Group Law Computations on Jacobians of Hyperelliptic Curves

Craig Costello[1,2,3][⋆] and Kristin Lauter[3]

[1] Information Security Institute
Queensland University of Technology, GPO Box 2434, Brisbane QLD 4001, Australia
craig.costello@qut.edu.au
[2] Mathematics Department
University of California, Irvine - Irvine, CA 92697-3875, USA
[3] Microsoft Research
One Microsoft Way, Redmond, WA 98052, USA
klauter@microsoft.com

Abstract. We derive an explicit method of computing the composition step in Cantor's algorithm for group operations on Jacobians of hyperelliptic curves. Our technique is inspired by the geometric description of the group law and applies to hyperelliptic curves of arbitrary genus. While Cantor's general composition involves arithmetic in the polynomial ring $\mathbb{F}_q[x]$, the algorithm we propose solves a linear system over the base field which can be written down directly from the Mumford coordinates of the group elements. We apply this method to give more efficient formulas for group operations in both affine and projective coordinates for cryptographic systems based on Jacobians of genus 2 hyperelliptic curves in general form.

Keywords: Hyperelliptic curves, group law, Jacobian arithmetic, genus 2.

1 Introduction

The field of curve-based cryptography has flourished for the last quarter century after Koblitz [31] and Miller [44] independently proposed the use of elliptic curves in public-key cryptosystems in the mid 1980's. Compared with traditional group structures like $\mathbb{F}_p^*$, elliptic curve cryptography (ECC) offers the powerful advantage of achieving the same level of conjectured security with a much smaller elliptic curve group. In 1989, Koblitz [32] generalized this idea by proposing Jacobians of hyperelliptic curves of arbitrary genus as a way to construct Abelian groups suitable for cryptography. Roughly speaking, hyperelliptic curves of genus g can achieve groups of the same size and security as elliptic curves, whilst being defined over finite fields with g times fewer bits[4]. At the same time however, increasing the genus of a hyperelliptic curve significantly increases the computational cost of performing a group operation in the corresponding Jacobian group. Thus, the question that remains of great interest to the public-key cryptography community is, under which circumstances elliptic curves are preferable, and vice versa. At the present time, elliptic curves carry on standing as the front-runner in most practical scenarios, but whilst both ECC and hyperelliptic curve cryptography (HECC) continue to enjoy a wide range of improvements, this question remains open in general. For a nice overview of the progress in this race and of the state-of-the-art in both cases, the reader is referred to the talks by Bernstein [4], and by Lange [39].

Cantor [6] was the first to give a concrete algorithm for performing computations in Jacobian groups of hyperelliptic curves over fields of odd characteristic. Shortly after, Koblitz [32] modified this algorithm to apply to fields of any characteristic. Cantor's algorithm makes use of the polynomial representation of group elements proposed by Mumford [46], and consists of two stages: (i) the *composition* stage, based on Gauss's classical composition of binary quadratic forms, which generally outputs an unreduced divisor, and (ii) the *reduction* stage, which transforms the unreduced divisor into the unique reduced divisor that is equivalent to the sum, whose existence is guaranteed by the Riemann-Roch theorem [33]. Cantor's algorithm has since been substantially optimized in work initiated by Harley [24], who was the first to obtain practical explicit formulas in genus 2, and extended by Lange [34, 38], who, among several others [43, 50, 45, 49], generalized and significantly improved Harley's original approach. Essentially, all of these improvements involve unrolling the

[⋆] This author acknowledges funding from the Australian-American Fulbright Commission, the Gregory Schwartz Enrichment Grant, the Queensland Government Smart State Ph.D. Fellowship, and an Australian Postgraduate Award.

[4] The security argument becomes more complicated once venturing beyond genus 2, where the attacks by Gaudry [17] and others [8, 21, 48] overtake the Pollard Rho method [47].

polynomial arithmetic implied by Cantor's algorithm into operations in the underlying field, and finding specialized shortcuts dedicated to each of the separate cases of input (see [35, §4]).

In this paper we propose an explicit alternative to unrolling Cantor's polynomial arithmetic in the composition phase. Our method is inspired by considering the geometric description of the group law and applies to hyperelliptic curves of any genus. The equivalence of the geometric group law and Cantor's algorithm was proven by Lauter [40] in the case of genus 2, but since then there has been almost no reported improvements in explicit formulas that benefit from this depiction. The notable exception being the work of Leitenberger [42], who used Gröbner basis reduction to show that in the addition of two distinct divisors on the Jacobian of a genus 2 curve, one can obtain explicit formulas to compute the required geometric function directly from the Mumford coordinates without (unrolling) polynomial arithmetic. Leitenberger's idea of obtaining the necessary geometric functions in a simple and elementary way is central to the theme of this paper, although we note that the affine addition formulas that result from our description (which do not rely on any Gröbner basis reduction) are significantly faster than the direct translation of those given in [42].

We use the geometric description of the group law to prove that the interpolating functions for the composition step can be found by writing down a linear system in the ground field to be solved in terms of the Mumford coordinates of the divisors. Therefore, the composition algorithm for arbitrary genera proposed in this work is immediately explicit in terms of arithmetic in $\mathbb{F}_q$, in contrast to Cantor's composition which operates in the polynomial ring $\mathbb{F}_q[x]$, the optimization of which calls for ad-hoc attention in each genus to unravel the $\mathbb{F}_q[x]$ operations into explicit formulas in $\mathbb{F}_q$.

To illustrate the value of our approach, we show that, for group operations on Jacobians of general genus 2 curves over large prime fields, the (affine and projective) formulas that result from this description are more efficient than their predecessors. Also, when applying this approach back to the case of genus 1, we are able to recover several of the tricks previously explored for merging simultaneous group operations to optimize elliptic curve computations.

The rest of this paper is organized as follows. We briefly touch on some more related work, before moving to Section 2 where we give a short background on hyperelliptic curves and the Mumford representation of Jacobian elements. Section 3 discusses the geometry of Jacobian arithmetic on hyperelliptic curves, and shows that we can use simple linear algebra to compute the required geometric functions from the Mumford coordinates. Section 4 is dedicated to illustrating how this technique results in fast explicit formulas in genus 2, whilst Section 5 generalizes the algorithm for all $g \geq 2$. As we hope this work will influence further progress in higher genus arithmetic, in Section 6 we highlight some further implications of adopting this geometrically inspired approach, before concluding in Section 7. MAGMA scripts that verify our proposed algorithms and formulas can be found in the appendices.

Related work. There are several high-level papers (e.g. [27, 25]) which discuss general methods for computing in Jacobians of arbitrary algebraic curves. In addition, there has also been work which specifically addresses arithmetic on non-hyperelliptic Jacobians from a geometric perspective (e.g. [13, 14]).

Khuri-Makdisi treated divisor composition on arbitrary algebraic curves with linear algebra techniques in [29] and [30]. In contrast to Khuri-Makdisi's deep and more general approach, our paper specifically aims to present an explicit algorithm in an implementation-ready format that is specific to hyperelliptic curves, much like his joint work with Abu Salem which applied his earlier techniques to present explicit formulas for arithmetic on $C_{3,4}$ curves [1]. Some other authors have also applied techniques from the realm of linear algebra to Jacobian operations: two notable examples being the work of Guyot *et al.* [23] and Avanzi *et al.* [2] who both used matrix methods to compute the resultant of two polynomials in the composition stage.

Since we have focused on general hyperelliptic curves, our comparison in genus 2 does not include the record-holding work by Gaudry [19], which exploits the Kummer surface associated with curves of a special form to achieve the current outright fastest genus 2 arithmetic for those curve models. Gaudry and Harley's second exposition [20] further describes the results in [24]. Finally, we do not draw comparisons with any work on real models of hyperelliptic curves, which usually result in slightly slower formulas than imaginary hyperelliptic curves, but we note that both Galbraith *et al.* [16] and Erickson *et al.* [11] achieve very competitive formulas for group law computations on real models of genus 2 hyperelliptic curves.

2 Background

We give some brief background on hyperelliptic curves and the Mumford representation of points in the Jacobian. For a more in depth discussion, the reader is referred to [3, §4] and [15, §11]. Over the field K, we use C_g to denote the general ("imaginary quadratic") hyperelliptic curve of genus g given by

$$C_g : y^2 + h(x)y = f(x), \quad h(x), f(x) \in K[x], \quad \deg(f) = 2g + 1, \quad \deg(h) \le g, \quad f \,\text{monic}, \tag{1}$$

with the added stipulation that no point $(x, y) \in \overline{K}$ simultaneously sends both partial derivatives $2y + h(x)$ and $f'(x) - h'(x)y$ to zero [3, §14.1]. As long as $\mathrm{char}(K) \neq 2g + 1$, we can isomorphically transform C_g into $\hat{C}_g$, given as $\hat{C}_g : y^2 + h(x)y = x^{2g+1} + \hat{f}_{2g-1}x^{2g-1} + ... + \hat{f}_1x + \hat{f}_0$, so that the coefficient of x^{2g} is zero [3, §14.13]. In the case of odd characteristic fields, it is standard to also annihilate the presence of $h(x)$ completely under a suitable transformation, in order to obtain a simpler model (we will make use of this in §4). We abuse notation and use C_g from hereon to refer to the simplified version of the curve equation in each context. Although the proofs in §3 apply to any K, it better places the intention of the discussion to henceforth regard K as a finite field $\mathbb{F}_q$.

We work in the Jacobian group $\mathrm{Jac}(C_g)$ of C_g, where the elements are equivalence classes of degree zero divisors on C_g. Divisors are formal sums of points on the curve, and degree of a divisor is the sum of the multiplicities of points in the support of the divisor. Two divisors are *equivalent* if their difference is a principal divisor, i.e. equal to the divisor of zeros and poles of a function. It follows from the Riemann-Roch Theorem that for hyperelliptic curves, each class D has a unique *reduced* representative of the form

$$\rho(D) = (P_1) + (P_2) + ... + (P_r) - r(P_\infty),$$

such that $r \le g$, $P_i \neq -P_j$ for $i \neq j$, no P_i satisfying $P_i = -P_i$ appears more than once, and with P_∞ being the point at infinity on C_g. We drop the ρ from hereon and, unless stated otherwise, assume divisor equations involve reduced divisors. When referring to the non-trivial elements in the reduced divisor D, we mean all $P \in \mathrm{supp}(D)$ where $P \neq P_\infty$, i.e. the elements corresponding to the effective part of D. For each of the r non-trivial elements appearing in D, write $P_i = (x_i, y_i)$. Mumford proposed a convenient way to represent such divisors as $D = (u(x), v(x))$, where $u(x)$ is a monic polynomial with $\deg(u(x)) \le g$ satisfying $u(x_i) = 0$, and $v(x)$ (which is not monic in general) with $\deg(v(x)) < \deg(u(x))$ is such that $v(x_i) = y_i$, for $1 \le i \le r$. In this way we have a one-to-one correspondence between reduced divisors and their so-called *Mumford representation* [46]. We use $\oplus$ (resp. $\ominus$) to distinguish group additions (resp. subtractions) between Jacobian elements from "additions" in formal divisor sums. We use $\bar{D}$ to denote the divisor obtained by taking the hyperelliptic involution of each of the non-trivial elements in the support of D.

When developing formulas for implementing genus g arithmetic, we are largely concerned with the frequent case that arises where both (not necessarily distinct) reduced divisors $D = (u(x), v(x))$ and $D' = (u'(x), v'(x))$ in the sum $D \oplus D'$ are such that $\deg(u(x)) = \deg(u'(x)) = g$. This means that $D = E - g(P_\infty)$ and $D' = E' - g(P_\infty)$, with both E and E' being effective divisors of degree g; from hereon we interchangeably refer to such divisors as *full degree* or *degree g* divisors, and we use $\hat{\mathrm{J}}\mathrm{ac}(C_g)$ to denote the set of all such divisor classes of full degree, where $\hat{\mathrm{J}}\mathrm{ac}(C_g) \subset \mathrm{Jac}(C_g)$. In Section 5.2 we discuss how to handle the special case when a divisor of degree less than g is encountered.

3 Computations in the Mumford function field

The purpose of this section is to show how to compute group law operations in Jacobians by applying linear algebra to the Mumford coordinates of divisors. The geometric description of the group law is an important ingredient in the proof of the proposed linear algebra approach (particularly in the proof of Proposition 7), so we start by reviewing the geometry underlying arithmetic on Jacobians of hyperelliptic curves.

