_4; \]

\[ u''_0 = -u_0 - u'_0 - u_1u'_1 - u''_1(u'_1 + u_1) + 2(s''_0u_1 + u_0 + w_3v_1) + (s''_0 + u_1)^2 + w_3(h_2(s'' + u_1) + h_1) - w_4f_4. \]

Expanding, simplifying and recalling that \( m_3 = u'_1 - u_1 \) and \( m_1 = u'_1(u'_1 - u_1) + u_0 - u'_0 \), results in the general explicit formula:

\[ u''_1 = 2s''_0 - m_3 + h_2w_3 - w_4; \]

\[ u''_0 = s''_0(s''_0 - 2m_3) + m_1 + w_3(h_2(s''_0 - u'_1) + h_1 + 2v_1) + w_4(u_1 + u'_1 - f_4). \]
The odd characteristic formula is:

\[ u''_1 = 2s''_0 - m_3 - w_4; \]
\[ u''_0 = s''_0(s''_0 - 2m_3) + m_1 + 2w_3v_1 + w_4(u_1 + u'_1). \]

The even characteristic formula is:

\[ u''_1 = m_3 + h_2w_3 + w_4; \]
\[ u''_0 = s''_0 + m_1 + w_3(h_2(s''_0 - u'_1) + h_1) + w_4(u_1 + u'_1). \]

Notice that novel to this thesis we were able to trade one multiplication for one square in the even characteristic because of the reuse of \( m_3 \). The cost of this step is \( 3M, 0S, 0C \) in odd characteristic and \( 2M, 1S, 0C \) in even characteristic.

**Step 5: Compute** \( v'' = -h - l' \mod u'' \)

Karatsuba reduction (Section 4.2.2) is used to compute \( v'' \), reducing a degree three polynomial modulo a degree two polynomial. Recall that \( l'_1 = s_1(u_1s''_0+u_0) + v_1 \) and \( l'_0 = s_1(u_0s''_0) + v_0 \) in Step 4. The \( l'_i \) are not used directly but the \( s''_0 \) and \( u_i \) that make up the \( l'_i \) are used instead, resulting a reduction of the number of multiplications (Section 4.2.9). The general and even characteristic explicit formula is:

\[ w_0 = (u''_1 - s''_0)(u_1 - u''_1); \]
\[ v''_1 = s_1(w_0 + u''_0 - u_0) - v_1 - h_1 + h_2u''_1; \]
\[ v''_0 = s_1(s''_0(u''_0 - u_0) + u''_0(u_1 - u''_1)) - v_0 - h_0 + h_2u''_0. \]

Specializing into odd characteristic, the explicit formula is:

\[ w_0 = (u''_1 - s''_0)(u_1 - u''_1); \]
\[ v''_1 = s_1(w_0 + u''_0 - u_0) - v_1; \]
\[ v''_0 = s_1(s''_0(u''_0 - u_0) + u''_0(u_1 - u''_1)) - v_0. \]
Table 5.1: Arbitrary Affine Addition Formula

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Setup system of equations for ( s \equiv (v' - v)/u' ) mod ( u ):</td>
<td>2M</td>
</tr>
<tr>
<td></td>
<td>( m_3 = u'_1 - u_1; m_4 = u_0 - u'_0; m_1 = m_4 + u'_1 m_3 );</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( m_2 = -u'_0 m_3; \tilde{v}_0 = v'_0 - v_0; \tilde{v}_1 = v'_1 - v_1 );</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Solve system of equations for ( s' = s'd' = s'_1 x + s'_0 ):</td>
<td>6M</td>
</tr>
<tr>
<td></td>
<td>( s'_0 = \tilde{v}_0 m_1 + \tilde{v}_1 m_2; s'_1 = v_0 m_3 + \tilde{v}_1 m_4; d' = m_4 m_1 - m_2 m_3 );</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(if ( d' = 0 ) branch to Cantor’s Algorithm)</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Compute ( s'' = x + s'_0/s'_1 ) and ( s_1 ):</td>
<td>1I, 5M, 2S</td>
</tr>
<tr>
<td></td>
<td>( w_1 = (d's'_1)^{-1}; w_2 = w_1 d'; w_3 = w_2 d' = (1/s_1); )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( w_4 = w_2^2; s_1 = s'_1 w_1 = (s_1); s''_0 = s'_0 w_2; )</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Compute ( u'' ):</td>
<td>3M</td>
</tr>
<tr>
<td></td>
<td>( u'' = 2s''_0 - m_3 + h_2 w_3 - w_4 );</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( u''_0 = s''_0 (s''_0 - 2m_3) + m_1 + w_3 (h_2 s''_0 - u'_1) + )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( + h_1 + 2v_1) + w_4 (u_1 + u'_1 - f_4); )</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Compute ( v'' = -h - l' ) mod ( u'' ):</td>
<td>5M</td>
</tr>
<tr>
<td></td>
<td>( w_0 = (u''_0 - s''_0) (u_1 - u''_0); )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( v''_0 = s_1 (w_0 + u''_0 - u_0) - v_1 - h_1 + h_2 u''_1; )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( v''_0 = s_1 (s''_0 (u_0 - u_0) + u''_0 (u_1 - u''_0)) - v_0 - h_0 + h_2 u''_0; )</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>I, 21M, 2S</td>
</tr>
</tbody>
</table>

The cost of this step is \( 5M, 0S, 0C \) in both characteristics.

The total cost of the explicit formula is \( 1I, 20M, 2S, 0C \) in odd characteristic, and \( 1I, 20M, 3S, 0C \) in even characteristic. Table 5.1 summarizes the formulae in arbitrary case. Tables 5.2 and 5.3 summarize the formulae in even and odd characteristic cases respectively.
**Table 5.2: Even Characteristic Affine Addition Formula**

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>IN:</strong></td>
<td>Reduced divisors $D = [u, v]$ and $D' = [u', v']$ with $u = x^2 + u_1x + u_0$, $v = v_1x + v_0$, $u' = x^2 + u'_1x + u'_0$, $v' = v'_1x + v'_0$, $h = h_2x^2 + h_1x + h_0$, $f = x^5 + f_4x^4 + f_3x^3 + f_2x^2 + f_1x + f_0$</td>
<td></td>
</tr>
<tr>
<td><strong>OUT:</strong></td>
<td>Reduced divisor $D'' = [u'', v''] = D + D'$ with $u'' = x^2 + u''_1x + u''_0$, $v'' = v''_1x + v''_0$.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><strong>Setup system of equations for</strong> $s \equiv (v' - v)/u'$ mod $u$:</td>
<td>2M</td>
</tr>
<tr>
<td></td>
<td>$m_3 = u'_1 - u_1$; $m_4 = u_0 - u'_0$; $m_1 = m_4 + u'_1m_3$;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$m_2 = -u'_0m_3$; $\tilde{v}_0 = v'_0 - v_0$; $\tilde{v}_1 = v'_1 - v_1$;</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td><strong>Solve system of equations for</strong> $s' = sd' = s'_1x + s'_0$:</td>
<td>6M</td>
</tr>
<tr>
<td></td>
<td>$s'_0 = \tilde{v}_0m_1 + \tilde{v}_1m_2$; $s'_1 = \tilde{v}_0m_3 + \tilde{v}_1m_4$; $d' = m_4m_1 - m_2m_3$;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(if $d' = 0$ branch to Cantor's Algorithm)</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td><strong>Compute</strong> $s'' = x + s'_0/s'_1$ and $s_1$:</td>
<td>1I, 5M, 2S</td>
</tr>
<tr>
<td></td>
<td>$w_1 = (d's'_1)^{-1}$; $w_2 = w_1d'; w_3 = w_2d' = (1/s_1)$;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$w_4 = w_2^2$; $s_1 = s_1^2w_1 = (s_1)$; $s'_0 = s'_0w_2$;</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td><strong>Compute</strong> $u''$:</td>
<td>2M, 1S</td>
</tr>
<tr>
<td></td>
<td>$u''_1 = h_2w_3 - w_4 - m_3$;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$u''_0 = s''_0^2 + m_1 + w_3(h_2(s''_0 - u''_1) + h_1) +$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$+w_4(u_1 + u'_1)$;</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td><strong>Compute</strong> $v'' = -h - l'$ mod $u''$:</td>
<td>5M</td>
</tr>
<tr>
<td></td>
<td>$w_0 = (u''_1 - s''_0)(u_1 - u''_1)$;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$v''_1 = s_1(w_0 + u''_0 - u_0) - v_1 - h_1 + h_2u''_1$;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$v''_0 = s_1(s''_0(u''_0 - u_0) + u''_0(u_1 - u''_1)) - v_0 - h_0 + h_2u''_0$;</td>
<td></td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>I, 20M, 3S</td>
<td></td>
</tr>
</tbody>
</table>
Reduced divisors $D = [u, v]$ and $D' = [u', v']$ with

\[
\begin{align*}
u &= x^2 + u_1 x + u_0, \quad v = v_1 x + v_0 \\
u' &= x^2 + u'_1 x + u'_0, \quad v' = v'_1 x + v'_0 \\
h &= h_2 x^2 + h_1 x + h_0, \\
f &= x^5 + f_4 x^4 + f_3 x^3 + f_2 x^2 + f_1 x + f_0
\end{align*}
\]

**Procedure**

**SETUP** system of equations for $\text{Reduced divisor } D'' = [u'', v''] = D + D'$ with

\[
\begin{align*}
\text{IN: } & u'' = x^2 + u''_1 x + u''_0, \quad v'' = v''_1 x + v''_0 \\
\text{OUT: } & \text{Reduced divisor } D'' = [u'', v''] = D + D'
\end{align*}
\]

\[
\begin{array}{|c|c|}
\hline
\text{Step} & \text{Procedure} \\
\hline
1 & \text{Setup system of equations for } s \equiv (v' - v)/u' \mod u: \\
& m_3 = u'_1 - u_1; \quad m_4 = u_0 - u'_0; \quad m_1 = m_4 + u'_1 m_3; \\
& m_2 = -u'_0 m_3; \quad \tilde{v}_0 = v'_0 - v_0; \quad \tilde{v}_1 = v'_1 - v_1; \\
\hline
2 & \text{Solve system of equations for } s' = sd = s'_1 x + s'_0: \\
& w_0 = (m_2 - \tilde{v}_0)(\tilde{v}_1 - m_1); \quad w_1 = (-\tilde{v}_0 - m_2)(\tilde{v}_1 + m_1); \\
& w_2 = (m_4 - \tilde{v}_0)(\tilde{v}_1 - m_3); \quad w_3 = (-\tilde{v}_0 - m_4)(\tilde{v}_1 + m_3); \\
& s'_0 = w_0 - w_1; \quad s'_1 = w_2 - w_3; \\
& d' = w_2 + w_3 - w_0 - w_1 - 2(m_2 - m_4)(m_1 + m_3); \\
& \text{(if } d' = 0 \text{ branch to Cantor's Algorithm)} \\
\hline
3 & \text{Compute } s'' = x + s''_0/s'_1 \text{ and } s_1: \\
& w_1 = (d's'_1)^{-1}; \quad w_2 = w_1 d'; \quad w_3 = w_2 d' = (1/s_1); \\
& w_4 = w_3^2; \quad s_1 = s''_0 w_1 = (s_1); \quad s''_0 = s'_0 w_2; \\
\hline
5 & \text{Compute } u'': \\
& u''_0 = 2s''_0 - m_3 - w_4; \\
& u''_0 = s''_0(s''_0 - 2m_3) + m_1 + 2w_3 v_1 + w_4(u_1 + u'_1); \\
\hline
6 & \text{Compute } v'': \\
& w_0 = (u''_1 - s''_0)(u_1 - u''_1); \\
& v''_1 = s_1(w_0 + u''_0 - u_0) - v_1; \\
& v''_0 = s_1(s''_0(u''_0 - u_0) + u''_0(u_1 - u''_1)) - v_0; \\
\hline
\end{array}
\]

**Total**

\[
\begin{align*}
\text{I, 20M, 2S}
\end{align*}
\]
5.1.2 Doubling

Our formula is an adaptation of the Harley method for divisor class doubling (Algorithm 3.2). The first difference between addition and doubling is how \( s' = sd' \) is computed, introducing two more steps to create the linear equations used to set up the system of equations for \( s \). The explicit formula also differs slightly in the computation of \( u' \) step. The first sub-expression \( k \equiv (f - hv - v^2)/u \mod u \) is computed using exact division (Section 4.2.3) and Karatsuba reduction (Section 4.2.2). The sub-expression \( s = k/h + 2v \mod u \) is computed by solving a system of equations for \( s' = sd' \), where \( d' = d \) in even characteristic and \( d' = 2d \) and \( d \) is the determinant of the system as described in (Section 4.2.7). All the rest of the steps follow closely from addition. A summary of this method is given in Algorithm 5.2 followed by a detailed description of how each step is converted to explicit formulae.

Algorithm 5.2 Summary of divisor class doubling

**Input:** \( D = [u_1, u_0, v_1, v_0] \) where,

\[
    u = x^2 + u_1 x + u_0 \\
    v = v_1 x + v_0 \\
    f = x^5 + f_4 x^4 + f_3 x^3 + f_2 x^2 + f_1 x + f_0 \\
    h = h_2 x^2 + h_1 x + h_0 \quad (h_2 = 1)
\]

**Output:** \( D' = [u'_1, u'_0, v'_1, v'_0] = 2D \) where,

\[
    u' = x^2 + u'_1 x + u'_0 \\
    v' = v'_1 x + v'_0
\]

1: Compute \( \tilde{v} = h + 2v \mod u \)

2: Compute \( k \equiv (f - hv - v^2)/u \mod u \)

3: Setup system of equations that solves for \( s \equiv k/\tilde{v} \mod u \)

4: Solve system of equations for \( s' \) up to a factor of \( d' \)

5: Compute \( s'' = s/s_1 = x + s''_0 \) and \( 1/s_1, 1/s'_1, s_1 \)

6: Compute \( u'' \) by equating \( x^4 \) and \( x^5 \) coefficients of \( uu'u'' = \frac{v^2 + hv' - f}{l_3^2} \) where \( l' = (s_1)s''u + v \)

7: Compute \( v' \equiv -h - l' \mod u' \)
Step 1: Compute $\tilde{v} = 2v + h \mod u$

First the polynomial

$$\tilde{v} = \tilde{v}_1 x + \tilde{v}_0$$

$$= 2v + h \mod u$$

$$= h_2 x^2 + (2v_1 + h_1)x + (2v_0 + h_0) \mod u$$

is computed. The explicit formula is:

$$\tilde{v}_1 = 2v_1 + h_1 - h_2 u_1; \quad \tilde{v}_0 = 2v_0 + h_0 - h_2 u_0.$$ 

In odd characteristic the formula is:

$$\tilde{v}_1 = 2v_1; \quad \tilde{v}_0 = 2v_0.$$ 

In even characteristic the formula is:

$$\tilde{v}_1 = h_1 - h_2 u_1; \quad \tilde{v}_0 = h_0 - h_2 u_0.$$ 

This step has no cost.

Step 2: Compute $k = (f - hv - v^2)/u \mod u$

For this step notice the division by $u$ is exact, so only the first four highest degree coefficients are needed since a degree five polynomial is divided by a degree two polynomial (as described in Section 4.2.3). First the $a = f - hv - v^2$ expression is computed, then $q = a/u = f - hv - v^2/u$, and finally $k \mod u$.

$$a = f - hv - v^2$$

$$v^2 = v_1^2 x^2 + ...$$

$$hv = h_2 v_1 x^3 + (h_2 v_0 + h_1 v_1)x^2 + ...$$
and so \( a = f - hv - v^2 = x^5 + a_4x^4 + a_3x^3 + a_2x^2 + \ldots \) where

\[
\begin{align*}
a_4 &= f_4, \\
a_3 &= f_3 - h_2v_1, \\
a_2 &= f_2 - h_2v_0 - h_1v_1 - v_1^2.
\end{align*}
\]

\( q = a/u \)

With the use of exact division, the intermediate quotient \( q = (f - hv - v^2)/u = a/u \) is found to be

\[
q = x^3 + q_2x^2 + q_1x + q_0,
\]

\( q_2 = (a_4 - u_1), \)

\( q_1 = (a_3 - a_4u_1 + u_1^2 - u_0), \)

\( q_0 = (a_2 + a_4(u_1^2 - u_0) - a_3u_1 + 2u_1u_0 - u_1^3). \)

\( k = q \mod u \)

Here the remainder of \( q \mod u \) is taken and then substituting in the values of \( q \) and \( a \) results in \( k = k_1x + k_0 \) where

\[
\begin{align*}
k_1 &= q_1 - u_0 - u_1(q_2 - u_1) \\
&= f_3 - h_2v_1 - 2f_4u_1 - 2u_0 + 3u_1^2 \\
&= 2(u_1^2 - f_4u_1 - u_0) + f_3 + u_1^2 - h_2v_1
\end{align*}
\]

\[
\begin{align*}
k_0 &= q_0 - u_0(q_2 - u_1), \\
&= f_2 - h_2v_0 - h_1v_1 - v_1^2 + f_4(u_1^2 - u_0) - (f_3 - h_2v_1)u_1 + \\
&\quad + 2u_1u_0 - u_1^3 - u_0(f_3 - h_2v_1 - f_4u_1 + u_1^2 - u_0 + u_0u_1 \\
&= u_1(-f_3 + h_2v_1 + u_1f_4 + 4u_0 - u_1^2) + f_2 - v_1^2 - h_1v_1 - h_2v_0 - 2u_0f_4.
\end{align*}
\]

The last equality is in the form best suited for implementation. Notice that Karatsuba reduction does not save a multiplication because both \( q \) and \( u \) are monic. The explicit
formula is:

\[ w_0 = v_1^2; \quad w_1 = u_1^2; \quad w_2 = f_3 + w_1; \quad w_3 = 2u_0; \]

\[ k_1 = 2(w_1 - f_4u_1) + w_2 - w_3 - h_2v_1; \]

\[ k_0 = u_1(2w_3 - w_2 + f_4u_1 + h_2v_1) + f_2 - w_0 - 2f_4u_0 - h_1v_1 - h_2v_0. \]

In odd characteristic the explicit formula is:

\[ w_0 = v_1^2; \quad w_1 = u_1^2; \quad w_2 = f_3 + w_1; \quad w_3 = 2u_0; \]

\[ k_1 = 2w_1 + w_2 - w_3; \]

\[ k_0 = u_1(2w_3 - w_2) + f_2 - w_0. \]

Notice that in the even characteristic setting the computation of \( v^2 \) is unneeded, so the even characteristic explicit formula is:

\[ w_1 = u_1^2; \]

\[ k_1 = w_1 - h_2v_1; \]

\[ k_0 = -u_1k_1 - v_1(h_1 + v_1) - h_2v_0. \]

The cost of this step is \( 1M, 2S, 0C \) in odd characteristic, and \( 2M, 1S, 0C \) in even characteristic.

**Step 3: Set up system of equations to solve for \( s = k/\tilde{v} \mod u \)**

In Section 3.2.2 the polynomial \( s = s_1x + s_0 \mod u \), where

\[ s \equiv \frac{k}{\tilde{v}} \mod u \]

is described as a sub-expression. Multiplying both sides with \( \tilde{v} \) results in

\[ k_1x + k_0 = (s_1x + s_0)(\tilde{v}_1x + \tilde{v}_0) = s_1\tilde{v}_1x^2 + (s_1\tilde{v}_0 + s_0\tilde{v}_1)x + s_0\tilde{v}_0 \mod x^2 + u_1x + u_0. \]

Taking \( s_1\tilde{v}_1x^2 + (s_1\tilde{v}_0 + s_0\tilde{v}_1)x + s_0\tilde{v}_0 \mod x^2 + u_1x + u_0 \) results in

\[ k_1x + k_0 = (s_1(\tilde{v}_0 - \tilde{v}_1u_1) + s_0(\tilde{v}_1))x + (s_1(-\tilde{v}_1u_0) + s_0(\tilde{v}_0)), \]
and $s_0, s_1$ can be found by solving
\[
k_0 = s_0(\bar{v}_0) + s_1(-\bar{v}_1 u_0),
\]
\[
k_1 = s_0(\bar{v}_1) + s_1(\bar{v}_0 - \bar{v}_1 u_1),
\]
from
\[
\begin{pmatrix}
\bar{v}_0 & -\bar{v}_1 u_0 \\
\bar{v}_1 & \bar{v}_0 - \bar{v}_1 u_1 \\
\end{pmatrix}
\times
\begin{pmatrix}
s_0 \\
s_1 \\
\end{pmatrix}
= \begin{pmatrix}
k_0 \\
k_1 \\
\end{pmatrix}.
\]
Transforming the system to solve for $s_i$ directly results in
\[
\begin{pmatrix}
\bar{v}_0 - \bar{v}_1 u_1 & \bar{v}_1 u_0 \\
-\bar{v}_1 & \bar{v}_0 \\
\end{pmatrix}
\times
\begin{pmatrix}
k_0 \\
k_1 \\
\end{pmatrix}
= \begin{pmatrix}
s_0 \\
s_1 \\
\end{pmatrix}.
\]
The explicit formula for setting up the linear system is:
\[
m_1 = \bar{v}_0 - \bar{v}_1 u_1; \quad m_2 = \bar{v}_1 u_0. \quad m_3 = -\bar{v}_1; \quad m_4 = \bar{v}_0;
\]
The cost of this step is $2M, 0S, 0C$ in both characteristics.

**Step 4: Solve system to get $s' = s'_x + s'_0 = k/\bar{v}$ mod $u$**

This step is identical to Step 2 of affine addition where $\bar{v}_i$ is replaced with $k_i$ for $i = 0, 1$. In the odd characteristic setting, we introduce a novel reduction of one multiplication in this step by using the “improved solution to a system of equations” technique (Section 4.2.8). In general the explicit formula is:
\[
s_0' = k_0 m_1 + k_1 m_2; \quad s_1' = k_0 m_3 + k_1 m_4;
\]
\[
d' = m_4 m_1 - m_2 m_3.
\]
In odd characteristic the explicit formula is:
\[
w_0 = (m_2 - k_0)(k_1 - m_1); \quad w_1 = (-k_0 - m_2)(k_1 + m_1);
\]
\[
w_2 = (m_4 - k_0)(k_1 - m_3); \quad w_3 = (-k_0 - m_4)(k_1 + m_3);
\]
\[
s_0' = w_0 - w_1; \quad s_1' = w_2 - w_3;
\]
\[
d' = w_2 + w_3 - w_0 - w_1 - 2(m_2 - m_4)(m_1 + m_3).
\]
The cost is $5M, 0S, 0C$ in odd characteristic and $6M, 0S, 0C$ in even characteristic.

**Step 5: Compute** $s'' = s/s_1 = x + s''_0$

This step is identical to Step 3 of affine addition. The explicit formula is:

\[
\begin{align*}
w_1 &= 1/(d's_1) = (1/d'^2s_1); \\
w_2 &= w_1d' = (1/d's_1); \\
w_3 &= w_2d' = (1/s_1); \\
w_4 &= w_3^2 = (1/s_1^2); \\
s_1 &= s_1^2w_1 = (s_1); \\
s''_0 &= s'_0w_2.
\end{align*}
\]

The cost of this step is $1I, 5M, 2S, 0C$ in both characteristics.

**Step 6: Compute** $u'$

The same technique of equating coefficients to compute $u'$ as for $u''$ in affine addition is used here. The same equality

\[
(x^2 + u_1x + u_0)(x^2 + u'_1x + u'_0) = \frac{l'^2 + l'h - f}{l'^2}
\]

is used as in affine addition. Equating the coefficients of $x^5$ and $x^4$ results in

\[
2u_1 + u'_1 = \frac{2l_3'l_2' + l_3'h2 - 1}{l'^2},
\]

and

\[
u'_0 + u_1^2 + 2u_1'u_1 = \frac{2l_1'l_3' + l_2'^2 + h_2l_2' + h_1l_1' - f_4}{l'^2},
\]

respectively. To compute $u'_1$ and $u'_0$, $l'$ is not directly used in our explicit formula but broken up into the additive parts of its coefficients including the $v$ portion and grouped into expressions in terms of the adjustments by $w_3 = 1/s_1$ and $w_4 = 1/s_1^2$ that are needed.
Adjusting with \( w_3 \) and \( w_4 \) accordingly we get

\[
\begin{align*}
    u_1'' &= -2u_1 + 2(s_0'' + u_1) + h_2w_3 - w_4; \\
    u_0'' &= -u_1^2 - u_1''(2u_1) + 2(s_0''u_1 + u_0 + w_3v_1) + (s_0'' + u_1)^2 + w_3(h_2(s_0'' + u_1) + h_1) - w_4f_4.
\end{align*}
\]

After manipulating the expressions to require the least number of multiplications, the explicit formula is:

\[
\begin{align*}
    u_1' &= 2s_0'' + h_2w_3 - w_4; \\
    u_0' &= s_0'' + w_3(h_2(s_0'' - u_1) + 2v_1 + h_1) + w_4(2u_1 - f_4).
\end{align*}
\]

In odd characteristic the explicit formula is:

\[
\begin{align*}
    u_1' &= 2s_0'' - w_4; \\
    u_0' &= s_0'' + 2(w_3v_1 + w_4u_1).
\end{align*}
\]

In even characteristic the explicit formula is:

\[
\begin{align*}
    u_1' &= h_2w_3 - w_4; \\
    u_0' &= s_0'' + w_3(h_2(s_0'' - u_1) + h_1).
\end{align*}
\]

The cost of this step is \( 2M, 1S, 0C \) in odd characteristic and \( 1M, 1S, 0C \) in even characteristic.

**Step 7:** Compute \( v' = -h - l' \mod u'' \)

This step is identical to Step 6 in affine addition. We introduce a novel reduction of one multiplication (Section 4.2.9). The general and even characteristic explicit formula is:

\[
\begin{align*}
    w_0 &= (u_1' - s_0'')(u_1 - u_1'); \\
    v_1' &= s_1(w_0 + u_0' - u_0) - v_1 - h_1 + h_2u_1'; \\
    v_0' &= s_1(s_0''(u_0' - u_0) + u_0'(u_1 - u_1')) - v_0 - h_0 + h_2u_0'.
\end{align*}
\]
Specializing into odd characteristic explicit formula is:

\[ w_0 = (u'_1 - s''_0)(u_1 - u'_1); \]
\[ v'_1 = s_1(w_0 + u'_0 - u_0) - v_1; \]
\[ v'_0 = s_1(s''_0(u'_0 - u_0) + u'_0(u_1 - u'_1)) - v_0. \]

The cost of this step is 5M, 0S, 0C in both characteristics.

The total cost is 1I, 20M, 5S, 0C in odd characteristic, and 1I, 21M, 4S, 0C in even characteristic. Table 5.4 summarizes the formulae in arbitrary case. Tables 5.5 and 5.6 summarize the formulae in even and odd characteristic cases respectively.

5.2 Projective Formulae

In this section we present the fastest projective formulae for doubling, addition and mixed addition of divisor classes in the frequent case. The projective setting is inversion free, trading one inversion for extra field operations. An explanation of this setting and terminology used is given in Section 4.3. A specialized version of using a system of equations to solve for \( s \) is implemented to solve for the \( s''_0 = s/s_1 \) directly, reducing the overall weight of the output (Section 4.3.1). We introduce adaptations of this method to the even characteristic case significantly saving on operations over previous best [27], and introduce the use of “improved solution to a system of equations” technique (Section 4.2.8) to save on operations in odd characteristic.

When implementing a left to right non-adjacent form or double base algorithm for scalar multiplication, all additions performed use the original divisor \( D_o = [u_{o1}, u_{o0}, v_{o1}, v_{o0}, z_o] \) as one of the inputs. The divisor \( D_o \) is affine to start, so \( z_o = 1 \). The number of field operations needed to compute an addition when one of the input divisor classes is affine is greatly reduced. This cases is called mixed addition; we cover this case separately from addition.

We give an exposition of the explicit formulae for divisor class addition in detail, then
<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Compute $\hat{v} \equiv h + 2v \mod u$: $\hat{v}_1 = 2v_1 + h_1 - h_2u_1; \hat{v}_0 = 2v_0 + h_0 - h_2u_0$;</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>Compute $k \equiv (f - hv - v^2)/u \mod u$: $w_0 = v_1^2; w_1 = u_1^2; w_2 = f_3 + w_1; w_3 = 2u_0$; $k_1 = 2(w_1 - f_4u_1) + w_2 - w_3 - h_2v_1$; $k_0 = u_1(2w_3 - w_2 + f_4u_1 + h_2v_1) + f_2 - w_0 - 2f_4u_0 - h_1v_1 - h_2v_0$;</td>
<td>3M, 2S</td>
</tr>
<tr>
<td>3</td>
<td>Setup system of equations for $s \equiv k/\hat{v} \mod u$: $m_1 = \hat{v}_0 - \hat{v}_1u_1; m_2 = \hat{v}_1u_0; m_3 = -\hat{v}_1; m_4 = \hat{v}_0$;</td>
<td>2M</td>
</tr>
<tr>
<td>4</td>
<td>Solve system of equations for $s' = sd' = s'_1x + s'_0$. $s'_0 = k_0m_1 + k_1m_2; s'_1 = k_0m_3 + k_1m_4; d' = m_4m_1 - m_2m_3$; (if $d'$ = 0 branch to Cantor’s Algorithm)</td>
<td>6M</td>
</tr>
<tr>
<td>5</td>
<td>Compute $s'' = x + s'_0/s'_1$ and $s_1$: $w_1 = (d's'_1)^{-1}; w_2 = w_1d'; w_3 = w_2d' = (1/s_1); w_4 = w_3^2; s_1 = s_2^2w_1 = (s_1); s'_0 = s'_0w_2$;</td>
<td>1I, 5M, 2S</td>
</tr>
<tr>
<td>6</td>
<td>Compute $u'$: $u'_0 = 2s''_0 + h_2w_3 - w_4$; $u'_0 = s''_0^2 + w_3(h_2(s''_0 - u_1) + 2v_1 + h_1) + w_4(2u_1 - f_4)$;</td>
<td>2M, 1S</td>
</tr>
<tr>
<td>7</td>
<td>Compute $v' = -h - l' \mod u'$: $w_0 = (u'_1 - s''_0)(u_1 - u'_1); v'_1 = s_1(u_0 - u') - v_1 - h_1 + h_2u'_1; v'_0 = s_1(s''_0(u_0 - u_0) + u'_0(u_1 - u'_1)) - v_0 - h_0 + h_2u'_0$;</td>
<td>5M</td>
</tr>
</tbody>
</table>

Table 5.4: Arbitrary Affine Doubling Formula
### Table 5.5: Even Characteristic Affine Doubling Formula

**IN:** Reduced divisor $D = [u, v]$ with

- $u = x^2 + u_1 x + u_0$, $v = v_1 x + v_0$
- $h = h_2 x^2 + h_1 x + h_0$,
- $f = x^5 + f_4 x^4 + f_3 x^3 + f_2 x^2 + f_1 x + f_0$

| **OUT:** Reduced divisor $D' = [u', v'] = 2D$ with |
| $u' = x^2 + u'_1 x + u'_0$, $v' = v'_1 x + v'_0$. |

<table>
<thead>
<tr>
<th><strong>Step</strong></th>
<th><strong>Procedure</strong></th>
<th><strong>Cost</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><strong>Compute</strong> $\tilde{v} \equiv h + 2v \mod u$: $\tilde{v}_1 = h_1 - h_2 u_1$; $\tilde{v}_0 = h_0 - h_2 u_0$;</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td><strong>Compute</strong> $k \equiv (f - hv - v^2)/u \mod u$: $w_1 = u_1^2$; $k_1 = w_1 - h_2 v_1$; $k_0 = -u_1 k_1 - v_1 (v_1 + h_1) - h_2 v_0$;</td>
<td>2M, 1S</td>
</tr>
<tr>
<td>3</td>
<td><strong>Setup system of equations for</strong> $s \equiv k/\tilde{v} \mod u$: $m_1 = \tilde{v}_0 - \tilde{v}_1 u_1$; $m_2 = \tilde{v}_1 u_0$; $m_3 = -\tilde{v}_1$; $m_4 = \tilde{v}_0$;</td>
<td>2M</td>
</tr>
<tr>
<td>4</td>
<td><strong>Solve system of equations for</strong> $s' = s d' = s'_1 x + s'_0$; $s'_0 = k_0 m_1 + k_1 m_2$; $s'_1 = k_0 m_3 + k_1 m_4$; $d' = m_4 m_1 - m_2 m_3$; (if $d' = 0$ branch to Cantor’s Algorithm)</td>
<td>6M</td>
</tr>
<tr>
<td>5</td>
<td><strong>Compute</strong> $s'' = x + s'_0 / s'_1$ and $s_1$: $w_1 = (d's'_1)^{-1}$; $w_2 = w_1 d'$; $w_3 = w_2 d' = (1/s_1)$; $w_4 = w_3^2$; $s_1 = s_2^2 w_1 = (s_1)$; $s_0'' = s'_0 w_2$;</td>
<td>1I, 5M, 2S</td>
</tr>
<tr>
<td>6</td>
<td><strong>Compute</strong> $u'$: $w_0 = (u'_1 - s_0'')(u_1 - u'_1)$; $v'_1 = s_1(u'_0 + u'_1 - u_0) - v_1 - h_1 + h_2 u'_1$; $v'_0 = s_1(s_0(u'_0 - u_0) + u'_0(u_1 - u'_1)) - v_0 - h_0 + h_2 u'_0$;</td>
<td>1M, 1S</td>
</tr>
<tr>
<td>7</td>
<td><strong>Compute</strong> $v' = -h - l' \mod u'$:</td>
<td>5M</td>
</tr>
</tbody>
</table>

**Total** | 1, 21M, 4S |
Table 5.6: Odd Characteristic Affine Doubling Formula

<table>
<thead>
<tr>
<th>IN:</th>
<th>Reduced divisor $D = [u, v]$ with</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$u = x^2 + u_1 x + u_0$, $v = v_1 x + v_0$</td>
</tr>
<tr>
<td></td>
<td>$h = h_2 x^2 + h_1 x + h_0$,</td>
</tr>
<tr>
<td></td>
<td>$f = x^5 + f_4 x^4 + f_3 x^3 + f_2 x^2 + f_1 x + f_0$</td>
</tr>
<tr>
<td>OUT:</td>
<td>Reduced divisor $D' = [u', v'] = 2D$ with</td>
</tr>
<tr>
<td></td>
<td>$u' = x^2 + u'_1 x + u'_0$, $v' = v'_1 x + v'_0$.</td>
</tr>
<tr>
<td>Step</td>
<td>Procedure</td>
</tr>
<tr>
<td>-------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>1</td>
<td>Compute $\hat{v} \equiv h + 2v \mod u$:</td>
</tr>
<tr>
<td></td>
<td>$\hat{v}_1 = 2v_1; \hat{v}_0 = 2v_0$;</td>
</tr>
<tr>
<td>2</td>
<td>Compute $k \equiv (f - hv - v^2)/u \mod u$:</td>
</tr>
<tr>
<td></td>
<td>$w_0 = v_1^2; w_1 = u_1^2; w_2 = f_3 + w_1; w_3 = 2u_0$;</td>
</tr>
<tr>
<td></td>
<td>$k_1 = 2w_1 + w_2 - w_3; k_0 = u_1(2w_3 - w_2) + f_2 - w_0$;</td>
</tr>
<tr>
<td>3</td>
<td>Setup system of equations for $s \equiv k/\hat{v} \mod u$:</td>
</tr>
<tr>
<td></td>
<td>$m_1 = \hat{v}_0 - \hat{v}_1 u_1; m_2 = \hat{v}_1 u_0; m_3 = -\hat{v}_1; m_4 = \hat{v}_0$;</td>
</tr>
<tr>
<td>4</td>
<td>Solve system of equations for $s' = sd' = s'_1 x + s'_0$:</td>
</tr>
<tr>
<td></td>
<td>$w_0 = (m_2 - k_0)(k_1 - m_1); w_1 = (-k_0 - m_2)(k_1 + m_1)$;</td>
</tr>
<tr>
<td></td>
<td>$w_2 = (m_4 - k_0)(k_1 - m_3); w_3 = (-k_0 - m_4)(k_1 + m_3)$;</td>
</tr>
<tr>
<td></td>
<td>$s'_0 = w_0 - w_1; s'_1 = w_2 - w_3$;</td>
</tr>
<tr>
<td></td>
<td>$d' = w_2 + w_3 - w_0 - w_1 - 2(m_2 - m_4)(m_1 + m_3)$;</td>
</tr>
<tr>
<td></td>
<td>(if $d' = 0$ branch to Cantor’s Algorithm)</td>
</tr>
<tr>
<td>5</td>
<td>Compute $s'' = x + s'_0/s'_1$ and $s_1$:</td>
</tr>
<tr>
<td></td>
<td>$w_1 = (d's'_1)^{-1}; w_2 = w_1 d'; w_3 = w_2 d' = (1/s_1)$;</td>
</tr>
<tr>
<td></td>
<td>$w_4 = w_3^2; s_1 = s''^2 w_1 = (s_1); s_0'' = s_0'' w_2$;</td>
</tr>
<tr>
<td>7</td>
<td>Compute $u'$:</td>
</tr>
<tr>
<td></td>
<td>$u'_0 = 2s_0'' - w_4; u'_1 = s_0''^4 + 2(w_3 v_1 + w_4 u_1)$;</td>
</tr>
<tr>
<td>8</td>
<td>Compute $v' = -h - l' \mod u'$:</td>
</tr>
<tr>
<td></td>
<td>$w_0 = (u'_1 - s''_0)(u_1 - u'_1)$;</td>
</tr>
<tr>
<td></td>
<td>$v'_1 = s_1(w_0 + u'_0 - u_0) - v_1$;</td>
</tr>
<tr>
<td></td>
<td>$v'_0 = s_1(s_0''(u'_0 - u_0) + u'_0(u_1 - u'_1)) - v_0$;</td>
</tr>
<tr>
<td>Total</td>
<td></td>
</tr>
</tbody>
</table>

81
mention the key differences in the case of mixed addition, and doubling.