Since the Jacobian of a hyperelliptic curve is the group of degree zero divisors modulo principal divisors, the group operation is formal addition modulo the equivalence relation. Thus two divisors D and D' can be added by finding a function whose divisor contains the support of both D and D', and then the sum is equivalent to the negative of the complement of that support. Such a function

$\ell(x)$ can be obtained by interpolating the points in the support of the two divisors. The complement of the support of D and D' in the support of $\mathrm{div}(\ell)$ consists of the other points of intersection of ℓ with the curve. In general those individual points may not be defined over the ground field for the curve. We are thus led to work with Mumford coordinates for divisors on hyperelliptic curves, since the polynomials in Mumford coordinates are defined over the base field and allow us to avoid extracting individual roots and working with points defined over extension fields.

For example, consider adding two full degree genus 3 divisors $D, D' \in \hat{\mathrm{J}}\mathrm{ac}(C_3/\mathbb{F}_q)$, with respective supports $\mathrm{supp}(D) = \{P_1, P_2, P_3\} \cup \{P_\infty\}$ and $\mathrm{supp}(D') = \{P'_1, P'_2, P'_3\} \cup \{P_\infty\}$, as in Figure 1. After computing the quintic function $\ell(x, y) = \sum_{i=0}^{5} \ell_i x^i$ that interpolates the six non-trivial points in the composition phase, computing the x-coordinates of the remaining (four) points of intersection explicitly would require solving

$$\ell_5^2 \cdot \prod_{i=1}^{3}(x - x_i) \cdot \prod_{i=1}^{3}(x - x'_i) \prod_{i=1}^{4}(x - \bar{x}_i) = \Big(\sum_{i=0}^{5} \ell_i x^i\Big)^2 - f(x)$$

for $\bar{x}_1$,$\bar{x}_2$,$\bar{x}_3$ and $\bar{x}_4$, which would necessitate multiple root extractions. On the other hand, the exact division $\prod_{i=1}^{4}(x - \bar{x}_i) = \Big(\big(\sum_{i=0}^{5} \ell_i x^i\big)^2 - f(x)\Big) / \Big(\ell_5^2 \cdot \prod_{i=1}^{3}(x - x_i) \cdot \prod_{i=1}^{3}(x - x'_i)\Big)$ can be computed very efficiently (and entirely over $\mathbb{F}_q$) by equating coefficients of x.

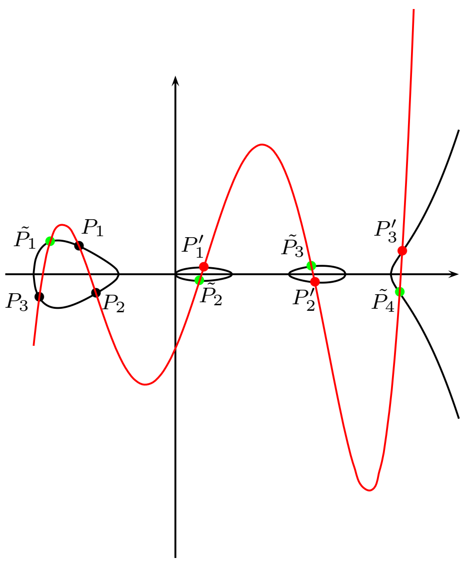

Fig. 1. The *composition* stage of a general addition on the Jacobian of a genus 3 curve C_3 over the reals $\mathbb{R}$: the 6 points in the combined supports of D and D' are interpolated by a quintic polynomial which intersects C in 4 more places to form the unreduced divisor $\tilde{D} = \tilde{P}_1 + \tilde{P}_2 + \tilde{P}_3 + \tilde{P}_4$.

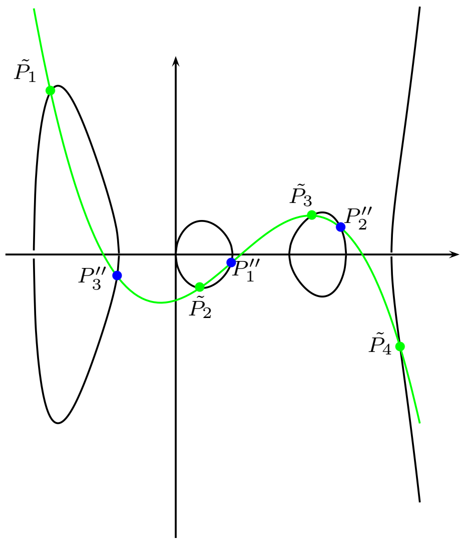

Fig. 2. The *reduction stage*: a (vertically) magnified view of the cubic function which interpolates the points in the support of $\tilde{D}$ and intersects C_3 in three more places to form $\bar{D}'' = (P''_1 + P''_2 + P''_3) \sim \tilde{D}$, the reduced equivalent of $\tilde{D}$.

Whilst the Mumford representation is absolutely necessary for efficient reduction, the price we seemingly pay in deriving formulas from the simple geometric description lies in the composition phase. In any case, finding the interpolating function $y = \ell(x)$ would be conceptually trivial if we knew the (x, y) coordinates of the points involved, but computing the function directly from the Mumford coordinates appears to be more difficult. In what follows we detail how this can be achieved in general, using only linear algebra over the base field. The meanings of the three propositions in this section are perhaps best illustrated through the examples that follow each of them.

Proposition 1. *On the Jacobian of a genus g hyperelliptic curve, the dense set $\hat{\mathrm{J}}\mathrm{ac}(C_g)$ of divisor classes with reduced representatives of full degree g can be described exactly as the intersection of g hypersurfaces of dimension (at most) $2g$.*

Proof. Let $D = \big(u(x), v(x)\big) = \big(x^g + \sum_{i=0}^{g-1} u_i x^i\,,\, \sum_{i=0}^{g-1} v_i x^i\big) \in \hat{\mathrm{J}}\mathrm{ac}(C_g(K))$ be an arbitrary degree g divisor class representative with $\mathrm{supp}(D) = \{(x_1, y_1), ..., (x_g, y_g)\} \cup \{P_\infty\}$, so that $u(x_i) = 0$ and

$v(x_i) = y_i$ for $1 \le i \le g$. Let $\Psi(x) = \sum_{i=0}^{g-1} \Psi_i x^i$ be the polynomial obtained by substituting $y = v(x)$ into the equation for C_g and reducing modulo the ideal generated by $u(x)$. Clearly, $\Psi(x_i) \equiv 0 \bmod \langle u(x) \rangle$ for each of the g non-trivial elements in $\mathrm{supp}(D)$, but since $\deg(\Psi(x)) \le g-1$, it follows that each of its g coefficients Ψ_i must be identically zero, implying that every element $D \in \hat{\mathrm{J}}\mathrm{ac}(C_g)$ of full degree g lies in the intersection of the g hypersurfaces $\Psi_i = \Psi_i(u_0, ..., u_{g-1}, v_0, ..., v_{g-1}) = 0$. On the other hand, each unique $2g$-tuple in K which satisfies $\Psi_i = 0$ for $1 \le i \le g$ defines a unique full degree representative $D \in \hat{\mathrm{J}}\mathrm{ac}(C_g(K))$ (cf. [15, ex 11.3.7]). □

Definition 2 (Mumford ideals). *We call the g ideals $\langle \Psi_i \rangle$ arising from the g hypersurfaces $\Psi_i = 0$ in Proposition 1 the Mumford ideals.*

Definition 3 (Mumford function fields). *The function fields of $\hat{\mathrm{J}}\mathrm{ac}(C_g)$ and $\hat{\mathrm{J}}\mathrm{ac}(C_g) \times \hat{\mathrm{J}}\mathrm{ac}(C_g)$ are respectively identified with the quotient fields of*

$$\frac{K[u_0, ..., u_{g-1}, v_0, ..., v_{g-1}]}{\langle \Psi_0, ..., \Psi_{g-1} \rangle} \quad \text{and} \quad \frac{K[u_0, ..., u_{g-1}, v_0, ..., v_{g-1}, u'_0, ..., u'_{g-1}, v'_0, ..., v'_{g-1}]}{\langle \Psi_0, ..., \Psi_{g-1}, \Psi'_0, ..., \Psi'_{g-1} \rangle},$$

which we call the Mumford function fields and denote by $K_{\mathrm{DBL}}^{\mathrm{Mum}} = K(\hat{\mathrm{J}}\mathrm{ac}(C_g))$ and $K_{\mathrm{ADD}}^{\mathrm{Mum}} = K(\hat{\mathrm{J}}\mathrm{ac}(C_g) \times \hat{\mathrm{J}}\mathrm{ac}(C_g))$ respectively. We abbreviate and use Ψ_i, Ψ'_i to differentiate between $\Psi_i = \Psi_i(u_0, ..., u_{g-1}, v_0, ..., v_{g-1})$ and $\Psi'_i = \Psi_i(u'_0, ..., u'_{g-1}, v'_0, ..., v'_{g-1})$ when working in $K_{\mathrm{ADD}}^{\mathrm{Mum}}$.

Example 4. Consider the genus 2 hyperelliptic curve defined by $C : y^2 = (x^5 + 2x^3 - 7x^2 + 5x + 1)$ over $\mathbb{F}_{37}$. A general degree two divisor $D \in \hat{\mathrm{J}}\mathrm{ac}(C)$ takes the form $D = (x^2 + u_1 x + u_0, v_1 x + v_0)$. Substituting $y = v_1 x + v_0$ into C and reducing modulo $\langle x^2 + u_1 x + u_0 \rangle$ gives

$$(v_1 x + v_0)^2 - (x^5 + 2x^3 - 7x^2 + 5x + 1) \equiv \Psi_1 x + \Psi_0 \equiv 0 \mod \langle x^2 + u_1 x + u_0 \rangle$$

where

$$\Psi_1(u_1, u_0, v_1, v_0) = 3\, u_0 {u_1}^2 - {u_1}^4 - {u_0}^2 + 2\, v_0 v_1 - {v_1}^2 u_1 + 2\,(u_0 - {u_1}^2) - 7u_1 - 5,$$
$$\Psi_0(u_1, u_0, v_1, v_0) = {v_0}^2 - {v_1}^2 u_0 + 2\, {u_0}^2 u_1 - {u_1}^3 u_0 - 2u_1 u_0 - 7u_0 - 1.$$

The number of tuples $(u_0, u_1, v_0, v_1) \in \mathbb{F}_{37}$ lying in the intersection of $\Psi_0 = \Psi_1 = 0$ is 1373, which is the number of degree 2 divisors on $\mathrm{Jac}(C)$, i.e. $\#\hat{\mathrm{J}}\mathrm{ac}(C) = 1373$. There are 39 other divisors on $\mathrm{Jac}(C)$ with degrees less than 2, each of which is isomorphic to a point on the curve, so that $\#\mathrm{Jac}(C) = \#\hat{\mathrm{J}}\mathrm{ac}(C) + \#C = 1373 + 39 = 1412$. Formulas for performing full degree divisor additions are derived inside the Mumford function field $K_{\mathrm{ADD}}^{\mathrm{Mum}} = \mathrm{Quot}(K[u_0, u_1, v_0, v_1, u'_0, u'_1, v'_0, v'_1]/\langle \Psi_0, \Psi_1, \Psi'_0, \Psi'_1 \rangle)$, whilst formulas for full degree divisor doublings are derived inside the Mumford function field $K_{\mathrm{DBL}}^{\mathrm{Mum}} = \mathrm{Quot}(K[u_0, u_1, v_0, v_1]/\langle \Psi_0, \Psi_1 \rangle)$.

Performing the efficient composition of two divisors amounts to finding the least degree polynomial function that interpolates the union of their (assumed disjoint) non-trivial supports. The following two propositions show that in the general addition and doubling of divisors, finding the interpolating functions in the Mumford function fields can be accomplished by solving linear systems.

Proposition 5 (General divisor addition). *Let D and D' be reduced divisors of degree g on* $\mathrm{Jac}(C_g)$ *such that* $\mathrm{supp}(D) = \{(x_1, y_1), ..., (x_g, y_g)\} \cup \{P_\infty\}$, $\mathrm{supp}(D') = \{(x'_1, y'_1), ..., (x'_g, y'_g)\} \cup \{P_\infty\}$ *and $x_i \ne x'_j$ for all $1 \le i, j \le g$. A function ℓ on C_g that interpolates the $2g$ non-trivial elements in* $\mathrm{supp}(D) \cup \mathrm{supp}(D')$ *can be determined by solving a linear system of dimension $2g$ inside the Mumford function field $K_{\mathrm{ADD}}^{\mathrm{Mum}}$.*

Proof. Let $D = \big(u(x), v(x)\big) = \big(x^g + \sum_{i=0}^{g-1} u_i x^i\,,\, \sum_{i=0}^{g-1} v_i x^i\big)$ and $D' = \big(u'(x), v'(x)\big) = \big(x^g + \sum_{i=0}^{g-1} u'_i x^i\,,\, \sum_{i=0}^{g-1} v'_i x^i\big)$. Let the polynomial $y = \ell(x) = \sum_{i=0}^{2g-1} \ell_i x^i$ be the desired function that interpolates the $2g$ non-trivial elements in $\mathrm{supp}(D) \cup \mathrm{supp}(D')$, i.e. $y_i = \ell(x_i)$ and $y'_i = \ell(x'_i)$ for $1 \le i \le g$. Focussing firstly on D, it follows that $v(x) - \ell(x) = 0$ for $x \in \{x_i\}_{1 \le i \le g}$. As in the proof of Proposition 1, we reduce modulo the ideal generated by $u(x)$ giving $\Omega(x) = v(x) - \ell(x) \equiv \sum_{i=0}^{g-1} \Omega_i x^i \equiv 0 \mod \langle x^g + \sum_{i=0}^{g-1} u_i x^i \rangle$. Since $\deg(\Omega(x)) \le g-1$ and $\Omega(x_i) = 0$ for $1 \le i \le g$, it follows that the g coefficients $\Omega_i = \Omega_i(u_0, ..., u_{g-1}, v_0, ..., v_{g-1}, \ell_0, ..., \ell_{2g-1})$ must be all identically zero. Each gives rise to an equation that relates the $2g$ coefficients of $\ell(x)$ linearly inside $K_{\mathrm{ADD}}^{\mathrm{Mum}}$. Defining $\Omega'(x)$ from D' identically and reducing modulo $u'(x)$ gives another g linear equations in the $2g$ coefficients of $\ell(x)$. □

Example 6. Consider the genus 3 hyperelliptic curve defined by $C : y^2 = x^7 + 1$ over $\mathbb{F}_{71}$, and take $D = \big(u(x), v(x)\big), D' = \big(u'(x), v'(x)\big) \in \hat{\mathrm{J}}\mathrm{ac}(C)$ as

$$D = \big(x^3 + 6x^2 + 41x + 33, 29x^2 + 22x + 47\big), D' = \big(x^3 + 18x^2 + 15x + 37, 49x^2 + 46x + 59\big).$$

We compute the polynomial $\ell(x) = \sum_{i=0}^{5} \ell_i x^i$ that interpolates the six non-trivial elements in $\mathrm{supp}(D) \cup \mathrm{supp}(D')$ using $\ell(x) - v(x) \equiv 0 \bmod \langle u(x)\rangle$ and $\ell(x) - v'(x) \equiv 0 \bmod \langle u'(x)\rangle$, to obtain Ω_i and Ω'_i for $0 \le i \le 2$. For D and D', we respectively have that

$$0 \equiv \sum_{i=0}^{5} \ell_i x^i - (29x^2 + 22x + 47) \equiv \Omega_2 x^2 + \Omega_1 x + \Omega_0 \quad \bmod \langle x^3 + 6x^2 + 41x + 33\rangle,$$

$$0 \equiv \sum_{i=0}^{5} \ell_i x^i - (49x^2 + 46x + 59) \equiv \Omega'_2 x^2 + \Omega'_1 x + \Omega'_0 \quad \bmod \langle x^3 + 18x^2 + 15x + 37\rangle,$$

with

$\Omega_2 = \ell_2 + 65\ell_3 + 66\ell_4 + 30\ell_5 - 29$; $\Omega_1 = \ell_1 + 30\ell_3 + 48\ell_5 - 22$; $\Omega_0 = \ell_0 + 38\ell_3 + 56\ell_4 + 23\ell_5 - 47$;
$\Omega'_2 = \ell_2 + 53\ell_3 + 25\ell_4 + 67\ell_5 - 49$; $\Omega'_1 = \ell_1 + 56\ell_3 + 20\ell_4 + 7\ell_5 - 46$; $\Omega'_0 = \ell_0 + 34\ell_3 + 27\ell_4 + 69\ell_5 - 59$.

Solving $\Omega_{0\le i\le 2}, \Omega'_{0\le i\le 2} = 0$ simultaneously for $\ell_0, ..., \ell_5$ gives $\ell(x) = 21x^5 + x^4 + 36x^3 + 46x^2 + 64x + 57$.

Proposition 7 (General divisor doubling). *Let D be a divisor of degree g representing a class on* $\mathrm{Jac}(C_g)$ *with* $\mathrm{supp}(D) = \{P_1, ..., P_g\} \cup \{P_\infty\}$*. A function ℓ on C_g such that each non-trivial element in* $\mathrm{supp}(D)$ *occurs with multiplicity two in* $\mathrm{div}(\ell)$ *can be determined by a linear system of dimension $2g$ inside the Mumford function field $K_{\mathrm{DBL}}^{\mathrm{Mum}}$.*

Proof. Let $D = \big(u(x), v(x)\big) = \big(x^g + \sum_{i=0}^{g-1} u_i x^i, \sum_{i=0}^{g-1} v_i x^i\big)$ and write $P_i = (x_i, y_i)$ for $1 \le i \le g$. Let the polynomial $y = \ell(x) = \sum_{i=0}^{2g-1} \ell_i x^i$ be the desired function that interpolates the g non-trivial elements of $\mathrm{supp}(D)$, and also whose derivative $\ell'(x)$ is equal to dy/dx on $C_g(x, y)$ at each such element. Namely, $\ell(x) = \sum_{i=0}^{2g-1} \ell_i x^i$ is such that $\ell(x_i) = y_i$ and $\frac{d\ell}{dx}(x_i) = \frac{dy}{dx}(x_i)$ on C for $1 \le i \le g$. This time the first g equations come from the direct interpolation as before, whilst the second g equations come from the general expression for the equated derivates, taking $\frac{d\ell}{dx}(x_i) = \frac{dy}{dx}(x_i)$ on C_g as

$$\sum_{i=1}^{g-1} i\ell_i x^{i-1} = \frac{(2g+1)x^{2g} + \sum_{i=1}^{2g-1} i f_i x^{i-1} + (\sum_{i=0}^{g} i h_i x^{i-1}) \cdot y}{2y + \sum_{i=0}^{g} h_i x^i}$$

for each x_i with $1 \le i \le g$. Again, it is easy to see that substituting $y = v(x)$ and reducing modulo the ideal generated by $u(x)$ will produce a polynomial $\Omega'(x)$ with degree less than or equal to $g - 1$. Since $\Omega'(x)$ has g roots, $\Omega'_i = 0$ for $0 \le i \le g - 1$, giving rise to the second g equations which importantly relate the coefficients of $\ell(x)$ linearly inside $K_{\mathrm{DBL}}^{\mathrm{Mum}}$. $\square$

Example 8. Consider the genus 3 hyperelliptic curve defined by $C : y^2 = x^7 + 5x + 1$ over $\mathbb{F}_{257}$, and take $D \in \hat{\mathrm{J}}\mathrm{ac}(C)$ as $D = (u(x), v(x)) = (x^3 + 57x^2 + 26x + 80, 176x^2 + 162x + 202)$. We compute the polynomial $\ell(x) = \sum_{i=0}^{5} \ell_i x^i$ that interpolates the three non-trivial points in $\mathrm{supp}(D)$, and also has the same derivative as C at these points. For the interpolation only, we obtain $\Omega_0, \Omega_1, \Omega_2$ (collected below) identically as in Example 6.

For $\Omega'_0, \Omega'_1, \Omega'_2$, equating dy/dx on C with $\ell'(x)$ gives

$$\frac{7x^6 + 5}{2y} \equiv 5\ell_5 x^4 + 4\ell_4 x^3 + 3\ell_3 x^2 + 2\ell_2 x + \ell_1 \quad \bmod \langle x^3 + 57x^2 + 26x + 80\rangle,$$

which, after substituting $y = 176x^2 + 162x + 202$, rearranges to give $0 \equiv \Omega'_2 x^2 + \Omega'_1 x + \Omega'_0$, where

$$\begin{aligned} \Omega_2 &= 118\ell_4 + 256\ell_2 + 57\ell_3 + 96\ell_5; & \Omega'_2 &= 76\ell_5 + 2541\ell_4 + 254\ell_3 + 166; \\ \Omega_1 &= 140\ell_4 + 256\ell_1 + 26\ell_3 + 82\ell_5; & \Omega'_1 &= 209 + 255\ell_2 + 104\ell_4 + 186\ell_5; \\ \Omega_0 &= 256\ell_0 + 80\ell_3 + 69\ell_5 + 66\ell_4; & \Omega'_0 &= 73\ell_5 + 63\ell_4 + 256\ell_1 + 31. \end{aligned}$$

Solving $\Omega_{0\le i\le 2}, \Omega'_{0\le i\le 2} = 0$ simultaneously for $\ell_0, ..., \ell_5$ gives $\ell(x) = 84x^5 + 213x^3 + 78x^2 + 252x + 165$.

This section showed that divisor composition on hyperelliptic curves can be achieved via linear operations in the Mumford function fields.

4 Generating explicit formulas in genus 2

This section applies the results of the previous section to develop explicit formulas for group law computations involving full degree divisors on Jacobians of genus 2 hyperelliptic curves. Assuming an underlying field of large prime characteristic, such genus 2 hyperelliptic curves $C'/\mathbb{F}_q$ can always be isomorphically transformed into $C_2/\mathbb{F}_q$ given by $C_2 : y^2 = x^5 + f_3x^3 + f_2x^2 + f_1x + f_0$, where $C_2 \cong C'$ (see §2). The Mumford representation of a general degree two divisor $D \in \hat{\mathrm{J}}\mathrm{ac}(C_2) \subset \mathrm{Jac}(C_2)$ is given as $D = (x^2 + u_1x + u_0, v_1x + v_0)$. From Proposition 1, we compute the $g = 2$ hypersurfaces whose intersection is the set of all such divisors $\hat{\mathrm{J}}\mathrm{ac}(C_2)$ as follows. Substituting $y = v_1x + v_0$ into the equation for C_2 and reducing modulo the ideal $\langle x^2 + u_1x + u_0\rangle$ gives the polynomial $\Psi(x)$ as

$$\Psi(x) \equiv \Psi_1x + \Psi_0 \equiv (v_1x + v_0)^2 - (x^5 + f_3x^3 + f_2x^2 + f_1x + f_0) \bmod \langle x^2 + u_1x + u_0\rangle,$$

where

$$\begin{aligned}\Psi_0 &= {v_0}^2 - f_0 + f_2u_0 - {v_1}^2u_0 + 2\,{u_0}^2u_1 - u_1f_3u_0 - {u_1}^3u_0,\\ \Psi_1 &= 2\,v_0v_1 - f_1 - {v_1}^2u_1 + f_2u_1 - f_3({u_1}^2 - u_0) + 3\,u_0{u_1}^2 - {u_1}^4 - {u_0}^2.\end{aligned} \tag{2}$$

We will derive doubling formulas inside $K_{\mathrm{ADD}}^{\mathrm{Mum}} = \mathrm{Quot}(K[u_0, u_1, v_0, v_1]/\langle\Psi_0, \Psi_1\rangle)$ and addition formulas inside $K_{\mathrm{ADD}}^{\mathrm{Mum}} = \mathrm{Quot}(K[u_0, u_1, v_0, v_1, u_0', u_1', v_0', v_1']/\langle\Psi_0, \Psi_1, \Psi_0', \Psi_1'\rangle)$. In §4.2 particularly, we will see how the ideal $\langle\Psi_0, \Psi_1\rangle$ is useful in simplifying the formulas that arise.

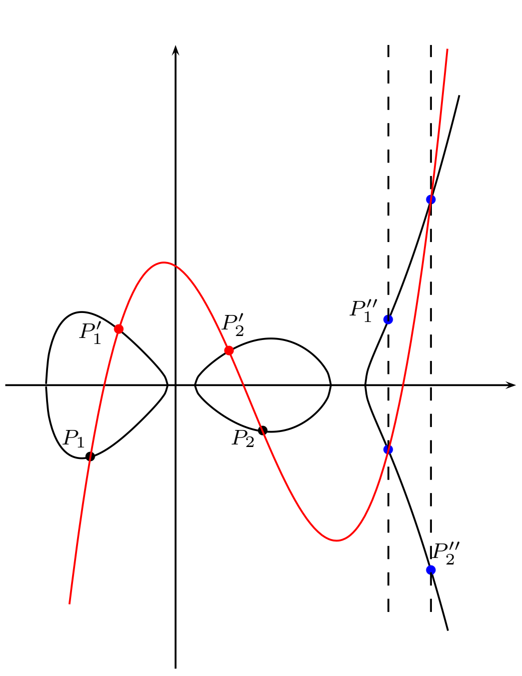

Fig. 3. The group law (general addition) on the Jacobian of the genus 2 curve C_2 over the reals $\mathbb{R}$, for $(P_1 + P_2) \oplus (P_1' + P_2') = P_1'' + P_2''$.