5.2.1 Addition

Algorithm 5.3 gives an overview of the steps involved in the explicit formula for addition in the projective setting. The explicit formula for projective addition follows the same path

Algorithm 5.3 Summary of divisor class addition in the projective setting

**Input:** $D = [u_1, u_0, v_1, v_0, z], \ D' = [u'_1, u'_0, v'_1, v'_0, z']$ where,

- $u = x^2 + (u_1/z)x + (u_0/z)$
- $v = (v_1/z)x + (v_0/z)$
- $u' = x^2 + (u'_1/z')x + (u'_0/z')$
- $v' = (v'_1/z')x + (v'_0/z')$
- $f = x^5 + f_4x^4 + f_3x^3 + f_2x^2 + f_1x + f_0$
- $h = h_2x^2 + h_1x + h_0$

**Output:** $D'' = [u''_1, u''_0, v''_1, v''_0, z''] = D + D'$ where,

- $u'' = x^2 + (u''_1/z'')x + (u''_0/z'')$
- $v'' = (v''_1/z'')x + (v''_0/z'')$

1: Adjust weights of the input values to have weight $\tilde{z} = zz'$
2: Set up system of equations that solves for $s'' \equiv (v' - v)/s_1u \mod u'$ and $i = 1/s_1$
3: Solve system to get $s''_0$ and $c$ with weight $d''$
4: Pre-computations and compute the output weight $z''$
5: Compute $u''$ by equating coefficients
6: Adjust $u$ to have the same weight as $u''$
7: Compute $v'' \equiv -h - l' \mod u''$
8: Adjust $u''$ to have a weight of $z''$

overall as affine addition with extra computations because of the weights involved. We begin with Step 1.
Step 1: Adjust weights of the input values to \( \tilde{z} = zz' \)

The input values are forced to have the same weight \( \hat{z} = zz' \), this helps make the rest of the computations easier. The explicit formula is:

\[
\begin{align*}
\tilde{v}_1 &= v_1 z'; \\
\tilde{v}_0 &= v_0 z'; \\
\tilde{u}_1 &= u_1 z'; \\
\tilde{u}_0 &= u_0 z'; \\
\tilde{v}_1' &= v_1' z; \\
\tilde{v}_0' &= v_0' z; \\
\tilde{u}_1' &= u_1' z; \\
\tilde{u}_0' &= u_0' z; \\
\hat{z} &= zz'.
\end{align*}
\]

The cost of this step is \( 9M, 0S, 0C \) in both characteristics.

Step 2: Set up system of equations for \( s'' \equiv (v' - v)/s_1 u' \mod u \) and \( i = 1/s_1 \)

The equation

\[
s_1 x + s_0 = \frac{(v_1 - v_1') x + (v_0 - v_0')}{x'^2 + u_1' x + u_0'} \mod x^2 + u_1 x + u_0,
\]

is manipulated to

\[
c(v' - v) = (x + s''_0)(u - u') - (u_1 - u_1')u'
\]

where \( c = 1/s_1 \) and \( s''_0 = s_0/s_1 \) is as in the affine case (Section 4.3.1). The weighted system is set up as

\[
\begin{pmatrix}
\tilde{u}_0 - \tilde{u}_0' \\
\tilde{v}_0' - \tilde{v}_0 \\
\tilde{u}_1 - \tilde{u}_1' \\
\tilde{v}_1' - \tilde{v}_1
\end{pmatrix}
\times
\begin{pmatrix}
s''_0 \\ c
\end{pmatrix}
= \begin{pmatrix}
\tilde{u}_0' (\tilde{u}_1 - \tilde{u}_1') \\
\tilde{u}_1' (\tilde{u}_0 - \tilde{u}_0') + \tilde{u}_0' (\tilde{u}_1' - \tilde{u}_1)
\end{pmatrix}.
\]

Fixing the coefficients to symbols used in the explicit formula results in,

\[
\begin{pmatrix}
m_4 & -m_2 \\
-m_3 & m_1
\end{pmatrix}
\times
\begin{pmatrix}
s''_0 \\ c
\end{pmatrix}
= \begin{pmatrix}
r_0 \\
r_1
\end{pmatrix}.
\]
The explicit formula is:

\[ m_1 = \tilde{v}'_1 - \tilde{v}_1; \quad m_2 = \tilde{v}_0 - \tilde{v}'_0; \]
\[ m_3 = \tilde{u}'_1 - \tilde{u}_1; \quad m_4 = \tilde{u}_0 - \tilde{u}'_0; \]
\[ w_0 = \hat{z}m_4; \quad w_1 = \tilde{u}'_1m_3; \]
\[ r_0 = -\tilde{u}'_0m_3; \quad r_1 = w_0 + w_1. \]

The values \( r_0 \) and \( r_1 \) have a weight of \( \hat{z}^2 \), and the \( m_i \) have a weight of \( \hat{z} \) with respect to their affine values. The cost of this step is \( 3M, 0S, 0C \) in both characteristics.

**Step 3: Solve system to get \( s'''_0 \) and \( c \) with weight \( d''' \)**

Here we introduce a novel reduction of one multiplication in odd characteristic due to the “improved solution to a system of equations” trick (Section 4.2.8). In general, the explicit formula is:

\[ s'''_0 = r_0m_1 + r_1m_2; \]
\[ c' = r_0m_3 + r_1m_4; \]
\[ d'' = m_4m_1 - m_2m_3; \]
\[ d''' = d''\hat{z}. \]

In odd characteristic the explicit formula is:

\[ w_0 = (m_2 - r_0)(r_1 - m_1); \quad w_1 = (-r_0 - m_2)(r_1 + m_1); \]
\[ w_2 = (m_4 - r_0)(r_1 - m_3); \quad w_3 = (-r_0 - m_4)(r_1 + m_3); \]
\[ s'''_0 = w_0 - w_1; c' = w_2 - w_3; \]
\[ d'' = w_2 + w_3 - w_0 - w_1 - 2(m_2 - m_4)(m_1 + m_3); \]
\[ d''' = d''\hat{z}. \]

Notice that \( d'' = d'\hat{z}^2 \) and \( d''' = d'\hat{z}^3 \) with respect to the affine value of \( d' \). Then \( s'''_0 = s''_0d'\hat{z}^3 = s''_0d''' \) and \( c' = d'\hat{z}^3/s_1 = d'''/s_1 \) have a weight of \( d''' \) with respect to their affine
values as well. The cost of this step is $7M, 0S, 0C$ for both general and even characteristic and the cost is $6M, 0S, 0C$ in odd characteristic.

**Step 4: Pre-computations and $z''$**

At this point $d'''$, $s'''_0$ ($s_0''$ with weight $d'''$), and $c'$ ($1/s_1$ with weight $d'''$) have been computed. All pre-computations are done to produce differently weighted versions of the affine values $s_0'' = s_0/s_1$, $1/s_1$ and $1/s_1^2$ that are used in the computation of $u''$ and $v''$. The output weight $z''$ that both $u''$ and $v''$ have is also computed. The explicit formula is:

\[
\begin{align*}
D &= d'''; \\
D' &= D\hat{z}; \\
S_0 &= s'''_0\hat{z}; \quad (s_0'' \text{ with a weight of } d'''; \hat{z}) \\
S_0' &= d''S_0; \quad (s_0'' \text{ with a weight of } D\hat{z}) \\
w_0 &= d''c; \quad (1/s_1 \text{ with a weight of } D) \\
S_1 &= \hat{z}c; \quad (1/s_1 \text{ with a weight of } d'''; \hat{z}) \\
S_1' &= w_0\hat{z}; \quad (1/s_1 \text{ with a weight of } D\hat{z}) \\
S_1'' &= S_1'\hat{z}; \quad (1/s_1 \text{ with a weight of } D\hat{z}^2) \\
S_{sq} &= S_1c; \quad (1/s_1^2 \text{ with a weight of } D\hat{z}) \\
z'' &= S''_1D'. \quad (D^2\hat{z}^3/s_1)
\end{align*}
\]

All of the values are adjusted to have a $D$ in their weights because in order to compute $u''$ a weighted $1/s_1^2$ and $s'''_0$ needs to be computed and only $c' = d'''/s_1$ and $s'''_0 = s''_0d'''$ are available. The cost of this step is $9M, 1S, 0C$ in both characteristics.

**Step 5: Compute $u''$**

The affine formulae for $u''_1$ and $u''_0$ from Step 5 of Table 5.1 are converted to our projective formulae directly. Recall that $r_1 = w_0 + w_1 = \hat{z}(\tilde{u}_0' - \tilde{u}_0) + \tilde{u}_1'(\tilde{u}_1' - \tilde{u}_1)$ has a weight of $\hat{z}^2$, and $m_3 = \tilde{u}_1' - \tilde{u}_1$ has a weight of $\hat{z}$. 85
With the pre-computed values from the last step, the affine formula for \( u_1'' \),

\[
  u_1'' = 2s''_0 + m_3 + h_2(1/s_1) - (1/s_1^2),
\]

becomes

\[
  u_1'' = 2S'_0 - Dm_3 + S'_1 h_2 - S_{sq}
\]

with a weight of \( D\hat{z} \). The affine formula for \( u_0'' \),

\[
  u_0'' = s''_0(s''_0 - 2m_3) + r_1 + (1/s_1)(h_2(s''_0 - u'_1) + h_1 + 2v_1) + (1/s_1^2)(u_1 + u'_1 - f_4),
\]

becomes

\[
  u_0'' = S_0(S_0 - 2d'''m_3) + Dr_1 + S_1(h_2S_0 - d'''(h_2\bar{u}'_1 + \hat{z}h_1 + 2v_1)) + S_{sq}(u_1 + u'_1 - \hat{z}f_4)
\]

with a weight of \( D\hat{z}^2 \). The general explicit formula is:

\[
  \hat{h}_1 = \hat{z}h_1;
\]

\[
  u_1'' = 2S'_0 - Dm_3 + S'_1 h_2 - S_{sq};
\]

\[
  u_0'' = S_0(S_0 - 2d'''m_3) + Dr_1 + S_1(h_2S_0 - d'''(h_2\bar{u}'_1 + \hat{h}_1 + 2v_1)) + S_{sq}(\bar{u}_1 + \bar{u}'_1 - \hat{z}f_4);
\]

The odd characteristic formula is

\[
  \hat{v}_1 = S'_1 \hat{v}_1;
\]

\[
  u_1'' = 2S'_0 - Dm_3 - S_{sq};
\]

\[
  u_0'' = S_0(S_0 - 2d'''m_3) + Dr_1 + 2\hat{v}_1 + S_{sq}(\bar{u}_1 + \bar{u}'_1);
\]

The even characteristic formula is

\[
  \hat{h}_1 = \hat{z}h_1;
\]

\[
  u_1'' = S'_1 h_2 - Dm_3 - S_{sq};
\]

\[
  u_0'' = S_0^2 + Dr_1 + S_1(h_2S_0 - d'''(h_2\bar{u}'_1 + \hat{h}_1)) + S_{sq}(\bar{u}_1 + \bar{u}'_1);
\]

The value \( v_1 \) is given a weight of \( \hat{z}^2 D/s_1 \) in the odd characteristic case because all other terms that have a \( 1/s_1 \) involve \( h \) and become zero. The cost of this step is \( 6M, 0S, 0C \) in odd characteristic and \( 5M, 1S, 1C \) in even characteristic.
**Step 6:** Adjust $u$ to the same weight as $u''$

The values $u_1$ and $u_0$ are given the same weights as $u''_1$ and $u''_0$ respectively because subtractions between the $u_i$ and $u''_i$ are computed in the computation of $v''$. The explicit formula is

$$\hat{u}_1 = \tilde{u}_1 D; \quad \hat{u}_0 = \tilde{u}_0 D'.$$

This step costs $2M, 0S, 0C$ in both characteristics.

**Step 7:** Compute $v'' = -h - l' \mod u''$

In this step the projective $v''_1$ and $v''_0$ are computed by adapting their affine formulae (Step 6 of Table 5.1) to include the pre-computed weights from Step 4. Recall the affine formula for $v_1$ and $v_0$:

$$v''_1 = s_1((u''_1 - s''_0)(u_1 - u''_1) + u''_0 - u_0) - v_1 - h_1 + h_2 u''_1; \quad v''_0 = s_1(s''_0(u''_0 - u_0) + u''_0(u_1 - u''_1)) - v_0 - h_0 + h_2 u''_0.$$

The explicit formulae for general and even characteristic is:

$$w_0 = (u''_1 - S'_0)(\hat{u}_1 - u''_1); \quad \hat{h}_0 = h_0 \hat{z};$$

$$v''_1 = \hat{z}(w_0 - S'_1(D(\hat{v}_1 + \hat{h}_1) - h_2 u''_1)) + D(u''_0 - \hat{u}_0); \quad v''_0 = S'_0(u''_0 - \hat{u}_0) + u''_0(\hat{u}_1 - u''_1) - S'_1(D'(\hat{v}_0 + \hat{h}_0) - h_2 u''_0).$$

The explicit formula for odd characteristic is,

$$\hat{v}_0 = S'_1 \hat{v}_0; \quad w_0 = (u''_1 - S'_0)(\hat{u}_1 - u''_1);$$

$$v''_1 = \hat{z}(w_0 + D(u''_0 - \hat{u}_0 - \hat{v}_1)); \quad v''_0 = S'_0(u''_0 - \hat{u}_0) + u''_0(\hat{u}_1 - u''_1) - D' \hat{v}_0.$$
The value $v_0$ is adjusted to have a weight of $\hat{z}^2D/s_1$ in odd characteristic because $h$ is zero. Both $v''_1$ and $v''_0$ have a weight of $\hat{z}^3D^2/s_1 = z''$. The cost of this step is $7M, 0S, 0C$ in odd characteristic and $9M, 0S, 1C$ in even characteristic.

**Step 8: Adjust $u''$ to have weight $z''$**

In this last step, $u''$ is adjusted to have a weight of $z'' = \hat{z}^3D^2/s_1$, the same weight as $v''$.

\[ u''_1 = S'_1u''_1; \quad u''_0 = S_1u''_0. \]

The cost of this step is $2M, 0S, 0C$ in both characteristics.

The total cost of projective addition is $44M, 1S$ in odd characteristic and $46M, 2S, 2C$ in even characteristic. The extra field operations for the even characteristic case come from having to deal with the $h_i$ in the computation (that in affine do not have to be multiplied with anything) of $v''$ and $u''$. Table 5.7 summarizes the formulae in arbitrary case. Tables 5.8 and 5.9 summarize the formulae in even and odd characteristic cases respectively.

### 5.2.2 Mixed Addition

In mixed addition, the input is the same as for addition, but $Z' = 1$. Many steps in the projective addition formula are simplified, most notably the pre-computations.

Notice that in Step 1, only four multiplications are needed, and that $\hat{z} = Z$. In Step 2, a weight of $\hat{Z}$ can be achieved instead of $\hat{Z}^2$ for $r_0$ and $r_1$, and so the resulting $i'$ and $s''_0$ have a weight of $d''$ instead of $d'' = d''\hat{z}$ for a savings of one multiplication in Step 3. The value $S'_1$ is not computed in Step 4, among other pre-computations that become unnecessary. The smaller weights acquired in Step 3 allow for all of $u''$ to have weight $D\hat{z}$ which further allows for $z'' = D^2s_1\hat{z}^2$ instead of $D^2s_1\hat{z}^3$. The cost of of mixed addition is $34M, 2S$ in odd characteristic and $37M, 2S, 1C$ in even characteristic. Table 5.10 summarizes the formulae in arbitrary case. Tables 5.11 and 5.12 summarize the formulae in even and odd characteristic cases respectively.
### Table 5.7: Arbitrary Projective Addition Formula

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Adjustments to input and the weight $\tilde{z} = zz'$: &lt;br&gt; $\bar{v}_1 = v_1 z'$; $\bar{v}_0 = v_0 z'$; $\bar{u}_1 = u_1 z'$; $\bar{u}_0 = u_0 z'$; &lt;br&gt; $\bar{v}_1' = v_1' z'$; $\bar{v}_0' = v_0' z'$; $\bar{u}_1' = u_1' z'$; $\bar{u}_0' = u_0' z'$; $\tilde{z} = zz'$;</td>
<td>9M</td>
</tr>
<tr>
<td>2</td>
<td>Setup system of equations for $s'' \equiv (v' - v)/s_1 u'$ mod $u$: &lt;br&gt; $m_1 = \bar{v}_1' - \bar{v}_1$; $m_2 = v_0 - \bar{v}_0'$; $m_3 = \bar{u}_1' - u_1$; $m_4 = \bar{u}_0 - u_0'$; &lt;br&gt; $w_0 = \tilde{z} m_4$; $w_1 = \bar{u}_1 m_3$; $r_0 = -\bar{u}_0 m_3$; $r_1 = w_0 + w_1$;</td>
<td>3M</td>
</tr>
<tr>
<td>3</td>
<td>Solve system of equations for $s''_0 = s'' s''_m$, $c = d''/s_1$: &lt;br&gt; $s''_0 = r_0 m_1 + r_1 m_2$; $c' = r_0 m_3 + r_1 m_4$; &lt;br&gt; $d'' = m_4 m_1 - m_2 m_3$; $d'' = d''$; &lt;br&gt; (if $d'' = 0$ branch to Cantor’s Algorithm)</td>
<td>7M</td>
</tr>
<tr>
<td>4</td>
<td>Pre-compute weighted $(1/s_1, s''_0, 1/s''_1)$ and $z''$: &lt;br&gt; $\bar{D} = d'' m_2$; $\bar{D}' = \bar{D} z$; $S_0 = s''_s z$; $S'_0 = d'' S_0$; $w_0 = d'' c'$; &lt;br&gt; $S_1 = \tilde{z} c'$; $S'_1 = w_0 z$; $S'_0 = S'_1 z$; $S_s q = S_1 c'$; $z'' = S''_1 D'$;</td>
<td>9M, 1S</td>
</tr>
<tr>
<td>5</td>
<td>Compute $u''$: &lt;br&gt; $h_1 = \tilde{z} h_1$; $u''_1 = 2 S''_0 - D m_3 + S''_1 h_2 - S_s q$; &lt;br&gt; $u''_0 = S_0 (S_0 - 2 d'' m_3) + D r_1 + S_1 (h_2 S_0 - d'' (h_2 u''_1 + h_1 + 2 v_1)) + S_s q (u_1 + u'_1 - \tilde{z} f_1)$;</td>
<td>7M, 1C</td>
</tr>
<tr>
<td>6</td>
<td>Adjust $u$ to the same weight as $u''$: &lt;br&gt; $\bar{u}_1 = D \bar{u}_1$; $\bar{u}_0 = D \bar{u}_0$;</td>
<td>2M</td>
</tr>
<tr>
<td>7</td>
<td>Compute $v'' = -h - l'$ mod $u''$: &lt;br&gt; $w_0 = (u'' - S'_0) (\bar{u}_1 - u''_1)$; $h_0 = h_0 \tilde{z}$; &lt;br&gt; $v''_1 = \tilde{z} (w_0 - S'_1 (D (\bar{v}_1 + h_1) - h_2 u''_1)) + D' (u''_0 - \bar{u}_0)$; &lt;br&gt; $v''_0 = S'_0 (w_0 - \bar{u}_0) + u''_0 (\bar{u}_1 - u''_1) - S'_1 (D' (\bar{v}_0 + h_0) - h_2 u''_0)$;</td>
<td>9M, 1C</td>
</tr>
<tr>
<td>8</td>
<td>Adjust $u''$ to the weight of $z''$: &lt;br&gt; $u''_1 = S'_1 u''_1$; $u''_0 = S_1 u''_0$.</td>
<td>2M</td>
</tr>
</tbody>
</table>

Total: 48M, 1S, 2C
Table 5.8: Even Characteristic Projective Addition Formula

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure</th>
<th>Cost</th>
</tr>
</thead>
</table>
| 1    | Adjustments to input and the weight \( \hat{z} = zz' \):  
\( \hat{v}_1 = v_1z' \); \( \hat{v}_0 = v_0z' \); \( \hat{u}_1 = u_1z' \); \( \hat{u}_0 = u_0z' \);  
\( \hat{v}'_1 = v'_1z' \); \( \hat{v}'_0 = v'_0z' \); \( \hat{u}'_1 = u'_1z' \); \( \hat{u}'_0 = u'_0z' \); \( \hat{z} = zz' \); | 9M |
| 2    | Set up system of equations for \( s'' \equiv (v' - v)/s_1u' \) mod \( u \):  
\( m_1 = \hat{v}'_1 - \hat{v}_1 \); \( m_2 = \hat{v}_0 - \hat{v}'_0 \); \( m_3 = \hat{u}'_1 - \hat{u}_1 \); \( m_4 = \hat{u}_0 - \hat{u}'_0 \);  
\( w_0 = \hat{z}m_4 \); \( w_1 = \hat{u}'_1m_3 \); \( r_0 = -\hat{u}'_0m_3 \); \( r_1 = w_0 + w_1 \); | 3M |
| 3    | Solve system of equations for \( s''_0 \equiv s''_0d''u', c = d''/s_1 \):  
\( s''_0 = r_0m_1 + r_1m_2 \); \( c' = r_0m_3 + r_1m_4 \);  
\( d'' = m_4m_1 - m_2m_3 \); \( d''' = d''\hat{z} \);  
(if \( d'' = 0 \) branch to Cantor’s Algorithm) | 7M |
| 4    | Pre-compute weighted \( (1/s_1, s''_0, 1/s''_1) \) and \( z''' \):  
\( D = d''/2 \); \( D' = D\hat{z} \); \( S_0 = s''_0\hat{z} \); \( s''_0 = d'''S_0 \); \( w_0 = d'''c' \);  
\( S_1 = \hat{z}c' \); \( S'_1 = w_0\hat{z} \); \( S''_1 = S'_1\hat{z} \); \( s_{sq} = S_1c' \); \( z''' = S''_1D' \); | 9M, 1S |
| 5    | Compute \( u'' \):  
\( \hat{h}_1 = \hat{z}\hat{h}_1 \); \( \hat{u}''_1 = S_1\hat{h}_2 - Dm_3 - s_{sq} \);  
\( \hat{u}'_1 = S''_0 + Dr_1 + S_1(h_2S_0 - d'''(h_2\hat{u}'_1 + \hat{h}_1)) + s_{sq}(\hat{u}_1 + \hat{u}'_1) \); | 5M, 1S, 1C |
| 6    | Adjust \( u \) to the same weight as \( u'' \):  
\( \hat{u}_1 = D\hat{u}'_1 \); \( \hat{u}_0 = D\hat{u}'_0 \); | 2M |
| 7    | Compute \( v'' = -h - l \) mod \( u'' \):  
\( w_0 = (u''_0 - S''_0)(\hat{u}_1 - u''_1) \); \( h_0 = h_0\hat{z} \);  
\( v''_0 = \hat{z}(w_0 - S'_1(D(\hat{v}_1 + \hat{h}_1) - h_2u''_1)) + D'(u''_0 - \hat{u}_0) \);  
\( v''_0 = S''_0(u''_0 - \hat{u}_0) + u''_0(\hat{u}_1 - u''_1) - S'_1(D'(\hat{v}_0 + \hat{h}_0) - h_2u''_0) \); | 9M, 1C |
| 8    | Adjust \( u'' \) to the weight of \( z''' \):  
\( \hat{u}'_1 = S'_1u''_1 \); \( \hat{u}'_0 = S_1u''_0 \); | 2M |
| Total | | 46M, 2S, 2C |
### Table 5.9: Odd Characteristic Projective Addition Formula

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Adjustments to input and the weight $\hat{z} = zz'$: $v_1 = v_1z'$; $v_0 = v_0z'$; $u_1 = u_1z'$; $\hat{u}_0 = u_0z'$; $v_1' = v_1'z'$; $\hat{v}_0 = v_0'z'$; $\hat{u}_0' = u_0'z'$; $\hat{z} = zz'$;</td>
<td>9M</td>
</tr>
<tr>
<td>2</td>
<td>Set up system of equations for $s'' \equiv (v' - v)/s_1 u'$ mod $u$: $m_1 = v_1' - v_1$; $m_2 = v_0 - v_0'$; $m_3 = u_1' - u_1$; $m_4 = u_0 - u_0'$; $w_0 = \hat{z} m_4$; $w_1 = \hat{u}_1 m_3$; $r_0 = -\hat{u}_0 m_3$; $r_1 = w_0 + w_1$;</td>
<td>3M</td>
</tr>
<tr>
<td>3</td>
<td>Solve system of equations for $s''_0 = s''_0 d''$, $c = d''/s_1$: $w_0 = (m_2 - r_0)(r_1 - m_1)$; $w_1 = (-r_0 - m_2)(r_1 + m_1)$; $w_2 = (m_4 - r_0)(r_1 - m_3)$; $w_3 = (-r_0 - m_4)(r_1 + m_3)$; $s''_0 = w_0 - w_1$; $c' = w_2 - w_3$; $d'' = w_2 + w_3 - w_0 - w_1 - 2(m_2 - m_4)(1 + m_3)$; $d'' = d'' \hat{z}$; (if $d'' = 0$ branch to Cantor’s Algorithm)</td>
<td>6M</td>
</tr>
<tr>
<td>4</td>
<td>Pre-compute weighted $(1/s_1$, $s''_0$, $1/s''_1$ and $z'': D = d''/D'$$ S_0 = s''_0 \hat{z}$; $S''_0 = d''/s''_0$; $w_0 = d''/c'; S_1 = \hat{z} c'$; $S''<em>1 = w_0 \hat{z}$; $S</em>{sg} = S_1 c'$; $z'' = S''_1 D'$;</td>
<td>9M, 1S</td>
</tr>
<tr>
<td>5</td>
<td>Compute $u''$: $v_1 = S_1 \hat{v}<em>1$; $u_1' = 2S_0 - D m_3 - S</em>{sg}$; $u_0'' = S_0(S_0 - 2d'' m_3) + D r_1 + 2\hat{v}<em>1 + S</em>{sg}(\hat{u}_1 + \hat{u}_1')$;</td>
<td>6M</td>
</tr>
<tr>
<td>6</td>
<td>Adjust $u$ to the same weight as $u''$: $\hat{u}_1 = D \hat{u}_1$; $\hat{u}_0 = D \hat{u}_0$;</td>
<td>2M</td>
</tr>
<tr>
<td>7</td>
<td>Compute $u'' = -h - l'$ mod $u''$: $v_0 = S_1 \hat{v}_0$; $w_0 = (u_1' - S''_0)(\hat{u}_1 - \hat{u}_1')$; $\hat{u}_1' = \hat{z}(w_0 + D(u''_0 - \hat{u}_0 - \hat{v}_1))$; $\hat{v}_0' = S_0(u''_0 - \hat{u}_0) + u''_0(\hat{u}_1 - \hat{u}_1') - D' \hat{v}_0$;</td>
<td>7M</td>
</tr>
<tr>
<td>8</td>
<td>Adjust $u''$ to the weight of $z''$: $u''_1 = S_1 u''_1$; $u'' = S_1 u''_0$.</td>
<td>2M</td>
</tr>
</tbody>
</table>