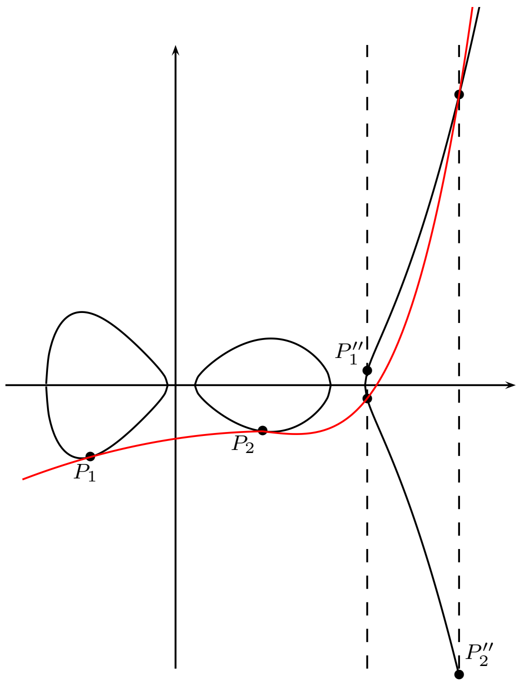

Fig. 4. A general point doubling on the Jacobian of a genus 2 curve C_2 over the reals $\mathbb{R}$, for $[2](P_1 + P_2) = P_1'' + P_2''$.

4.1 General divisor addition in genus 2

Let $D = (x^2 + u_1x + u_0, v_1x + v_0), D' = (x^2 + u_1'x + u_0', v_1'x + v_0') \in \hat{\mathrm{J}}\mathrm{ac}(C_2)$ be two divisors with $\mathrm{supp}(D) = \{P_1, P_2\} \cup \{P_\infty\}$ and $\mathrm{supp}(D') = \{P_1', P_2'\} \cup \{P_\infty\}$, such that no P_i has the same x coordinate as P_j' for $1 \le i, j \le 2$. Let $D'' = (x^2 + u_1''x + u_0'', v_1''x + v_0'') = D \oplus D'$. The composition step in the addition of D and D' involves building the linear system inside $K_{\mathrm{ADD}}^{\mathrm{Mum}}$ that solves to give the coefficients ℓ_i of the cubic polynomial $y = \ell(x) = \sum_{i=0}^{3} \ell_i x^i$ which interpolates P_1, P_2, P_1', P_2'. Following Proposition 5, we have

$$\begin{aligned}0 &\equiv \Omega_1x + \Omega_0 \equiv \ell_3x^3 + \ell_2x^2 + \ell_1x + \ell_0 - (v_1x + v_0) && \bmod\langle x^2 + u_1x + u_0\rangle,\\ &\equiv (\ell_3({u_1}^2 - u_0) - \ell_2u_1 + \ell_1 - v_1)x + (\ell_3u_1u_0 - \ell_2u_0 + \ell_0 - v_0) && \bmod\langle x^2 + u_1x + u_0\rangle,\end{aligned} \tag{3}$$

which provides two equations ($\Omega_1 = 0$ and $\Omega_0 = 0$) relating the four coefficients of the interpolating polynomial linearly inside $K_{\text{ADD}}^{\text{Mum}}$. Identically, interpolating the support of D' produces two more linear equations which allow us to solve for the four ℓ_i as

$$\begin{pmatrix} 1 & 0 & -u_0 & u_1u_0 \\ 0 & 1 & -u_1 & u_1^2 - u_0 \\ 1 & 0 & -u_0' & u_1'u_0' \\ 0 & 1 & -u_1' & u_1'^2 - u_0' \end{pmatrix} \cdot \begin{pmatrix} \ell_0 \\ \ell_1 \\ \ell_2 \\ \ell_3 \end{pmatrix} = \begin{pmatrix} v_0 \\ v_1 \\ v_0' \\ v_1' \end{pmatrix}.$$

Observe that the respective subtraction of rows 1 and 2 from rows 3 and 4 gives rise to a smaller system that can be solved for ℓ_2 and ℓ_3, as

$$\begin{pmatrix} u_0 - u_0' & u_1'u_0' - u_1u_0 \\ u_1 - u_1' & (u_1'^2 - u_0') - (u_1^2 - u_0) \end{pmatrix} \cdot \begin{pmatrix} \ell_2 \\ \ell_3 \end{pmatrix} = \begin{pmatrix} v_0' - v_0 \\ v_1' - v_1 \end{pmatrix}. \quad (4)$$

Remark 9. We will see in Section 5.1 that for all $g \geq 2$, the linear system that arises in the computation of $\ell(x)$ can always be trivially reduced to be of dimension g, but for now it is useful to observe that once we solve the dimension $g = 2$ matrix system for ℓ_i with $i \geq g$, calculating the remaining ℓ_i where $i < g$ is computationally straightforward.

The next step is to determine the remaining intersection points of $y = \ell(x)$ on C_2. Since $y = \ell(x)$ is cubic, its substitution into C_2 will give a degree six equation in x. Four of the roots will correspond to the four non-trivial points in $\text{supp}(D) \cup \text{supp}(D')$, whilst the remaining two will correspond to the two x coordinates of the non-trivial elements in $\text{supp}(\bar{D}'')$, which are the same as the x coordinates in $\text{supp}(D'')$ (see the intersection points in Figure 3). Let the Mumford representation of $\bar{D}''$ be $\bar{D}'' = (x^2 + u_1''x + u_0'', -v_1''x - v_0'')$; we then have

$$(x^2 + u_1x + u_0) \cdot (x^2 + u_1'x + u_0') \cdot (x^2 + u_1''x + u_0'') = \frac{(\ell_0 + \ell_1x + \ell_2x^2 + \ell_3x^3)^2 - f(x)}{\ell_3^2}.$$

Equating coefficients is an efficient way to compute the exact division required above to solve for $u''(x)$. For example, equating coefficients of x^5 and x^4 above respectively gives

$$u_1'' = -u_1 - u_1' - \frac{1 - 2\ell_2\ell_3}{\ell_3^2}; \qquad u_0'' = -(u_0 + u_0' + u_1u_1' + (u_1 + u_1')u_1'') + \frac{2\ell_1\ell_3 + \ell_2^2}{\ell_3^2}. \quad (5)$$

It remains to compute v_1'' and v_0''. Namely, we wish to compute the linear function that interpolates the points in $\text{supp}(D'')$. Observe that reducing $\ell(x)$ modulo $\langle x^2 + u_1''x + u_0''\rangle$ gives the linear polynomial $-v_1''x + -v_0''$ which interpolates the points in $\text{supp}(\bar{D}'')$, i.e. those points which are the involutions of the points in $\text{supp}(D'')$. Thus, the computation of v_1'' and v_0'' amounts to negating the result of $\ell(x) \bmod \langle x^2 + u_1''x + u_0''\rangle$. From equation (3) then, it follows that

$$v_1'' = -(\ell_3(u_1''^2 - u_0'') - \ell_2u_1'' + \ell_1), \qquad v_0'' = -(\ell_3u_1''u_0'' - \ell_2u_0'' + \ell_0). \quad (6)$$

We summarize the process of computing a general addition $D'' = D \oplus D'$ on $\hat{\text{J}}\text{ac}(C_2)$, as follows. *Composition* involves constructing and solving the linear system in (4) for ℓ_2 and ℓ_3 before computing ℓ_0 and ℓ_1 via (3), whilst *reduction* involves computing u_1'' and u_0'' from (5) before computing v_1'' and v_0'' via (6). The explicit formulas for these computations are in Table 1, where **I**, **M** and **S** represent the costs of an $\mathbb{F}_q$ inversion, multiplication and squaring respectively. We postpone comparisons with other works until after the doubling discussion.

Remark 10. The formulas for computing v_0'' and v_1'' in (6) include operations involving $u_1''^2$ and $u_1''u_0''$. Since those quantities are also needed in the first step of the addition formulas (see the first line of Table 1) for any subsequent additions involving the divisor D'', it makes sense to carry those quantities along as extra coordinates to exploit these overlapping computations. It turns out that an analogous overlap arises in geometric group operations for all $g \geq 2$, but for now we remark that both additions and doublings on genus 2 curves will benefit from extending the generic affine coordinate system to include two extra coordinates u_1^2 and u_1u_0.

	AFFINE ADDITION		
Input:	$D = (u_1, u_0, v_1, v_0, U_1 = u_1^2, U_0 = u_1 u_0)$, $D' = (u'_1, u'_0, v'_1, v'_0, U'_1 = u'^2_1, U'_0 = u'_1 u'_0)$		Operations in $\mathbb{F}_q$
	$\sigma_1 \leftarrow u_1 + u'_1$, $\Delta_0 \leftarrow v_0 - v'_0$, $\Delta_1 \leftarrow v_1 - v'_1$, $M_1 \leftarrow U_1 - u_0 - U'_1 + u'_0$, $M_2 \leftarrow U'_0 - U_0$,		
	$M_3 \leftarrow u_1 - u'_1$, $M_4 \leftarrow u'_0 - u_0$, $t_1 \leftarrow (M_2 - \Delta_0) \cdot (\Delta_1 - M_1)$, $t_2 \leftarrow (-\Delta_0 - M_2) \cdot (\Delta_1 + M_1)$,		2**M**
	$t_3 \leftarrow (-\Delta_0 + M_4) \cdot (\Delta_1 - M_3)$, $t_4 \leftarrow (-\Delta_0 - M_4) \cdot (\Delta_1 + M_3)$,		2**M**
	$\ell_2 \leftarrow t_1 - t_2$ $\ell_3 \leftarrow t_3 - t_4$, $d \leftarrow t_3 + t_4 - t_1 - t_2 - 2(M_2 - M_4) \cdot (M_1 + M_3)$,		1**M**
	$A \leftarrow 1/(d \cdot \ell_3)$, $B \leftarrow d \cdot A$, $C \leftarrow d \cdot B$, $D \leftarrow \ell_2 \cdot B$, $E \leftarrow \ell_3^2 \cdot A$, $CC \leftarrow C^2$,		**I** + 5**M** + 2**S**
	$u''_1 \leftarrow 2D - CC - \sigma_1$, $u''_0 \leftarrow D^2 + C \cdot (v_1 + v'_1) - ((u''_1 - CC) \cdot \sigma_1 + (U_1 + U'_1))/2$,		2**M** + 1**S**
	$U''_1 \leftarrow u''^2_1$, $U''_0 \leftarrow u''_1 \cdot u''_0, v''_1 \leftarrow D \cdot (u_1 - u''_1) + U''_1 - u''_0 - U_1 + u_0$,		2**M** + 1**S**
	$v''_0 \leftarrow D \cdot (u_0 - u''_0) + U''_0 - U_0$, $v''_1 \leftarrow E \cdot v''_1 + v_1$ $v''_0 \leftarrow E \cdot v''_0 + v_0$.		3**M**
Output:	$D'' = \rho(D \oplus D') = (u''_1, u''_0, v''_1, v''_0, U''_1 = u''^2_1, U''_0 = u''_1 u''_0)$.	**Total**	**I** + 17**M** + 4**S**
	PROJECTIVE ADDITION		
Input:	$D = (U_1, U_0, V_1, V_0, Z)$, $D' = (U'_1, U'_0, V'_1, V'_0, Z')$,		Operations
	$ZZ \leftarrow Z_1 \cdot Z_2$, $U1Z \leftarrow U_1 \cdot Z_2$, $U1Z' \leftarrow U'_1 \cdot Z_1$, $U1ZS \leftarrow U1Z^2$, $U1ZS' \leftarrow U1Z'^2$,		3**M** + 2**S**
	$U0Z \leftarrow U_0 \cdot Z_2$, $U0Z' \leftarrow U'_0 \cdot Z_1$, $V1Z \leftarrow V_1 \cdot Z_2$, $V1Z' \leftarrow V'_1 \cdot Z_1$,		4**M**
	$M_1 \leftarrow U1ZS - U1ZS' + ZZ \cdot (U0dZ - U0Z)$, $M_2 \leftarrow U1Z' \cdot U0Z' - U1Z \cdot U0Z$;		3**M**
	$M_3 \leftarrow U1Z - U1Z'$, $M_4 \leftarrow U0Z' - U0Z$, $z_1 \leftarrow V0 \cdot Z_2 - V0' \cdot Z_1$, $z_2 \leftarrow V1Z - V1Z'$,		2**M**
	$t_1 \leftarrow (M_2 - z_1) \cdot (z_2 - M_1)$, $t_2 \leftarrow (-z_1 - M_2) \cdot (z_2 + M_1)$,		2**M**
	$t_3 \leftarrow (-z_1 + M_4) \cdot (z_2 - M_3)$, $t_4 \leftarrow (-z_1 - M_4) \cdot (z_2 + M_3)$,		2**M**
	$\ell_2 \leftarrow t_1 - t_2$, $\ell_3 \leftarrow t_3 - t_4$, $d \leftarrow t_3 + t_4 - t_1 - t_2 - 2 \cdot (M_2 - M_4) \cdot (M_1 + M_3)$,		1**M**
	$A \leftarrow d^2$, $B \leftarrow \ell_3 \cdot ZZ$, $C \leftarrow \ell_2 \cdot B$, $D \leftarrow d \cdot B$, $E \leftarrow \ell_3 \cdot B$, $F \leftarrow U1Z \cdot E$, $G \leftarrow ZZ \cdot E$,		6**M** + 1**S**
	$H \leftarrow U0Z \cdot G$, $J \leftarrow D \cdot G$, $K \leftarrow Z_2 \cdot J$, $U''_1 \leftarrow 2 \cdot C - A - E \cdot (U1Z + U1Z')$,		4**M**
	$U''_0 \leftarrow \ell_2^2 \cdot ZZ + D \cdot (V1Z + V1Z') - ((U''_1 - A) \cdot (U1Z + U1Z') + E \cdot (U1ZS + U1ZS'))/2$,		4**M** + 1**S**
	$V''_1 \leftarrow U''_1 \cdot (U''_1 - C) + F \cdot (C - F) + E \cdot (H - U''_0)$,		3**M**
	$V''_0 \leftarrow H \cdot (C - F) + U''_0 \cdot (U''_1 - C)$, $V''_1 \leftarrow V''_1 \cdot ZZ + K \cdot V_1$, $V''_0 \leftarrow V''_0 + K \cdot V_0$,		5**M**
	$U''_1 \leftarrow U''_1 \cdot D \cdot ZZ$, $U''_0 \leftarrow U''_0 \cdot D$, $Z'' \leftarrow ZZ \cdot J$.		4**M**
Output:	$D'' = \rho(D \oplus D') = (U''_1, U''_0, V''_1, V''_0, Z'')$.	**Total**	43**M** + 4**S**
	AFFINE DOUBLING		
Input:	$D = (u_1, u_0, v_1, v_0, U_1 = u_1^2, U_0 = u_1 u_0)$, with constants f_2, f_3		Operations
	$vv \leftarrow v_1^2$, $vu \leftarrow (v_1 + u_1)^2 - vv - U_1$, $M_1 \leftarrow 2v_0 - 2vu$, $M_2 \leftarrow 2v_1 \cdot (u_0 + 2U_1)$,		1**M** + 2**S**
	$M_3 \leftarrow -2v_1$, $M_4 \leftarrow vu + 2v_0$, $z_1 \leftarrow f_2 + 2U_1 \cdot u_1 + 2U_0 - vv$, $z_2 \leftarrow f_3 - 2u_0 + 3U_1$,		1**M**
	$t_1 \leftarrow (M_2 - z_1) \cdot (z_2 - M_1)$, $t_2 \leftarrow (-z_1 - M_2) \cdot (z_2 + M_1)$,		2**M**
	$t_3 \leftarrow (M_4 - z_1) \cdot (z_2 - M_3)$, $t_4 \leftarrow (-z_1 - M_4) \cdot (z_2 + M_3)$,		2**M**
	$\ell_2 \leftarrow t_1 - t_2$, $\ell_3 \leftarrow t_3 - t_4$, $d \leftarrow t_3 + t_4 - t_1 - t_2 - 2(M_2 - M_4) \cdot (M_1 + M_3)$,		1**M**
	$A \leftarrow 1/(d \cdot \ell_3)$, $B \leftarrow d \cdot A$, $C \leftarrow d \cdot B$, $D \leftarrow \ell_2 \cdot B$, $E \leftarrow \ell_3^2 \cdot A$,		**I** + 5**M** + 1**S**
	$u''_1 \leftarrow 2D - C^2 - 2u_1$, $u''_0 \leftarrow (D - u_1)^2 + 2C \cdot (v_1 + C \cdot u_1)$, $U''_1 \leftarrow u''^2_1$, $U''_0 \leftarrow u''_1 \cdot u''_0$,		3**M** + 3**S**
	$v''_1 \leftarrow D \cdot (u_1 - u''_1) + U''_1 - U_1 - u''_0 + u_0$, $v''_0 \leftarrow D \cdot (u_0 - u''_0) + U''_0 - U_0$,		2**M**
	$v''_1 \leftarrow E \cdot v''_1 + v_1$, $v''_0 \leftarrow E \cdot v''_0 + v_0$.		2**M**
Output:	$D'' = \rho([2]D) = (u''_1, u''_0, v''_1, v''_0, U''_1 = u''^2_1, U''_0 = u''_1 u''_0)$.	**Total**	**I** + 19**M** + 6**S**
	PROJECTIVE DOUBLING		
Input:	$D = (U_1, U_0, V_1, V_0, Z)$, curve constants f_2, f_3		Operations
	$UU \leftarrow U_1 \cdot U_0$, $U_{1S} \leftarrow U_1^2$, $Z_S \leftarrow Z^2$, $V0Z \leftarrow V0 \cdot Z$, $U0Z \leftarrow U0 \cdot Z$, $V_{1S} \leftarrow V1^2$,		3**M** + 3**S**
	$UV \leftarrow (V_1 + U_1)^2 - V_{1S} - U_{1S}$, $M_1 \leftarrow 2 \cdot V0Z - 2 \cdot UV$, $M_2 \leftarrow 2 \cdot V1 \cdot (U0Z + 2 \cdot U_{1S})$,		1**M** + 1**S**
	$M_3 \leftarrow -2 \cdot V_1$, $M_4 \leftarrow UV + 2 \cdot V0Z$, $z_1 \leftarrow Z \cdot (f_2 \cdot Z_S - V_{1S}) + 2 \cdot U_1 \cdot (U_{1S} + U0Z)$,		2**M**
	$z_2 \leftarrow f_3 \cdot Z_S - 2 \cdot U0Z + 3 \cdot U_{1S}$, $t_1 \leftarrow (M_2 - z1) \cdot (z_2 - M_1)$, $t_2 \leftarrow (-z_1 - M_2) \cdot (z_2 + M_1)$,		2**M**
	$t_3 \leftarrow (-z_1 + M_4) \cdot (z_2 - M_3)$, $t_4 \leftarrow (-z_1 - M_4) \cdot (z_2 + M_3)$,		2**M**
	$\ell_2 \leftarrow t_1 - t_2$, $\ell_3 \leftarrow t_3 - t_4$, $d \leftarrow t_3 + t_4 - t_1 - t_2 - 2 \cdot (M_2 - M_4) \cdot (M_1 + M_3)$,		1**M**
	$A \leftarrow \ell_2^2$, $B \leftarrow \ell_3^2$, $C \leftarrow ((\ell_2 + \ell_3)^2 - A - B)/2$, $D \leftarrow B \cdot Z$, $E \leftarrow B \cdot U_1$,		2**M** + 3**S**
	$F \leftarrow d^2$, $G \leftarrow F \cdot Z$, $H \leftarrow ((d + \ell_3)^2 - F - B)/2$, $J \leftarrow H \cdot Z$, $K \leftarrow V_1 \cdot J$, $L \leftarrow U0Z \cdot B$,		4**M** + 2**S**
	$U''_1 \leftarrow 2 \cdot C - 2 \cdot E - G$, $U''_0 \leftarrow A + U_1 \cdot (E - 2 \cdot C + 2 \cdot G) + 2 \cdot K$,		1**M**
	$V''_1 \leftarrow (C - E - U''_1) \cdot (E - U''_1) + B \cdot (L - U''_0)$, $V''_0 \leftarrow L \cdot (C - E) + (U''_1 - C) \cdot U''_0$.		4**M**
	$V''_1 \leftarrow V''_1 \cdot Z + K \cdot D$, $V''_0 \leftarrow V''_0 + V0Z \cdot H \cdot D$, $M \leftarrow J \cdot Z$, $U''_1 \leftarrow U''_1 \cdot M$, $U''_0 \leftarrow U''_0 \cdot J$,		7**M**
	$Z'' \leftarrow M \cdot D$.		1**M**
Output:	$D'' = \rho([2]D) = (U''_1, U''_0, V''_1, V''_0, Z'')$.	**Total**	30**M** + 9**S**

Table 1. Explicit formulas for a divisor addition $D'' = D \oplus D'$ involving two distinct degree 2 divisors on $\mathrm{Jac}(C_2)$, and for divisor doubling $D'' = [2]D$ of a degree 2 divisor on $\mathrm{Jac}(C_2)$. A MAGMA script is provided in Appendix A.

4.2 General divisor doubling in genus 2

Let $D = (x^2 + u_1x + u_0, v_1x + v_0) \in \hat{\text{J}}\text{ac}(C_2)$ be a divisor with $\text{supp}(D) = \{P_1, P_2\} \cup \{P_\infty\}$. To compute $[2]D = D \oplus D$, we seek the cubic polynomial $\ell(x) = \sum_{i=0}^{3} \ell_i x^i$ that has zeroes of order two at both $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$. We can immediately make use of the equations arising out of the interpolation of $\text{supp}(D)$ in (3) to obtain the first $g = 2$ equations.

There are two possible approaches to obtaining the second set of $g = 2$ equations. The first is the geometric flavored approach that was used in the proof of Proposition 7 and in Example 8, which involves matching the derivatives. The second involves reducing the substitution of $\ell(x)$ into C_g by $\langle u(x)^2 \rangle$ to ensure the prescribed zeros are of multiplicity two, and using the associated Mumford ideals to linearize the equations. For the purpose of presenting both approaches, we will illustrate the latter approach in this subsection, but it is important to highlight that the guaranteed existence of linear equations follows from the expression gained when matching derivatives in the geometric approach.

We start by setting $y = \ell(x)$ into C_2 and reducing modulo the ideal $\langle (x^2 + u_1x + u_0)^2 \rangle$, which gives

$$\Omega(x) = \Omega_0 + \Omega_1 x + \Omega_2 x^2 + \Omega_3 x^3 \equiv (\ell_0 + \ell_1 x + \ell_2 x^2 + \ell_3 x^3)^2 - f(x) \mod \langle (x^2 + u_1x + u_0)^2 \rangle$$

where

$$\begin{aligned}
\Omega_0 &= \ell_3^2(2u_0^3 - 3u_1^2u_0^2) + 4\ell_3\ell_2u_1u_0^2 - 2\ell_3\ell_1u_0^2 + \ell_0^2 - \ell_2^2u_0^2 - 2u_1u_0^2 - f_0,\\
\Omega_1 &= 6\ell_3^2(u_1u_0^2 - u_1^3u_0) + 2\ell_3\ell_2(4u_1^2u_0 - u_0^2) + 2\ell_1\ell_0 - 4\ell_3\ell_1u_0u_1 - 2\ell_2^2u_0u_1 - 4u_1^2u_0 + u_0^2 - f_1,\\
\Omega_2 &= 3\ell_3^2(u_0^2 - u_1^4) + \ell_1^2 - \ell_2^2(u_1^2 + 2u_0) - 2u_0u_1 - 2u_1^3 + 4\ell_3\ell_2(u_1^3 + u_0u_1) - 2\ell_3\ell_1(2u_0 + u_1^2)\\
&\quad + 2\ell_2\ell_0 - f_2,\\
\Omega_3 &= 2\ell_3^2(3u_1u_0 - 2u_1^3) + 2\ell_2\ell_1 + 2\ell_3\ell_2(3u_1^2 - 2u_0) - 2\ell_2^2u_1 - 4\ell_3\ell_1u_1 + 2\ell_3\ell_0 - 3u_1^2 + 2u_0 - f_3.
\end{aligned}$$

It follows that $\Omega_i = 0$ for $0 \leq i \leq 3$. Although we now have four more equations relating the unknown ℓ_i coefficients, these equations are currently nonlinear. We linearize by substituting the linear equations taken from (3) above, and reducing the results modulo the Mumford ideals given in (2). We use the two linear equations $\tilde{\Omega}_2, \tilde{\Omega}_3$ resulting from Ω_2, Ω_3, given as

$$\begin{aligned}
\tilde{\Omega}_2 &= 4\ell_1v_1 + 2\ell_2(v_0 - 2v_1u_1) - 6\ell_3u_0v_1 - 2u_0u_1 - 2u_1^3 - 3v_1^2 - f_2,\\
\tilde{\Omega}_3 &= 2v_1\ell_2 + \ell_3(2v_0 - 4u_1v_1) + 2u_0 - 3u_1^2 - f_3,
\end{aligned}$$

which combine with the linear interpolating equations (in (3)) to give rise to the linear system

$$\begin{pmatrix} -1 & 0 & u_0 & -u_1u_0 \\ 0 & -1 & u_1 & -u_1^2 + u_0 \\ 0 & 4v_1 & -2v_1u_1 + 2v_0 & -6u_0v_1 \\ 0 & 0 & 2v_1 & -4v_1u_1 + 2v_0 \end{pmatrix} \cdot \begin{pmatrix} \ell_0 \\ \ell_1 \\ \ell_2 \\ \ell_3 \end{pmatrix} = \begin{pmatrix} -v_0 \\ -v_1 \\ f_2 + 2u_1u_0 + 2u_1^3 + 3v_1^2 \\ f_3 - 2u_0 + 3u_1^2 \end{pmatrix}.$$

As was the case with the divisor addition in the previous section, we can first solve a smaller system for ℓ_2 and ℓ_3, by adding the appropriate multiple of the second row to the third row above, to give

$$\begin{pmatrix} 2v_1u_1 + 2v_0 & -2u_0v_1 - 4v_1u_1^2 \\ 2v_1 & -4v_1u_1 + 2v_0 \end{pmatrix} \cdot \begin{pmatrix} \ell_2 \\ \ell_3 \end{pmatrix} = \begin{pmatrix} f_2 + 2u_1u_0 + 2u_1^3 - v_1^2 \\ f_3 - 2u_0 + 3u_1^2 \end{pmatrix}.$$

After solving the above system for ℓ_2 and ℓ_3, the process of obtaining $D'' = [2]D = (x^2 + u_1''x + u_0'', v_1''x + v_0'')$ is identical to the case of addition in the previous section, giving rise to the analogous explicit formulas in Table 1.