**Total:** 44M, 1S, 0C
## Table 5.10: Arbitrary Mixed Addition Formula

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Adjustments of input:</td>
<td>4M</td>
</tr>
<tr>
<td></td>
<td>( \tilde{v}'_1 = v'_1 z; \tilde{v}'_0 = v'_0 z; \tilde{u}'_1 = u'_1 z; \tilde{u}'_0 = u'_0 z; )</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Set up system of equations for ( s'' = (v' - v)/s_1 u' ) mod ( u ):</td>
<td>2M</td>
</tr>
<tr>
<td></td>
<td>( m_1 = \tilde{v}'_1 - v_1; m_2 = v_0 - \tilde{v}'_0; m_3 = \tilde{u}'_1 - u_1; m_4 = u_0 - \tilde{u}'_0; )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( r_0 = -\tilde{u}'_0 m_3; r_1 = m_4 + \tilde{u}'_1 m_3; )</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Solve system of equations for ( s''' = s'' d'' ), ( c = d''/s_1 ):</td>
<td>6M</td>
</tr>
<tr>
<td></td>
<td>( s''_0 = r_0 m_1 + r_1 m_2; c' = r_0 m_3 + r_1 m_4; d'' = m_4 m_1 - m_2 m_3; )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(if ( d'' = 0 ) branch to Cantor’s Algorithm)</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Pre-compute weighted ( (1/s_1, s''_0, 1/s''_1) ) and ( z'' ):</td>
<td>7M, 2S</td>
</tr>
<tr>
<td></td>
<td>( D = d''^2; D' = D z; S_0 = s''_0 z; S'_0 = d'' S_0; S_1 = d'' c'; )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( S''<em>1 = S_1 z; S'</em>{sq} = c'; S''<em>0 = S</em>{sq} z; z'' = S''_1 D' )</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Compute ( u'' ):</td>
<td>7M, 1C</td>
</tr>
<tr>
<td></td>
<td>( \hat{h}_1 = h_1 z; u''_1 = 2 S''_0 - D m_3 + S''<em>1 h_2 - S''</em>{sq}; )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( u''_0 = s''_0 (S_0 - 2 d'' m_3) + D r_1 + c' (h_2 S_0 - d'' (h_2 \tilde{u}'_1 + \hat{h}_1 + 2 v_1)) + )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( + S_{sq} (u_1 + \tilde{u}'_1 - z f_1); )</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Adjust ( u ) to the same weight as ( u'' ):</td>
<td>2M</td>
</tr>
<tr>
<td></td>
<td>( \hat{u}_1 = D u_1; \hat{u}_0 = D u_0; )</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Compute ( v'' = -h - l' ) mod ( u'' ):</td>
<td>9M, 1C</td>
</tr>
<tr>
<td></td>
<td>( w_0 = (u''_1 - S''_0)(\hat{u}_1 - u''_0); h_0 = h_0 z; )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( v''_1 = w_0 - S'_1 (D(v_1 + h_1) - h_2 u''_1) + D(u''_0 - \hat{u}_0); )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( v''_0 = S'_0 (u''_0 - \hat{u}_0) + u''_0 (\hat{u}_1 - u''_1) - S'_1 (D(v_0 + h_0) - h_2 u''_0); )</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Adjust ( u'' ) to the weight of ( z'' ):</td>
<td>2M</td>
</tr>
<tr>
<td></td>
<td>( u''_1 = S''_1 u''_1; u''_0 = S''_0 u''_0 )</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>39M, 2S, 2C</td>
</tr>
</tbody>
</table>
Table 5.11: Even Characteristic Mixed Addition Formula

| IN: | Reduced divisors $D = [u, v, z]$ and $D' = [u', v', 1]$ with |
|     | $u = x^2 + (u_1/z)x + (u_0/z), v = (v_1/z)x + (v_0/z)$ |
|     | $u' = x^2 + u'_1x + u'_0, \quad v' = v'_1x + v'_0$ |
|     | $h = h_2x^2 + h_1x + h_0,$ |
|     | $f = x^5 + f_4x^4 + f_3x^3 + f_2x^2 + f_1x + f_0$ |
| OUT: | Reduced divisor $D'' = [u'', v'', z''] = D + D'$ with |
|     | $u'' = x^2 + (u''_1/z'')x + (u''_0/z''), \quad v'' = (v''_1/z'')x + (v''_0/z'')$ |

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><strong>Adjustments of input:</strong></td>
<td>4M</td>
</tr>
<tr>
<td></td>
<td>$\bar{v}'_1 = v'_1z; \quad \bar{v}'_0 = v'_0z; \quad \bar{u}'_1 = u'_1z; \quad \bar{u}'_0 = u'_0z;$</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td><strong>Set up system of equations for</strong> $s'' \equiv (v' - v)/s_1u'$ mod $u$:</td>
<td>2M</td>
</tr>
<tr>
<td></td>
<td>$m_1 = \bar{v}'_1 - v_1; m_2 = v_0 - \bar{v}'_0; m_3 = \bar{u}'_1 - u_1; m_4 = u_0 - \bar{u}'_0;$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$r_0 = -\bar{u}'_0m_3; r_1 = m_4 + \bar{u}'_1m_3;$</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td><strong>Solve system of equations for</strong> $s'''_0 = s''_0d''_0, c = d''/s_1$:</td>
<td>6M</td>
</tr>
<tr>
<td></td>
<td>$s'''_0 = r_0m_1 + r_1m_2; \quad c' = r_0m_3 + r_1m_4; d'' = m_4m_1 - m_2m_3;$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(if $d'' = 0$ branch to Cantor’s Algorithm)</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td><strong>Pre-compute weighted</strong> $(1/s_1, s''_0, 1/s'_1)$ and $z'':$</td>
<td>7M, 2S</td>
</tr>
<tr>
<td></td>
<td>$D = d''_0; \quad D' = Dz; \quad S_0 = s'''_0z; \quad S'_0 = d''S_0; \quad S_1 = d''c';$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$S'<em>1 = S_1z; \quad S</em>{sq} = c'^2; \quad S''<em>0 = S</em>{sq}z; \quad z'' = S'_1D'$;</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td><strong>Compute</strong> $u''$:</td>
<td>6M, 1C</td>
</tr>
<tr>
<td></td>
<td>$h_1 = h_1z; \quad u''_1 = S'_1 h_2 - Dm_3 - S''_0;$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$u''_0 = s'''_0 S_0 + Dr_1 + c'(h_2S_0 - d''(h_2\bar{u}'<em>1 + h_1)) + S</em>{sq}(u_1 + \bar{u}'_1);$</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td><strong>Adjust</strong> $u$ to the same weight as $u''$:</td>
<td>2M</td>
</tr>
<tr>
<td></td>
<td>$\bar{u}_1 = Du_1; \quad \bar{u}_0 = Du_0;$</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td><strong>Compute</strong> $v'' = -h - l'$ mod $u''$:</td>
<td>9M, 1C</td>
</tr>
<tr>
<td></td>
<td>$w_0 = (u''_1 - S''_0)(\bar{u}_1 - u''_1); \quad h_0 = h_0z;$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$v''_1 = u_0 - S'_1(D(v_1 + h_1) - h_2u''_1) + D(u''_0 - \bar{u}_0);$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$v''_0 = S''_0(u''_0 - \bar{u}_0) + u''_0(\bar{u}_1 - u''_1) - S'_1(D(v_0 + h_0) - h_2u''_0);$</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td><strong>Adjust</strong> $u''$ to the weight of $z'':$</td>
<td>2M</td>
</tr>
<tr>
<td></td>
<td>$u''_1 = S'_1 u''_1; \quad u''_0 = S'_1 u''_0.$</td>
<td></td>
</tr>
</tbody>
</table>

Total: 38M, 2S, 2C

93
Table 5.12: Odd Characteristic Mixed Addition Formula

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Adjustments of input:</td>
<td>4M</td>
</tr>
<tr>
<td></td>
<td>$\bar{v}_1' = v_1'; \bar{v}_0' = v_0'; \bar{u}_1' = u_1'; \bar{u}_0' = u_0';$</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Set up system of equations for $s'' \equiv (v' - v)/s_1 u'$ mod $u$:</td>
<td>2M</td>
</tr>
<tr>
<td></td>
<td>$m_1 = \bar{v}_1' - v_1; m_2 = v_0 - \bar{v}_0'; m_3 = \bar{u}_1' - u_1; m_4 = u_0 - \bar{u}_0';$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$r_0 = -\bar{u}_0' m_3; r_1 = m_4 + \bar{u}_1' m_3;$</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Solve system of equations for $s''_0 = s''_0 d''; c = d''/s_1$:</td>
<td>5M</td>
</tr>
<tr>
<td></td>
<td>$w_0 = (m_2 - r_0)(r_1 - m_1); w_1 = (-r_0 - m_2)(r_1 + m_1);$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$w_2 = (m_4 - r_0)(r_1 - m_3); w_3 = (-r_0 - m_4)(r_1 + m_3);$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$s''_0 = w_0 - w_1; c' = w_2 - w_3;$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$d'' = w_2 + w_3 - w_0 - w_1 - 2(m_2 - m_4)(m_1 + m_3);$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(if $d'' = 0$ branch to Cantor’s Algorithm)</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Pre-compute weighted $(1/s_1; s''_0; 1/s''_0$ and $z'':$</td>
<td>7M, 2S</td>
</tr>
<tr>
<td></td>
<td>$D = d''; D' = Dz; S_0 = s''_0 z; S'_0 = d'' S_0; S_1 = d'' c';$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$S'<em>1 = S_1 z; S</em>{sq} = c'^2; S'<em>{sq} = S</em>{sq} z; z'' = S'_1 D'$;</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Compute $u'':$</td>
<td>6M</td>
</tr>
<tr>
<td></td>
<td>$\hat{v}_1 = S_1 v_0; \hat{u}_1' = 2S_0' - D m_3 - S'_1; \hat{v}_0'' = s''_0(S_0 - 2d'' m_3) + D r_1 + 2\hat{v}<em>1 + S</em>{sq}(u_1 + \bar{u}_1');$</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Adjust $u$ to the same weight as $u'':$</td>
<td>2M</td>
</tr>
<tr>
<td></td>
<td>$\hat{u}_1 = D u_1; \hat{u}_0 = D u_0;$</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Compute $v'' = -h - l' \text{ mod } u'':$</td>
<td>6M</td>
</tr>
<tr>
<td></td>
<td>$\hat{v}_0 = S_1 v_0; w_0 = (u_1' - S_0')(\hat{u}_1 - \hat{u}_1'); \hat{v}_1' = w_0 + D'(u_0'' - \hat{u}_0 - \hat{v}_1);$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$\hat{v}_0'' = S_0'(u_0'' - \hat{u}_0) + u_0''(u_1'' - u_1'') - D'\hat{v}_0;$</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Adjust $u''$ to the weight of $z'':$</td>
<td>2M</td>
</tr>
<tr>
<td></td>
<td>$u_1'' = S'_1 u_1'; u_0'' = S'_1 u_0';$</td>
<td></td>
</tr>
</tbody>
</table>

Total: 34M, 2S, 0C
5.2.3 Double

Here is an overview of the steps involved in the explicit formula for doubling in the projective setting. The explicit formula for projective doubling follows the same path as affine doubling.

**Algorithm 5.4 Summary of divisor class addition in the projective setting**

**Input:** \( D = [u_1, u_0, v_1, v_0, z] \) where,

- \( u = x^2 + (u_1/z)x + (u_0/z) \)
- \( v = (v_1/z)x + (v_0/z) \)
- \( f = x^5 + f_4x^4 + f_3x^3 + f_2x^2 + f_1x + f_0 \)
- \( h = h_2x^2 + h_1x + h_0 \) (\( h_2 = 1 \))

**Output:** \( D' = [u'_1, u'_0, v'_1, v'_0, z'] = 2D \) where,

- \( u' = x^2 + (u'_1/z')x + (u'_0/z') \)
- \( v' = (v'_1/z')x + (v'_0/z') \)

1: Compute \( \tilde{v} = 2v + h \mod u \)
2: Compute \( k \) where \( k_1 \) has weight \( z^2 \) and \( k_0 \) has weight \( z^3 \)
3: Set up system of equations that solves for \( s'' \equiv k/s_1 \tilde{v} \mod u \) and \( i = 1/s_1 \)
4: Solve system to get \( s_0'' \) and \( c \) with weight \( d'' \)
5: Pre-computations and compute the output weight \( z' \)
6: Compute \( u'' \) by equating coefficients
7: Adjust \( u \) to have the same weight as \( u'' \)
8: Compute \( u'' \equiv -h - l' \mod u'' \)
9: Adjust \( u'' \) to have weight \( z'' \)

with extra computations because of the weights involved.

There are a few differences between projective doubling and addition, but the techniques for converting to projective coordinates stay the same. The key difference that results in a large reduction of multiplications is that the weight of both input divisor classes is the same, so step one of addition can be skipped saving nine multiplications. The value \( \tilde{v} = 2v + h \) and \( k \) need to be computed before setting up the system of equations, \( \tilde{v} \) is computed with a weight of \( z \), \( k_1 \) with a weight of \( z^2 \) and \( k_0 \) with a weight of \( z^3 \).

After setting up the system of equations and solving (with a novel reduction of one multiplication in odd characteristic as described in (Section 4.2.8), \( d'' = d'z^4 \) has a weight of \( z^4 \), \( s''_0 = s'_0z^5d' = s_0d''z \) has a weight of \( d''z \) and \( c = (1/s_1)z^4d' = (1/s_1)d'' \) has a weight of \( d'' \) with respect to their affine values. The value \( d'' = d''z \) does not need to be computed.
because of how the weights work in the system of equations, saving another multiplication. There are similar pre-computations involved resulting in \( z' = D^2 z^3 / s_1 \) where \( D = d'^2 \), and the weights of \( u'' \) and \( v'' \) are \( z' \), the same as in addition.

The cost of projective doubling is \( 31M, 5S, 2C \) in odd characteristic and \( 36M, 4S, 2C \) in even characteristic. Refer to Table 5.13 summarizing the formulae in arbitrary case. Tables 5.14 and 5.15 summarize the formulae in even and odd characteristic cases respectively.

5.3 Alternate Representations of Divisor Classes

In this section we describe alternate representations of divisor classes that in certain cases have been shown to reduce the number of field operations required to compute additions, doubles and triples at the cost of using extra field elements to describe the divisor classes. There are two affine representations that require inversions denoted as the “semi-affine” and “geometric” settings, and a representation for the projective setting that requires three auxiliary field elements over the regular projective setting denoted as the “new-coordinates” setting. We describe all three representations starting with the semi-affine setting.

5.3.1 Semi-Affine Setting

The semi-affine setting introduced by Balamohan in [2] takes advantage of two auxiliary field elements that are used to represent a divisor class in Mumford representation in the affine setting. The polynomial \( v \) is represented with a weight \( Z \), and the value \( z = Z^2 \) is used to save a squaring, resulting in a divisor class representation requiring six field elements. This representation is beneficial when computing the \( v'' \) output polynomial in Harley’s method. Instead of adjusting parts of \( v'' \) at the end with \( s_1 \) to produce the affine representation of \( v'' \), the computation of \( s_1 \) is omitted and \( v'' \) is computed with a weight of \( s_1 \). The savings in operations from not computing \( s_1 \) and adjusting \( v'' \) is greater than the overhead of dealing with the extra weight, resulting in a reduction of field operations over all. Although no
Table 5.13: Arbitrary Projective Doubling Formula

<table>
<thead>
<tr>
<th>IN:</th>
<th>Reduced divisor $D = [u, v, z]$ with $u = x^2 + (u_1/z)x + (u_0/z), v = (v_1/z)x + (v_0/z)$ $h = h_2x^2 + h_1x + h_0$, $f = x^5 + f_4x^4 + f_3x^3 + f_2x^2 + f_1x + f_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>OUT:</td>
<td>Reduced divisor $D' = [u', v', z'] = 2D$ with $u' = x^2 + (u'_1/z')x + (u'_0/z'), v' = (v'_1/z')x + (v'_0/z')$.</td>
</tr>
<tr>
<td>Step</td>
<td>Procedure</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>1</td>
<td>Compute $\tilde{v} \equiv h + 2v \mod u$: $\tilde{h}_1 = \tilde{h}_1z; \tilde{h}_0 = \tilde{h}_0z; \tilde{v}_1 = 2v_1 + \tilde{h}_1 - h_2u_1; \tilde{v}_0 = 2v_0 + \tilde{h}_0 - h_2u_0$;</td>
</tr>
<tr>
<td>2</td>
<td>Compute $k \equiv (f - hv - v^2)/u$: $Z = z^2; w_1 = u_1^2; w_2 = Zf_3 + w_1; w_3 = z u_0; w_4 = z v_1$; $k_1 = 2(w_1 - z f_4 u_1) + w_2 - 2w_3 - h_2w_4$; $k_0 = u_1(4w_3 - w_2 - z f_4 u_1 + h_2w_4) + z(zf_2 - f_4u_0 - h_2v_0) - v_1(h_1 + v_1)$;</td>
</tr>
<tr>
<td>3</td>
<td>Set up system of equations for $s'' \equiv k/s_1\tilde{v} \mod u$: $w_0 = \tilde{v}_1z; m_1 = -k_1; m_2 = k_0; m_3 = -\tilde{v}_1; m_4 = z\tilde{v}_0$; $r_0 = w_0u_0; r_1 = \tilde{v}_1u_1 + m_4$;</td>
</tr>
<tr>
<td>4</td>
<td>Solve system for $s''_0 = s_0/s_1$ and $c = 1/s_1$: $s''_0 = r_0m_1 + r_1m_2; c' = r_0m_3 + r_1m_4; d'' = m_4m_1 - m_2m_3$; (if $d'' = 0$ branch to Cantor’s Algorithm)</td>
</tr>
<tr>
<td>5</td>
<td>Pre-computations and $z'$: $S_0 = d''s''_0^m; S_1 = cz; S'_1 = d''S'_1; S''_1 = S'<em>1z; S</em>{sq} = S_1c'$; $D = d''^2; D' = Dz; z' = D'S'_1$;</td>
</tr>
<tr>
<td>6</td>
<td>Compute $u'$: $u'_1 = 2S_0 + h_2S'<em>1 - S</em>{sq}$; $u'_0 = s''_0m^2 + S_1(h_2s''_0 - d''(h_2u_1 + 2v_1 + \tilde{h}<em>1)) + S</em>{sq}(2u_1 - zf_4)$;</td>
</tr>
<tr>
<td>7</td>
<td>Adjust $u$ to have the same weight as $u'$: $\tilde{u}_1 = Du_1; \tilde{u}_0 = D'u_0$;</td>
</tr>
<tr>
<td>8</td>
<td>Compute $v' = -h - l' \mod u'$: $w_0 = (u'_1 - S_0)(\tilde{u}_1 - u'_1); v'_1 = z(w_0 - S'_1(D(v_1 + \tilde{h}_1) - h_2u'_1)) + D'(u'_0 - \tilde{u}_0); v'_0 = S_0(u'_0 - \tilde{u}_0) + u'_0(\tilde{u}_1 - u'_1) - S'_1(D'(v_0 + \tilde{h}_0) - h_2u'_0)$;</td>
</tr>
<tr>
<td>9</td>
<td>Adjust $u'$ to have weight $z'$: $u'_1 = u'_1S''_1; u'_0 = u'_0S'_1$.</td>
</tr>
<tr>
<td>Total</td>
<td></td>
</tr>
</tbody>
</table>
Table 5.14: Even Characteristic Projective Doubling Formula

<table>
<thead>
<tr>
<th>IN: Reduced divisor $D = [u, v, z]$ with $u = x^2 + (u_1/z)x + (u_0/z), v = (v_1/z)x + (v_0/z)$ $h = h_2x^2 + h_1x + h_0,$ $f = x^5 + f_4x^4 + f_3x^3 + f_2x^2 + f_1x + f_0$</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>OUT: Reduced divisor $D' = [u', v', z'] = 2D$ with $u' = x^2 + (u'_1/z')x + (u'_0/z'), v' = (v'_1/z')x + (v'_0/z').$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Step</td>
<td>Procedure</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>1</td>
<td>Compute $\overline{v} \equiv h + 2v \mod u$: $h_1 = h_1z; h_0 = h_0z; \overline{v}_1 = h_1 - h_2u_1; \overline{v}_0 = \tilde{h}_0 - h_2u_0$;</td>
</tr>
<tr>
<td>2</td>
<td>Compute $k \equiv (f - hv - v^2)/u \mod u$: $Z = z^2; w_0 = zv_1; w_1 = u'_1; w_2 = v_0Z; k_1 = w_1 - h_2w_0; k_0 = -w_1k_1 - w_0(v_1 + \tilde{h}_1) - h_2w_2$;</td>
</tr>
<tr>
<td>3</td>
<td>Set up system of equations for $s'' \equiv k/s_1\overline{v} \mod u$: $w_0 = \overline{v}_1z; m_1 = -k_1; m_2 = k_0; m_3 = -\overline{v}_1; m_4 = z\overline{v}_0; r_0 = w_0u_0; r_1 = \overline{v}_1u_1 + m_4;$</td>
</tr>
<tr>
<td>4</td>
<td>Solve system for $s'' = s_0/s_1$ and $c = 1/s_1$: $s''_0 = r_0m_1 + r_1m_2; c' = r_0m_3 + r_1m_4; d'' = m_4m_1 - m_2m_3; (if d'' = 0 branch to Cantor’s Algorithm)$</td>
</tr>
<tr>
<td>5</td>
<td>Pre-computations and $z'$: $S_0 = d''s''_0; S_1 = cz; S'_1 = d''s'_1; S''_1 = S'<em>1z; S</em>{sq} = S_1c'; D = d''; D' = Dz; z' = D'S''_1$;</td>
</tr>
<tr>
<td>6</td>
<td>Compute $u'$: $u'_1 = h_2S'<em>1 - S</em>{sq}; u'_0 = s''_0u_0 + S_1(h_2s'' - d''(h_2u_1 + \tilde{h}_1));$</td>
</tr>
<tr>
<td>7</td>
<td>Adjust $u$ to have the same weight as $u'$: $\tilde{u}_1 = Du_1; \tilde{u}_0 = D'u_0$;</td>
</tr>
<tr>
<td>8</td>
<td>Compute $v' = -h - l' \mod u'$: $w_0 = (u'_1 - S_0)(\tilde{u}_1 - u'_1); v'_1 = z(w_0 - S'_1(D(v_1 + \tilde{h}_1) - h_2u'_1)) + D'(v_0' - \tilde{u}_0); v'_0 = S_0(u'_0 - \tilde{u}_0) + u'_0(\tilde{u}_1 - u'_1) - S'_1(D'(v_0 + \tilde{h}_0) - h_2u'_0);$</td>
</tr>
<tr>
<td>9</td>
<td>Adjust $u'$ to have weight $z'$: $u'_1 = u'_1S'_1; u'_0 = u'_0S'_1.$</td>
</tr>
<tr>
<td>Total</td>
<td>36M, 4S, 2C</td>
</tr>
</tbody>
</table>
### Table 5.15: Odd Characteristic Projective Doubling Formula

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>IN:</strong></td>
<td>Reduced divisor $D = [u, v, z]$ with $u = x^2 + (u_1/z)x + (u_0/z)$, $v = (v_1/z)x + (v_0/z)$, $h = h_2x^2 + h_1x + h_0$, $f = x^5 + f_4x^4 + f_3x^3 + f_2x^2 + f_1x + f_0$</td>
<td></td>
</tr>
<tr>
<td><strong>OUT:</strong></td>
<td>Reduced divisor $D' = [u', v', z'] = 2D$ with $u' = x^2 + (u_1'/z')x + (u_0'/z')$, $v' = (v_1'/z')x + (v_0'/z')$.</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Compute $\tilde{v} \equiv h + 2v \mod u$: $\tilde{v}_1 = 2v_1$; $\tilde{v}_0 = 2v_0$;</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>Compute $k \equiv (f - hv - v^2)/u \mod w$: $Z = z^2$; $w_0 = v_1^2$; $w_1 = u_1^2$; $w_2 = Zf_3 + w_1$; $w_3 = zw_0$; $k_1 = 2(w_1 - w_3) + w_2$; $k_0 = u_1(4w_3 - w_2) + z(Zf_2 - w_0)$;</td>
<td>3M, 3S, 2C</td>
</tr>
<tr>
<td>3</td>
<td>Set up system of equations for $s'' \equiv k/s_1 \tilde{v} \mod u$: $w_0 = zw_0$; $m_1 = -k_1$; $m_2 = k_0$; $m_3 = -\tilde{v}_1$; $m_4 = 2w_0$; $r_0 = w_3\tilde{v}_1$; $r_1 = \tilde{v}_1u_1 + m_4$;</td>
<td>3M</td>
</tr>
<tr>
<td>4</td>
<td>Solve system for $s'' = s_0/s_1$ and $c == 1/s_1$: $w_1 = (m_2 - r_0)(r_1 - m_1)$; $w_2 = (-r_0 - m_2)(r_1 + m_1)$; $w_3 = (m_4 - r_0)(r_1 - m_3)$; $w_4 = (-r_0 - m_4)(r_1 + m_3)$; $s'' = w_1 - w_2$; $c' = w_3 - w_4$; $d'' = w_3 + w_4 - w_1 - w_2 - 2(m_2 - m_4)(m_1 + m_3)$; (if $d'' = 0$ branch to Cantor’s Algorithm)</td>
<td>5M</td>
</tr>
<tr>
<td>5</td>
<td>Pre-computations and $z'$: $S_0 = d''s''_0$; $S_1 = cz$; $S'_1 = d''S_1$; $S''_1 = S'<em>1z$; $S</em>{sq} = S_1c'$; $D = d''z$; $D' = Dz$; $z' = D'S''_1$;</td>
<td>7M, 1S</td>
</tr>
<tr>
<td>6</td>
<td>Compute $u'$: $\hat{v}_1 = S'_1\hat{v}_1$; $u'<em>1 = 2S_0 - S</em>{sq}$; $u'_0 = s''_0 + 2\hat{v}<em>1 + 2S</em>{sq}\hat{u}_1$;</td>
<td>2M, 1S</td>
</tr>
<tr>
<td>7</td>
<td>Adjust $u$ to have the same weight as $u'$: $\hat{u'_1} = D\hat{u}_1$; $\hat{u'_0} = D'\hat{u}_0$;</td>
<td>2M</td>
</tr>
<tr>
<td>8</td>
<td>Compute $v' = -h - l' \mod u'$: $\tilde{v}_0 = S'_1\tilde{v}_0$; $w_0 = (u'_1 - S_0)(\tilde{u}_1 - u'_1)$; $v'_1 = zw_0 + D(u'_0 - \hat{u}_0 - \hat{v}_1)$; $v'_0 = S_0(u'_0 - \hat{u}_0) + u_0(\hat{u}_1 - u'_1) - D'\tilde{v}_0$;</td>
<td>7M</td>
</tr>
<tr>
<td>9</td>
<td>Adjust $u'$ to have weight $z'$: $u'_1 = u'_1S''_1$; $u'_0 = u'_0S'_1$.</td>
<td>2M</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>31M, 5S, 2C</td>
<td></td>
</tr>
</tbody>
</table>
savings are produced in his work over prior methods for the regular affine setting with no auxiliary coordinates, this technique produces the fastest doubling, addition and tripling formulae in the six coordinate affine setting for the odd characteristic case.

In even characteristic, the semi-affine setting falls short due to the overhead of computing with a weighted $v$ when $h \neq 0$. Instead, the six coordinate geometric setting Section (5.3.2) produces the fastest affine doubling and addition formulae for general Weierstrass form, and specialized 4 coordinate affine setting produces the fastest formulae when $h(x) = x$.

Our contributions of using the “improved solution to a system of equations” technique (Section 4.2.8) directly applies to the semi-affine setting in odd characteristic. The technique produces the exact same values as Cramer’s rule respecting weights in weighted computations, resulting in a savings of one field multiplication in for doubling and addition, and two field multiplications for tripling in both affine and projective settings.

5.3.2 Geometric Setting

The geometric setting introduced by Costello et. al. [9], uses the geometric method described in Section 3.3 to compute divisor class addition and doubling for the odd characteristic case. The authors notice that certain values computed at the end of a doubling and addition are also computed in the beginning of any subsequent operations. As a result two auxiliary elements $U_0 = u_0u_1$ and $U_1 = u_1^2$ are passed along with the regular affine representation to avoid computing the same values twice.

In odd characteristic, the six element geometric setting requires more field operations than the semi-affine setting with our contributions. In even characteristic the six element geometric setting requires the fewest number of field operations for doubling and addition due to our extension of the geometric method to work over arbitrary fields (Section 3.3). We present the even characteristic formulae in Appendix A.1.
5.3.3 New-coordinates Setting

The new-coordinate setting described by Lange in [27], requires three auxiliary elements over the regular projective setting to represent a divisor class. The regular projective setting five element formulas are adapted to the eight element versions as follows. Let $Z_1, Z_2$ be two different weights and let $z_1 = Z_1^2$ and $z_2 = Z_2^2$ be their squares. The input divisor is represented by

$$D = [u_1, u_0, v_1, v_0, Z_1, Z_2, z_1, z_2]$$

where $u = x^2 + u_1/z_1 + u_0/z_1$ and $v = v_1/Z_1^2 Z_2 + v_0/Z_1^3 Z_2$.

The new-coordinates setting is designed to minimize weight computations based on the weights acquired by different sub-expressions in the Harely method. This setting does not produce faster formulae in even characteristic due to complications by $h$ not being zero.

5.4 Comparisons

In this section we present explicit formulae cost differences between previous work and our contributions. We compare our findings to most special cases and settings that have been looked at in literature. In the table we include affine settings where the original four coordinates $[u_1, u_0, v_1, v_0]$ are the input, and settings where there are two auxiliary coordinates over the original four. There are two different six coordinate settings, the “geometric” setting introduced in [9] denoted as 6GE and the “semi-affine” setting introduced in [2] denoted as 6SA. Note that the semi-affine setting is specialized for odd characteristic and produces relatively complex formulae compared to the geometric six coordinate setting. We also include comparisons of five and eight coordinate versions of the projective setting, meaning the regular projective setting, and the “new coordinate” projective setting introduced in [27] that has three auxiliary coordinates. See Section 5.3 for a description of these settings.

In the even characteristic setting we further include the special case where $h(x) = x f_4 = f_3 = f_2 = 0$ that has been looked at by others [40]. Our contributions are not only faster in
standard affine and projective settings, but make almost all other special cases and settings faster as well. The comparisons are given in Tables 5.17 and 5.16.

5.5 Assessment for Non-adjacent Form Algorithms

Here we assess which coordinate systems are best suited for implementation of divisor double-base scalar multiplication algorithms based on Tables 5.16 and 5.17.

- In even characteristic where \( h(x) = x \), the regular 4 coordinate affine setting is best suited for implementing affine non-adjacent form scalar multiplication algorithms and the regular 5 coordinate projective setting is best suited for implementing projective non-adjacent form scalar multiplication algorithms.
Table 5.17: Comparisons of our Odd Characteristic formulae.

<table>
<thead>
<tr>
<th>Inv</th>
<th># coords</th>
<th>Conditions</th>
<th>Double</th>
<th>Addition</th>
<th>Mixed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>M</td>
<td>S</td>
<td>C</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td></td>
<td>22 5</td>
<td>-</td>
<td>22 3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This work</td>
<td>22 5</td>
<td>-</td>
<td>22 2</td>
</tr>
<tr>
<td></td>
<td>6GE</td>
<td></td>
<td>20 5</td>
<td>-</td>
<td>20 2</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>20 4</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This work</td>
<td>20 4</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td>38 6</td>
<td>2</td>
<td>47 4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This work</td>
<td>30 9</td>
<td>2</td>
<td>43 4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>32 5</td>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td>31 5</td>
<td>2</td>
<td>44 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This work</td>
<td>32 7</td>
<td>2</td>
<td>47 7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>30 7</td>
<td>2</td>
<td>-</td>
</tr>
</tbody>
</table>

The explicit formula for doubling in the affine setting when \( h(x) = x \) comes from \([39]\) and is given in Table 5.18, the affine addition, projective doubling, and projective addition formulae can be easily obtained by setting the values of \( h(x) \) accordingly in their respective tables.

- In odd characteristic, the 6SA coordinate setting is best suited for implementing affine non-adjacent form scalar multiplication algorithms and the 8 coordinate projective setting is best suited for implementing projective non-adjacent form scalar multiplication algorithms. Explicit formulae for both the 6SA setting and 8 coordinate projective setting are given in Tables 5.19, 5.20, 5.21 and 5.22.
Table 5.18: Even Characteristic Affine Doubling $h(x) = x$

**IN:** Reduced divisor $D = [u, v]$ with

$u = x^2 + u_1x + u_0$, $v = v_1x + v_0$

$h = x$, $f = x^5 + f_1x + f_0$

**OUT:** Reduced divisor $D' = [u', v'] = 2D$ with

$u' = x^2 + u'_1x + u'_0$, $v' = v'_1x + v'_0$.