4.3 Comparisons of formulas in genus 2

Table 2 draws comparisons between the explicit formulas obtained from the above approach and the explicit formulas presented in previous work. In implementations where inversions are expensive compared to multiplications (i.e. **I** > 20**M**), it can be advantageous to adopt projective formulas which avoid inversions altogether. Our projective formulas compute scalar multiples faster than

$\mathbb{F}_q$ inversions	Previous work	#	Doubling		Addition		Mixed	
I		coords	**M**	**S**	**M**	**S**	**M**	**S**
2	Harley [24, 20]	4	30	-	24	3	-	
	Lange [34]	4	24	6	24	3	-	
	Matsuo *et al.* [43]	4	27	-	25	-	-	
1	Takahashi [50]	4	29	-	25	-	-	
	Miyamoto *et al.* [45]	4	27	-	26	-	-	
	Lange [38]	4	22	5	22	3	-	
	This work	**6**	**19**	**6**	**17**	**4**	-	
-	Wollinger and Kovtun [52]	5	39	6	46	4	39	4
	Lange [36, 38]	5	38	6	47	4	40	3
	Fan *et al.* [12]	5	39	6	-	-	38	3
	Fan *et al.* [12]	8	35	7	-	-	36	5
	Lange [37, 38]	8	34	7	47	7	36	5
	This work	**5**	**30**	**9**	**43**	**4**	**36**	**5**

Table 2. Comparisons between our explicit formulas for genus 2 curves over prime fields and previous formulas using CRT based composition.

all previous projective formulas for general genus 2 curves. We also note that our homogeneous projective formulas require only 5 coordinates in total, which is the heuristic minimum for projective implementations in genus 2.

In the case of the affine formulas, it is worth commenting that, unlike the case of elliptic curves where point doublings are generally much faster than additions, affine genus 2 operations reveal divisor additions to be the significantly cheaper operation. In cases where an addition would usually follow a doubling to compute $[2]D \oplus D'$, it is likely to be computationally favorable to instead compute $(D \oplus D') \oplus D$, provided temporary storage of the additional intermediate divisor is not problematic.

Lastly, the formulas in Table 1 all required the solution to a linear system of dimension 2. This would ordinarily require 6 $\mathbb{F}_q$ multiplications, but we applied Hisil's trick [26, eq. 3.8] to instead perform these computations using 5 $\mathbb{F}_q$ multiplications. In implementations where extremely optimized multiplication routines give rise to $\mathbb{F}_q$ addition costs that are relatively high compared to $\mathbb{F}_q$ multiplications, it may be advantageous to undo such tricks (including **M**-**S** trade-offs) in favor of a lower number of additions.

5 The general description

This section presents the algorithm for divisor composition on hyperelliptic Jacobians of any genus g. The general method for reduction has essentially remained the same in all related publications following Cantor's original paper (at least in the case of low genera), but we give a simple geometric interpretation of the number of reduction rounds required in Section 5.3 below.

5.1 Composition for $g \geq 2$

We extend the composition described for genus 2 in sections 4.1 and 4.2 to hyperelliptic curves of arbitrary genus. Importantly, there are two aspects of this general description to highlight.

(i) In contrast to Cantor's general description of composition which involves polynomial arithmetic, this general description is immediately explicit in terms of $\mathbb{F}_q$ arithmetic.
(ii) The required function $\ell(x)$ is of degree $2g-1$ and therefore has $2g$ unknown coefficients. Thus, we would usually expect to solve a linear system of dimension $2g$, but the linear system that requires solving in the Mumford function field is actually of dimension g.

Henceforth we use $\mathbf{M} \cdot \mathbf{x} = \mathbf{z}$ to denote the associated linear system of dimension g, and we focus our discussion on the structure of $\mathbf{M}$ and $\mathbf{z}$.

In the case of a general divisor addition, $\mathbf{M}$ is computed as $\mathbf{M} = \mathbf{U} - \mathbf{U}'$, where $\mathbf{U}$ and $\mathbf{U}'$ are described by D and D' respectively. In fact, as for the system derived from coordinates of points above, the matrix $\mathbf{M}$ is completely dependent on $u(x)$ and $u'(x)$, whilst the vector $\mathbf{z}$ depends entirely on $v(x)$ and $v'(x)$. Algorithm 1 details how to build $\mathbf{U}$ (resp. $\mathbf{U}'$), where the first column of $\mathbf{U}$ is initialized as the Mumford coordinates $\{u_i\}_{1 \leq i < g}$ of D, and the remaining $g^2 - g$ entries are computed by proceeding across the columns and taking $\mathbf{U}_{i,j} = u_{i-1} \cdot \mathbf{U}_{g,j-1} + \mathbf{U}_{i-1,j-1}$. This relationship is obtained by a careful generalization of the process that computed (4) from (3) in the case of genus 2.

Algorithm 1 General composition (addition) of two distinct divisors. A MAGMA script is provided in Appendix C.

```
Input: $D = \{u_i, v_i\}_{0 \le i \le g-1}$, $D' = \{u'_i, v'_i\}_{0 \le i \le g-1}$.
Output: $\ell(x) = \sum_{i=0}^{2g-1} \ell_i x^i$ such that $\mathrm{supp}(D) \cup \mathrm{supp}(D') \subset \mathrm{supp}(\mathrm{div}(\ell))$.
 1: $\mathbf{U}, \mathbf{U}', \mathbf{M} \leftarrow \{0\}^{g \times g} \in \mathbb{F}_q^{g \times g}$, $\mathbf{z} \leftarrow \{0\}^g \in \mathbb{F}_q^g$.
 2: for $i$ from 1 to $g$ do
 3:     $\mathbf{U}_{g+1-i,1} \leftarrow -u_{g-i}$ ; $\mathbf{U}'_{g+1-i,1} \leftarrow -u'_{g-i}$
 4: end for
 5: for $j$ from 2 to $g$ do
 6:     $\mathbf{U}_{1,j} \leftarrow \mathbf{U}_{g,j-1} \cdot \mathbf{U}_{1,1}$ ; $\mathbf{U}'_{1,j} \leftarrow \mathbf{U}'_{g,j-1} \cdot \mathbf{U}'_{1,1}$.
 7:     for $i$ from 2 to $g$ do
 8:         $\mathbf{U}_{i,j} \leftarrow \mathbf{U}_{g,j-1} \cdot \mathbf{U}_{i,1} + \mathbf{U}_{i-1,j-1}$ ; $\mathbf{U}'_{i,j} \leftarrow \mathbf{U}'_{g,j-1} \cdot \mathbf{U}'_{i,1} + \mathbf{U}'_{i-1,j-1}$.
 9:     end for
10: end for
11: $\mathbf{M} \leftarrow \mathbf{U} - \mathbf{U}'$.
12: for $i$ from 1 to $g$ do
13:     $\mathbf{z}_i \leftarrow v_{i-1} - v'_{i-1}$
14: end for
15: Solve $\mathbf{M} \cdot \mathbf{x} = \mathbf{z}$
16: Compute $\tilde{\mathbf{x}} = \mathbf{U} \cdot \mathbf{x}$
17: for $i$ from 1 to $g$ do
18:     $\tilde{\mathbf{x}}_i \leftarrow v_{g-i} - \tilde{\mathbf{x}}_i$
19: end for
20: return $\ell(x)$ (from $\tilde{\mathbf{x}} = \{\ell_0, ..., \ell_{g-1}\}$ and $\mathbf{x} = \{\ell_g, ..., \ell_{2g-1}\}$)
```

Depending on the genus, we remark that Algorithm 1 will most likely not be the fastest way to compute $\mathbf{M}$. Instead, we propose that a faster routine is likely to be achieved by using Algorithm 1 to determine the algebraic expression for each of the elements in $\mathbf{M}$, and tailor making optimized formulas to generate its entries, in the same way that the previous section did for genus 2.

In addition, there is alternative way to view the structure (and computation) of the matrix $\mathbf{M}$. This follows from observing that both $\mathbf{U}$ and $\mathbf{U}'$ can actually be written as a sum of g matrices that are computed as outer products; let $\mathbf{c} = (c_1, .., c_g), \tilde{\mathbf{c}} = (\tilde{c}_1, ..., \tilde{c}_g) \in \mathbb{F}_q^g$ be two vectors that are derived solely from the g Mumford coordinates belonging to D, then

$$\mathbf{U} = \begin{pmatrix} c_1\tilde{c}_1 & c_1\tilde{c}_2 & \dots & c_1\tilde{c}_g \\ c_2\tilde{c}_1 & c_2\tilde{c}_2 & \dots & c_2\tilde{c}_g \\ \vdots & \dots & \ddots & \vdots \\ c_{g-1}\tilde{c}_1 & c_{g-1}\tilde{c}_2 & \dots & c_{g-1}\tilde{c}_g \\ c_g\tilde{c}_1 & c_g\tilde{c}_2 & \dots & c_g\tilde{c}_g \end{pmatrix} + \begin{pmatrix} 0 & 0 & \dots & 0 \\ 0 & c_1\tilde{c}_1 & \dots & c_1\tilde{c}_{g-1} \\ \vdots & \dots & \ddots & \vdots \\ 0 & c_{g-2}\tilde{c}_2 & \dots & c_{g-2}\tilde{c}_{g-1} \\ 0 & c_{g-1}\tilde{c}_2 & \dots & c_{g-1}\tilde{c}_{g-1} \end{pmatrix} + \dots + \begin{pmatrix} 0 & \dots & 0 & 0 \\ 0 & \dots & 0 & 0 \\ \vdots & \dots & \ddots & \vdots \\ 0 & \dots & 0 & 0 \\ 0 & \dots & 0 & c_1\tilde{c}_1 \end{pmatrix}.$$

Example 11. Assume a general genus 3 curve and let the Mumford representations of the divisors D and D' be as usual. The matrix $\mathbf{U}$ is given as

$$\begin{pmatrix} -u_0 & u_2u_0 & -u_2^2u_0 + u_1u_0 \\ -u_1 & u_2u_1 - u_0 & -u_2^2u_1 + u_1^2 + u_2u_0 \\ -u_2 & u_2^2 - u_1 & -u_2^3 + 2u_2u_1 - u_0 \end{pmatrix} = \begin{pmatrix} -u_0 & u_2u_0 & (-u_2^2 + u_1)u_0 \\ -u_1 & u_2u_1 & (-u_2^2 + u_1)u_1 \\ -u_2 & u_2^2 & (-u_2^2 + u_1)u_2 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 \\ 0 & -u_0 & u_2u_0 \\ 0 & -u_1 & u_2u_1 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -u_0 \end{pmatrix},$$

and $\mathbf{U}'$ is given identically. In this case $\mathbf{c} = (u_0, u_1, u_2)^T$ and $\tilde{\mathbf{c}} = (-1, u_2, -u_2^2 + u_1)^T$. Setting $\mathbf{M} = \mathbf{U} - \mathbf{U}'$ and $\mathbf{z} = (v_0 - v'_0, v_1 - v'_1, v_2 - v'_2)^T$, we find the $g = 3$ coefficients ℓ_3, ℓ_4 and ℓ_5 of the quintic $\ell(x) = \sum_{i=0}^{5} \ell_i x^i$ that interpolates the 6 non-trivial elements in $\mathrm{supp}(D) \cup \mathrm{supp}(D')$ by solving $\mathbf{M} \cdot \mathbf{x} = \mathbf{z}$ for $\mathbf{x} = (\ell_3, \ell_4, \ell_5)^T$. The remaining coefficients are found via a straightforward matrix multiplication as $\tilde{\mathbf{x}} = (\ell_0, \ell_1, \ell_2)^T = \mathbf{U} \cdot \mathbf{x}$.

The immediate observation in general is that $\mathbf{c}\tilde{\mathbf{c}}^T$ is the only outer product that requires computation in order to determine $\mathbf{U}$ entirely.

For general divisor doublings the description of the linear system is much longer; this is because the right hand side vector $\mathbf{z}$ is slightly more complicated than in the case of addition: as is the case with general Weierstrass elliptic curves, additions tend to be independent of the curve constants whilst doublings do not. We reiterate that, for low genus implementations at least, Algorithm 2 is

Algorithm 2 General composition (doubling) of a unique divisor with itself

Input: $D = \{u_i, v_i\}_{0 \le i \le g-1}$ and curve coefficients $f_0, f_1, ..., f_{2g-1}$.

Output: $\ell(x) = \sum_{i=0}^{2g-1} \ell_i x^i$ such that each non-trivial element in $\text{supp}(D)$ occurs with multiplicity two in $\text{div}(\ell)$.

```
 1: $\mathbf{U}, \mathbf{M} \leftarrow \{0\}^{g \times g} \in \mathbb{F}_q^{g \times g}$, $\mathbf{v} \leftarrow \{0\}^{g-1} \in \mathbb{F}_q^{g-1}$, $\mathbf{z} \leftarrow \{0\}^{g} \in \mathbb{F}_q^{g}$
 2: for $i$ from 1 to $g$ do
 3:     $\mathbf{U}_{g+1-i,1} \leftarrow -u_{g-i}$
 4: end for
 5: for $j$ from 2 to $g$ do
 6:     $\mathbf{U}_{1,j} \leftarrow \mathbf{U}_{g,j-1} \cdot \mathbf{U}_{1,1}$.
 7:     for $i$ from 2 to $g$ do
 8:         $\mathbf{U}_{i,j} \leftarrow \mathbf{U}_{g,j-1} \cdot \mathbf{U}_{i,1} + \mathbf{U}_{i-1,j-1}$.
 9:     end for
10: end for
11: $u_{\text{extra}} \leftarrow \mathbf{U}_{g,1} \cdot \mathbf{U}_{g,g} + \mathbf{U}_{g-1,g}$.
12: for $i$ from 1 to $g$ do
13:     $\mathbf{M}_{g+1-i,1} \leftarrow v_{g-i}$
14: end for
15: for $j$ from 2 to $g$ do
16:     $\mathbf{M}_{i,j} \leftarrow \mathbf{M}_{i,j} + \mathbf{U}_{g,j-1} \cdot \mathbf{M}_{i,1} + \mathbf{M}_{g,j-1} \cdot \mathbf{U}_{i,1} + \mathbf{M}_{i-1,j-1}$.
17: end for
18: for $i$ from 1 to $g-1$ do
19:     $\mathbf{z}_{g+1-i} \leftarrow \mathbf{z}_{g+1-i} + 2 \cdot \mathbf{U}_{g,1} \cdot \mathbf{U}_{g+1-i,1} + \mathbf{U}_{g-i,1} + \mathbf{U}_{g,i+1} + f_{2g-i}$.
20:     for $j$ from 1 to $i$ do
21:         $\mathbf{z}_{g-i} \leftarrow \mathbf{z}_{g-i} + f_{2g-1-i+j} \cdot U_{g,j}$.
22:         $\mathbf{v}_i \leftarrow \mathbf{v}_i - \mathbf{M}_{g+1-j,1} \cdot \mathbf{M}_{g-i+j,1}$.
23:     end for
24: end for
25: $\mathbf{z}_1 \leftarrow \mathbf{z}_1 + 2 \cdot \mathbf{U}_{g,1} \cdot \mathbf{U}_{1,1} + f_g$.
26: $\mathbf{z}_{g-1} \leftarrow \mathbf{z}_{g-1} + \mathbf{v}_1$.
27: for $i$ from 3 to $g$ do
28:     for $j$ from 2 to $i-1$ do
29:         $\mathbf{z}_{g+1-i} \leftarrow \mathbf{z}_{g+1-i} + \mathbf{v}_{i-j} \cdot \mathbf{U}_{g,j-1}$.
30:     end for
31:     $\mathbf{z}_{g+1-i} \leftarrow \mathbf{z}_{g+1-i} + \mathbf{v}_{i-1}$.
32: end for
33: $\mathbf{z}_{1,1} \leftarrow \mathbf{z}_{1,1} + u_{\text{extra}}$.
34: for $i$ from 1 to $g$ do
35:     $\mathbf{z}_i \leftarrow \mathbf{z}_i / 2$.
36: end for
37: Solve $\mathbf{M} \cdot \mathbf{x} = \mathbf{z}$
38: Compute $\tilde{\mathbf{x}} = -\mathbf{U} \cdot \mathbf{x}$
39: for $i$ from 1 to $g$ do
40:     $\tilde{\mathbf{x}}_i \leftarrow v_{g-i} + \tilde{\mathbf{x}}_i$
41: end for
42: return $\ell(x)$ (from $\tilde{\mathbf{x}} = \{\ell_0, ..., \ell_{g-1}\}$ and $\mathbf{x} = \{\ell_g, ..., \ell_{2g-1}\}$)
```

intended to obtain the algebraic expressions for each element in **M**; as was the case with genus 2, a faster computational route to determining the composition function will probably arise from genus specific attention that derives tailor-made explicit formulas. Besides, the general consequence of Remark 10 is that many (if not all) of the values constituting **U** will have already been computed in the previous point operation, and can therefore be temporarily stored and reused.

5.2 Handling special cases

The description of divisor composition herein naturally encompasses the special cases where either (or both) of the divisors have degree less than g. In fact, Proposition 1 trivially generalizes to describe the set of divisors on $\text{Jac}(C_g)$ whose effective parts have degree $d \le g$, and can therefore be used to obtain the Mumford ideals associated with special input divisors[5].

This will often result in fewer rounds of reduction and a simpler linear system. For example, whilst the general addition of two full degree divisors in genus 3 requires an additional round of

[5] Perhaps the most general consequence of Proposition 1 is using it to describe (or enumerate) the entire Jacobian by summing over all d, as $\#\text{Jac}(C_g) = \#C_g + \sum_{d=2}^{g} n_d$, where n_d is the number of $2d$-tuples lying in the intersection of the d associated hypersurfaces.

reduction after the first points of intersection are found (see Figure 1 and Figure 2), it is easy to see that any group operation on a genus 3 curve involving a divisor of degree less than 3 will give rise to a reduced divisor immediately. Clearly, the linear systems in these cases are smaller, and therefore the explicit formulas arising in these special cases will always be much faster, in agreement with all prior expositions (cf. [3, §14]). In higher genus implementations that do not explicitly account for all special cases of inputs, Katagi *et al.* [28] noted that it can still be very advantageous to explicitly implement and optimize *one* of the special cases.

5.3 Reduction in low genera

Gaudry's chapter [18] gives an overview of different algorithms (and complexities) for the reduction phase. Our experiments lead us to believe that the usual method of reduction is still the most preferable for small g. In genus 2 we saw that point additions and doublings do not require more than one round of reduction, i.e. the initial interpolating function intersects C_2 in at most two more places (refer to Figure 3), immediately giving rise to the reduced divisor that is the sum. In genus $g \geq 3$ however, this is generally not the case. Namely, the initial interpolating function intersects C_g in more than g places, giving rise to an unreduced divisor that requires further reduction. We restate Cantor's complexity argument concerning the number of rounds of reduction ([6, §4]) in a geometric way in the following proposition.

Proposition 12. *In the addition of any two reduced divisor classes on the Jacobian of a genus g hyperelliptic curve, the number of rounds of further reduction required to form the reduced divisor is at most $\lfloor \frac{g-1}{2} \rfloor$, with equality occurring in the general case.*

Proof. For completeness note that addition on elliptic curves in Weierstrass form needs no reduction, so take $g \geq 2$. The composition polynomial $y = \ell(x)$ with the $2g$ prescribed zeros (including multiplicities) has degree $2g-1$. Substituting $y = \ell(x)$ into $C_g : y^2 + h(x)y = f(x)$ gives an equation of degree $\max\{2g+1, 3g-1, 2(2g-1)\} = 2(2g-1)$ in x, for which there are at most $2(2g-1) - 2g = 2g-2$ new roots. Let n_t be the maximum number of new roots after t rounds of reduction, so that $n_0 = 2g-2$. While $n_t > g$, reduction is not complete, so continue by interpolating the n_t new points with a polynomial of degree $n_t - 1$, producing at most $2(n_t - 1) - n_t = n_t - 2$ new roots. It follows that $n_t = 2g - 2t - 2$, and since $t, g \in \mathbb{Z}$, the result follows.

□

6 Further implications and potential

This section is intended to further illustrate the potential of coupling a geometric approach with linear algebra when performing arithmetic in Jacobians. It is our hope that the suggestions in this section encourage future investigations and improvements.

We start by commenting that our algorithm can naturally be generalized to much more than standard divisor additions and doublings. Namely, given any set of divisors $D_1, ..., D_n \in C_g$ and any corresponding set of scalars $r_1, ..., r_n \in \mathbb{Z}$, we can theoretically compute $D = \sum_{i=1}^{n} [r_i] D_i$ at once, by first prescribing a function that, for each $1 \leq i \leq n$, has a zero of order r_i at each of the non-trivial points in the support of D_i. Note that if $r_i \notin \mathbb{Z}^+$, then prescribing a zero of order r_i at some point P is equivalent to prescribing a pole of order $-r_i \in \mathbb{Z}^+$ at P instead. We first return to genus 1 to show that this technique can be used to recover several results that were previously obtained by alternatively merging or overlapping consecutive elliptic curve computations (cf. [10, 7]).

Simultaneous operations on elliptic curves. In the case of genus 1, the Mumford representation of reduced divisors is trivial, i.e. if $P = (x_1, y_1)$, the Mumford representation of the associated divisor is $D_P = (x - x_1, y_1)$, and the associated Mumford ideal is (isomorphic to) the curve itself. However, we can again explore using the Mumford representation as an alternative to derivatives in order to generate the required linear systems arising from prescribing multiplicities of greater than one. In addition, when unreduced divisors in genus 1 are encountered, the Mumford representation becomes non-trivial and very necessary for efficient computations.

To double-and-add or point triple on an elliptic curve, we can prescribe a parabola $\ell(x) = \ell_2 x^2 + \ell_1 x + \ell_0 \in \mathbb{F}_q(E)$ with appropriate multiplicities in advance, as an alternative to Eisenträger

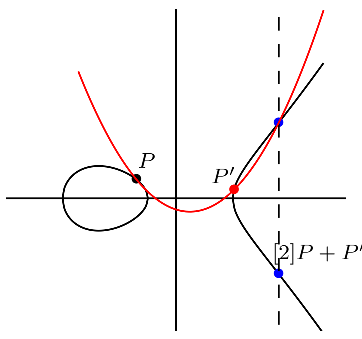

Fig. 5. Computing $[2]P+P'$ by prescribing a parabola which intersects E at P, P' with multiplicities two and one respectively.

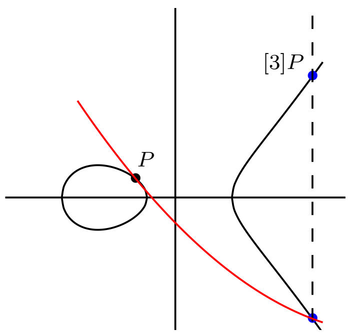

Fig. 6. Tripling the point $P \in E$ by prescribing a parabola which intersects E at P with multiplicity three.

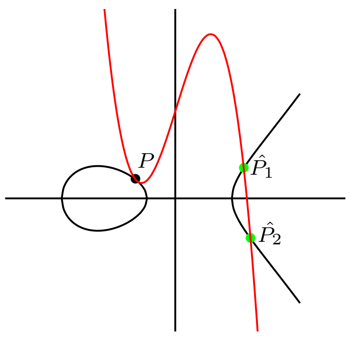

Fig. 7. Quadrupling the point $P \in E$ by prescribing a cubic which intersects E at P with multiplicity four.

et al.'s technique of merging two consecutive chords into a parabola [10]. Depending on the specifics of an implementation, computing the parabola in this fashion offers the same potential advantage as that presented by Ciet *et al.* [7]; we avoid any intermediate computations and bypass computing $P+P'$ or $[2]P$ along the way. When tripling the point $P = (x_P, y_P) \in E$, the parabola is determined from the three equalities $\ell(x)^2 \equiv x^3 + f_1x + f_0 \bmod \langle (x-u_0)^i \rangle$ for $1 \leq i \leq 3$, from which we take one of the coefficients that is identically zero in each of the three cases. As one example, we found projective formulas which compute triplings on curves of the form $y^2 = x^3 + f_0$ and cost $3\mathbf{M} + 10\mathbf{S}$ (see Appendix A). These are the second fastest tripling formulas reported across all curve models [5], being only slightly slower (unless $\mathbf{S} < 0.75\mathbf{M}$) than the formulas for tripling-oriented curves introduced by Doche *et al.* [9] which require $6\mathbf{M} + 6\mathbf{S}$.