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><strong>Compute resultant</strong> $r$ of $u$ and $h + 2v$: $r = u_0$; (if $r = 0$ branch to Cantor’s Algorithm)</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td><strong>Compute almost inverse</strong> $\text{inv} \equiv r/\tilde{v}$ mod $u_1$: $\text{inv}_0 = 1$; $\text{inv}_0 = u_1$</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td><strong>Compute</strong> $k \equiv (f - hv - v^2)/u$ mod $u$: $w_0 = v_1^2, w_1 = u_1^2, k_1 = w_1$; $t_1 = u_1k_1, k_0 = t_1 + w_0 + v_1$</td>
<td>1M, 2S</td>
</tr>
<tr>
<td>4</td>
<td><strong>Compute</strong> $s' \equiv \text{kinv}$ mod $u$: $t_2 = u_0k_0, s'_1 = k_0, s'_0 = (u_0 + u_1)(k_0 + k_1) + t_1 + t_2$;</td>
<td>2M</td>
</tr>
<tr>
<td>5</td>
<td><strong>Compute</strong> $s_1$ and $s_0u_1$: $t_3 = t_2^{-1}, w_3 = r^2t_3(= 1/s_1), w_4 = w_3^2(= 1/s_1^2); s_1 = s_1^2t_3(= s_1), t_6 = t_1 + k_1s_1(= s_0u_1)$</td>
<td>1I, 3M, 3S</td>
</tr>
<tr>
<td>6</td>
<td><strong>Compute</strong> $l' = su$: $l'_0 = s'_0, l'_1 = t_6 + s'_1, l'_2 = w_1, l'_3 = s_1$</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td><strong>Compute</strong> $u' = 1/s_1^2((su + h + v)^2 + f)/u^2$: $u'_1 = w_4, u'_0 = u_4k_1^2 + k_1 + w_3$;</td>
<td>1M, 1S</td>
</tr>
<tr>
<td>8</td>
<td><strong>Compute</strong> $v' \equiv h + l' + v$ mod $u'$: $t_4 = w_3, t_7 = t_4 + l'_2, t_5 = t_7u'_0$; $v'_1 = (l'_3 + t_7)(u'_0 + u'_1) + t_4 + t_5 + 1 + l'_1 + v_1$; $v'_0 = t_5 + l'_0 + v_0$</td>
<td>2M</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>I, 9M, 6S</td>
</tr>
<tr>
<td>Step</td>
<td>Procedure</td>
<td>Cost</td>
</tr>
<tr>
<td>------</td>
<td>-----------</td>
<td>------</td>
</tr>
<tr>
<td>1</td>
<td>Compute ( v' = Zv' ): ( v'_1 = v'_1Z; v'_0 = v'_0Z );</td>
<td>2M</td>
</tr>
<tr>
<td>2</td>
<td>Setup system of equations for ( s'' = x + s'_0 ) and ( c = 1/s_1 ):</td>
<td>2M</td>
</tr>
<tr>
<td></td>
<td>( m_1 = v_1 - v'_1; m_2 = v'_0 - v_0; m_3 = u'_1 - u_1 );</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( m_4 = u_0 - u'_0; r_0 = -u'_0m_3; r_1 = -u'_1m_3 - m_4 );</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Solve system of equations for ( s'' = x + s'_0 ) and ( c = 1/s_1 ):</td>
<td>11, 7M</td>
</tr>
<tr>
<td></td>
<td>( w_0 = (m_2 - r_0)(r_1 - m_1); w_1 = (r_0 - m_2)(r + m_1) );</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( w_2 = (m_4 - r_0)(r_1 - m_3); w_3 = (r_0 - m_4)(r_1 + m_3) );</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( d' = w_2 + w_3 - w_0 - w_1 - 2(m_2 - m_4)(m_1 + m_3); t = d'^{-1} );</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( s''_0 = t(w_0 - w_1); c = t(w_2 - w_3) );</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(if ( d' = 0 ) branch to Cantor’s Algorithm)</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Compute weights ( Z'', z'' ) and pre-computations:</td>
<td>3M, 1S</td>
</tr>
<tr>
<td></td>
<td>( Z'' = cZ; z'' = Z''^2 );</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( v_1 = v_1c(= v_1Z''); v_0 = v_0c(= v_0Z'') );</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Compute ( u'' ):</td>
<td>2M</td>
</tr>
<tr>
<td></td>
<td>( u''_1 = 2s''_0 - m_3 - z'' );</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( u''_0 = s''_0(s''_0 - 2m_3) - r_1 + 2v_1 + z''(u_1 + u'_1) );</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Compute ( v'' = -h - l' \bmod u'' ):</td>
<td>3M</td>
</tr>
<tr>
<td></td>
<td>( w_0 = (u''_1 - s''_0)(u_1 - u'_1) );</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( v''_1 = w_0 + u''_0 - u_0 - v_1 );</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( v''_0 = s''_0(u''_0 - u_0) + u''_0(u_1 - u''_1) - v_0 );</td>
<td></td>
</tr>
</tbody>
</table>

Total \( 1, 19M, 1S \)
### Table 5.20: Odd Characteristic 6SA Doubling Formula

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Compute $\bar{v} \equiv h + 2v \mod u$;  $ar{v}_1 = 2v_1$;  $v_0 = 2v_0$;</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>Compute $k \equiv (f - hv - v^2)/u \mod u$;  $w_0 = v_1^2$;  $w_1 = u_1^2$;  $w_2 = f_3 + w_1$;  $w_3 = 2u_0$;  $k_1 = z(2w_1 + w_2 - w_3)$;  $k_0 = z(u_1(2w_3 - w_2) + f_2) - w_0$;</td>
<td>3M, 2S</td>
</tr>
<tr>
<td>3</td>
<td>Setup system of equations for $s \equiv k/\bar{v} \mod u$;  $m_1 = -k_1$;  $m_2 = k_0$;  $m_3 = -v_1$;  $m_4 = v_0$;  $r_1 = -\bar{v}_0 + v_1u_1$;  $r_0 = \bar{v}_1u_0$;</td>
<td>2M</td>
</tr>
<tr>
<td>4</td>
<td>Solve system of equations for $s' = s'd' = s'_1x + s'_0$;  $w_0 = (m_2 - k_0)(k_1 - m_1)$;  $w_1 = -(k_0 - m_2)(k_1 + m_1)$;  $w_2 = (m_4 - k_0)(k_1 - m_3)$;  $w_3 = -(k_0 - m_4)(k_1 + m_3)$;  $d' = w_2 + w_3 - w_0 - w_1 - 2(m_2 - m_4)(m_1 + m_3); t = d'^{-1}$;  $s'_0 = t(w_0 - w_1)$;  $c = t(w_2 - w_3)$; (if $d' = 0$ branch to Cantor’s Algorithm)</td>
<td>11, 7M</td>
</tr>
<tr>
<td>5</td>
<td>Compute weights $Z'$, $z'$ and pre-computations:  $Z' = cZ$;  $z' = Z'^{m_2}$;  $v_1 = v_1c(= v_1Z')$;  $v_0 = v_0c(= v_0Z')$;</td>
<td>3M, 1S</td>
</tr>
<tr>
<td>6</td>
<td>Compute $u'$:  $u'_1 = 2s'_0 - z'; u'_0 = s'^{m_2} + 2(v_1 + z'u_1)$;</td>
<td>1M, 1S</td>
</tr>
<tr>
<td>7</td>
<td>Compute $v' = -h - t' \mod u'$:  $w_0 = (u'_1 - s''_0)(u_1 - u'_1)$;  $v'_1 = w_0 + u'_0 - u_0 - v_1$;  $v'_0 = s''_0(u'_0 - u_0) + u'_0(u_1 - u'_1) - v_0$;</td>
<td>3M</td>
</tr>
</tbody>
</table>

**Total**: 1, 19M, 4S
Table 5.21: Odd Characteristic 8 Coordinate Mixed Addition Formula

| IN: | Reduced divisors $D = [u, v, Z_1, z_1, Z_2, z_2]$ and $D' = [u', v', 1, 1, 1, 1]$ with $u = x^2 + u_1/z_1 + u_0/z_1$, $v = v_1/Z_1^3Z_2 + v_0/Z_1^3Z_2$ $u' = x^2 + u'_1x + u'_0$, $v' = v'_1x + v'_0$ $h = 0$, $f = x^5 + f_3x^4 + f_3x^3 + f_2x^2 + f_1x + f_0$ |
| OUT: | Reduced divisor $D'' = [u'', v'', z''_1, z''_2, z''_1, z''_2] = D + D'$ with $u = x^2 + u_1/z''_1 + u_0/z''_1$, $v = v_1/Z_1''Z_2'' + v_0/Z_1''Z_2''$ |
| Step | Procedure | Cost |
| 1 Adjustments of input: | | 6M |
| $w_1 = Z_1Z_2$, $w_2 = w_1z_1$, $v'_1 = v'_1w_2$, $v'_0 = v'_0w_2$, $u'_1 = u'_1z_1$, $u'_0 = u'_0z_1$ | | |
| 2 Set up system of equations for $s'' \equiv (v' - v)/s_1u'$ mod $u$: | | 2M |
| $m_1 = v_1 - v'_1$, $m_2 = v'_0 - v_0$, $m_3 = u'_1 - u_1$, $m_4 = u_0 - u'_0$, $r_0 = -u'_0m_2$, $r_1 = m_4 + u'_1m_2$ | | |
| 3 Solve system of equations for $s''_0 = s''_0d''_0$, $c' = d''/s_1$: | | 5M |
| $w_0 = (m_2 - r_0)(r_1 - m_1)$, $w_1 = (-r_0 - m_2)(r_1 + m_1)$, $w_2 = (m_4 - r_0)(r_1 - m_3)$, $w_3 = (-r_0 - m_4)(r_1 + m_3)$, $s''_0 = w_0 - w_1$, $c' = w_2 - w_3$, $d'' = w_2 + w_3 - w_0 - w_1 - 2(m_2 - m_4)(m_1 + m_3)$ (if $d'' = 0$ branch to Cantor’s Algorithm) | | |
| 4 Pre-computations and weights: | | 6M, 4S |
| $d_2 = d''z_2$, $w_4 = d''c'$, $w_5 = c'w_1$, $w_6 = w_2$, $w_7 = s''_0z_1$, $w_8 = d''w_7$, $Z''_2 = w_5Z_1$, $Z''_1 = Z_2''$, $Z''_1 = d''Z_1$, $z''_1 = Z''_1$, $d''_0$ | | |
| 5 Adjust $v$: | | 2M |
| $V_1 = v_1w_4$, $V_0 = v_0w_4$ | | |
| 6 Compute $u''$: | | 5M |
| $u''_0 = 2w_8 - d_2m_3 - z''_2$, $u''_0 = s''_0(w_7 - 2d''m_3) + d_2r_1 + 2v_1 + w_0(u_1 + u'_1)$ | | |
| 7 Adjust $u$ to the same weight as $u''$: | | 2M |
| $u_1 = d_2u_1$, $u_0 = d_2u_0$ | | |
| 8 Compute $v'' = -h - l'$ mod $u'':$ | | 5M |
| $w_9 = (u''_1 - w_8)(u_1 - u''_1)$, $v''_0 = w_9 + z''_1(u''_0 - u_0 - v_1)$, $v''_0 = w_8(u''_0 - u_0) + u''_0(u_1 - u''_1) - z''_1v_0$ | | |
| Total | | 33M, 4S, 0C |
Table 5.22: Odd Characteristic 8 Coordinate Doubling Formula

| IN: Reduced divisor $D = [u, v, Z_1, z_1, Z_2, z_2]$ with $u = x^2 + u_1/z_1 + u_0/z_1$, $v = v_1/Z_1^3 Z_2 + v_0/Z_1^3 Z_2$ |
| OUT: Reduced divisor $D' = [u', v', Z'_1, z'_1, Z'_2, z'_2] = 2D$ with $u = x^2 + u_1/z'_1 + u_0/z'_1$, $v = v_1/Z_1^3 Z_2 + v_0/Z_1^3 Z_2$ |

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Compute $\bar{v} \equiv h + 2v \mod u$: $\bar{v}_1 = 2v_1; \bar{v}_0 = 2v_0$</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>Compute $k \equiv (f - hv - v^2)/u \mod u$: $w_0 = v_1^2; w_1 = u_1^2; w_2 = z_1^2; w_3 = z_1 u_0$; $w_4 = w_2 f_3 + w_1; w_5 = 2w_3; k_1 = z_2(2w_1 + w_4 - w_5)$; $k_0 = z_2(u_1(2w_5 - w_4) + w_2 f_2 z_1) - w_0$</td>
<td>5M, 3S, 2C</td>
</tr>
<tr>
<td>3</td>
<td>Set up system of equations for $s'' \equiv k/s_1 \bar{v} \mod u$: $m_1 = -k_1; m_2 = k_0; m_3 = -\bar{v}_1; m_4 = z_1 \bar{v}_0$; $r_0 = w_3 \bar{v}_1; r_1 = \bar{v}_1 u_1 + m_4$</td>
<td>3M</td>
</tr>
<tr>
<td>4</td>
<td>Solve system for $s'' = s_0/s_1$ and $c = 1/s_1$: $w_1 = (m_2 - r_0)(r_1 - m_1); w_2 = (-r_0 - m_2)(r_1 + m_1)$; $w_3 = (m_4 - r_0)(r_1 - m_3); w_4 = (-r_0 - m_4)(r_1 + m_3)$; $s''_0 = w_1 - w_2; c' = w_3 - w_4$; $d'' = w_3 + w_4 - w_1 - w_2 - 2(m_2 - m_4)(m_1 + m_3)$; (if $d'' = 0$ branch to Cantor’s Algorithm)</td>
<td>5M</td>
</tr>
<tr>
<td>5</td>
<td>Pre-computations and weights: $w_0 = c' Z_2; Z'_1 = d'' z_1; z'_1 = Z_1^2; Z'_2 = Z_1 w_0$; $z'_2 = Z_2^2; w_7 = s'' Z'_1$</td>
<td>4M, 2S</td>
</tr>
<tr>
<td>6</td>
<td>Adjust $v$: $w_8 = d'' c'; \hat{v}_1 = w_8 v_1; \hat{v}_0 = w_8 v_0$</td>
<td>3M</td>
</tr>
<tr>
<td>7</td>
<td>Compute $u'$: $u'_1 = 2w_7 - z'_2; u'_0 = s''_0 u^2 + 2(\hat{v}_1 + u''_0 u_1)$</td>
<td>1M, 2S</td>
</tr>
<tr>
<td>8</td>
<td>Adjust $u$ to have the same weight as $u'$: $w_9 = Z'_1 d''; \tilde{u}_1 = w_9 u_1; u_0 = w_9 u_0$</td>
<td>3M</td>
</tr>
<tr>
<td>9</td>
<td>Compute $v' = -h - l' \mod u'$: $w_0 = (u'_1 - u'_1)(\tilde{u}_1 - u'_1)$; $w'_1 = w_0 + z'_1(u'_0 - \tilde{u}_0 - \hat{v}_1)$; $w'_0 = w_7(u'_0 - \tilde{u}_0) + u'_0 (\tilde{u}_1 - u'_1) - z'_1 \hat{v}_0$</td>
<td>5M</td>
</tr>
</tbody>
</table>

| Total | 29M, 7S, 2C |
Chapter 6

Divisor Tripling
In this chapter we describe our explicit formulae for tripling in the affine and projective settings. We found that a specialized double and add construction for tripling produces explicit formulae that require fewer field operations than any other to date. We also tried creating tripling formulae that use the dedicated tripling methods NUCUBE [22] and the geometric method as described in Section 3.3; we give an exposition of these methods as well.

We describe each tripling formula and then summarize each formula in a table and give a cost analysis. As in the previous chapter, when giving counts for the explicit formulae we denote a field inversion with \( I \) a field multiplication with \( M \) and a field squaring with \( S \). Recall that in even characteristic \( f_4 = f_3 = f_2 = 0, h_2 = 1 \) and in odd characteristic \( h(x) = 0 \) and \( f_4 = 0 \) as described in Section 4.1.2. We remind the reader that \( h_2 \) is left in the formulae descriptions so that specializing to curve equations where \( h_2 = 0 \) can easily be seen.

We first present our fastest formulae using the specialized double-and-add method for the affine and projective settings. We describe how our formulae adapt to alternate settings for representing divisors and give tables that compare our formulae to previous state of the art in most other settings. Finally we describe the dedicated tripling methods we tried and give a summary of their drawbacks.

### 6.1 Specialized Double and Add

The formulae presented in this section are new to this work and require fewer field operations than any other to date for the regular projective and affine settings. We introduce two inversion, one inversion and projective versions of tripling with significant improvements over using an addition after a double. In the odd characteristic setting, the only other work for tripling is due to Bamalohan in[2]. The author considers the projective setting and only an affine setting that requires two extra field elements to represent a divisor class that he
dubs the “semi-affine” setting. Not only do we produce the fastest tripling formulas for the regular affine case, we improve on his work reducing the cost of tripling by $2M$ in the projective and the semi-affine settings. In the even characteristic setting, we are the first to introduce tripling algorithms, so we compare our costs to the cost of the previous best doubling and addition formulae added together.

We first present affine double and add formulae with two inversions and one inversion, and then the projective formula. Two inversion tripling formulae can be advantageous over one inversion depending on the cost of an inversion, so we present both. To produce these formulae, we combine our doubling and addition formulae from Chapter 5 and simplify. The tripling-specific techniques we use omit the computation of $v'$ (Section 6.1.1) in the double portion and compute $u''$ modulo $u$ the equate coefficients in the add portion (Section 4.3.2).

In this work we introduce adapting the use a system of equations to solve for $s$ (Section 4.2.7) to significantly decrease the operation count increase when holding off on the first inversion in one inversion tripling (Section 4.3.1). Most noticeably in even characteristic where inversion costs are small ($10M$), the operation cost of one inversion tripling is the fastest tripling overall, even faster than settings with auxiliary elements representing the divisor class. We also extend the idea of computing $u''$ modulo $u$ then equating coefficients to arbitrary fields (Section 4.3.2), further reducing the number of operations for projective tripling in the even characteristic case.

Notice that all of our double-and-add method tripling formulae are based on the Harley method. The tripling specific techniques used to lower the complexity of our tripling algorithms only apply to the Harley method resulting in relatively faster formulae. We did not find any techniques for the geometric method other than reusing values which resulted in geometric double-and-add formulae to be less efficient.

We start with presenting the main tripling-specific technique used here, then we present two inversion tripling, one inversion tripling, and finally projective double-and-add tripling.
6.1.1 Omit Computation of $v'$

Fan and Gong introduced a technique that avoids the computation of the intermediate $v'$ polynomial when computing an addition right after another addition to emulate a double-then-add \[13\]. Bamalohan in \[2\] adapts this idea to implementing an addition right after a double to compute $3D = 2D + D$.

Consider $[u_1, u_0, v_1, v_0]$, where $u = x^2 + u_1x + u_0$ and $v = v_1x + v_0$, to be the input to our tripling algorithm, and likewise $[u'_1, u'_0, v'_1, v'_0]$ the intermediate output of the doubling portion. Note that all polynomial sub-expressions in the addition portion are marked with a tilde for readability. In the naive implementation of the Harley add and the Harley double algorithms (Algorithms 3.1 and 3.2), we have that

$$\tilde{s} \equiv \frac{v' - v}{u} \mod u'$$

and

$$v' \equiv -h - (su + v) \mod u'$$

respectively. Substituting $v'$ into $\tilde{s}$ yields,

$$\tilde{s} \equiv \frac{v' - v}{u} \equiv \frac{-su - h - 2v}{u} \equiv -s - \frac{h + 2v}{u} \mod u'.$$

Notice now that to compute $\tilde{s}$ we do not need to compute $v'$ in the double portion anymore. This shortcut saves field operations in all three versions of the double then add tripling approach.

Note that taking advantage of this simplification still permits the use of the system of equations trick to solve for $\tilde{s}$ (Section 4.2.7) in the addition portion where $\tilde{s} \equiv -s - \frac{h + 2v}{u} \mod u'$ results in

$$\hat{s} \equiv \tilde{s} + s \equiv -\frac{h + 2v}{u} \mod u'.$$

Then $\hat{s}$ is solved for using a system of equations as before, and finally $\tilde{s}$ is extracted with a subtraction by $s$. 

112
6.1.2 Two Inversion

Here we introduce our fastest two inversion formula. A two inversion formula may be useful in settings where inversions take few field operations. The produce our two inversion formula, we construct a double-then-add using the same techniques from Sections 5.1.1 and 5.1.2, but introduce the trick that omits the computation of the intermediate $v'$ in the doubling portion as described in Section 6.1.1. A summary of this method is given in Algorithm 6.1 followed by a detailed description of how each step is converted to explicit formulae.
Algorithm 6.1 Summary of two inversion divisor class tripling

**Input:** \( D = [u_1, u_0, v_1, v_0] \), where,
\[
\begin{align*}
u &= x^2 + u_1 x + u_0 \\
v &= v_1 x + v_0 \\
f &= x^5 + f_4 x^4 + f_3 x^3 + f_2 x^2 + f_1 x + f_0 \\
h &= h_2 x^2 + h_1 x + h_0
\end{align*}
\]

**Output:** \( D'' = [u_1'', u_0'', v_1'', v_0''] = 3D \) where,
\[
\begin{align*}
u'' &= x^2 + u_1'' x + u_0'' \\
v'' &= v_1'' x + v_0''
\end{align*}
\]

1: Compute \( \hat{v} = h + 2v \mod u \)
2: Compute \( k \equiv (f - hv - v^2)/u \mod u \)
3: Set up system of equations that solves for \( s \equiv k/\hat{v} \mod u \)
4: Solve system of equations for \( s' = s \cdot d' \), up to a factor of \( d' \)
5: Compute \( s'' = s/s_1 = x + s_0' \) and \( 1/s_1, 1/s_1^2, s_1 \)
6: Compute \( u' \) by equating coefficients of \( u^2 u' = \frac{v^2 + hv' - f}{v'^2} \)
7: Set up system of equations that solves for \( \hat{s} = \hat{s} + s \equiv -(2v + h)/u \mod u' \)
8: Solve system of equations for \( \hat{s} = \hat{s} + s \), adjust for and extract \( \hat{s} \), up to a factor of \( \hat{d}' \)
9: Compute \( \bar{s}' = \bar{s}/\bar{s}_1 = x + s_0'' \) and \( 1/\bar{s}_1, 1/\bar{s}_1^2, \bar{s}_1 \)
10: Compute \( u'' \) by equating coefficients of \( u u' u'' = \frac{v^2 + hv' - f}{v'^2} \)
11: Compute \( v'' \equiv -h - \hat{v}' \mod u'' \)

**Steps 1-4:** Compute \( s' = s_1' x + s_0' \)

These steps are exactly the same as for doubling in the affine case. Refer to Section 5.1.2

Steps 1–4 for a detailed explanation of these steps. The following intermediate values were computed:

\[
\hat{v}_1; \hat{v}_0; k_1; k_0; m_1; m_2; m_3; m_4; s_0'; s_1'; d'.
\]

The total cost of Steps 1-4 is \( 8M, 2S, 0C \) in odd characteristic and \( 10M, 1S, 0C \) in even characteristic.

**Step 5:** Compute \( s'' = x + s_0/s_1 \)

In this step we come across the first difference compared to the regular affine doubling.

Notice this where we first obtain savings from not having to compute \( v' \). Montgomery’s inversion trick (Section 4.2.5) is used to invert \( s_1' \) and \( d' \) at the same time. However, the
computation of $s_1$ is traded for the computation $1/d'$, saving $1S$. The value $t = 1/d'$ is needed for computing adjustments in the addition portion of the algorithm. The explicit formula is:

\[
\begin{align*}
  w_1 &= \frac{1}{d's'_1}; \\
  w_2 &= d'w_1; \\
  w_3 &= d'w_2; \\
  w_4 &= w^2_3; \\
  t &= s'_1 w_1; \\
  s''_0 &= s'_0 w_2.
\end{align*}
\]

The cost of this step is $1I, 5M, 1S, 0C$ in both characteristics.

**Step 6: Compute $u'$**

This step follows exactly from Step 6 of Section 5.1.2. The explicit formula is:

\[
\begin{align*}
  u''_1 &= 2s''_0 + h_2w_3 - w_4; \\
  u''_0 &= s''_0 + w_3(h_2(s''_0 - u_1) + 2v_1 + h_1) + w_4(2u_1 - f_4).
\end{align*}
\]

In odd characteristic the explicit formula is:

\[
\begin{align*}
  u''_1 &= 2s''_0 - w_4; \\
  u''_0 &= s''_0 + 2(w_3v_1 + w_4u_1).
\end{align*}
\]

In even characteristic the explicit formula is:

\[
\begin{align*}
  u''_1 &= h_2w_3 - w_4; \\
  u''_0 &= s''_0 + w_3(h_2(s''_0 - u_1) + h_1).
\end{align*}
\]

The cost of this step is $2M, 1S, 0C$ in odd characteristic and $1M, 1S, 0C$ in even characteristic.
**Step 7: Set up system for** \( \hat{s} \equiv \tilde{s} + s \equiv -\frac{h+2v}{u} \mod u' \)

In this step, a system to solve for

\[
\hat{s} \equiv \tilde{s} + s \equiv -\frac{h}{u} - 2v \mod u'
\]

is set up as described in Section [6.1.1](#).

Let \( \tilde{v}' = \tilde{v}'_1 x + \tilde{v}'_0 \equiv -h - 2v \mod u' \), resulting in

\[
\hat{s} \equiv \frac{\tilde{v}'}{u} \mod u'.
\]

Multiplying by \( u \) on both sides results in

\[
(x^2 + u_1 x + u_0)(\hat{s}_1 x + \hat{s}_0) = \tilde{v}'_1 x + \tilde{v}'_0 \mod x^2 + u'_1 x + u'_0,
\]

producing the linear equations

\[
\tilde{v}'_0 = \hat{s}_0(u_0 - u'_0) + \hat{s}_1(u'_0(u'_1 - u_1)),
\]
\[
\tilde{v}'_1 = \hat{s}_0(-u'_1 - u_1)) + \hat{s}_1(u_0 - u'_0 + u'_1(u'_1 - u_1)).
\]

The polynomial \( \hat{s} = \hat{s}_1 x + \hat{s}_0 \) can be obtained by solving

\[
\begin{pmatrix}
  u_0 - u'_0 & u'_0(u'_1 - u_1) \\
  -(u'_1 - u_1) & u_0 - u'_0 + u'_1(u'_1 - u_1)
\end{pmatrix}
\times
\begin{pmatrix}
  \hat{s}_0 \\
  \hat{s}_1
\end{pmatrix}
= \begin{pmatrix}
  \tilde{v}'_0 \\
  \tilde{v}'_1
\end{pmatrix}.
\]

The general and even characteristic explicit formula is:

\[
\tilde{v}'_1 = h_2 u'_1 - h_1 - 2v_1; \quad \tilde{v}'_0 = h_2 u'_0 - h_0 - 2v_0;
\]
\[
m_3 = u'_1 - u_1; \quad m_4 = u_0 - u'_0; \quad m_1 = m_4 + u'_1 m_3; \quad m_2 = u'_0 m_3.
\]

Specializing to odd characteristic the explicit formula is:

\[
\tilde{v}'_1 = -2v_1; \quad \tilde{v}'_0 = -2v_0;
\]
\[
m_3 = u'_1 - u_1; \quad m_4 = u_0 - u'_0; \quad m_1 = m_4 + u'_1 m_3; \quad m_2 = u'_0 m_3.
\]

The cost of this step is \( 2M, 0S, 0C' \) in both characteristics.
Step 8: Solve system to get $\hat{s} = \hat{s}_1 x + \hat{s}_0 = \tilde{v}'/u \mod u'$ and extract $\tilde{s}'$

The system is solved by either using Cramer’s rule in even characteristic or the “improved solution to a system of equations” in odd characteristic (Section 4.2.8), resulting in $\hat{s}' = \tilde{d}' \hat{s} = \hat{s}_1' x + \hat{s}_0'$ where $\tilde{d}' = 2\tilde{d}$ in odd characteristic and $\tilde{d}' = \tilde{d}$ in even. Notice that $\tilde{d}$ is the determinant of the linear two by two system. Recall that $s' = sd'$ was computed instead of $s$ in Step 4, and $t' = 1/d'$ was computed in Step 5 so

$$\hat{s}' = \hat{s}d' = \tilde{d}'(\hat{s} + s) = \tilde{s}' + \tilde{d}'s = \tilde{d}'\hat{s} + \tilde{d}'s't.$$  

Let $\overline{t}' = td' = \tilde{d}'/d'$ and $s = \overline{t}'s'$, resulting in

$$\tilde{s}' = \hat{s}' - s.$$

In general the explicit formula is:

$$\begin{align*}
\hat{s}_0' &= \overline{v}_0'm_1 + \overline{v}_1'm_2; & \hat{s}_1' &= \overline{v}_0'm_3 + \overline{v}_1'm_4; \\
\tilde{d}' &= m_4m_1 - m_2m_3; & \overline{t}' &= td'; \\
s_1 &= \hat{s}_1'\overline{t}; & s_0 &= \hat{s}_0'\overline{t}; \\
\tilde{s}_1' &= \hat{s}_1' - s_1; & \tilde{s}_0' &= \hat{s}_0' - s_0.
\end{align*}$$

The explicit formula in odd characteristic is:

$$\begin{align*}
w_0 &= (m_2 - \overline{v}_0')(\overline{v}_1' - m_1); & w_1 &= (-\overline{v}_0' - m_2)(\overline{v}_1' + m_1); \\
w_2 &= (m_4 - \overline{v}_0')(\overline{v}_1' - m_3); & w_3 &= (-\overline{v}_0' - m_4)(\overline{v}_1' + m_3); \\
\hat{s}_0' &= w_0 - w_1; & \hat{s}_1' &= w_2 - w_3; \\
\tilde{d}' &= w_2 + w_3 - w_0 - w_1 - 2(m_2 - m_4)(m_1 + m_3); \\
\overline{t}' &= td'; \\
s_1 &= \hat{s}_1'\overline{t}; & s_0 &= \hat{s}_0'\overline{t}; \\
\tilde{s}_1' &= \hat{s}_1' - s_1; & \tilde{s}_0' &= \hat{s}_0' - s_0.
\end{align*}$$

The cost of this step is $8M, 0S, 0C$ in odd characteristic and $9M, 0S, 0C$ in even characteristic.
Step 9: Compute $s'' = x + \frac{\tilde{s}_0}{\tilde{s}_1}$ and $\tilde{s}_1$

In this step, Montgomery’s inversion trick (Section 4.2.5) is used again to find the inverse of $\tilde{s}_1$ and $\tilde{d}$ at the same time. This allows us to adjust for the extra $\tilde{d}$ weights in the intermediate equations thus far, as well as to efficiently ensure $u''$ is monic when it is computed. The explicit formula is:

\[
\begin{align*}
  w_1 &= \frac{1}{d's'_1} = (\frac{1}{d^2\tilde{s}_1}) \\
  w_2 &= \tilde{r}w_1 = (\frac{1}{d'\tilde{s}_1}) \\
  \tilde{s}_1 &= \tilde{s}_1^2w_1 = (\frac{\tilde{s}_1}{s_1}) \\
  w_3 &= \tilde{d}'w_2 = (\frac{1}{s_1}) \\
  w_4 &= w_3^2 = (\frac{1}{s_1^2}) \\
  \tilde{z}'' &= \tilde{z}'w_2 = (\frac{\tilde{z}_0}{s_1})
\end{align*}
\]

The cost of this step is $1I, 5M, 2S, 0C$ in both characteristics.

Step 10: Compute $u''$

This step is exactly the same as described in Step 4 of Section 5.1.1. The general explicit formula is:

\[
\begin{align*}
  u''_1 &= 2\tilde{z}'' - m_3 + h_2w_3 - w_4; \\
  u''_0 &= \tilde{s}_0(\tilde{s}''_0 - 2m_3) + m_1 + w_3(h_2(\tilde{s}''_0 - u'_1) + h_1 + 2v_1) + w_4(u_1 + u'_1 - f_4).
\end{align*}
\]

The odd characteristic formula is:

\[
\begin{align*}
  u''_1 &= 2\tilde{z}'' - m_3 - w_4; \\
  u''_0 &= \tilde{s}_0(\tilde{s}''_0 - 2m_3) + m_1 + 2w_3v_1 + w_4(u_1 + u'_1).
\end{align*}
\]
The even characteristic formula is:

\[ u''_1 = m_3 + h_2 w_3 + w_4; \]

\[ u''_0 = \tilde{s}''_0 + m_1 + w_3(h_2(\tilde{s}''_0 - u'_1) + h_1) + w_4(u_1 + u'_1). \]

The cost of this step is 3M, 0S in odd characteristic and 2M, 1S in even characteristic.

**Step 11: Compute** \( v'' \equiv -h - \tilde{l} \pmod{u''} \)

This step is exactly the same as described in Step 5 of Section [5.1.1](#). The general explicit formulas is:

\[ w_0 = (u''_1 - \tilde{s}''_0)(u_1 - u''_1); \]

\[ v''_1 = \tilde{s}_1(w_0 + u''_0 - u_0) - v_1 - h_1 + h_2 u''_1; \]

\[ v''_0 = \tilde{s}_1(\tilde{s}''_0(u''_0 - u_0) + u''_0(u_1 - u''_1)) - v_0 - h_0 + h_2 u''_0. \]

Specializing into odd characteristic, the explicit formula is:

\[ w_0 = (u''_1 - \tilde{s}''_0)(u_1 - u''_1); \]

\[ v''_1 = \tilde{s}_1(w_0 + u''_0 - u_0) - v_1; \]

\[ v''_0 = \tilde{s}_1(\tilde{s}''_0(u''_0 - u_0) + u''_0(u_1 - u''_1)) - v_0. \]

The cost of this step is 5M, 0S, 0C in both characteristics.

The total cost is 2I, 38M, 6S, 0C in odd characteristic, and 2I, 39M, 6S, 0C in even characteristic. Table [6.1](#) summarizes the formulae in arbitrary case. Tables [6.2](#) and [6.3](#) summarize the formulae in even and odd characteristic cases respectively.
### Table 6.1: Arbitrary 2 Inversion Tripling Formula

<table>
<thead>
<tr>
<th>IN: Reduced divisor $D = [u,v]$ with $u = x^2 + u_1x + u_0$, $v = v_1x + v_0$, $h = h_2x^2 + h_1x + h_0$, $f = f_3x^5 + f_4x^4 + f_3x^3 + f_2x^2 + f_1x + f_0$</th>
<th>OUT: Reduced divisor $D'' = [u'',v''] = 3D$ with $u'' = x^2 + u''_1x + u''_0$, $v'' = v''_1x + v''_0$.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step</td>
<td>Procedure</td>
</tr>
<tr>
<td>1</td>
<td>Compute $\hat{v} \equiv h + 2v \mod u$: $\hat{v}_1 = 2v_1 + h_1 - h_2u_1; \hat{v}_0 = 2v_0 + h_0 - h_2u_0$</td>
</tr>
<tr>
<td>2</td>
<td>Compute $k \equiv (f - hv - v^2)/u \mod u$: $w_0 = v^2; w_1 = u^2_1; w_2 = f_3 + w_1; w_3 = 2u_0; k_1 = 2(w_1 - f_4u_1) + w_2 - w_3 - h_2v_1; k_0 = u_1(2w_3 - w_2 + f_4u_1 + h_2v_1) + f_2 - w_0 - 2f_4u_0 - h_1v_1 - h_2v_0$</td>
</tr>
<tr>
<td>3</td>
<td>Set up system of equations for $s \equiv k/\hat{v} \mod u$: $m_1 = \hat{v}_1; m_2 = -\hat{v}_0; m_3 = \hat{v}_1u_1 - \hat{v}_0; m_4 = \hat{v}_1u_0$</td>
</tr>
<tr>
<td>4</td>
<td>Solve system of equations for $s' = sd' = s'_1x + s'_0$: $s'_0 = k_0m_1 + k_1m_2; s'_1 = k_0m_3 + k_1m_4; d' = m_4m_1 - m_2m_3$ (if $d' = 0$ branch to Cantor’s Algorithm)</td>
</tr>
<tr>
<td>5</td>
<td>Compute $s'' = x + s'_0/s'_1$ and $s_1$: $w_1 = (d's'_1)^{-1}; w_2 = w_1d'; w_3 = w_2d' = (1/s_1); w_4 = u_3^2; t = w_1s'_1 = (1/d'); s''_0 = s'_0w_2$</td>
</tr>
<tr>
<td>6</td>
<td>Compute $u'$: $u'_1 = 2s''_0 + h_2w_3 - w_4; u'_0 = s''_0u_2^2 + u_3(h_2(s''_0 - u_1) + 2v_1 + h_1) + w_4(2u_1 - f_4)$</td>
</tr>
<tr>
<td>7</td>
<td>Set up system for $\hat{s} \equiv s + s \equiv -h + 2v \mod u'$: $\hat{v}'_1 = h_2u'_1 - h_1 - 2v_1; \hat{v}'_0 = h_2u'_0 - h_0 - 2v_0; m_3 = u'_1 - u_1; m_4 = u_0 - u'_0; m_1 = m_4 + u'_1m_3; m_2 = u'_0m_3$</td>
</tr>
<tr>
<td>8</td>
<td>Solve system for $\hat{s} = s_1x + \hat{s}_0 = v'/u \mod u'$: $s'_0 = \hat{v}'_0m_1 + \hat{v}'_1m_2; s'_1 = \hat{v}'_0m_3 + \hat{v}'_1m_4; d' = m_4m_1 - m_2m_3; \tilde{d}' = td'; s_1 = s'_1\tilde{d}'; s_0 = s'_0\tilde{d}'\tilde{d}; s''_1 = s'_1 - s_1; s''_0 = \hat{s}_0 - s_0$</td>
</tr>
<tr>
<td>9</td>
<td>Compute $s'' = x + s''_0/s''_1$ and $s_1$: $w_1 = (d's'_1)^{-1}; w_2 = w_1d'; w_3 = w_2d' = (1/s_1); w_4 = u_3^2; s_1 = s''_0w_1 = (s''_1); s''_0 = s''_0w_2$</td>
</tr>
<tr>
<td>10</td>
<td>Compute $u''$: $u''_1 = 2s''_0 - m_3 + h_2w_3 - w_4; u''_0 = s''_0(s''_0 - 2m_3) + m_1 + w_3(h_2(s''_0 - u'_1) + h_1 + 2v_1) + u_4(u_1 + u'_1 - f_4)$</td>
</tr>
<tr>
<td>11</td>
<td>Compute $v'' = -h - l' \mod u''$: $w_0 = (u''_1 - s''_0)(u_1 - u''_1); v''_1 = \hat{s}_1(w_0 + u''_0 - u_0) - v_1 - h_1 + h_2u''_1; v''_0 = \hat{s}_1(s''_0(u''_0 - u_0) + u''_0(u_1 - u''_1)) - v_0 - h_0 + h_2u''_0$</td>
</tr>
<tr>
<td>Total</td>
<td>2I, 42M, 6S, 0C</td>
</tr>
</tbody>
</table>
### Table 6.2: Even Characteristic 2 Inversion Tripling Formula

<table>
<thead>
<tr>
<th>IN:</th>
<th>Reduced divisor $D = [u, v]$ with $u = x^2 + u_1 x + u_0$, $v = v_1 x + v_0$, $h = h_2 x^2 + h_1 x + h_0$, $f = f_2 x^5 + f_1 x^4 + f_2 x^2 + f_1 x + f_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>OUT:</td>
<td>Reduced divisor $D'' = [u'', v''] = 3D$ with $u'' = x^2 + u''_1 x + u''_0$, $v'' = v''_1 x + v''_0$.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Compute $\hat{v} \equiv h + 2v \mod u$: $\hat{v}_1 = h_1 - h_2 u_1; \hat{v}_0 = h_0 - h_3 u_0$</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>Compute $k \equiv (f - hv - v^2)/u \mod u$: $w_1 = u_1^2; k_1 = w_1 - h_2 v_1; k_0 = -u_1 k_1 - v_1 (v_1 + h_1) - h_2 v_0$</td>
<td>2M, 1S</td>
</tr>
<tr>
<td>3</td>
<td>Set up system of equations for $s \equiv k/\hat{v} \mod u$: $m_1 = v_1; m_2 = -\hat{v}_0; m_3 = \hat{v}_1 u_1 - \hat{v}_0; m_4 = \hat{v}_1 u_0$</td>
<td>2M</td>
</tr>
<tr>
<td>4</td>
<td>Solve system of equations for $s' = sd' = s'_1 x + s'_0$: $s'_0 = k_0 m_1 + k_1 m_2; s'_1 = k_0 m_3 + k_1 m_4; d' = m_4 m_1 - m_2 m_3$; (if $d' = 0$ branch to Cantor’s Algorithm)</td>
<td>6M</td>
</tr>
<tr>
<td>5</td>
<td>Compute $s'' = x + s'_0/s'_1$ and $s_1$: $w_1 = (d's'_1)^{-1}; w_2 = w_1 d'; w_3 = w_2 d' = (1/s_1); w_4 = w_3^2; t = w_1 s'_1 = (1/d'); s'_0 = s'_0 w_2$</td>
<td>11, 5M, 1S</td>
</tr>
<tr>
<td>6</td>
<td>Compute $u'$: $u'_1 = h_2 w_3 - w_4; u'_0 = s''_0 w_3 + (h_2 (s''_0 - u_1) + h_1)$</td>
<td>1M, 1S</td>
</tr>
<tr>
<td>7</td>
<td>Set up system for $\hat{s} \equiv \hat{s} + s \equiv -u + 2v \mod u'$: $\hat{v}'_1 = h_2 u'_1 - h_1; \hat{v}'_0 = h_2 u'_0 - h_0; m_3 = u'_1 - u_1; m_4 = u_0 - u'_0; m_1 = m_4 + u'_4 m_3; m_2 = u'_0 m_3$</td>
<td>2M</td>
</tr>
<tr>
<td>8</td>
<td>Solve system for $s'' = x + s'_0/s'_1$ and $s_1$: $w_1 = (d's'_1)^{-1}; w_2 = w_1 d'; w_3 = w_2 d' = (1/s_1); w_4 = w_3^2; s_1 = s''_1 w_1 = (s_1); s''_0 = s''_0 w_2$</td>
<td>9M</td>
</tr>
<tr>
<td>9</td>
<td>Compute $u''$: $u''_1 = h_2 w_3 - w_4 - m_3; u''_0 = s''_0 w_3 + m_1 + w_3 (h_2 (s''_0 - u'_1) + h_1) + w_4 (u_1 + u'_1)$</td>
<td>11, 5M, 2S</td>
</tr>
<tr>
<td>10</td>
<td>Compute $v'' = -h - l' \mod u''$: $w_0 = (u'_1 - s''_0) (u_1 - u'_1)$; $v''_0 = \hat{s}_1 (w_0 + u''_0 - u_0) - v_1 - h_1 + h_2 u''_1$; $v''_0 = \hat{s}_1 (s''_0 (u''_0 - u_0) + u''_0 (u_1 - u'_1)) - v_0 - h_0 + h_2 u''_0$</td>
<td>5M</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>2I, 39M, 6S, 0C</td>
</tr>
</tbody>
</table>

Total 121
### Table 6.3: Odd Characteristic 2 Inversion Tripling Formula

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Compute $\tilde{v} \equiv h + 2v \mod w$: $\tilde{v}_1 = 2v_1$; $\tilde{v}_0 = 2v_0$</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>Compute $k \equiv (f - hv - v^2)/u \mod w$: $w_0 = v_1^2$; $w_1 = u_1^2$; $w_2 = f_3 + w_1$; $w_3 = 2u_0$; $k_1 = 2w_1 + w_2 - w_3$; $k_0 = u_1(2w_3 - w_2) + f_2 - w_0$</td>
<td>1M, 2S</td>
</tr>
<tr>
<td>3</td>
<td>Set up system of equations for $s \equiv k/\tilde{v} \mod w$: $m_1 = \tilde{v}_1$; $m_2 = -\tilde{v}_0$; $m_3 = \tilde{v}_1u_1 - \tilde{v}_0$; $m_4 = \tilde{v}_1u_0$</td>
<td>2M</td>
</tr>
<tr>
<td>4</td>
<td>Solve system of equations for $s' = sd' = s'\tilde{x} + s'_0$: $w_0 = (m_2 - k_0)(k_1 - m_1)$; $w_1 = (-k_0 - m_2)(k_1 + m_1)$; $w_2 = (m_4 - k_0)(k_1 - m_3)$; $w_3 = (-k_0 - m_4)(k_1 + m_3)$; $s'_0 = w_0 - w_1$; $s'_1 = w_2 - w_3$; $d' = w_2 + w_3 - w'_0 - w_1 - 2(m_2 - m_4)(m_1 + m_3)$; (if $d' = 0$ branch to Cantor’s Algorithm)</td>
<td>5M</td>
</tr>
<tr>
<td>5</td>
<td>Compute $s'' = x + s'_0/s'_1$ and $s_1$: $w_1 = d's'_1; w_2 = w_1d'; w_3 = w_2d' = (1/s_1); w_4 = w_3^2; t = w_1s'_1 = (1/d'); s''<em>0 = s'</em>{01}; w_2$</td>
<td>11, 5M, 1S</td>
</tr>
<tr>
<td>6</td>
<td>Compute $u'$: $u'_1 = 2s''_0 - w_3$; $u'_0 = s''_0 + 2(w_3v_1 + w_4u_1)$</td>
<td>2M, 1S</td>
</tr>
<tr>
<td>7</td>
<td>Set up system for $s \equiv s + s \equiv -\frac{x + x\tilde{x}}{8} \mod u'$: $\tilde{v}'_1 = -2v_1$; $\tilde{v}'_0 = -2v_0$; $m_3 = v'_1 = v'_0$; $m_4 = u_0 - u'_0$; $m_1 = m_4 + u'_1m_3$; $m_2 = u'_0m_3$</td>
<td>2M</td>
</tr>
<tr>
<td>8</td>
<td>Solve system for $s = s_1x + s_0 = \tilde{v}'/u \mod u'$: $w_0 = (m_2 - \tilde{v}'_0)(\tilde{v}'_1 - m_1)$; $w_1 = (-\tilde{v}'_0 - m_2)(\tilde{v}'_1 + m_1)$; $w_2 = (m_4 - \tilde{v}'_0)(\tilde{v}'_1 - m_3)$; $w_3 = (-\tilde{v}'_0 - m_4)(\tilde{v}'_1 + m_3)$; $s''_0 = w_0 - w_1$; $s''_1 = w_2 - w_3$; $d'' = w_2 + w_3 - w_0 - w_1 - 2(m_2 - m_4)(m_1 + m_3)$; $t'' = t\tilde{d}'; s_1 = s''_1t'; s_0 = s''_0t'w_2$; $s''_1 = s''_1 - s_1$; $s''_0 = s''_0 - s_0$</td>
<td>8M</td>
</tr>
<tr>
<td>9</td>
<td>Compute $s'' = x + s''_0/s''_1$ and $s_1$: $w_1 = (d''s'_1)^{-1}$; $w_2 = w_1d''; w_3 = w_2d'' = (1/s'_1)$; $w_4 = w_3^2; s_1 = s''_1w_1 = (s'_1); s''_0 = s''_0w_2$</td>
<td>11, 5M, 2S</td>
</tr>
<tr>
<td>10</td>
<td>Compute $u''$: $u''_1 = 2s''_0 - m_3 - w_4$; $u''_0 = s''_0(s''_0 - 2m_3) + m_1 + 2w_3v_1 + w_4(u_1 + u'_1)$</td>
<td>3M</td>
</tr>
<tr>
<td>11</td>
<td>Compute $v'' = -h - l' \mod u''$: $w_0 = (u'_1 - s''_0)(u_1 - u'_1)$; $v''_1 = s_1(w_0 + u''_0 - u_0) - v_1$; $v''_0 = s_1(s''_0(u''_0 - u_0) + u''_0(u_1 - u''_0)) - v_0$</td>
<td>5M</td>
</tr>
</tbody>
</table>

Total: 21, 38M, 6S, 0C
6.1.3 One Inversion

In this section we present our fastest one inversion formula. To produce the one inversion formula, we use the same techniques from Section 6.1.2 but instead of setting up a system to solve for \( s \) we set up a system to solve for monic \( s'' \) i.e., \( s''_0 = s_0/s_1 \) and \( c = 1/s_1 \) to reduce the adjustments needed when computing the addition portion (Section 4.3.1). We omit the first inversion, producing a weight of \( d^2 \) on \( u' \), then when solving the second system of equation, we adjust accordingly. A summary of this method is given in Algorithm 6.2, followed by a detailed description of how each step is converted to explicit formulae.

Algorithm 6.2 Summary of one inversion divisor class tripling

**Input:** \( D = [u_1, u_0, v_1, v_0] \), where,
\[
\begin{align*}
    u &= x^2 + u_1x + u_0 \\
    v &= v_1x + v_0 \\
    f &= x^5 + f_4x^4 + f_3x^3 + f_2x^2 + f_1x + f_0 \\
    h &= h_2x^2 + h_1x + h_0
\end{align*}
\]

**Output:** \( D'' = [u''_1, u''_0, v''_1, v''_0] = 3D \) where,
\[
\begin{align*}
    u'' &= x^2 + u''_1x + u''_0 \\
    v'' &= v''_1x + v''_0
\end{align*}
\]

1: Compute \( \hat{v} = h + 2v \mod u \)
2: Compute \( k \equiv (f - hv - v^2)/u \mod u \)
3: Set up system of equations that solves for \( s'' \equiv k/s_1 \hat{v} \mod u \)
4: Solve system of equations for \( s''_0 = s_0/s_1 \) and \( c = 1/s_1 \) up to a factor of \( d' \)
5: Compute \( u' \) with weight \( d'^2 \) by equating coefficients of \( uu'u' = \frac{\tilde{r}^2 + h\tilde{r} - f}{\tilde{r}_3^2} \)
6: Pre-computations and weighted \( \tilde{v}' = -(2v + h) \mod u' \)
7: Set up system of equations that solves for \( s'' = \tilde{s} + s \equiv -(2v + h)/u \mod u' \)
8: Solve system of for \( s'''' = \tilde{s}'' + s'' \) up to a factor of \( d'' \), adjust, and extract \( \tilde{s}'''' \)
9: Compute \( 1/\tilde{s}_1, 1/\tilde{s}_1', \) and \( \tilde{s}_1 \)
10: Compute \( u'' \) by equating coefficients of \( uu'u'' = \frac{\tilde{r}^2 + h\tilde{r} - f}{\tilde{r}_3^2} \)
11: Compute \( v'' \equiv -h - \tilde{v} \mod u'' \)

We trade 3\( M \), 1\( S \) for one inversion as compared to the two inversion formula (Section 6.1.2). We begin with the doubling portion of the formula.
Steps 1-2: Compute \( \tilde{v} \) and \( k \)

These steps are exactly the same as for doubling in the affine case. Refer to Section 5.1.2, Steps 1–2 for a detailed explanation of these steps. The following intermediate values

\[ \tilde{v}_1; \tilde{v}_0; k_1; k_0, \]

are computed. The total cost of Steps 1-2 is \( 1M, 2S, 0C \) in odd characteristic and \( 2M, 1S, 0C \) in even characteristic.

Step 3: Set up system of equations to solve for \( s'' = k/s_1 \tilde{v} \mod u \)

In this step, for \( c = 1/s_1 \), a system of equations is set up for the equality

\[ s'' \equiv \frac{ck}{\tilde{v}} \mod u, \]

as described in Section 4.3.1. Multiplying both sides by \( \tilde{v} \) results in

\[ c(k_1 x + k_0) = (x + s''_0)(\hat{v}_1 x + \hat{v}_0) = \hat{v}_1 x^2 + (\hat{v}_0 + s''_0 \hat{v}_1)x + s''_0 \hat{v}_0 \mod x^2 + u_1 x + u_0. \]

Taking \( \hat{v}_1 x^2 + (\hat{v}_0 + s''_0 \hat{v}_1)x + s''_0 \hat{v}_0 \mod x^2 + u_1 x + u_0 \) results in

\[ ck_1 x + ck_0 = (\hat{v}_0 - \hat{v}_1 u_1 + s''_0 (\hat{v}_1))x + (\hat{v}_1 u_0 + s_0 (\hat{v}_0)), \]

and the values \( s''_0, c \) can be computed by solving

\[ \hat{v}_1 u_0 = s''_0 (\hat{v}_0) + c(-k_0), \]

\[ \hat{v}_1 u_1 - \hat{v}_0 = s''_0 (\hat{v}_1) + c(-k_1), \]

from

\[
\begin{pmatrix}
\hat{v}_0 & -k_0 \\
\hat{v}_1 & -k_1
\end{pmatrix}
\begin{pmatrix}
s''_0 \\
c
\end{pmatrix}
=
\begin{pmatrix}
\hat{v}_1 u_0 \\
\hat{v}_1 u_1 - \hat{v}_0
\end{pmatrix}.
\]

The explicit formula for setting up the linear system is

\[ m_1 = -k_1; \quad m_2 = k_0; \quad m_3 = -\hat{v}_1; \quad m_4 = \hat{v}_0; \]

\[ r_0 = \hat{v}_1 u_0; \quad r_1 = \hat{v}_1 u_1 - \hat{v}_0. \]

The cost of step one is \( 2M, 0S, 0C \) in both characteristics.
Step 4: Solve system to get \( s''_0 = s_0/s_1 \) and \( c = 1/s_1 \)

This step is almost identical to Step 2 of affine addition (Section 5.1.1). The only difference is that \( \tilde{v}_i \) is replaced with \( r_i \). Both \( c \) and \( s''_0 \) result in having a weight of \( d' \). In general the explicit formula is:

\[
\begin{align*}
s''_0 &= r_0m_1 + r_1m_2; \quad c = r_0m_3 + r_1m_4; \\
d' &= m_4m_1 - m_2m_3.
\end{align*}
\]

In odd characteristic the explicit formula is:

\[
\begin{align*}
w_0 &= (m_2 - r_0)(r_1 - m_1); \quad w_1 = (-r_0 - m_2)(r_1 + m_1); \\
w_2 &= (m_4 - r_0)(r_1 - m_3); \quad w_3 = (-r_0 - m_4)(r_1 + m_3); \\
s''_0 &= w_0 - w_1; \quad c = w_2 - w_3; \\
d' &= w_2 + w_3 - w_0 - w_1 - 2(m_2 - m_4)(m_1 + m_3).
\end{align*}
\]

The cost is \( 5M, 0S, 0C \) in odd characteristic and \( 6M, 0S, 0C \) in even characteristic.

Step 5: Compute \( u' \)

The polynomial \( u' \) is computed in the same way as in Step 7 of Section 5.1.2 but \( c = d'/s_1 \) and \( s''_0 = d's_0/s_1 \). There are terms in \( u_1 \) and \( u_0 \) that need to be multiplied by \( 1/s_1^2 \), so all of \( u'' \) is computed with a weight of \( d'2 \). The explicit formula is:

\[
\begin{align*}
c_2 &= c^2; \\
u''_1 &= d'(2s''_0 + h_2c) - c_2; \\
u''_0 &= s''_0 + c(h_2s''_0 - d'(h_2u_1 + 2v_1 + h_1)) + c_2(2u_1 - f_4).
\end{align*}
\]

In odd characteristic the explicit formula is:

\[
\begin{align*}
c_2 &= c^2; \\
u''_1 &= 2d's''_0 - c_2; \\
u''_0 &= s''_0 + 2(d'cv_1 + c_2u_1).
\end{align*}
\]
In even characteristic the explicit formula is:

\[ c_2 = c^2; \]
\[ u''_1 = d'h_2c - c_2; \]
\[ u''_0 = s''_0 + c(h_2s''_0 - d'(h_2u_1 + h_1)). \]

The cost of this step is \(4M, 2S, 0C\) in odd characteristic and \(3M, 2S, 0C\) in even characteristic.

**Step 6: Pre-computations and weighted \( \tilde{v}' = -(2v + h) \mod u' \)**

The coefficients of \( u \) are adjusted to have the same weight \( D' = d^2 \) as \( u' \). The coefficients of \( \tilde{v}' = -2v - h \mod u' \) are also computed with a weight of \( D' \). The general explicit formula is:

\[ D' = d^2 \]
\[ \hat{u}_1 = u_1D'; \quad \hat{u}_0 = u_0D'; \]
\[ \tilde{v}'_1 = h_2u'_1 - D'(h_1 - 2v_1); \quad \tilde{v}'_0 = h_2u'_0 - D'(h_0 - 2v_0). \]

In odd characteristic the explicit formula is,

\[ D' = d^2 \]
\[ \hat{u}_1 = u_1D'; \quad \hat{u}_0 = u_0D'; \]
\[ \tilde{v}'_1 = -2D'v_1; \quad \tilde{v}'_0 = -2D'v_0. \]

In even characteristic, the explicit formula is,

\[ D' = d^2 \]
\[ \hat{u}_1 = u_1D'; \quad \hat{u}_0 = u_0D'; \]
\[ \tilde{v}'_1 = h_2u'_1 - D'h_1; \quad \tilde{v}'_0 = h_2u'_0 - D'h_0. \]

The cost of this step is \(4M, 1S, 0C\) in odd characteristic and \(2M, 1S, 2C\) in even characteristic.
Step 7: Set up the system for \( \hat{s}'' \equiv \bar{s}'' + s'' \) and \( \hat{z} \equiv \bar{z} + i \)

In this step, instead of computing \( \hat{s} \), \( \hat{s}''_0 = \hat{s}_0 / \hat{s}_1 \) and \( \hat{c} = 1 / \hat{s}_1 \) are computed directly because \( c = 1 / s_1 \) and \( s''_0 \) are available from the doubling portion, resulting in an easy extraction of \( \bar{s}''_0 \) and \( 1 / \bar{c} \).

Let \( \bar{v}' = \bar{v}'_1 x + \bar{v}'_0 \equiv -h - 2v \mod u' \), then from

\[
\hat{s}'' \equiv \frac{\hat{c} \bar{v}'}{u'} \mod u'
\]

and adjusting for the extra weight of \( D' \) results in the system,

\[
\begin{pmatrix}
\hat{u}_0 - u'_0 & -\bar{v}'_0 \\
\hat{u}_1 - u'_1 & -\bar{v}'_0
\end{pmatrix}
\times
\begin{pmatrix}
\hat{s}''_0 \\
\hat{c}
\end{pmatrix}
= \begin{pmatrix}
u'_0 (\hat{u}_1 - u'_1) \\
-D' (\hat{u}_0 - u'_0) + u'_1 (\hat{u}_1 - u'_1)
\end{pmatrix}.
\]

The general explicit formula for this step is:

\[
m_1 = \bar{v}'_0; \quad m_2 = \bar{v}'_1; \quad m_3 = u'_1 - \hat{u}_1; \quad m_4 = \hat{u}_0 - u'_0; \\
r_1 = -D'm_4 - u'_1 m_3; \quad r_0 = -u'_0 m_3.
\]

The cost of this step is \( 3M, 0S, 0C \) in both characteristics.

Step 8: Solve system for \( \hat{s}''_0 \) and \( \hat{c} = 1 / \hat{s}_1 \) and extract \( \bar{s}''_0 \) and \( \bar{c} \)

The system is solved using either Cramer’s rule in even characteristic or the “improved solution to a system of equations” in odd characteristic (Section 4.2.8), resulting in \( \hat{s}''_0 = D' \hat{d} \hat{s}_0 / \hat{s}_1 \) and \( \hat{c} = D' \hat{d}' / \hat{s}_1 \) where \( \hat{d}' \) has a weight of \( D'^2 \). Recall that \( s''_0 = s''_0 \hat{d}' \) and \( c = \hat{d}' / s_1 \) instead of \( s''_0 \) and \( 1 / s_1 \) in Step 4. Thus,

\[
\hat{s}''_0 = \hat{s}''_0 \hat{d}' D' = \hat{d}' D' (\hat{s}''_0 + s''_0) = \hat{d}' D' \hat{s}''_0 + \hat{d}' D' s''_0 = \hat{d}' D' \hat{s}_0 + \hat{d}' D' s''_0.
\]

Let \( \bar{\hat{d}} = \hat{d}' D' \) and \( \bar{d}'' = \bar{\hat{d}} \hat{d}' \), resulting in

\[
\bar{s}''_0 = \bar{s}''_0 - \bar{d}'' \bar{s}''_0
\]
with a weight of $\tilde{d}'$. The value $\tilde{c}$ is found exactly in the same way. In general the explicit formula is:

\[ \hat{s}'''_0 = r_0 m_1 + r_1 m_2; \quad \hat{c} = r_0 m_3 + r_1 m_4; \]
\[ \hat{d}' = m_4 m_1 - m_2 m_3; \quad \hat{d}'' = \hat{d}' \hat{d}'; \quad \hat{d}' = \hat{d} D'; \]
\[ \tilde{s}'''_0 = \hat{s}'''_0 - \hat{s}'''_0 d''; \quad \tilde{c} = \hat{c} - c \hat{d}''; \]

The explicit formula in odd characteristic is:

\[ w_0 = (m_2 - r_0)(r_1 - m_1); \quad w_1 = (-r_0 - m_2)(r_1 + m_1); \]
\[ w_2 = (m_4 - r_0)(r_1 - m_3); \quad w_3 = (-r_0 - m_4)(r_1 + m_3); \]
\[ \hat{s}'''_0 = w_0 - w_1; \quad \hat{c} = w_2 - w_3; \]
\[ \tilde{d}' = w_2 + w_3 - w_0 - w_1 - 2(m_2 - m_4)(m_1 + m_3); \]
\[ \hat{d}'' = \hat{d}' \hat{d}'; \quad \hat{d}' = \hat{d} D'; \]
\[ \tilde{s}'''_0 = \hat{s}'''_0 - \hat{s}'''_0 d''; \quad \tilde{c} = \hat{c} - c \hat{d}''; \]

The cost of this step is $9M, 0S, 0C$ in odd characteristic and $10M, 0S, 0C$ in even characteristic.
Step 9: Compute \( \tilde{s}'' = x + \frac{\tilde{s}_0}{\tilde{s}_1} \) and \( \tilde{s}_1 \)

The inverse of \( \tilde{c} \) and \( \tilde{d}' \) is found at the same time using Montgomery’s inversion trick (Section 4.2.5). Then the values \( \tilde{s}_1, 1/\tilde{s}_1, 1/\tilde{s}_1^2 \) and \( \tilde{s}''_0 \) are computed. The explicit formula is:

\[
\begin{align*}
    w_1 &= \frac{1}{\tilde{d}' \tilde{c}}; & \left( \frac{\tilde{s}_1}{\tilde{d}^2} \right) \\
    w_2 &= \tilde{c} w_1; & \left( \frac{1}{\tilde{d}'} \right) \\
    \tilde{s}_1 &= \tilde{d}^2 w_1; \\
    w_3 &= \tilde{c} w_2; & \left( \frac{1}{\tilde{s}_1} \right) \\
    w_4 &= w_3^2; & \left( \frac{1}{\tilde{s}_1^2} \right) \\
    \tilde{s}''_0 &= \tilde{s}_0'' w_2.
\end{align*}
\]

The cost of this step is \(1I, 5M, 2S, 0C\) in both characteristics.

Step 10: Compute \( u'' \)

In this step, \( u'' \) is computed almost in the same was as Step 4 of Section 5.1.1. Notice though that the value \( r_1 \) computed in Step 7 cannot be reused as done in Step 4 of Section 5.1.1 because \( r_1 \) has a weight of \( D^2 \). The general explicit formula is:

\[
\begin{align*}
    u''_1 &= 2\tilde{s}_0'' - m_3 + h_2 w_3 - w_4; \\
    u''_0 &= \tilde{s}_0'' (\tilde{s}_0'' - 2m_3) + u_0 - u'_0 + u_1' (u'_1 - u_1) + w_3 (h_2 (\tilde{s}_0'' - u'_1) + h_1 + 2v_1) + w_4 (u_1 + u'_1 - f_4). \\
\end{align*}
\]

The odd characteristic formula is:

\[
\begin{align*}
    u'_1 &= 2\tilde{s}_0'' - m_3 - w_4; \\
    u'_0 &= \tilde{s}_0'' (\tilde{s}_0'' - 2m_3) + u_0 - u'_0 + u_1' (u'_1 - u_1) + 2w_3 v_1 + w_4 (u_1 + u'_1). \\
\end{align*}
\]

The even characteristic formula is:

\[
\begin{align*}
    u''_1 &= m_3 + h_2 w_3 + w_4; \\
    u''_0 &= \tilde{s}_0'' + u_0 - u'_0 + u_1' (u'_1 - u_1) + w_3 (h_2 (\tilde{s}_0'' - u'_1) + h_1) + w_4 (u_1 + u'_1). \\
\end{align*}
\]
The cost of this step is 4M, 0S, 0C in odd characteristic and 3M, 1S, 0C in even characteristic.

**Step 11: Compute** $v'' \equiv -h - \bar{l}' \pmod{u''}$

This step is exactly the same as described in Step 5 of Section 5.1.1. The general explicit formula is:

$w_0 = (u''_1 - \tilde{s}'_0)(u_1 - u''_1)$;

$v''_1 = \tilde{s}_1(w_0 + u''_0 - u_0) - v_1 - h_1 + h_2 u''_1$;

$v''_0 = \tilde{s}_1(\tilde{s}''_0(u''_0 - u_0) + u''_0(u_1 - u''_1)) - v_0 - h_0 + h_2 u''_0$.

Specializing to odd characteristic, the explicit formula is:

$w_0 = (u''_1 - \tilde{s}'_0)(u_1 - u''_1)$;

$v''_1 = \tilde{s}_1(w_0 + u''_0 - u_0) - v_1$;

$v''_0 = \tilde{s}_1(\tilde{s}''_0(u''_0 - u_0) + u''_0(u_1 - u''_1)) - v_0$.

The cost of this step is 5M, 0S, 0C in both characteristics.

The total cost is 1I, 42M, 7S in odd characteristic, and 1I, 41M, 7S, 2C in even characteristic. Refer to Table 6.4 summarizing the formulae in arbitrary case. Tables 6.5 and 6.6 summarize the formulae in even and odd characteristic cases respectively.
### Table 6.4: Arbitrary 1 Inversion Tripling Formula

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure</th>
<th>Cost</th>
</tr>
</thead>
</table>
| IN:  | Reduced divisor $D = [u, v]$ with  
$u = x^2 + u_1x + u_0$,  
v = v_1x + v_0$,  
h = h_2x^2 + h_1x + h_0$,  
f = f_5x^5 + f_4x^4 + f_3x^3 + f_2x^2 + f_1x + f_0 |  |
| OUT: | Reduced divisor $D'' = [u'', v''] = 3D$ with  
u'' = x^2 + u''x + u_0$,  
v'' = v''x + v_0$. |  |
| 1    | Compute $\hat{v} \equiv h + 2v \text{ mod } u$:  
$\hat{v}_1 = 2v_1 + h_1 - h_2u_1$;  
$\hat{v}_0 = 2v_0 + h_0 - h_2u_0$; | - |
| 2    | Compute $k \equiv (f - hv - v^2)/u \text{ mod } u$:  
w_0 = v_0^2$;  
w_1 = u_1^2$;  
w_2 = f_3 + w_1$;  
w_3 = 2u_0$;  
k_1 = 2(w_1 - f_4u_1) + w_2 - w_3 + h_2v_1$;  
k_0 = u_1(2w_3 - w_2 + f_4u_1 + h_2v_1) +  
+ f_2 - w_0 - 2f_4u_0 - h_1v_1 - h_2v_2$; | 3M, 2S |
| 3    | Set up system of equations for $s'' = k/s_1\hat{v} \text{ mod } u$:  
m_1 = -k_1;  
m_2 = k_0;  
m_3 = -\hat{v}_1;  
m_4 = \hat{v}_0$;  
r_0 = \hat{v}_1u_0;  
r_1 = \hat{v}_1u_1 - \hat{v}_0$; | 2M |
| 4    | Solve system to get $s'''_0 = s_0/s_1$ and $c = 1/s_1$:  
s'''_0 = r_0m_1 + r_1m_2;  
c = r_0m_3 + r_1m_4;  
d' = m_4m_1 - m_2m_3$;  
(if $d' = 0$ branch to Cantor’s Algorithm) | 6M |
| 5    | Compute $u'$:  
c_2 = c^2$;  
u''_1 = d'(2s'''_0 + h_2c) - c_2$;  
u''_0 = s'''_0^2 + (h_2s'''_0 - d'(h_2u_1 + 2v_1 + h_1)) + c_2(2u_1 - f_4)$; | 4M, 2S |
| 6    | Pre-computations and weighted $\tilde{v}' = -(2v + h) \text{ mod } u'$:  
$D' = d'^2$;  
u_1 = u_1D'$;  
u_0 = u_0D'$;  
$\tilde{v}'_0 = h_2u'_1 - D'(h_1 - 2v_1)$;  
$\tilde{v}'_1v_0 = h_2u'_0 - D'(h_0 - 2v_0)$; | 4M, 1S |
| 7    | Set up system for $s'''' \equiv s'' + s'''$ and $\hat{c} \equiv \hat{c} + c$:  
m_1 = \tilde{v}_0;  
m_2 = \tilde{v}_1;  
m_3 = u'_1 - \tilde{u}_1;  
m_4 = \tilde{u}_0 - u_0$;  
r_1 = -D'm_4 - u'_1m_3;  
r_0 = -u'_0m_3$; | 3M |
| 8    | Solve system for $s'''_0$ and $\hat{c} \equiv 1/s_1$ and extract $\tilde{s}'''_0$ and $\hat{c}$:  
$s'''_0 = r_0m_1 + r_1m_2;  
\hat{c} = r_0m_3 + r_1m_4;  
d' = m_4m_1 - m_2m_3$;  
d'' = d'd';  
$\tilde{s}'''_0 = s'''_0 - s'''_0D';  
\hat{c} = \hat{c} - cd''$; | 10M |
| 9    | Compute $\tilde{s}'''_1 \equiv x + \frac{s_0}{s_1}$ and $\tilde{s}'_1$:  
w_1 = (d'\hat{c})^{-1}$;  
w_2 = c_1w_1;  
\tilde{s}'_1 = d''w_2$;  
w_3 = c_2w_1;  
w_4 = w_3^2$;  
$s'''w_3$;  
$s'''_0 = s''w_3$; | 11, 5M, 2S |
| 10   | Compute $u''$:  
u''_1 = 2s'''_0 - m_3 + h_2w_3 - w_4;  
u''_0 = s'''_0(s'''_0 - 2m_3) + w_0 - u_0' + u'_1(u'_1 - u_1) +  
+ w_0h_2(s'''_0 - u'_1) + h_1 + 2v_1 + w_1(u_1 + u'_1 - f_4)$; | 4M |
| 11   | Compute $u'' = -h - l' \text{ mod } u'$:  
w_0 = (u_1' - s'''_0)(u_1 - u_1')$;  
v''_1 = \tilde{s}_1(w_0 + u_0' - u_0) - v_1 - h_1 + h_2u_0'$;  
v''_0 = \tilde{s}_1(s'''_0(u'' - u_0) + u''_0(u_1 - u_1')) - v_0 - h_0 + h_2u_0'$; | 5M |
| Total | I, 46M, 7S, 0C |  |
Table 6.5: Even Characteristic 1 Inversion Tripling Formula

| IN: Reduced divisor $D = [u, v]$ with $u = x^2 + u_1x + u_0$, $v = v_1x + v_0$, $h = h_2x^2 + h_1x + h_0$, $f = f_5x^5 + f_4x^4 + f_3x^3 + f_2x^2 + f_1x + f_0$ |  |
| OUT: Reduced divisor $D'' = [u'', v''] = 3D$ with $u'' = x^2 + u''_1x + u''_0$, $v'' = v''_1x + v''_0$. |  |

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Compute $\hat{v} \equiv h + 2v \bmod u$; $\hat{v}_1 = h_1 - h_2u_1$; $\hat{v}_0 = h_0 - h_2u_0$</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>Compute $k \equiv (f - hv - v^2)/u \bmod u$; $w_1 = u_1^2$; $k_1 = w_1 - h_2v_1$; $k_0 = -u_1k_1 - v_1(v_1 + h_1) - h_2v_0$</td>
<td>2M, 1S</td>
</tr>
<tr>
<td>3</td>
<td>Set up system of equations for $s'' = k/s_1\hat{v}$ mod $u$; $m_1 = -k_1$; $m_2 = k_0$; $m_3 = -\hat{v}_1$; $m_4 = \hat{v}_0$; $r_0 = \hat{v}_1u_0$; $r_1 = \hat{v}_1u_1 - \hat{v}_0$;</td>
<td>2M</td>
</tr>
<tr>
<td>4</td>
<td>Solve system to get $s''_0 = s_0/s_1$ and $c = 1/s_1$; $s''_m = r_1m_3 + r_1m_4$; $c = r_0m_3 + r_1m_4$; $d'' = m_4m_1 - m_2m_3$; (if $d'' = 0$ branch to Cantor’s Algorithm)</td>
<td>6M</td>
</tr>
<tr>
<td>5</td>
<td>Compute $u'$; $c_2 = c_1^2$; $u'_1 = d'h_2c - c_2$; $u''_m = s''_m^2 + c(h_2s''_0 - d'(h_2u_1 + h_1))$</td>
<td>3M, 2S</td>
</tr>
<tr>
<td>6</td>
<td>Pre-computations and weighted $\hat{v}' = -(2v + h) \bmod u'$</td>
<td>2M, 1S, 2C</td>
</tr>
<tr>
<td>7</td>
<td>Set up system for $\hat{\hat{s}''} \equiv s'' + s''$ and $\hat{c} \equiv \hat{c} + c$; $m_1 = \hat{\hat{v}}_0$; $m_2 = \hat{\hat{v}}_1$; $m_3 = \hat{u}_1 - \hat{u}_1$; $m_4 = \hat{u}_0 - \hat{u}_0$; $r_1 = -D'm_4 - u'_1m_3$; $r_0 = -u'_0m_3$;</td>
<td>3M</td>
</tr>
<tr>
<td>8</td>
<td>Solve system for $\hat{s}_0''$ and $\hat{c} = 1/s_1$ and extract $\hat{s}_0''$ and $\hat{c}$; $\hat{s}_0'' = r_0m_1 + r_1m_2$; $\hat{c} = r_0m_3 + r_1m_4$; $d'' = m_4m_1 - m_2m_3$; $d'' = d'd'$; $\hat{\hat{d}} = \hat{d}'\hat{d}'$; $\hat{s}_0'' = \hat{s}_0'' - \hat{s}_0''d''$; $\hat{c} = \hat{c} - c\hat{d}''$;</td>
<td>10M</td>
</tr>
<tr>
<td>9</td>
<td>Compute $\hat{s}_0'' = x + s_0''/s_1$ and $\hat{s}_1$; $w_1 = (d'c)^{-1}$; $w_2 = \hat{c}w_1$; $\hat{s}_1 = \hat{d}^2w_1$; $w_3 = \hat{c}w_2$; $w_4 = w_3^2$; $s_0'' = s_0'''w_4$;</td>
<td>2I, 5M, 2S</td>
</tr>
<tr>
<td>10</td>
<td>Compute $u''$; $u_1'' = h_2w_3 - w_4 - m_3$; $u_0'' = s_0''^2 + u_0 - u_0' + u_1'(u_1' - u_1) + w_3(h_2(s_0'' - u_1') + h_1) + w_4(u_1 + u_1')$;</td>
<td>3M, 1S</td>
</tr>
<tr>
<td>11</td>
<td>Compute $v'' = -h - l' \bmod u''$; $w_0 = (u_1'' - s_0'')(u_1 - u_1')$; $v_0'' = \hat{s}_1(w_0 + u_0' - u_0) - v_1 - h_1 + h_2u''_0$; $v_0'' = \hat{s}_1(s_0''(u_0'' - u_0) + u_0'(u_1 - u_1')) - v_0 - h_0 + h_2u''_0$;</td>
<td>5M</td>
</tr>
</tbody>
</table>

Total 1, 4I, 7S, 2C
IN: Reduced divisor $D = [u, v]$ with
\[
\begin{aligned}
    u &= x^2 + u_1 x + u_0, \\
    v &= v_1 x + v_0, \\
    h &= h_2 x^2 + h_1 x + h_0, \\
    f &= f_5 x^5 + f_4 x^4 + f_3 x^3 + f_2 x^2 + f_1 x + f_0
\end{aligned}
\]

OUT: Reduced divisor $D'' = [u'', v''] = 3D$ with
\[
\begin{aligned}
    u'' &= x^2 + u_1'' x + u_0''; \\
    v'' &= v_1'' x + v_0''
\end{aligned}
\]

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Compute $\tilde{v} \equiv h + 2v \bmod w$: $\tilde{v}_1 = 2v_1; \tilde{v}_0 = 2v_0$;</td>
<td>$-$</td>
</tr>
<tr>
<td>2</td>
<td>Compute $k \equiv (f - hv - v^2) / u \bmod w$: $w_0 = v_1^2; w_1 = u_1^2; w_2 = f_3 + w_1; w_3 = 2u_0; k_1 = 2w_1 + w_2 - w_3; k_0 = u_1(2w_3 - w_2) + f_2 - w_0$;</td>
<td>1M, 2S</td>
</tr>
<tr>
<td>3</td>
<td>Set up system of equations for $s'' = k / s_1 \tilde{v} \bmod w$: $m_1 = -k_1; m_2 = k_0; m_3 = -\tilde{v}_1; m_4 = \tilde{v}_0; r_0 = \tilde{v}_1 u_0; r_1 = \tilde{v}_1 u_1 - \tilde{v}_0$;</td>
<td>2M</td>
</tr>
<tr>
<td>4</td>
<td>Solve system to get $s''_1 = s_0 / s_1$ and $c = 1 / s_1$; $w_0 = (m_2 - r_0)(r_1 - m_1); w_1 = (-r_0 - m_2)(r_1 + m_1); w_2 = (m_4 - r_0)(r_1 - m_3); w_3 = (-r_0 - m_4)(r_1 + m_3); s''_0 = w_0 - w_1; c = w_2 - w_3; d'' = w_2 + w_3 - w_0 - w_1 - 2(m_2 - m_4)(m_1 + m_3); (if $d'' = 0$ branch to Cantor’s Algorithm)</td>
<td>5M</td>
</tr>
<tr>
<td>5</td>
<td>Compute $u''$: $c_2 = c_2^2; u''_1 = 2d'' s''_0 - c_2; u''_0 = s''_0 + 2(d'' u_1 + c_2 u_1);$</td>
<td>4M, 2S</td>
</tr>
<tr>
<td>6</td>
<td>Pre-computations and weighted $\tilde{v}' = -(2v + h) \bmod u''$: $D' = d''^2; \tilde{u}_1 = u_1 D'; \tilde{u}_0 = u_0 D'; \tilde{v}'_1 = -2D' v_1; \tilde{v}'_0 = -2D' v_0$;</td>
<td>4M, 1S</td>
</tr>
<tr>
<td>7</td>
<td>Set up system for $s'' = s'' + s''$ and $\tilde{c} = \tilde{c} + c$: $m_1 = \tilde{v}_0; m_2 = \tilde{v}_0; m_3 = u_1 - u_1; m_4 = u_0 - u_0; r_1 = -D' m_4 - u_0 m_3; r_0 = -u''_1 m_3$;</td>
<td>3M</td>
</tr>
<tr>
<td>8</td>
<td>Solve system for $s''_0$ and $c = 1 / s_1$ and extract $s''_0$ and $\tilde{c}$: $w_0 = (m_2 - r_0)(r_1 - m_1); w_1 = (-r_0 - m_2)(r_1 + m_1); w_2 = (m_4 - r_0)(r_1 - m_3); w_3 = (-r_0 - m_4)(r_1 + m_3); s''_0 = w_0 - w_1; \tilde{c} = w_2 - w_3; d'' = w_2 + w_3 - w_0 - w_1 - 2(m_2 - m_4)(m_1 + m_3); d'' = d'' d''; d'' = d'' D'; s''_0 = s''_0 - s'' d''; \tilde{c} = \tilde{c} - c d''$;</td>
<td>9M</td>
</tr>
<tr>
<td>9</td>
<td>Compute $s'' = x + \frac{n}{s_1}$ and $\tilde{s}_1$: $w_1 = (d'' \tilde{c})^{-1}; w_2 = \tilde{c} w_1; s_1 = d''^2 w_1; w_3 = \tilde{c} w_2; w_4 = w_3^2; s''_0 = s''_0 w_2$;</td>
<td>II, 5M, 2S</td>
</tr>
<tr>
<td>10</td>
<td>Compute $u''$: $u''_1 = 2s''_0 - m_3 - u_4; u''_0 = s''_0 (s''_0 - 2m_3) + u_0 - u_0 + u'_1 (u'_1 - u_1) + 2u_3 v_1 + w_4 (u_1 + u'_1)$;</td>
<td>4M</td>
</tr>
<tr>
<td>11</td>
<td>Compute $u'' = -h - h' \bmod u''$: $w_0 = (u''_1 - s''_0) (u_1 - u'_1); v'_1 = s_1 (u_0 + u''_0 - u_0) - v_1; v''_0 = s_1 (s''_0 (u''_0 - u_0) + u''_0 (u_1 - u'_1)) - v_0$;</td>
<td>5M</td>
</tr>
</tbody>
</table>

Total: I, 42M, 7S, 0C
6.1.4 Projective

In this section we present our fastest projective tripling formula. We use the same techniques as from Section 6.1.2 but keep track of an extra coordinate or the “weight” $z$ instead of computing inversions. An explanation of this setting and terminology used can be found in Section 4.3. In the addition portion, we use a trick to find the value $u''$ by computing $u''$ modulo the input polynomial $u$ and equating coefficients to extract $u''_0$ as described in Section 4.3.2.

We start with an overview of the steps. In odd characteristic we introduce the use of the “improved system of equations” technique to save $2M$ over the work done by Balamohan in [2]. In even characteristic we introduce a new formula that is significantly faster over previous state of the art. A summary of this method is given in Algorithm 6.3 followed by a detailed description of how each step is converted to explicit formulae.

**Steps 1: Compute weighted $\hat{v} = 2v + h \mod u$**

Recall that every $u_i$ and $v_i$ from the input divisor class has a weight of $z$. The coefficients of $\hat{v} = 2v + h \mod u$ are computed to have a weight of $z$. The general explicit formulas is:

\[
\hat{h}_1 = h_1 z; \quad \hat{h}_0 = h_0 z; \\
\hat{v}_1 = 2v_1 + \hat{h}_1 - h_2 u_1; \quad \hat{v}_0 = 2v_0 + \hat{h}_0 - h_2 u_0.
\]

In odd characteristic the explicit formula is:

\[
\hat{v}_1 = 2v_1; \quad \hat{v}_0 = 2v_0.
\]

In even characteristic the explicit formula is:

\[
\hat{h}_1 = h_1 z; \quad \hat{h}_0 = h_0 z; \\
\hat{v}_1 = \hat{h}_1 - h_2 u_1; \quad \hat{v}_0 = \hat{h}_0 - h_2 u_0.
\]

The cost of this step is $0M, 0S, 0C$ in odd characteristic and $0M, 0S, 2C$ in even characteristic.
Algorithm 6.3 Summary of projective divisor class tripling

Input: \( D = [u_1, u_0, v_1, v_0, z] \), where,
\[
\begin{align*}
    u &= x^2 + (u_1/z)x + (u_0/z) \\
    v &= (v_1/z)x + (v_0/z) \\
    f &= x^5 + f_4x^4 + f_3x^3 + f_2x^2 + f_1x + f_0 \\
    h &= h_2x^2 + h_1x + h_0 \quad (h_2 = 1)
\end{align*}
\]

Output: \( D'' = [u_1'', u_0'', v_1'', v_0'', z''] = 3D \) where,
\[
\begin{align*}
    u'' &= x^2 + (u_1''/z'')x + (u_0''/z'') \\
    v'' &= (v_1''/z'')x + (v_0''/z'')
\end{align*}
\]

1: Compute weighted \( \tilde{v} = 2v + h \mod u \) with weight \( z \)
2: Compute weighted \( k \equiv (f - hv - v^2)/u \mod u \), \( k_1 \) with weight \( z^2 \) and \( k_0 \) with weight \( z^3 \).
3: Set up weighted system of equations to solve for \( s = k/\tilde{v} \mod u \)
4: Solve system of equations for \( s'' \), \( s_0'' = s_0/s_1 \) and \( c = 1/s_1 \) up to a factor of \( d' \)
5: Pre-computations and compute \( z' \)
6: Compute \( u' \) with weight \( z' \) by equating coefficients of \( u^2u' = \frac{u^2+h'v-f}{l_2} \)
7: Pre-computations and \( \tilde{v}' = -(2v + h) \mod u' \) with a weight of \( z' \)
8: Set up system of equations that solves for \( \tilde{s} = \tilde{s} + s \equiv -(2v + h)/u \mod u' \)
9: Solve system for \( \tilde{s}' = \tilde{s}_1x + \tilde{s}_0 \), up to a factor of \( d'' \)
10: Extract \( \tilde{s} \) from \( \tilde{s} = \tilde{s} + s \) with weight \( d'' \), compute \( \tilde{s} = s - \tilde{s} \) with weight \( d'' \)
11: Pre-computations and compute \( z'' \)
12: Compute \( u_0'' \) by equating coefficients after computing \( u'' \mod u \)
13: Compute \( u_1'' \) by equating coefficients \( uu'u'' = \frac{u^2+h''v-f}{l_2} \)
14: Compute \( v'' \equiv -h - \tilde{l}' \mod u'' \)
Steps 2: Compute weighted \( k \equiv (f - hv - v^2)/u \mod u \)

The coefficients of \( k \) are computed just like in Step 2 of [5.1.2] but \( k_1 \) is computed with a weight of \( z^2 \) and \( k_0 \) with a weight of \( z^3 \). The general explicit formula is:

\[
Z = z^2;
\]

\[
w_1 = u_1^2; \quad w_2 = Zf_3 + w_1; \quad w_3 = zu_0; \quad w_4 = zv_1;
\]

\[
k_1 = 2(w_1 - zf_4u_1) + w_2 - 2w_3 - h_2w_4;
\]

\[
k_0 = u_1(4w_3 - w_2 + zf_4u_1 + h_2w_4) + z(zf_2 - 2f_4u_0 - h_2v_0) - v_1(h_1 + v_1)).
\]

In odd characteristic the explicit formula is:

\[
Z = z^2;
\]

\[
w_0 = v_1^2; \quad w_1 = u_1^2; \quad w_2 = Zf_3 + w_1; \quad w_3 = zu_0;
\]

\[
k_1 = 2(w_1 - w_3) + w_2;
\]

\[
k_0 = u_1(4w_3 - w_2) + z(Zf_2 - w_0).
\]

Notice that in the even characteristic the computation of \( v^2 \) can be omitted, so the explicit formula is:

\[
Z = z^2;
\]

\[
w_0 = zv_1; \quad w_1 = u_1^2; \quad w_2 = Zv_0;
\]

\[
k_1 = w_1 - h_2w_0;
\]

\[
k_0 = -u_1k_1 - w_0(v_1 + \hat{h}_1) - h_2w_2.
\]

The cost of this step is \( 3M, 3S, 2C \) in odd characteristic, and \( 4M, 2S, 0C \) in even characteristic.
Step 3: Set up weighted system of equations to solve for $s_0 = k/\hat{v} \mod u$

In this step, the same affine values are computed as in Step 3 Section 5.1.2 but with some adjustment for the weights on $\hat{v}$ and $k$. The weighted system of equations is set up as:

$$
\begin{pmatrix}
z\hat{v}_0 & -k_0 \\
\hat{v}_1 & -k_1
\end{pmatrix} \times 
\begin{pmatrix}
s''_0 \\
c
\end{pmatrix} = 
\begin{pmatrix}
z\hat{v}_1 u_0 \\
\hat{v}_1 u_1 - z\hat{v}_0
\end{pmatrix}.
$$

In general the explicit formula is:

$$
w_0 = \hat{v}_1 z;
$$

$$
m_1 = -k_1; \quad m_2 = k_0; \quad m_3 = -\hat{v}_1; \quad m_4 = z\hat{v}_0;
$$

$$
r_0 = w_0 u_0; \quad r_1 = \hat{v}_1 u_1 + m_4.
$$

Specializing to odd characteristic, the explicit formula is:

$$
w_0 = zv_0;
$$

$$
m_1 = -k_1; \quad m_2 = k_0; \quad m_3 = -\hat{v}_1; \quad m_4 = 2w_0;
$$

$$
r_0 = w_3 \hat{v}_1; \quad r_1 = \hat{v}_1 u_1 + m_4.
$$

The cost of this step is $3M, 0S, 0C$ in odd characteristic, and $4M, 0S, 0C$ in even characteristic.

Step 4: Solve for $s''', s'''_0 = s_0/s_1$ and $c = 1/s_1$ with weight $d''$

In this step we introduce a novel reduction of one multiplication in odd characteristic due to the “improved solution to a system of equations” trick (Section 4.2.8). In even characteristic, the system is solved using Cramer’s rule. In general, the explicit formula is:

$$
s'''_0 = r_0 m_1 + r_1 m_2; \quad c' = r_0 m_3 + r_1 m_4;
$$

$$
d'' = m_4 m_1 - m_2 m_3.
$$
In odd characteristic the explicit formula is:

\[ w_1 = (m_2 - r_0)(r_1 - m_1); \quad w_2 = (-r_0 - m_2)(r_1 + m_1); \]
\[ w_3 = (m_4 - r_0)(r_1 - m_3); \quad w_4 = (-r_0 - m_4)(r_1 + m_3); \]
\[ s'''_0 = w_1 - w_2; \quad c' = w_3 - w_4; \]
\[ d'' = w_3 + w_4 - w_1 - w_2 - 2(m_2 - m_4)(m_1 + m_3). \]

Notice that \( d'' = d'z^2 \) has a weight of \( z^2 \) with respect to the affine value of \( d' \). Then \( s'''_0 = s''_0 d'z^3 = s''_0 d''z \) has a weight of \( d''z \) and \( c' = d'z^2/s_1 = d''/s_1 \) has a weight of \( d'' \) with respect to their affine values as well. The cost of this step is \( 6M, 0S, 0C \) for both general and even characteristic and the cost is \( 5M, 0S, 0C \) in odd characteristic.

**Step 5: Pre-computations and \( z'' \)**

At this point \( d'', s'''_0 (s''_0 \text{ with weight } d''z) \), and \( c' (1/s_1 \text{ with weight } d'') \) have been computed. All pre-computations are done to produce differently weighted versions of the affine values \( s''_0, 1/s_1 \) and \( 1/s_1^2 \) that are used in the computation of \( u'' \) and \( v'' \). The output weight \( z'' \) that both \( u'' \) and \( v'' \) have is also computed. Based on the values needed to compute \( u' \); \( S'_1 \) can be omitted in odd characteristic and \( S_{sq} \) can be omitted in even characteristic. The general explicit formula is:

\[ d''' = zd''; \]
\[ z' = d'''^2; \]
\[ S_0 = s'''_0 d'''^2; \quad (s'''_0 \text{ with a weight of } d'''^2z^2) \]
\[ c'' = c'z; \quad (1/s_1 \text{ with a weight of } d''z) \]
\[ S_1 = d''c''; \quad (1/s_1 \text{ with a weight of } d''^2z) \]
\[ S'_1 = S_1z; \quad (1/s_1 \text{ with a weight of } d''^2z^2) \]
\[ S_{sq} = c''c; \quad (1/s_1^2 \text{ with a weight of } d''^2z) \]
\[ S'_{sq} = c''^2; \quad (1/s_1^2 \text{ with a weight of } d''^2z^2) \]
The explicit formula in odd characteristic is:

\[ d''' = zd''; \]
\[ z' = d''^2; \]
\[ S_0 = s'''_0 d'''; \quad (s'''_0 \text{ with a weight of } d''^2 z^2) \]
\[ c'' = c' z; \quad (1/s_1 \text{ with a weight of } d'' z) \]
\[ S_1 = d'' c''; \quad (1/s_1 \text{ with a weight of } d''^2 z) \]
\[ S_{sq} = c'' c; \quad (1/s_1^2 \text{ with a weight of } d''^2 z) \]
\[ S'_{sq} = c''^2. \quad (1/s_1^2 \text{ with a weight of } d''^2 z^2) \]

The explicit formula in even characteristic is:

\[ d''' = zd''; \]
\[ z' = d''^2; \]
\[ S_0 = s'''_0 d'''; \quad (s'''_0 \text{ with a weight of } d''^2 z^2) \]
\[ c'' = c' z; \quad (1/s_1 \text{ with a weight of } d'' z) \]
\[ S_1 = d'' c''; \quad (1/s_1 \text{ with a weight of } d''^2 z) \]
\[ S'_1 = S_1 z; \quad (1/s_1 \text{ with a weight of } d''^2 z^2) \]
\[ S'_{sq} = c''^2. \quad (1/s_1^2 \text{ with a weight of } d''^2 z^2) \]

All of the values are adjusted to have a \( d''^2 \) in their weights because in order to compute \( u'' \) a weighted \( 1/s_1^2 \) and \( s'''_0 \) need to be computed and only \( c' = d''/s_1 \) and \( s'''_0 = s'''_0 d'' z \) are available to compute with. The cost of this step is \( 5M, 2S, 0C \) in both characteristics.
Step 6: Compute $u''$

In this step, $u'$ is computed with a weight of $z'$ with respect to its affine value described in Step 7 of Section 5.1.2. The general explicit formula is:

$$u'_1 = 2S_0 + h_2S'_1 - S'_{sq};$$

$$u'_0 = s''m^2 + w_1(h_2s''_0 - d''(h_2u_1 + 2v_1 + \hat{h}_1)) + S_{sq}(2u_1 - zf_4).$$

In odd characteristic, the explicit formula is:

$$\tilde{v}_1 = S_1v_1;$$

$$u'_1 = 2S_0 - S'_{sq};$$

$$u'_0 = s''m^2 + 2\tilde{v}_1 + 2S_{sq}u_1.$$

In even characteristic, the explicit formula is:

$$u'_1 = h_2S'_1 - S'_{sq};$$

$$u'_0 = s''m^2 + w_1(h_2s''_0 - d''(h_2u_1 + \hat{h}_1)).$$

The cost of this step is $2M, 1S, 0C$ in both characteristics.

Step 7: Pre-computations and $\bar{v}' = -(2v + h) \mod u'$

In this step $u$ is adjusted to have the same weight as $u'$ because both are used in the system of equations that solves for $\hat{s}$ in the next step. The coefficients of $\bar{v}' = -(2v + h) \mod u'$ are computed with a weight of $z''^2/s_1$ in even characteristic and $z'/s_1$ in odd. The weight has to be higher in even because of the $h_2u_i$ term. Keeping the $1/s_1$ in the weight of $\bar{v}'$ helps with adjustments in the next system of equations solved in Step 9. The general explicit formula is:

$$w_1 = d''d''';$$

$$\hat{u}_0 = u_0w_1; \quad \hat{u}_1 = u_1w_1;$$

$$\tilde{h}'_1 = h_1z'; \quad \bar{h}'_0 = h_0z';$$

$$\bar{v}_1 = v_1w_1; \quad \bar{v}_0 = v_0w_1;$$

$$\bar{v}'_1 = S_1(h_2u'_1 - \bar{h}'_1 - 2\bar{v}_1); \quad \bar{v}'_0 = S_1(h_2u'_0 - \bar{h}'_0 - 2\bar{v}_0).$$
In odd characteristic the explicit formula is:

\[ w_1 = d''d'''; \]
\[ \hat{u}_0 = u_0w_1; \quad \hat{u}_1 = u_1w_1; \]
\[ \tilde{v}_0 = S_1v_0; \]
\[ \tilde{v}_1 = -2\tilde{v}_1; \quad \tilde{v}_0 = -2\tilde{v}_0. \]

In even characteristic the explicit formula is:

\[ w_1 = d''d'''; \]
\[ \hat{u}_0 = u_0w_1; \quad \hat{u}_1 = u_1w_1; \]
\[ \tilde{h}'_1 = h_1z'; \quad \tilde{h}'_0 = h_0z'; \]
\[ \tilde{v}_1 = v_1w_1; \quad \tilde{v}_0 = v_0w_1; \]
\[ \tilde{v}_1 = S_1(h_2u'_1 - \tilde{h}'_1); \quad \tilde{v}_0 = S_1(h_2u'_0 - \tilde{h}'_0). \]

The cost of this step is 4M, 0S, 0C in odd characteristic and 7M, 0S, 2C in even characteristic.

**Step 8: Set up system of equations that solves for** \( \hat{s} = \tilde{s} + s \equiv -(2v + h)/u \mod u' \)

In this step, a weighted system that solves for

\[ \hat{s} \equiv \tilde{s} + s \equiv \tilde{v}' \mod u' \]

is set up as described in Section 6.1.1. Multiplying by \( u \) on both sides results in

\[(x^2 + u_1x + u_0)(\hat{s}_1x + \hat{s}_0) = \tilde{v}'_1x + \tilde{v}'_0 \mod x^2 + u'_1x + u'_0,\]

producing the linear equations

\[ \tilde{v}'_0 = \hat{s}_0(u_0 - u'_0) + \hat{s}_1(u'_0(u'_1 - u_1)), \]
\[ \tilde{v}'_1 = \hat{s}_0(-(u'_1 - u_1)) + \hat{s}_1(u_0 - u'_0 + u'_1(u'_1 - u_1)). \]
The coefficients of the weighted polynomial \( \hat{s} = \hat{s}_1 x + \hat{s}_0 \) can be solved for by setting up

\[
\begin{pmatrix}
  u_0 - u'_0 \\
  - (u'_1 - u_1) \\
\end{pmatrix}
\begin{pmatrix}
  u'_0 (u'_1 - u_1) \\
  z' (u_0 - u'_0) + u'_1 (u'_1 - u_1) \\
\end{pmatrix}
\times
\begin{pmatrix}
  \hat{s}_0 \\
  \hat{s}_1 \\
\end{pmatrix}
= \begin{pmatrix}
  \hat{\tilde{v}}'_0 \\
  \hat{\tilde{v}}'_1 \\
\end{pmatrix}.
\]

The general explicit formula for this step is:

\[
m_3 = u'_1 - u_1; \quad m_4 = u_0 - u'_0; \quad m_1 = z'm_4 + u'_1m_3; \quad m_2 = u'_0m_3.
\]

The cost of this step is \( 3M, 0S, 0C \) in both characteristics.

**Step 9: Solve system for \( \hat{s} = \hat{s}_1 x + \hat{s}_0 = \hat{\tilde{v}}'/u \mod u' \)**

The system is solved by either using Cramer’s rule in even characteristic or the “improved solution to a system of equations” in odd characteristic (Section 4.2.8), resulting in \( \hat{s}' = \hat{\tilde{d}}'' \hat{s} = \hat{s}'_1 x + \hat{s}'_0 \) where \( \hat{\tilde{d}}'' = 2\hat{\tilde{d}}' \) in odd characteristic and \( \hat{\tilde{d}}'' = \hat{\tilde{d}}' \) in even. Notice that \( \hat{\tilde{d}}' \) is the determinant of the linear two by two system. In odd characteristic the weight of the resulting \( \hat{s}'_1 = \hat{s}_1 \hat{\tilde{d}}'' / s_1 \) and \( \hat{s}'_0 = \hat{s}_0 \hat{\tilde{d}}'' / s_1 \) is \( \hat{\tilde{d}}'' / s_1 \). In even characteristic the weight of the resulting \( \hat{s}'_1 = \hat{s}_1 \hat{\tilde{d}}'' \) is \( \hat{\tilde{d}}'' \) and \( \hat{s}'_0 = \hat{s}_0 \hat{\tilde{d}}'' \) is \( z' \hat{\tilde{d}}'' \). In general the explicit formula is:

\[
\hat{s}'_0 = \tilde{v}'_0m_1 + \tilde{v}'_1m_2; \quad \hat{s}'_1 = \tilde{v}'_0m_3 + \tilde{v}'_1m_4;
\]

\[
\hat{\tilde{d}}'' = m_4m_1 - m_2m_3.
\]

The explicit formula in odd characteristic is :

\[
w_0 = (m_2 - \tilde{v}'_1)(\tilde{v}'_1 - m_1); \quad w_1 = (-\tilde{v}'_0 - m_2)(\tilde{v}'_1 + m_1);
\]

\[
w_2 = (m_4 - \tilde{v}'_1)(\tilde{v}'_1 - m_3); \quad w_3 = (-\tilde{v}'_0 - m_4)(\tilde{v}'_1 + m_3);
\]

\[
\hat{s}'_0 = w_0 - w_1; \quad \hat{s}'_1 = w_2 - w_3;
\]

\[
\hat{\tilde{d}}'' = w_2 + w_3 - w_0 - w_1 - 2(m_2 - m_4)(m_1 + m_3).
\]

The cost of this step is \( 5M, 0S, 0C \) in odd characteristic and \( 6M, 0S, 0C \) in even characteristic.
Step 10: Extract $\tilde{s}$ from $\hat{s} = \tilde{s} + s$ with weight $\hat{d''}$, compute $\bar{s} = s - \tilde{s}$ with weight $\hat{d''}$

In this step, the affine $\tilde{s} = \hat{s} - s$ and $\bar{s} = s - \tilde{s}$ are computed with weight $\hat{d''}/s_1$ for $\tilde{s}'_1$ and $\bar{s}'_1$ and weight $\tilde{z}'\hat{d''}/s_1$ for $\tilde{s}'_0$ and $\bar{s}_0$ in both characteristics, where $\hat{d''} = \tilde{d'}z'^3$. The number of operations needed for the computation of $u''$ and $v''$ is reduced using these weights. The coefficients of $\tilde{s}'$ are used in Steps 13 and 14 to solve for $u'_1''$ and $v''$. The coefficients of $\bar{s}$ are used in Step 12 to compute $u''$ using the $u''$ modulo $u$ technique as described in Section 4.3.2.

Recall that $S_0 = s_0 z'/s_1$ was computed in earlier steps, so

$$\tilde{s}'_1 = \frac{\hat{s}_0 \hat{d}''}{s_1} \tilde{z}' (\tilde{s}_0 + s_0) = \frac{\hat{d}''}{s_1} \tilde{s}_0 + \frac{\hat{d}''}{s_1} s_0 \Rightarrow \tilde{s}'_0 \tilde{z}' = \tilde{s}'_0 + \hat{d}'' S_0,$$

in odd characteristic, and

$$\tilde{s}'_0 = \frac{\hat{s}_0 \hat{d}''}{s_1} \tilde{z}' (\tilde{s}_0 + s_0) = \frac{\hat{d}''}{s_1} \tilde{s}_0 + \frac{\hat{d}''}{s_1} s_0 = \tilde{s}'_0 + \hat{d}'' S_0,$$

in even characteristic. Similarly,

$$\tilde{s}'_1 = \frac{\hat{s}_0 \hat{d}''}{s_1} \tilde{s}'_0 = \frac{\hat{d}''}{s_1} (\tilde{s}_1 + s_1) = \frac{\hat{d}''}{s_1} \tilde{s}_1 + \frac{\hat{d}''}{s_1} = \tilde{s}'_1 + \hat{d}''.$$

The resulting equations for $\tilde{s}'_1$ and $\tilde{s}'_0$ are,

$$\tilde{s}'_1 = \tilde{s}'_1 - \hat{d}''; \quad \tilde{s}'_0 = \tilde{z}' \tilde{s}'_0 - S_0 \hat{d}'',$$

in odd characteristic and,

$$\tilde{s}'_1 = \tilde{s}'_1 - \hat{d}''; \quad \tilde{s}'_0 = \tilde{s}'_0 - S_0 \hat{d}''.$$

The resulting equations for $\bar{s}_0$ and $\bar{s}'_1$ are,

$$\bar{s}_1 = \bar{s}'_1 - 2\hat{d}''; \quad \bar{s}_0 = \bar{s}'_0 - 2S_0 \hat{d}'',$$

in odd characteristic and,

$$\bar{s}_1 = \bar{s}'_1 - \hat{d}''; \quad \bar{s}_0 = \bar{s}'_0.$$

143
in even characteristic. The general explicit formula is:

\[
\begin{align*}
    w_0 &= S_0 \hat{d}''; \\
    w_1 &= z' \hat{\bar{s}}_0' \\
    \bar{s}_1 &= \bar{s}_1 - \hat{\bar{s}}_0''; \\
    \bar{s}_1' &= z' \bar{s}_1; \\
    \bar{s}_0' &= \bar{s}_0' - w_0; \\
    \bar{s}_1 &= \bar{s}_1' - \hat{\bar{s}}_0''; \\
    \bar{s}_0 &= \bar{s}_0' - w_0.
\end{align*}
\]

In odd characteristic, the explicit formula is:

\[
\begin{align*}
    w_0 &= S_0 \hat{d}''; \\
    w_1 &= z' \hat{\bar{s}}_0' \\
    \bar{s}_1 &= \bar{s}_1 - \hat{\bar{s}}_0''; \\
    \bar{s}_1' &= z' \bar{s}_1; \\
    \bar{s}_0' &= \bar{s}_0' - w_0; \\
    \bar{s}_1 &= \bar{s}_1' - \hat{\bar{s}}_0''; \\
    \bar{s}_0 &= \bar{s}_0' - w_0.
\end{align*}
\]

In even characteristic the explicit formula is:

\[
\begin{align*}
    w_0 &= S_0 \hat{d}''; \\
    \bar{s}_1 &= \bar{s}_1 - \hat{\bar{s}}_0''; \\
    \bar{s}_1' &= z' \bar{s}_1; \\
    \bar{s}_0' &= \bar{s}_0' - w_0; \\
    \bar{s}_1 &= \bar{s}_1' - \hat{\bar{s}}_0''; \\
    \bar{s}_0 &= \bar{s}_0' - w_0.
\end{align*}
\]

The cost of this step is \(3M, 0S, 0C\) in odd characteristic and \(2M, 0S, 0C\) in even characteristics.

**Step 11: Pre-computations**

In this step, values that are used in the computation of \(u''\) and \(v''\) are pre-computed along with their weight \(z''\). Weighted versions of \(1/\bar{s}_1\) and \(1/\bar{s}_0\) are computed for Steps 12 - 14. There are differences in the weight between even and odd characteristic; note that using the even characteristic route will work in the arbitrary case. In general and even characteristic
the explicit formula is:

\[
\tilde{s}_{sq} = \tilde{s}_1^2; \\
\tilde{S}_0 = \tilde{s}_1^t \tilde{s}_0^t; \\
\tilde{d} = \tilde{d}'' \tilde{S}_1 \tilde{d}'''; \\
\tilde{D} = \tilde{d}^2; \\
\tilde{s}_1'' = \tilde{s}_1^t d'''; \\
\tilde{s}_q'' = \tilde{s}_1''; \\
\tilde{S}_1 = \tilde{d} \tilde{s}_1''.
\]

In odd characteristic the explicit formula is:

\[
\tilde{s}_{sq} = \tilde{s}_1^2; \\
\tilde{S}_0 = \tilde{s}_1^t \tilde{s}_0^t; \\
\tilde{d} = \tilde{c}'' \tilde{d}'''; \\
\tilde{D} = \tilde{d}^2; \\
\tilde{s}_1'' = \tilde{s}_1^t d'''; \\
\tilde{s}_q'' = \tilde{s}_1''; \\
\tilde{S}_1 = \tilde{d} \tilde{s}_1''.
\]

The cost of this step is 4M, 3S, 0C in odd characteristic, and 6M, 3S, 0C in even characteristic.

**Step 12: Compute** $u''_0 = u''_0$ **modulo** $u$ and $u''_0$

In this step, $u''_0$ is computed by computing $u''$ modulo $u$ as described in Section 4.3.2. The value $u''_0$ is computed from equating coefficients of Equation 4.7. In odd characteristic, the modulo $u$ method requires a weight of $\tilde{s}_1^2$, and the only way to compute that is with $\tilde{s}_1^t$ which results in $\tilde{s}_1^2 \tilde{d}''$. The weight of $u''_0$ is $\tilde{D} \tilde{s}_1^2$ in both characteristics. The value $\hat{u}_0$ is adjusted.
with $\tilde{s}_{sq}$ because there is a factor of $\tilde{s}_1^2$ on the left side of equation 4.7 in Section 4.3.2. The general explicit formula is:

$$u_0''' = \tilde{d}''(\bar{s}_0\tilde{v}_0' + \bar{s}_1\tilde{v}_1'u_0');$$

$$u_0'' = u_0''' + \hat{u}_0\tilde{s}_{sq}.$$  

The cost of this step is $5M, 0S, 0C$ in both characteristics.

**Step 13: Compute $u_1''$**

In this step the weighted value $u_1''$ is computed similarly to Step 5 of Section 5.2 but with different weights. The weight of $u_1''$ is $\bar{D}\tilde{s}_1^2$. In general the explicit formula is:

$$u_1''' = 2\tilde{S}_0 + \tilde{s}_1''\tilde{d}h_2 - \bar{D} - \tilde{s}_{sq}u_1';$$

$$u_1'' = u_1''' + \hat{u}_1\tilde{s}_{sq}.$$  

In odd characteristic, the explicit formula is:

$$u_1''' = 2\tilde{S}_0 - \bar{D} - \tilde{s}_{sq}u_1';$$

$$u_1'' = u_1''' + \hat{u}_1\tilde{s}_{sq}$$

In even characteristic, the explicit formula is:

$$u_1''' = \tilde{s}_1''\tilde{d}h_2 - \bar{D} - \tilde{s}_{sq}u_1';$$

$$u_1'' = u_1''' + \hat{u}_1\tilde{s}_{sq}.$$  

The cost of this step is $2M, 0S, 0C$ in odd characteristic and $3M, 0S, 0C$ in even characteristic.

**Step 14: Compute weighted $v'' \equiv -h - \bar{l}' \pmod{u''}$**

In this step the weighted coefficients of $v''$ are computed. Notice that $u_0''' = u_0'' - \tilde{s}_{sq}\hat{u}_0$ and $u_1''' = u_1'' - \tilde{s}_{sq}\hat{u}_1$ are available, so $(u_1'' - \hat{u}_1)$ and $(u_0'' - \hat{u}_0)$ are replaced in Step 7 of Section 5.2 with their respective values here. Also notice $u_0$ and $u_1$ do no need to be adjusted as done
in Step 6 of 5.2. The coefficients of $v''$ are computed with a weight of $\tilde{D}^2\tilde{s}_1^3$. The general and even characteristic formula is:

\[
v''_1 = (u''_1 - \tilde{S}_0)u''_0 + \tilde{s}''_{sq}u'''_0 - \tilde{S}_1(\tilde{s}_{sq}(\tilde{v}_1 + \tilde{h}_1) - h_2u''_1);
\]
\[
v''_0 = \tilde{S}_0u''_0 + u''_0u''_1 - \tilde{S}_1(\tilde{s}_{sq}(\tilde{v}_0 + \tilde{h}_0) - h_2u''_0).
\]

In odd characteristic the formula is:

\[
w_0 = \tilde{d}''s_1';
\]
\[
\tilde{v}''_1 = \tilde{v}_1w_0; \quad \tilde{v}''_0 = \tilde{v}_0w_0;
\]
\[
v''_1 = (u''_1 - \tilde{S}_0)u''_0 + \tilde{s}''_{sq}(u'''_0 - \tilde{v}''_1);
\]
\[
v''_0 = \tilde{S}_0u''_0 + u''_0u''_1 - \tilde{s}''_{sq}\tilde{v}''_0.
\]

The cost of this step is $8M, 0S, 0C$ in both characteristics.

**Step 15: Adjust $u''$ to have weight $z''$ and compute $z''$**

In this step the weight of $u''$ is adjusted to have the same weight has $v''$. The output weight $z''$ is also computed. The general the explicit formula is:

\[
u''_0 = u''_0\tilde{S}_1;
\]
\[
u''_1 = u''_1\tilde{S}_1;
\]
\[
z'' = \tilde{s}''_{sq}\tilde{S}_1.
\]

The cost of this step is $3M, 0S, 0C$ in both characteristics.

The total cost is $55M, 9S, 2C$ in odd characteristic, and $64M, 8S, 4C$ in even characteristic. Table 6.7 summarizes the formulae in arbitrary case. Tables 6.8 and 6.9 summarize the formulae in even and odd characteristic cases respectively.
### Table 6.7: Arbitrary Projective Tripling Formula

**IN:** Reduced divisor \( D = [u, v, z] \) with
\[
\begin{align*}
    u &= x^2 + (u_1/z)x + (u_0/z), \\
    v &= (u_1/z)x + (v_0/z), \\
    h &= h_2x^3 + h_1x + h_0, \\
    f &= f_5x^5 + f_4x^4 + f_3x^3 + f_2x^2 + fx + f_0
\end{align*}
\]

**OUT:** Reduced divisor \( D'' = \left[ u'', v'', z'' \right] = 3D \) with
\[
\begin{align*}
    u'' &= x^2 + (u''_1/z'')x + (u''_0/z''), \\
    v'' &= (v''_1/z'')x + (v''_0/z'')
\end{align*}
\]

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Compute ( \bar{v} \equiv h + 2v \mod u' )</td>
<td>2C</td>
</tr>
<tr>
<td>2</td>
<td>Compute ( k \equiv (f - hv - v'')/u \mod w' )</td>
<td>6M, 2S, 2C</td>
</tr>
<tr>
<td>3</td>
<td>Set up system of equations for ( s'' \equiv k/s_1 \bar{v} \mod u' )</td>
<td>4M</td>
</tr>
<tr>
<td>4</td>
<td>Solve system of equations for ( s''_0 = s_0/s_1 ) and ( c'' = 1/s_1 )</td>
<td>6M</td>
</tr>
<tr>
<td>5</td>
<td>Pre-computations and ( z' ):</td>
<td>6M, 2S</td>
</tr>
<tr>
<td>6</td>
<td>Compute ( u' ):</td>
<td>3M, 1S</td>
</tr>
<tr>
<td>7</td>
<td>Pre-computations and ( \bar{v}' = -(2v + h) \mod u' ):</td>
<td>7M, 2C</td>
</tr>
<tr>
<td>8</td>
<td>Set up system for ( s = s + s \equiv -(2v + h)/u \mod u' ):</td>
<td>3M</td>
</tr>
<tr>
<td>9</td>
<td>Solve system for ( s = s_1x + s_0 = \bar{v}'/u \mod u' ):</td>
<td>6M</td>
</tr>
<tr>
<td>10</td>
<td>Extract ( s ) from ( s = s + s ):</td>
<td>3M</td>
</tr>
<tr>
<td>11</td>
<td>Pre-computations:</td>
<td>6M, 3S</td>
</tr>
<tr>
<td>12</td>
<td>Compute ( u''_0 = u_0'' \mod u ) and ( u''_0 ):</td>
<td>5M</td>
</tr>
<tr>
<td>13</td>
<td>Compute ( u''_1 ):</td>
<td>3M</td>
</tr>
<tr>
<td>14</td>
<td>Compute weighted ( u'''' \equiv h - f' \mod u'' ):</td>
<td>8M</td>
</tr>
<tr>
<td>15</td>
<td>Adjust ( u'''' ) to have weight ( z'' ) and compute ( z'' ):</td>
<td>3M</td>
</tr>
</tbody>
</table>

**Total:** 69M, 8S, 6C
Table 6.8: Even Characteristic Projective Tripling Formula

IN: Reduced divisor $D = [u, v, z]$ with
\[ u = x^2 + (u_0/z)x + (v_0/z), \]
\[ v = (v_1/z)x + (v_0/z), \]
\[ h = h_2x^2 + h_1x + h_0, \]
\[ f = f_5x^5 + f_4x^4 + f_3x^3 + f_2x^2 + f_1x + f_0 \]

OUT: Reduced divisor $D'' = [u'', v'', z''] = 3D$ with
\[ u'' = x^2 + (u''_0/z')x + (v''_0/z'), \]
\[ v'' = (v''_1/z')x + (v''_0/z'), \]
\[ S = 3u_0 - 9u_0^2 + 8u_0u_1 + u_1^2 \]

Step | Procedure | Cost
--- | --- | ---
1 | Compute $\hat{v} = \hat{h} + 2\bar{v}$ mod $u$; \n\[ h_1 = h_1z; h_0 = h_0z; \hat{v}_1 = \hat{h}_1 - h_2u_1; \hat{v}_0 = \hat{h}_0 - h_2u_0; \] | 2C
2 | Compute $k = (f - \nu u - \nu'')/u$ mod $u$; \n\[ Z = z^2; w_0 = zv_1; w_1 = u''_1; w_2 = v_0z; k_1 = w_1 - h_2w_0; \]
\[ k_0 = -u_1k_1 - w_0(v_1 + h_1) - h_2w_2; \] | 4M, 2S
3 | Set up system of equations for $s'' \equiv k/s_1\nu$ mod $u$; \n\[ u_0 = \hat{v}_1z; m_1 = -k_1; m_2 = k_0; m_3 = -\hat{v}_1; m_4 = z\hat{v}_0; \]
\[ r_0 = w_0u_0; r_1 = \hat{v}_1u_1 + u_4; \] | 4M
4 | Solve system of equations for $s_0'' = s_0/s_1$ and $c = 1/s_1$: \n\[ s_0'' = r_0m_1 + r_1m_2; c' = r_0m_3 + r_1m_4; d'' = m_4m_1 - m_2m_3; \]
(if $d'' = 0$ branch to Cantor’s Algorithm) | 6M
5 | Pre-computations and $z'$; \n\[ d'' = zd''''; z' = d''''; S_0 = d''''s_0; c'' = c'/z; \]
\[ S_1 = d''d'; S''_2 = (c'')^2; \] | 5M, 2S
6 | Compute $u'$; \n\[ u_1' = \hat{h}_1s_1'' - S''_2; u_0' = \hat{h}_2s_1'' - d'((h_2u_1'' + \hat{h}_1)); \] | 2M, 1S
7 | Pre-computations and $d' = -(2v + h)$ mod $u'$; \n\[ w_1 = d''u'; u_0 = u_0w_1; \hat{v}_1 = u_1w_1; h_1' = h_1z'; h_0' = h_0z'; \]
\[ \hat{v}_0 = \hat{v}_1 - h_1u_1 + h_1u_0; \] | 7M, 2C
8 | Set up system for $\tilde{s} = \tilde{s} + s \equiv -(2v + h)/u$ mod $u'$; \n\[ m_1 = u_1 - u_1; m_4 = u_4 - u_0'; m_1 = z'm_4 + u_4'm_3; m_2 = u_0'm_3; \] | 3M
9 | Solve system for $\hat{s} = \hat{s}_1x + s_0 = \tilde{s}'/u$ mod $u'$; \n\[ \hat{s}_0'' = \hat{s}'_0m_1 + \hat{s}'_0m_2; \hat{s}_1'' = \hat{s}'_0m_3 + \hat{s}'_0m_4; \hat{d}' = m_4m_1 - m_2m_3; \] | 6M
10 | Extract $s$ from $s = \hat{s} + s$, compute $\hat{s} = s - \hat{s}$; \n\[ w_0 = S_0d''; \hat{s}_1 = \hat{s}'_1 - d''; \hat{s}_2' = z''\hat{s}_1; \hat{s}_0 = \hat{s}_0 - w_0; \]
\[ s_1 = \hat{s}_1; s_0 = \hat{s}_0; \] | 2M
11 | Pre-computations: \n\[ s_2 = \hat{s}_2; S_0 = \hat{s}'_0s_0; \hat{d} = \hat{d}''S_1d'''; \hat{D} = \hat{d}''; \]
\[ \hat{s}''_1 = \hat{s}'_0d''''; \hat{s}_2'' = \hat{s}'_0r_2''; S_1 = \hat{d}''s_1''; \] | 6M, 3S
12 | Compute $u''_0 = u_0''$ mod $u$ and $u''_0'$; \n\[ u''_0 = d''(s_0v_0'' + s_1v_1''); u''_0' = u''_0' + \hat{u}a_1s_0''; \] | 5M
13 | Compute $u''_1$; \n\[ u''_1 = \hat{s}'_1dh_2 - \hat{d} - \hat{s}_2u_1'' + \hat{u}_0s_2''; \] | 3M
14 | Compute weighted $u''''_1 \equiv -h - l'$ mod $u''$; \n\[ u''_1' = (u''_1 - S_0)u''_1''' + S_2u''''_0'' - S_1(s_0(v_1 + h_1') - h_2u_1'''); \]
\[ u''_1'' = \hat{S}_0u''''_1 + u''''_0u''_1 - S_1(s_0\hat{v}_0 + \hat{h}_1') - h_2u_1'''; \] | 8M
15 | Adjust $u''''$ to have weight $z''$ and compute $z''$; \n\[ u''_0'' = u''_0S_1; u''''_1 = u''_0''S_1; z'''' = \hat{s}_2''S_1. \] | 3M

Total | | 64M, 8S, 4C
Table 6.9: Odd Characteristic Projective Tripling Formula

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Compute ( \hat{v} \equiv h + 2v \mod u; ) ( \hat{v}_1 = 2v_1; \hat{v}_0 = 2v_0; )</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>Compute ( k \equiv (f - hv - v^2)/u \mod u; ) ( \hat{Z} = z^2; u_0 = v_1^2; w_1 = u_1^2; w_2 = Zf_3 + w_1; w_3 = zu_0; )</td>
<td>3M, 3S, 2C</td>
</tr>
<tr>
<td>3</td>
<td>Set up system of equations for ( s'' \equiv k/s_1 \hat{v} \mod u; ) ( u_0 = zv_0; m_1 = -k_1; m_2 = k_2; m_3 = -\hat{v}_1; m_4 = 2w_0; ) ( r_0 = w_0\hat{v}_1; r_1 = r_1u_1 + m_4; )</td>
<td>3M</td>
</tr>
<tr>
<td>4</td>
<td>Solve system of equations for ( s'' = s_0/s_1 ) and ( c'' = 1/s_1; ) ( w_1 = (m_2 - r_0)(r_1 - m_1); w_2 = (-r_0 - m_2)(r_1 + m_1); w_3 = (m_1 - r_0)(r_1 - m_3); w_4 = (-r_0 - m_3)(r_1 + m_3); ) ( s''_0 = w_1 - w_2; ) ( \hat{d}' = w_3 - w_4; ) ( d'' = w_3 + w_4 - w_2 - 2(m_2 - m_4)(m_1 + m_3); ) ( (if \ d'' = 0 \text{ branch to Cantor’s Algorithm}) )</td>
<td>5M</td>
</tr>
<tr>
<td>5</td>
<td>Pre-computations and ( z'': ) ( d'' = zw''; z'' = d''w''; S_0 = d''s''_0; c'' = c’’z’’; S_1 = d''c’’w’’; S_1 = zS_1; S_0 = c’’c; S_3 = (c’’w’’)(^2; )</td>
<td>5M, 2S</td>
</tr>
<tr>
<td>6</td>
<td>Compute ( \hat{u}'': ) ( \hat{v}_1 = S_1\hat{v}_1; u_1 = 2S_0 - S_0'; ) ( u''_0 = u''_0 + 2\hat{v}_1 + 2S_3u_1; )</td>
<td>2M, 1S</td>
</tr>
<tr>
<td>7</td>
<td>Pre-computations and ( \hat{d}' = -(2v + h) \mod u'': ) ( u_1 = d''\hat{d}''; u_0 = u_0u_1; u_1 = u_1u_1; ) ( \hat{v}_0 = S_1u_1; )</td>
<td>4M</td>
</tr>
<tr>
<td>8</td>
<td>Set up system for ( s = s + s' \equiv (2v + h)/u \mod u': ) ( m_2 = u_1' - u_1; m_3 = u_0 - u_0' + m_1 = z't_4u_1 + u_0'm_3; m_2 = u_0'm_3; )</td>
<td>3M</td>
</tr>
<tr>
<td>9</td>
<td>Solve system for ( s = s'x + s_0 = \hat{d}'/u \mod u': ) ( u_0 = (m_2 - \hat{v}_0')(r_1' - m_1); w_1 = (\hat{v}_0' - m_2)(r_1' + m_1); w_2 = (m_1 - \hat{v}_0')(r_1' - m_3); w_3 = (\hat{v}_0' - m_3)(r_1' + m_3); ) ( s''_0 = w_3 - w_1; ) ( \hat{S}_1 = w_2 - w_3; ) ( d'' = w_2 + w_3 - w_0 - w_1 - 2(m_2 - m_4)(m_1 + m_3); )</td>
<td>5M</td>
</tr>
<tr>
<td>10</td>
<td>Extract ( s ) from ( s = s' + s ) compute ( s = s' + s; ) ( u_0 = S_0d''; u_1 = zS_0'; ) ( S_1 = S_1 - d''; ) ( \hat{S}_1 = z'S_1; ) ( \hat{s}_0 = \hat{s}_0' - w_0; ) ( s_1 = \hat{s}_1' - 2\hat{d}'; ) ( S_0 = w_1 - 2w_0; )</td>
<td>3M</td>
</tr>
<tr>
<td>11</td>
<td>Pre-computations: ( \hat{s}_0 = \hat{s}_0'; S_0 = s_0; ) ( \hat{d} = c''\hat{d}''; \hat{D} = \hat{d}'; ) ( \hat{S}_1 = \hat{d}'s''_0; ) ( S_1 = dS_1'; )</td>
<td>4M, 3S</td>
</tr>
<tr>
<td>12</td>
<td>Compute ( u_0'' = u_0'' \mod u ) and ( u_0''; ) ( u_0'' = d''(s''_0' + \hat{s}_0'; u_0); u_0'' = u_0'' + u_0\hat{s}_0''; )</td>
<td>5M</td>
</tr>
<tr>
<td>13</td>
<td>Compute ( \hat{u}'': ) ( u_3'' = 2S_0 - \hat{D} - \hat{s}_0'\hat{u}_0''; u_1'' = u_1'' + \hat{u}_1\hat{s}_0''\hat{u}_0''; )</td>
<td>2M</td>
</tr>
<tr>
<td>14</td>
<td>Compute weighted ( v'' \equiv -h - l' \mod u''; ) ( u_0'' = d''\hat{s}_1; v_0'' = v_0u_0; ) ( v_0'' = v_0u_0; ) ( v_1'' = (u_1'' - \hat{S}_0)u_1'' + \hat{s}_0''u_0'' + \hat{e}_1; ) ( v_2'' = \hat{s}_0''\hat{u}_0'' + u_0''u_0'' - \hat{s}_0''\hat{u}_0''; )</td>
<td>8M</td>
</tr>
<tr>
<td>15</td>
<td>Adjust ( u'' ) to have weight ( z'' ) and compute ( z'': ) ( u_0'' = u_0''S_1; u_1'' = u_1'S_1; z'' = \hat{s}_0''S_1; )</td>
<td>3M</td>
</tr>
</tbody>
</table>

Total 55M, 9S, 2C
6.1.5 Comparisons

In Tables 6.10 and 6.11 we present explicit formulae cost differences between previous work and our contributions. We compare our findings to most special cases and settings that have been look at in literature. In our tables we include affine settings where the original four coordinates \([u_1, u_0, v_1, v_0]\) are the input, and settings where there are two auxiliary coordinates over the original four. There are two different six coordinate settings, the “geometric” setting introduced in [9] denoted as 6GE and the “semi-affine” setting introduced in [2] denoted as 6SA. Note that the six coordinate geometric setting requires the least number of field operations for two inversion tripling just by implementing an add right after a double. None of the techniques used for simplifying the Harley method double-and-add tripling formulae apply to the geometric method, making the one inversion formula much slower, and so it is not presented here. The explicit formulae for two inversion tripling using the 6GE setting can be found in Appendix A.1.

Furthermore, we include comparisons of five and eight coordinate versions of the projective setting, meaning the regular projective setting, and the “new coordinate” projective setting introduced in [27] that has three auxiliary coordinates. See Section 5.3 for a description of these settings. For even characteristic we further include the special case where \(h(x) = x, f_4 = f_3 = f_2 = 0\) that has been looked at by others [40]. Refer to Table 6.11 for odd characteristic comparisons and Table 6.10 for even characteristic comparisons.

6.2 Assessment for Double-base Algorithms

Here we assess which coordinate systems are best suited for implementation of divisor double-base scalar multiplication algorithms based on Tables 6.10 and 6.11.

- In even characteristic where \(h(x) = x\), the regular 4 coordinate affine setting is best suited for implementing affine double-base scalar multiplication algorithms and the regular 5 coordinate projective setting is best suited for
Table 6.10: Even Characteristic Tripling Formula Comparisons.

<table>
<thead>
<tr>
<th>Inv</th>
<th># coords</th>
<th>Conditions</th>
<th>Triple</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>M</td>
</tr>
<tr>
<td>Even Characteristic</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4GE</td>
<td>This work</td>
<td>38</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>This work</td>
<td>44</td>
</tr>
<tr>
<td></td>
<td>6GE</td>
<td>This work</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>6SA</td>
<td>This work</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>This work</td>
<td>41</td>
</tr>
<tr>
<td></td>
<td>6SA</td>
<td>This work</td>
<td>52</td>
</tr>
<tr>
<td>-</td>
<td>5</td>
<td>[27] This work</td>
<td>83</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>This work</td>
<td>83</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>66</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>[27] This work</td>
<td>31</td>
</tr>
<tr>
<td></td>
<td>6GE</td>
<td>This work</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>6SA</td>
<td>This work</td>
<td>30</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>This work</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td>6SA</td>
<td>This work</td>
<td>42</td>
</tr>
<tr>
<td>-</td>
<td>5</td>
<td>[40] This work</td>
<td>73</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>This work</td>
<td>54</td>
</tr>
</tbody>
</table>

$h(x) = x, f_2 = f_3 = f_4 = 0$
implementing projective non-adjacent form scalar multiplication algorithms.

The explicit formula for doubling in the affine setting when $h(x) = x$ comes from [39] and is given in Table 5.18, the affine addition, tripling and all projective formulae can be easily obtained by setting the values of $h(x)$ accordingly in their respective tables.

- In odd characteristic when implementing affine double-base scalar multiplication algorithms it is not clear which affine setting is best, the 6SA coordinate setting which has faster doubling and additions or 4 coordinate affine setting which has faster triplings. It is also not clear which projective setting is best, the 8 coordinate projective setting has faster doubling, but the 5 coordinate settings have faster addition and tripling. Explicit formulae for addition and doubling in both the 6SA setting and 8 coordinate projective setting are given in Tables 5.19, 5.20, 5.21 and 5.22. The explicit formulae for tripling in both

<table>
<thead>
<tr>
<th>Inv I</th>
<th># Coords</th>
<th>Conditions</th>
<th>Triple M</th>
<th>S</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4</td>
<td>[12]</td>
<td>44 7</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>This work</td>
<td>38 6</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>6GE</td>
<td>9</td>
<td></td>
<td>35 10</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>6SA</td>
<td>2</td>
<td>This work</td>
<td>36 5</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>-</td>
<td>- 7</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>6SA</td>
<td>2</td>
<td>This work</td>
<td>46 7</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>5</td>
<td>[27]</td>
<td>85 10 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>[9]</td>
<td>73 13 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>[2]</td>
<td>57 9 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>This work</td>
<td>55 9 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>[27]</td>
<td>79 14 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>[2]</td>
<td>55 10 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>This work</td>
<td>53 10 2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 6.11: Odd Characteristic Tripling Formula Comparisons.
settings are given in Tables 6.12 and 6.13 where the one inversion formula is used in the affine case assuming the usual \( I = 30M \).

6.3 Dedicated Tripling Methods

In this section we describe two methods for computing a dedicated divisor class tripling, the geometric method and the algebraic NUCUBE method [22]. Both methods compute a divisor class tripling directly instead of computing a double then addition.

Let \( D = [u, v] \) be the input divisor class to be tripled. Recall that \( D_{sr} = [U, V] \) is the unreduced representation of the output divisor class \( 3D = D'' = [u'', v''] \). The geometric method follows the same path as geometric doubling (Section 3.3.2) but produces an interpolating polynomial \( l' = V \) that interpolates the two non trivial points in the support of \( D \) with multiplicity three instead of two. The NUCUBE method algebraically computes \( V \) directly and uses a continued fraction expansion technique to compute the reduced output divisor class \( D'' \). Both methods produce a degree five polynomial \( V \), resulting in an increase of complexity due to dealing with higher degree polynomials and extra reduction steps.

We first present the geometric method to dedicated tripling and its drawbacks, followed by the NUCUBE method. Finally we give a summary for the dedicated tripling methods.

6.3.1 Geometric

In this section we describe a dedicated tripling algorithm using the geometric method. One way to implement a triple without using the double-and-add method is to compute the interpolating polynomial for the points in the support of the divisor class to be tripled with a multiplicity of three. Several problems arise using this method, most notably solving a four by four linear system and the need for a reduction step. We give an overview of this method and then a summary of its drawbacks relative to the double-and-add methods presented in Section 6.1.
Table 6.12: Odd Characteristic 6SA 1 Inversion Tripling Formula

<table>
<thead>
<tr>
<th>IN:</th>
<th>Reduced divisor $D = [u, v, Z, z]$ with $u = x^2 + u_1x + u_0$, $v = (v_1/Z)x + v_0/Z$ $h = 0$, $f = x^5 + f_4x^4 + f_3x^3 + f_2x^2 + f_1x + f_0$</th>
</tr>
</thead>
</table>

| OUT: | Reduced divisor $D'' = [u'', v'', Z'', z''] = 3D$ with $u'' = x^2 + u''_1x + u''_0$, $v'' = (v''_1/Z')x + v''_0/Z''$. |

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><strong>Compute</strong> $\hat{v} \equiv h + 2v \bmod u$ $\hat{v}_1 = 2v_1$; $\hat{v}_0 = 2v_0$;</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td><strong>Compute</strong> $k \equiv (f - hv - v^2)/u \bmod u$ $w_0 = v'_1^2$; $w_1 = u'_1^2$; $w_2 = f_3 + w_1$; $w_3 = 2u_0$; $k_0 = z(2w_1 + w_2 - w_3)$; $k_0 = z(u_1(2u_3 - w_2) + f_3) - w_0$;</td>
<td>3M, 2S</td>
</tr>
<tr>
<td>3</td>
<td><strong>Setup system of equations for</strong> $s \equiv k/\hat{v} \bmod u$: $m_1 = -k_1$; $m_2 = k_0$; $m_3 = -v_1$; $m_4 = \hat{v}_0$; $r_1 = -v_0 + \hat{v}_1u_1$; $r_0 = \hat{v}_1u_0$;</td>
<td>2M</td>
</tr>
<tr>
<td>4</td>
<td><strong>Solve system of equations for</strong> $s' = sd' = s(x + s_0)$: $w_0 = (m_3 - k_0)(k_1 - m_1)$; $w_1 = (-k_0 - m_2)(k_1 + m_1)$; $w_2 = (m_4 - k_0)(k_1 - m_3)$; $w_3 = (-k_0 - m_4)(k_1 + m_3)$; $d' = w_2 + w_3 - w_0 - w_1 - 2(m_2 - m_4)(m_1 + m_3)$; $s''_0 = w_0 - w_1$; $c' = w_2 - w_3$; (if $d' = 0$ branch to Cantor’s Algorithm)</td>
<td>5M</td>
</tr>
<tr>
<td>5</td>
<td><strong>Compute weights</strong> $Z'; z'$ and pre-computations: $Z' = c'Z$; $z' = Z^{m_2}$; $w_5 = d'c'$; $\hat{v}_1 = v_1w_5$; $\hat{v}_0 = v_0w_5$;</td>
<td>4M, 1S</td>
</tr>
<tr>
<td>6</td>
<td><strong>Compute</strong> $d'$: $w_6 = ds''_0; u'_1 = 2w_6 - z'; u'_0 = s''_0 + 2(\hat{v}_1 + z'u_1)$;</td>
<td>2M, 1S</td>
</tr>
<tr>
<td>7</td>
<td><strong>Adjust</strong> $u$: $d_3 = d'^2$; $\hat{u}_1 = u_1d_3$; $\hat{u}_0 = u_0d_3$;</td>
<td>2M, 1S</td>
</tr>
<tr>
<td>8</td>
<td><strong>Set up system for</strong> $s'' \equiv -(s - 2\hat{v}/u)/s_1 \bmod u'$: $r_0 = -2\hat{v}_0$; $r_1 = -2\hat{v}_1$; $m_3 = u'_1 - u_1$; $m_4 = u_0 - u'_0$; $m_1 = d_2m_4 + u'_1m_3$; $m_2 = -u'_0m_3$;</td>
<td>3M</td>
</tr>
<tr>
<td>9</td>
<td><strong>Solve system for</strong> $s'' \equiv -(s - 2\hat{v}/u)/s_1 \bmod u'$: $w_0 = (m_3 - r_0)(r_1 - m_1)$; $w_1 = (-r_0 - m_2)(r_1 + m_1)$; $w_2 = (m_4 - r_0)(r_1 - m_3)$; $w_3 = (-r_0 - m_4)(r_1 + m_3)$; $d' = w_2 + w_3 - w_0 - w_1 - 2(m_2 - m_4)(m_1 + m_3)$; $w_7 = (w_2 - w_3)d_2 - d'$; $\bar{d} = d_2w_7$; $s''_0 = (w_0 - w_1)d_2 - \bar{d}w_6$; $t = d'd_2^{-1}$; $s''_0 = s''_0/t$; $\bar{c} = d't$;</td>
<td>11, 11M</td>
</tr>
<tr>
<td>10</td>
<td><strong>Pre-computations and adjust</strong> $u'$ and $\hat{v}$ (weighted $v$) : $Z'' = d'\hat{c}Z'; z'' = Z'^{m_2}$; $\hat{c} = v_2t$; $u'_0 = \hat{c}u'_0$; $u'_1 = \hat{c}u'_1$; $\hat{v}_1 = v_1\hat{c}$; $\hat{v}_0 = v_0\hat{c}$;</td>
<td>7M, 1S</td>
</tr>
<tr>
<td>11</td>
<td><strong>Compute</strong> $u''$: $u''_0 = 2s''_0 + u_1 - u'_1 - z''; u''_0 = s''_0 + (u_1 - u'_1)(2s''_0 - u'_1) + + (w_0 - u'_0) + 2\hat{v}_1 + (u_1 + u'_1)z''$;</td>
<td>2M, 1S</td>
</tr>
<tr>
<td>11</td>
<td><strong>Compute</strong> $v'' \equiv -h - h' \bmod u''$: $w_0 = (u_1 - s''_0)(u_1 - u'_1); v'_1 = w_0 + u''_0 - u_0 - v_1; v''_0 = s''_0(w_0 - u''_0) + u''_0(u_1 - u'_1)) - v_0$;</td>
<td>3M</td>
</tr>
<tr>
<td>Total</td>
<td>1, 44M, 7S, 0C</td>
<td></td>
</tr>
</tbody>
</table>
Table 6.13: Odd Characteristic 8 Coordinate Tripling Formula

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Compute $v \equiv h + 2v \mod u$; $v_1 = 2v_1$; $v_0 = 2v_0$;</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>Compute $k \equiv (f - he - e^2)/u \mod u$; $w_5 = v_1^2$; $w_1 = u_1^2$; $w_2 = z_1^2$; $w_3 = z_1 u_0$; $w_4 = w_2 f_3 + w_1$; $w_5 = 2w_5$; $k_1 = z_2(2w_1 + w_4 - w_5)$; $b_0 = z_2(u_2 w_2 - w_5 + w_2 f_3 + f_3 x + f_0)$</td>
<td>5M, 3S, 2C</td>
</tr>
<tr>
<td>3</td>
<td>Set up system of equations for $s'/s \equiv k/s \equiv v \mod u$;</td>
<td>3M</td>
</tr>
<tr>
<td>4</td>
<td>Solve system for $s_0 = s_0/s_1$ and $c = 1/s_1$; $w_0 = (m_2 - r_0)(r_1 - m_1)$; $w_2 = (-r_0 - m_2)(r_1 + m_1)$; $w_3 = (m_4 - r_0)(r_1 - m_3)$; $w_4 = (-r_0 - m_4)(r_1 + m_3)$; $b_0 = w_1 - w_2$; $c = w_3 - w_4$; $d' = w_3 + w_4 - w_1 - w_2 - 2(m_2 - m_4)(m_1 + m_3)$; (if $d' = 0$ branch to Cantor's Algorithm)</td>
<td>5M</td>
</tr>
<tr>
<td>5</td>
<td>Pre-computations and weights; $w_5 = u Z_4$; $Z_1 = d' Z_1'$; $Z_2 = Z_2 w_6$; $Z_2' = Z_2 w_2$; $w_6 = s_0/2 u_1$; $w_0 = s_0^2 + 2(v_1 + u_2 u_1)$;</td>
<td>4M, 2S</td>
</tr>
<tr>
<td>6</td>
<td>Adjust $v$; $w_8 = d' c'$; $v_1 = w_8 v_1$; $v_0 = w_8 v_0$;</td>
<td>3M</td>
</tr>
<tr>
<td>7</td>
<td>Compute $u'$; $u_1' = 2w_7 - u_2'$; $u_0' = s_0^2 + 2(v_1 + u_2 u_1)$;</td>
<td>1M, 2S</td>
</tr>
<tr>
<td>8</td>
<td>Adjust $u$ to have the same weight as $u'$; $w_9 = Z_2' d'$; $u_1 = w_9 u_1$; $u_0 = w_9 u_0$;</td>
<td>3M</td>
</tr>
<tr>
<td>9</td>
<td>Set up system for $s = (v - v')/u \mod u'$; $m_1 = -2v_1$; $m_2 = 2v_0$; $m_3 = u_1 - u_1'$; $m_4 = u_0 - u_0'$; $r_1 = z_1 m_4 - u_1 m_3$; $r_0 = -u_1' m_3$;</td>
<td>3M</td>
</tr>
<tr>
<td>10</td>
<td>Solve system for $s = s_1 x + s_0 = v'/u \mod u'$; $w_9 = (m_2 - r_0)(r_1 - m_1)$; $w_2 = (m_4 - r_0)(r_1 - m_3)$; $w_3 = (r_0 - m_4)(r_1 + m_3)$; $b_0 = w_9 - w_1$; $w_4 = w_2 - w_3$; $l_1 = w_4 z_1'$; $d'' = w_3 + w_4 - w_1 - 2(m_2 - m_4)(m_1 + m_3)$; $w_5 = l_0 + w_4(u_1' - u_1)$; $w_0 = l_1 - d''$; $w_3 = z_1' w_5$; $w_5 = w_0 d'' z_0'' = -w_5 + w_4$;</td>
<td>9M</td>
</tr>
<tr>
<td>11</td>
<td>Compute $u = s - s'$; $s_1 = l_1 - 2d''$; $s_0 = -w_5 + w_4$;</td>
<td>-</td>
</tr>
<tr>
<td>12</td>
<td>Compute $u_0'' = u_0''$ mod $u$ and $u_0''$; $w_0 = l_1 s_1$; $w_3 = s_0^2 + w_8 u_0 u_1$; $u_0'' = u_0 + u_0 u_7$;</td>
<td>4M, 1S</td>
</tr>
<tr>
<td>13</td>
<td>Pre-computations and weights; $Z_1' = w_3 z_1'$; $z_1' = z_1$; $z_2' = d' Z_1'$; $Z_2' = z_2' Z_2$; $w_3 = w_3 w_9$; $w_0 = d' w_3$; $l_1 = w_0 v_1$; $v_0' = w_0 v_0$;</td>
<td>6M, 2S</td>
</tr>
<tr>
<td>14</td>
<td>Compute $u_1''$; $u_1'' = 2w_3 - u_1'' u_1'' + z_2'' = u_1'' + u_1 u_7$;</td>
<td>2M</td>
</tr>
<tr>
<td>15</td>
<td>Compute weighted $v'' \equiv -h - l \mod u''$; $v''_1 = -(u_1'' - u_1') u_1'' + z_1''(u_0'' - v_0')$; $v''_0 = u_0'' u_0'' + u_1'' u_1'' - z_1'' v_0'$;</td>
<td>5M</td>
</tr>
</tbody>
</table>

Total | 53M, 10S, 2C |
Overview of the Geometric Method

Given the input divisor class $D = [u, v]$, we want to compute $3D = D'' = [u'', v'']$. Tripling involves building a six by six linear system of equations that solves for the coefficients of $l'$, a polynomial that interpolates the non-trivial points in the support of $D$ with multiplicity three. The degree of $l'$ is five because it now interpolates six points instead of four as described in geometric doubling.

To obtain the simplest equations we use the approach of reducing the substitution of $l' = y$ into the curve $C$ modulo $u^3$ instead of $u^2$ to ensure that a multiplicity of three for the points in the support of $D$ is achieved and take the simplest ones. The equations produced are used to set up the six by six linear system that needs to be solved for the coefficients of $l'$. The system can be reduced to a four by four similarly to the reduction done in Section 4.2.11. The complexity of the set up is already much higher than in geometric doubling.

Next we come across one of the major problems with using this method. The solution to solving a four by four linear system of equations using Cramer’s rule takes 40 field multiplications. This cost is already almost the total cost of our fastest one inversion double-and-add tripling formula. Solving the system produces four of the six coefficients needed, so the last two are computed using the equations produced for the system as done in geometric doubling. The rest is computed similarly to geometric doubling, equating coefficients to produce a reduced representation of $D''$. Since $l'$ originally has degree five, one more reduction step is needed to compute the proper reduced form for the output divisor class $D''$. A summary of this method is given in Algorithm 6.4.

Summary of Drawbacks

The problems that arise in the geometric dedicated tripling method are:

- The linear equations used to create the system of equations are much more complex than in geometric doubling.
- The solution to a four by four linear system of equations is required resulting
Algorithm 6.4 Geometric Dedicated Tripling

Input: $D = [u, v]$
Output: $D'' = [u'', v''] = 3D$

Compute $l'$:
1: Construct polynomial equations by reducing the substitution of $l' = y$ into the curve $C$ modulo $u^3$
2: Set up six by six system of equations linear in coefficients of $l'$ and reduce to four by four system
3: Solve linear system for four of the coefficients of $l'$
4: Solve for the other two coefficients using equations from Step 1
5: Compute $u''' = \frac{v''^2 + hl' - f}{u^2}$ made monic
6: Compute $v'' \equiv -h - l' \mod u''$
7: Compute $u'' = \frac{v''^2 + hu'' - f}{u^2}$
8: Compute $v'' \equiv -h - v''' \mod u''$

in 40 multiplications using Cramer’s rule.

- The degree of $l' = V$ is five, so an extra reduction step is required to get the output into reduced form.

All of these problems greatly contribute to the total number of field operations needed to compute tripling using this route making this method not viable. Specifically, our fastest one inversion tripling formulae require only 42 multiplications and 7 squares, practically the cost of Step 3 in Algorithm 6.4.

6.3.2 NUCUBE

In this section we describe a dedicated tripling method called NUCUBE introduced by Jacobson et. al. in [22] where the authors describe ideal cubing algorithms in imaginary function fields. The Jacobian of a hyperelliptic curve is isomorphic to a factor group of function fields, and the correspondence between ideals in those fields and divisors of the corresponding hyperelliptic curve is well-known [23]. The authors describe an algorithm that works for imaginary hyperelliptic curves of any genus and arbitrary input; for this description we specialize to our genus two frequent case setting only. We give an overview of
this method and then a summary of its drawbacks relative to the double-and-add methods presented in Section 6.1.

Overview of the NUCUBE Method
Recall $D = [u, v]$ is the input divisor class to be tripled and that $D_{sr} = [U, V]$ is the unreduced representation of the output divisor class $3D = D'' = [u'', v'']$. The main idea is that a dedicated tripling algorithm using the algebraic Harley method computes $V$ that has degree 5. The authors in [22] give an exposition of this process. Once $V$ is computed, two rounds of Cantor’s reduction step can be taken to produce the output polynomials representing $D''$. Instead the authors use continued fraction expansions to circumvent the computation of $D_{sr}$ and come to the reduced output directly, reducing the complexity of the reduction process. We show that either way the field operation cost is higher than our double-and-add tripling formulae, even before reaching the reduction step. We note that both reduction paths require every coefficient of $s$, so no coefficients can be omitted in the computation of $s$. Using the continued fractions path for reduction did produce fewer field operation counts than applying Cantor’s reduction step twice so we present the NUCUBE algorithm, refer to Algorithm 6.5 for an overview.

Summary of Drawbacks
The problems that arise in the algebraic dedicated tripling method are:

- Cannot compute $t$ and $k$ modulo $u$ as done in Harley doubling and addition because $s$ in this setting is computed modulo $u^2$ instead of $u$, so an extended Euclidean algorithm is used to compute $t$. The polynomial $k$ has degree three instead of one and polynomial $t$ has degree one.

- Several polynomial multiplications and reductions modulo $u^2$ are required to compute $s$ where the intermediate polynomials grow up to degree five in Step 5. In odd characteristic this step costs 46M, 2S, already more operations than
Algorithm 6.5 NUCUBE Dedicated Tripling

**Input:** \( D = [u, v] \)

**Output:** \( D'' = [u'', v''] = 3D \)

**Composition**

1: Compute \( \hat{v} = 2v - h \)
2: Compute \( k = \frac{t + hv - v^2}{u} \)
3: Compute \( t \) where \( 1 = ru + t\hat{v} \) (omit computation of \( r \))
4: Compute \( u_2 = u^2 \)
5: Compute \( s = kt(2 + t(ktu - \hat{v})) \) (mod \( u_2 \))

**Reduction**

6: Compute \( q \) and \( r \) where \( u_2 = qs + r \)
7: Compute \( V' = su + v \) (mod \( u_2 \))
8: Compute \( m_1 = \frac{(uk + (V'-v)q)}{u_2} \)
9: Compute \( m_2 = \frac{r(v+V'-h)+k}{u_2} \)
10: Compute \( u'' = rm_1 - qm_2 \)
11: Compute \( V'' = \frac{u''+u'}{4} - v + h \)
12: Compute \( v'' = V'' \) (mod \( u'' \))

our fastest one inversion tripling formula. In even characteristic this step costs 31M, 2S, 2C, more than our fastest tripling formula when combined with the four previous steps. Notice that the cost is much lower in even characteristic because two of the coefficients of \( u_2 = u^2 \) have factors of 2 in them equating to 0.

- The degree of \( V' = V - v \) is five (before reducing modulo \( u_2 \)), so either an extra reduction step or the continued fractions reduction method are required to get the output into reduced form.

- All coefficients of \( s \) are required in the reduction portion of the algorithm, namely every coefficient of \( s \) is computed.

All of these problems, most notably Step 5, greatly contribute to the total number of field operations needed to compute tripling using this route; making this method not viable.
6.3.3 Dedicated Tripling Summary

The dedicated tripling algorithms presented in this section are more complex than their double-and-add counterparts described in Section 6.1 due to the intermediate polynomial $V$ (part of the unreduced representation of $D''$) growing to have a degree of five. Dedicated tripling methods deal with larger degree intermediate polynomials and extra reduction steps, resulting in an increase to the complexity of the resulting explicit formulae. Tripling specific techniques for the double-and-add method omit the computation of the first degree three $V$ polynomial and only deal with a degree three $V$ polynomial representing part of the unreduced divisor $D''$ in the addition portion resulting in less complex and faster explicit formulae.

6.4 Summary

All versions of our double-and-add Harley method tripling formulae are the fastest to date for regular affine and projective settings. The improvements we made in this work extend to alternate settings that have been considered in literature as well, producing the fastest explicit formula over all in almost all cases and settings.
Chapter 7

Conclusion
Elliptic curve cryptography is an accepted cryptographic standard in wireless technologies, smart cards and other areas of communication. The smaller bit length that is required for the same security as compared to more traditional protocols like RSA is ideal for settings where memory is limited. Genus 2 hyperelliptic curve cryptography has been shown to be computationally comparable \[39\] to elliptic curve cryptography although elliptic curves hold many speed records \[3\].

The most important and time consuming operation in hyperelliptic curve protocols is divisor class scalar multiplication. There are many approaches to computing a scalar multiple of a divisor class. We focus on the generally used approach of using a non-adjacent form algorithm that only takes advantage of additions and doubles, or the newer approach of using a double-base algorithm that also takes advantage of triples. We were able to directly increase the efficiency of both non-adjacent form and double-base scalar multiplication algorithms by reducing the number of field operations required for the doubling, addition and tripling operations.

Based on our results, we recommend the following settings for efficient divisor class arithmetic on genus two imaginary hyperelliptic curves given in Weierstrass form:

<table>
<thead>
<tr>
<th>Table 7.1: Recommendations for Non-Adjacent Form Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Odd Characteristic</strong></td>
</tr>
<tr>
<td>Setting</td>
</tr>
<tr>
<td>---------</td>
</tr>
<tr>
<td>6SA</td>
</tr>
<tr>
<td>New Coord 8</td>
</tr>
<tr>
<td><strong>Even Characteristic</strong></td>
</tr>
<tr>
<td>( h(x) = x, f_2 = f_3 = f_4 = 0 )</td>
</tr>
<tr>
<td>Setting</td>
</tr>
<tr>
<td>---------</td>
</tr>
<tr>
<td>Affine 4</td>
</tr>
<tr>
<td>Projective 5</td>
</tr>
</tbody>
</table>
Table 7.2: Recommendations for Double-Base Implementation

<table>
<thead>
<tr>
<th>Setting</th>
<th>Double</th>
<th>Addition</th>
<th>Triple</th>
</tr>
</thead>
<tbody>
<tr>
<td>Affine 4</td>
<td>1I, 20M, 5S</td>
<td>1I, 20M, 3S</td>
<td>1I, 42M, 7S</td>
</tr>
<tr>
<td>6SA</td>
<td>1I, 19M, 4S</td>
<td>1I, 19M, 1S</td>
<td>1I, 44M, 7S</td>
</tr>
<tr>
<td>Projective 5</td>
<td>31M, 5S, 2C</td>
<td>34M, 2S</td>
<td>55M, 9S, 2C</td>
</tr>
<tr>
<td>New coord 8</td>
<td>29M, 7S, 2C</td>
<td>33M, 4S</td>
<td>53M, 10S, 2C</td>
</tr>
</tbody>
</table>

**Odd Characteristic**

$h(x) = x, f_2 = f_3 = f_4 = 0$

<table>
<thead>
<tr>
<th>Setting</th>
<th>Double</th>
<th>Addition</th>
<th>Triple</th>
</tr>
</thead>
<tbody>
<tr>
<td>Affine 4</td>
<td>1I, 9M, 6S</td>
<td>1I, 19M, 3S</td>
<td>1I, 33M, 5S</td>
</tr>
<tr>
<td>Projective 5</td>
<td>26M, 4S</td>
<td>33M, 2S</td>
<td>52M, 9S</td>
</tr>
</tbody>
</table>

- In even characteristic with specializing the curve equation to have $h(x) = x$ as done in literature, the regular 4 coordinate affine setting has the fastest doubling, addition and tripling formulae. For the projective setting, all three operations are fastest in the standard 5 coordinate projective setting. Assuming an inversion is 10M, the “Affine 4” setting is faster overall for all three operations. Thus, we recommend using the the affine 4 coordinate representation for both non-adjacent form and double-base implementations of scalar multiplication algorithms.

- In odd characteristic, the 6 coordinate semi-affine setting yields the fastest formulae for doubling and addition and is best suited for affine Non-adjacent form scalar multiplication algorithms. Tripling is faster in the regular affine setting so it is unclear which affine setting is best suited for implementing with double-base scalar multiplication algorithms. For inversion-free formulae, the
new coordinates (8 coordinate) representation has faster doubling and tripling but slower addition than the regular projective setting. It is unclear whether the new coordinates setting or the regular projective setting is better suited for non-adjacent form scalar multiplication algorithms. It is also unclear which is better suited for double-base scalar multiplication algorithms. Implementing with the regular projective or new coordinates settings is the most efficient over all when assuming the cost of an inversion to be the usual $30M$.

7.1 Future Work

There are a number of possibilities for extensions of this work. First, all of these techniques can be extended to real hyperelliptic curves in an effort to improve on the formulae described in [12]. They may also improve the state-of-the-art in genus 3.

The fastest implementations of genus two cryptography [3] make use of special classes of curves related to Kummer surfaces [18] and efficiently-computatable automorphisms [16, 15]. It remains to be seen whether an efficient tripling formula on the Kummer surface exists, and whether tripling can be used effectively in conjunction with GLV-type methods.

Very recent work by Costello and Hisil [8], introduces yet another projective coordinate setting that uses one auxiliary element over the regular projective setting. Applying our contributions to that setting may lead to even faster explicit formulae.
Bibliography

discrete logarithms over hyperelliptic curves of large genus over gf(q). In *Theoretical

[2] B. Balamohan. Accelerating the scalar multiplication on genus 2 hyperelliptic curve


2006.

raphy*. Chapman and Hall/CRC, 6000 Broken Sound Parkway NW, Boca Raton, FL,


University of Technology, 2010.

number and function fields. In Advances in Mathematics of Communications, number 2

continued fraction expansions. Advances in Coding theory and Cryptology, 3:201–244,
2007.


[27] T. Lange. Formulae for arithmetic on genus 2 hyperelliptic cruves. Applicable Algebra

[28] K. Lauter. The equivalence of geometric and algebraic group laws for jacobians of genus

[29] K. Lauter. The advantages of elliptic curve cryptography for wireless security. IEEE


Appendix A

A.1 6GE Explicit Formulae in Even Characteristic

In this appendix we present six coordinate “geometric” setting formulae for doubling, addition, and tripling. These formulae are produced by using the methodology from Section 3.3 and is an extension of the work done in [9] to even characteristic fields. An explanation of the six coordinate geometric setting is described in Section 5.3.
<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure</th>
<th>Cost</th>
</tr>
</thead>
</table>
| 1    | Setup system of equations to solve for $l'$:  
$m_1 = U'_1 - u'_0 - U_1 + u_0$;  
$m_2 = U_0 - U'_0$;  
$m_3 = u'_1 - u_1$;  
$m_4 = u_0 - u'_0$;  
r_0 = v'_0 - v_0;  
r_1 = v'_1 - v_1$; | - |
| 2    | Solve system of equations for $l'$:  
l_2 = r_0m_1 + r_1m_2;  
l_3 = r_0m_3 + r_1m_4;  
d = m_4m_1 - m_2m_3; | 6M |
| 3    | Compute $l'_3, 1/l'_3, 1/l'_3, l'_3$:  
$w_1 = (dl_3)^{-1}$;  
w_2 = w_1d;  
w_3 = w_2d = (1/l'_3);  
w_4 = w_2^2;  
w_5 = l_2w_2;  
l'_3 = l'_3w_1 = (l'_3); | 1I, 5M, 2S |
| 4    | Compute $u''$:  
$u'' = m_3 + h_2w_3 - w_4$;  
$u'_0 = -u_0 - u'_0 - u_1u'_1 + u'_1(u_1 + u'_1) + w_5^2 + (w_5h_2 + h_1)w_3$; | 3M, 1S |
| 5    | Compute $v'' = -h - l' \mod u''$:  
$U''_1 = u''u_1^2;  
U''_0 = u''u_0u'_1;  
v''_0 = w_5(u_0 - u''_0) + U''_0 - U_0$;  
v''_1 = w_5(u_1 - u''_1) + U''_1 - u''_0 - U_1 + u_0$;  
v''_0 = l'_3v''_0 - v_0 - h_2u''_0;  
v''_1 = l'_3v''_0 - v_1 - h_1 + h_2u''_1$; | 5M, 1S |
| Total | | I, 19M, 4S |
### Table A.2: 6GE Even Char Doubling Formula

<p>| IN: | Reduced divisor $D = [u, v, U_1, U_0]$ with $u = x^2 + u_1x + u_0$, $v = v_1x + v_0$, $U_1 = u_1^2$, $U_0 = u_1u_0$ $h = h_2x^2 + h_1x + h_0$, $f = f_5x^5 + f_4x^4 + f_3x^3 + f_2x^2 + f_1x + f_0$ |
| OUT: | Reduced divisor $D' = [u', v', U'_1, U'_0] = 2D$ with $u' = x^2 + u'_1x + u'_0$, $v' = v'_1x + v'_0$, $U'_1 = u_1^2$, $U'_0 = u_1u_0$. |</p>
<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Setup system of equations to solve for $l'$: $m_1 = h_0 - h_2v_0$; $m_2 = u_0h_1 - h_2U_0$; $m_3 = h_2u_1 - h_1$; $m_4 = u_1h_1 - h_2(U_1 + u_0) + h_0$; $r_0 = -h_1v_1 - v_0^2 - h_2v_0$; $r_1 = U_1 - h_2v_1$;</td>
<td>1S, 3C</td>
</tr>
<tr>
<td>2</td>
<td>Solve system of equations for $l'$: $l_2 = r_0m_1 + r_1m_2$; $l_3 = r_0m_3 + r_1m_4$; $d = m_4m_1 - m_2m_3$;</td>
<td>6M</td>
</tr>
<tr>
<td>3</td>
<td>Compute $l_3/l_4$, $1/l_3^2$, $1/l_3^2$, $1/l_3$: $w_1 = (dl_3)^{-1}$; $w_2 = w_1d$; $w_3 = w_2d = (1/l_3)$; $w_4 = w_2^2$; $w_5 = l_2w_2$; $l_4/l_2 = l_2w_1 = (l_4)$;</td>
<td>1I, 5M, 2S</td>
</tr>
<tr>
<td>4</td>
<td>Compute $u'$: $u'_1 = h_2w_3 - w_4$; $u'_0 = w_2^2 + (w_3h_2 + h_1)w_3 - U_1$;</td>
<td>1M, 1S</td>
</tr>
<tr>
<td>5</td>
<td>Compute $v' = -h - l' \mod u'$: $U'_1 = u'_1^2$; $U'_0 = u'_0u'_1$; $v'' = w_5(u_0 - u'_0) + U'_0 - U_0$; $v'_1 = w_5(u_1 - u'_1) + U'_1 - u'_0 - U_1 + u_0$; $v'_0 = l'_3v''_0 - v_0 - h_0 + h_2u'_0$; $v'_1 = l'_3v''_1 - v_1 - h_1 + h_2u'_1$</td>
<td>5M, 1S</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>1, 17M, 5S, 3C</td>
</tr>
</tbody>
</table>

173
### Table A.3: 6GE Even Char Tripling Formula

<table>
<thead>
<tr>
<th>IN:</th>
<th>Reduced divisor $D = [u, v, U_1, U_0]$ with $u = x^2 + u_1x + u_0, v = v_1x + v_0, U_1 = u_1^2, U_0 = u_1u_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$h = h_2x^2 + h_1x + h_0, \quad f = f_3x^5 + f_4x^4 + f_3x^3 + f_2x^2 + f_1x + f_0$</td>
</tr>
</tbody>
</table>

| OUT: | Reduced divisor $D'' = [u'', v'', U''_1, U''_0] = 2D$ with $u'' = x^2 + u''_1x + u''_0, v'' = v''_1x + v''_0, U''_1 = u''_1, U''_0 = u''_1u''_0$ |

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Setup system of equations to solve for $l'$: $m_1 = h_0 - h_2v_0; m_2 = u_0h_1 - h_2U_0; m_3 = h_2u_1 - h_1; m_4 = u_1h_1 - h_2(U_1 + u_0) + h_0; r_0 = -h_1v_1 - v_1^2 - h_2v_0; r_1 = U_1 - h_2v_1;$</td>
<td>1S, 3C</td>
</tr>
<tr>
<td>2</td>
<td>Solve system of equations for $l'$: $l_2 = r_0m_1 + r_1m_2; l_3 = r_0m_3 + r_1m_4; d = m_4m_1 - m_2m_3;$</td>
<td>6M</td>
</tr>
<tr>
<td>3</td>
<td>Compute $v'_3, l'_3, 1/l'_3, 1/l'_3^2, l'_2/l'_3$: $w_1 = (dl'_3)^{-1}; w_2 = w_1d; w_3 = w_2d = (1/l'_3); w_4 = w_3^2; w_5 = l_2w_2; l'_3 = l'_2/w_1 = (l'_3);$</td>
<td>11, 5M, 2S</td>
</tr>
<tr>
<td>4</td>
<td>Compute $u'$: $u'_1 = h_2w_3 - w_4; u'_0 = w_3^2 + (w_5h_2 + h_1)w_3 - U_1;$</td>
<td>1M, 1S</td>
</tr>
<tr>
<td>5</td>
<td>Compute $v' = -h - l' \mod u'$: $U'_1 = u''_1^2; U'_0 = u''_0u'_1; v''_0 = w_5(u_0 - u'_0) + U'_0 - U'_0; v'_1 = w_5(u_1 - u'_1) + U'_1 - u'_0 - U_1 + u_0; v'_0 = l'_3v''_0 - v_0 + h_2u'_0; v'_1 = l'_3v''_0 - v_1 - h_1 + h_2u'_1$</td>
<td>5M, 1S</td>
</tr>
<tr>
<td>6</td>
<td>Setup system of equations to solve for $l'$: $m_1 = U'_1 - u'_0 - U'_1 + u_0; m_2 = U_0 - U'_0; m_3 = u'_1 - u_1; m_4 = u_0 - u'_0; r_0 = v'_0 - v_0; r_1 = v'_1 - v_1;$</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>Solve system of equations for $l'$: $l_2 = r_0m_1 + r_1m_2; l_3 = r_0m_3 + r_1m_4; d = m_4m_1 - m_2m_3;$</td>
<td>6M</td>
</tr>
<tr>
<td>8</td>
<td>Compute $v''_3, l''_3, 1/l''_3, 1/l''_3^2, l''_2/l''_3$: $w_1 = (dl''_3)^{-1}; w_2 = w_1d; w_3 = w_2d = (1/l''_3); w_4 = w_3^2; w_5 = l''_2w_2; l''_3 = l''_2/w_1 = (l''_3);$</td>
<td>11, 5M, 2S</td>
</tr>
<tr>
<td>9</td>
<td>Compute $u''$: $u''_1 = m_3 + h_2w_3 - w_4; u''_0 = -u_0 - u'_0 - u_1u'_1 + u'_1(u_1 + u'_1) + w_3^2 + (w_5h_2 + h_1)w_3;$</td>
<td>3M, 1S</td>
</tr>
<tr>
<td>10</td>
<td>Compute $v'' = -h - l'' \mod u''$: $U''_1 = u''_1^2; U''_0 = u''_0u''_1; v''_0 = w_5(u_0 - u''_0) + U''_0 - U'_0; v''_1 = w_5(u_1 - u'_1) + U''_1 - u'_0 - U_1 + u_0; v''_0 = l''_3v''_0 - v_0 + h_2u''_0; v''_1 = l''_3v''_0 - v_1 - h_1 + h_2u''_1$</td>
<td>5M, 1S</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>21, 36M, 9S, 3C</td>
</tr>
</tbody>
</table>
A.2 Maple Code

In this appendix section we present maple code that produces the linear equations that can be used to solve for $l'$ in our adaptation of the geometric method for divisor class doubling to arbitrary fields. We present how we found all the linear equations in the code below. Note that we used the equations $\Omega_0$, $\Omega_1$, $\Omega_5$ and $\Omega_7$ in Section 3.3 because computing with them requires the least number of field operations.

```maple
F := x -> x^5 + f3*x^3 + f2*x^2 + f1*x + f0;
H := x -> h2*x^2 + h1*x + h0;
C := (x,y) -> y^2 + y*H(x) - F(x);
U := x -> x^2 + u1*x + u0;
V := x -> v1*x + v0;
L := x -> l3*x^3 + l2*x^2 + l1*x + l0;

#############################################################
# # Definitions #
# #
#############################################################

#############################################################
# # Finding the Relations #
# # We compute the basic relations along with the derivative #
# relations and the modulo relations to make sure we use the #
# most efficient relations to solve the system. #
# #
#############################################################

################## First Set of Relations ####################
### Reduce V(x) - L(x) mod U(x).
I0 := simplify(V(x)-L(x),[U(x)],plex(x));
Omega0 := coeff(I0,x,0); Omega1 := coeff(I0,x,1);

################### Derivative Relations #####################
#Take derivatives
dC := implicitdiff(C(x,y),y,x);
dL := diff(L(x),x);

### Reduce dC - (H(x) + 2v(x))dL mod U(x).
I1:=simplify(numer(dC) - (H(x) + 2*(V(x)))*dL,[U(x)],plex(x));
Omega2 := coeff(I1,x,0); Omega3 := coeff(I1,x,1);
```

175
> # Modulo U(x)^2 Relations #
> I2 := simplify(C(x,L(x)),[U(x)^2],plex(x));
> Omega4 := coeff(I2,x,0); Omega5 := coeff(I2,x,1);
> Omega6 := coeff(I2,x,2); Omega7 := coeff(I2,x,3);

> #### List of all relations, using mindeg to simplify. ####
> Omega2:=factor(simplify(Omega2,[Omega0,Omega1],mindeg));
> Omega3:=factor(simplify(Omega3,[Omega0,Omega1],mindeg));
> Omega4:=factor(simplify(Omega4,[Omega0,Omega1],mindeg));
> Omega5:=factor(simplify(Omega5,[Omega0,Omega1],mindeg));
> Omega6:=factor(simplify(Omega6,[Omega0,Omega1],mindeg));
> Omega7:=factor(simplify(Omega7,[Omega0,Omega1],mindeg));