We can quadruple the point P by prescribing a cubic function $\ell(x) = \ell_3x^3 + \ell_2x^2 + \ell_1x + \ell_0$ which intersects E at P with multiplicity four (see Figure 7). This time however, the cubic is zero on E in two other places, resulting in an unreduced divisor $D_{\hat{P}} = \hat{P}_1 + \hat{P}_2$, which we can represent in Mumford coordinates as $D_{\hat{P}} = (\hat{u}(x), \hat{v}(x))$ (as if it were a reduced divisor in genus 2). Our experiments agree with prior evidence that it is unlikely that point quadruplings will outperform consecutive doublings in the preferred projective cases, although we believe that one application which could benefit from this description is pairing computations, where interpolating functions are necessary in the computations. To reduce $D_{\hat{P}}$, we need the line $y = \hat{\ell}(x)$ joining $\hat{P}_1$ with $\hat{P}_2$, which can be computed via $\hat{\ell}(x) \equiv \ell(x) \bmod \langle \hat{u}(x) \rangle$. The update to the pairing function requires both $\ell(x)$ and $\hat{\ell}(x)$, as $f_{\text{upd}} = \ell(x)/\hat{\ell}(x)$. We claim that it may be attractive to compute a quadrupling in this fashion and only update the pairing function once, rather than two doublings which update the pairing functions twice, particularly in implementations where inversions don't compare so badly against multiplications [41]. It is also worth pointing out that in a quadruple-and-add computation, the unreduced divisor $D_{\hat{P}}$ need not be reduced before adding an additional point P'. Rather, it could be advantageous to immediately interpolate $\hat{P}_1$, $\hat{P}_2$ and P' with a parabola instead.

Simultaneous operations in higher genus Jacobians. Increasing the prescribed multiplicity of a divisor not only increases the degree of the associated interpolating function (and hence the linear system), but also generally increases the number of rounds of reduction required after composition. In the case of genus 1, we can get away with prescribing an extra zero (double-and-add or point tripling) without having to encounter any further reduction, but for genus $g \geq 2$, this will not be the case in general. For example, even when attempting to simultaneously compute $[2]D + D'$ for two general divisors $D, D' \in \text{Jac}(C_2)$, the degree of the interpolating polynomial becomes 5, instead of 3, and the dimension of the linear system that arises can only be trivially reduced from 6 to 4. Our preliminary experiments seem to suggest that unless the linear system can be reduced further, it is likely that computing $[2]D+D'$ simultaneously using our technique won't be as fast as computing two consecutive straightforward operations. However, as in the previous paragraph, we argue that such a trade-off may again become favorable in pairing computations where computing the higher-degree interpolating function would save a costly function update.

Explicit formulas in genus 3 and 4. Developing explicit formulas for hyperelliptic curves of genus 3 and 4 has also received some attention [51, 53, 22]. It will be interesting to see if the composition technique herein can further improve these results. In light of Remark 10 and the general description in Section 5, the new entries in the matrix $\mathbf{M}$ will often have been already computed in the previous point operation, suggesting an obvious extension of the coordinates if the storage space permits it. Therefore the complexity of our proposed composition essentially boils down to the complexity of solving the dimension g linear system in $\mathbb{F}_q$, and so it would also be interesting to determine for which (practically useful) genera one can find tailor-made methods of solving the special linear system that arises in Section 5.1.

Characteristic two, special cases, and more coordinates. Although the proofs in Section 3 were for arbitrary hyperelliptic curves over general fields, Section 4 simplified the exposition by focusing only on finite fields of large prime characteristic. Of course, it is possible that the description herein can be tweaked to also improve explicit formulas in the cases of special characteristic two curves (see [3, §14.5]). In addition, it is possible that the geometrically inspired derivation of explicit formulas for special cases of inputs will enhance implementations which make use of these (refer to Section 5.2). Finally, we only employed straightforward homogeneous coordinates to obtain the projective versions of our formulas. As was the case with the previous formulas based on Cantor's composition, it is possible that extending the projective coordinate system will give rise to even faster formulas.

7 Conclusion

This paper presents a new and explicit method of divisor composition for hyperelliptic curves. The method is based on using simple linear algebra to derive the required geometric functions directly from the Mumford coordinates of Jacobian elements. In contrast to Cantor's composition which operates in the polynomial ring $\mathbb{F}_q[x]$, the algorithm we propose is immediately explicit in terms of $\mathbb{F}_q$ operations. We showed that this achieves the current fastest general group law formulas in genus 2, and pointed out several other potential improvements that could arise from this exposition.

8 Acknowledgements

We wish to thank Huseyin Hisil and Michael Naehrig for many fixes and improvements to an earlier version of this paper.

References

1. F. K. Abu Salem and K. Khuri-Makdisi. Fast Jacobian group operations for $c_{3,4}$ curves over a large finite field. *CoRR*, abs/math/0610121, 2006.
2. R. Avanzi, N. Thériault, and Z. Wang. Rethinking low genus hyperelliptic jacobian arithmetic over binary fields: interplay of field arithmetic and explicit formulæ. *Journal of Mathematical Cryptology*, 2(3):227–255, 2008.
3. R. M. Avanzi, H. Cohen, C. Doche, G. Frey, T. Lange, K. Nguyen, and F. Vercauteren. *The Handbook of Elliptic and Hyperelliptic Curve Cryptography.* CRC, 2005.
4. D. J. Bernstein. Elliptic vs. hyperelliptic, part I. Talk at ECC, September 2006.
5. D. J. Bernstein and T. Lange. Explicit-formulas database. http://www.hyperelliptic.org/EFD.
6. D. G. Cantor. Computing in the Jacobian of a hyperelliptic curve. *Mathematics of computation*, 48(177):95–101, January 1987.
7. M. Ciet, M. Joye, K. Lauter, and P. L. Montgomery. Trading inversions for multiplications in elliptic curve cryptography. *Designs, Codes and Cryptography*, 39(2):189–206, 2006.
8. C. Diem. An index calculus algorithm for plane curves of small degree. In F. Hess, S. Pauli, and M. E. Pohst, editors, *ANTS*, volume 4076 of *Lecture Notes in Computer Science*, pages 543–557. Springer, 2006.
9. C. Doche, T. Icart, and D. R. Kohel. Efficient scalar multiplication by isogeny decompositions. In *PKC 2006 [54]*, pages 191–206, 2006.
10. K. Eisenträger, K. Lauter, and P. L. Montgomery. Fast elliptic curve arithmetic and improved Weil pairing evaluation. In M. Joye, editor, *CT-RSA*, volume 2612 of *Lecture Notes in Computer Science*, pages 343–354. Springer, 2003.
11. S. Erickson, M. J. Jacobson, N. Shang, and S. Shen A. Stein. Efficient formulas for real hyperelliptic curves of genus 2 in affine representation. In C. Carlet and B. Sunar, editors, *Arithmetic of finite fields*, volume 4547 of *Lecture Notes in Computer Science*, pages 202–218. Springer Berlin / Heidelberg, 2010.
12. X. Fan, G. Gong, and D. Jao. Efficient pairing computation on genus 2 curves in projective coordinates. In R. M. Avanzi, L. Keliher, and F. Sica, editors, *Selected Areas in Cryptography*, volume 5381 of *Lecture Notes in Computer Science*, pages 18–34. Springer Berlin / Heidelberg, 2009.

13. S. Flon, R. Oyono, , and C. Ritzenthaler. Fast addition on non-hyperelliptic genus 3 curves. *Algebraic geometry and its applications*, 5(3):227–256, 2008.
14. S. Flon and R. Oyono. Fast arithmetic on jacobians of picard curves. In F. Bao, R. H. Deng, and J. Zhou, editors, *Public Key Cryptography*, volume 2947 of *Lecture Notes in Computer Science*, pages 55–68. Springer, 2004.
15. S. D. Galbraith. *Mathematics of Public Key Cryptography.* URL: http://www.math.auckland.ac.nz/∼sgal018/crypto-book/crypto-book.html, 0.9 edition, February 11, 2011.
16. S. D. Galbraith, M. Harrison, and D. J. Mireles Morales. Efficient hyperelliptic arithmetic using balanced representation for divisors. In A. van der Poorten and A. Stein, editors, *Algorithmic Number Theory*, volume 5011 of *Lecture Notes in Computer Science*, pages 342–356. Springer Berlin / Heidelberg, 2008.
17. P. Gaudry. An algorithm for solving the discrete log problem on hyperelliptic curves. In B. Preneel, editor, *EUROCRYPT*, volume 1807 of *Lecture Notes in Computer Science*, pages 19–34. Springer, 2000.
18. P. Gaudry. *Hyperelliptic curves and the HCDLP*, volume 317 of *London Mathematical Society Lecture Notes*, chapter VII, pages 133–150. Cambridge University Press, 2005.
19. P. Gaudry. Fast genus 2 arithmetic based on Theta functions. *Journal of Mathematical Cryptology*, 1(3):243–265, 2007.
20. P. Gaudry and R. Harley. Counting points on hyperelliptic curves over finite fields. In W. Bosma, editor, *ANTS*, volume 1838 of *Lecture Notes in Computer Science*, pages 313–332. Springer, 2000.
21. P. Gaudry, E. Thomé, N. Thériault, and C. Diem. A double large prime variation for small genus hyperelliptic index calculus. *Math. Comput.*, 76(257):475–492, 2007.
22. M. Gonda, K. Matsuo, K. Aoki, J. Chao, and S. Tsujii. Improvements of addition algorithm on genus 3 hyperelliptic curves and their implementation. *IEICE Transactions on Fundamentals of Electronics Communications and Computer Sciences*, pages 89–96, 2005.
23. C. Gurot, K. Kaveh, and V. M. Patankar. Explicit algorithm for the arithmetic on the hyperelliptic Jacobians of genus 3. *Journal of the Ramanujan Mathematical Society*, 19:75–115, 2004.
24. R. Harley. Fast arithmetic on genus 2 curves. See http://cristal.inria.fr/∼harley/hyper for `C` source code and further explanations.
25. F. Hess. Computing Riemann-Roch spaces in algebraic function fields and related topics. *J. Symb. Comput.*, 33(4):425–445, 2002.
26. H. Hisil. *Elliptic curves, group law, and efficient computation.* PhD thesis, Queensland University of Technology, 2010.
27. M. A. Huang and D. Ierardi. Efficient algorithms for the Riemann-Roch problem and for addition in the Jacobian of a curve. *J. Symb. Comput.*, 18(6):519–539, 1994.
28. M. Katagi, I. Kitamura, T. Akishita, and T. Takagi. Novel efficient implementations of hyperelliptic curve cryptosystems using degenerate divisors. In C. H. Lim and M. Yung, editors, *WISA*, volume 3325 of *Lecture Notes in Computer Science*, pages 345–359. Springer, 2004.
29. K. Khuri-Makdisi. Linear algebra algorithms for divisors on an algebraic curve. *Math. Comput.*, 73(245):333–357, 2004.
30. K. Khuri-Makdisi. Asymptotically fast group operations on jacobians of general curves. *Math. Comput.*, 76(260):2213–2239, 2007.
31. N. Koblitz. Elliptic curve cryptosystems. *Mathematics of computation*, 48(177):203–209, 1987.
32. N. Koblitz. Hyperelliptic cryptosystems. *J. Cryptology*, 1(3):139–150, 1989.
33. S. Lang. *Introduction to algebraic geometry.* Addison-Wesley, 1972.
34. T. Lange. *Efficient arithmetic on hyperelliptic curves.* PhD thesis, Universität-Gesamthochschule Essen, 2001.
35. T. Lange. Efficient arithmetic on genus 2 hyperelliptic curves over finite fields via explicit formulae. Cryptology ePrint Archive, Report 2002/121, 2002. http://eprint.iacr.org/.
36. T. Lange. Inversion-free arithmetic on genus 2 hyperelliptic curves. Cryptology ePrint Archive, Report 2002/147, 2002. http://eprint.iacr.org/.
37. T. Lange. Weighted coordinates on genus 2 hyperelliptic curves. Cryptology ePrint Archive, Report 2002/153, 2002. http://eprint.iacr.org/.
38. T. Lange. Formulae for arithmetic on genus 2 hyperelliptic curves. *Appl. Algebra Eng. Commun. Comput.*, 15(5):295–328, 2005.
39. T. Lange. Elliptic vs. hyperelliptic, part II. Talk at ECC, September 2006.
40. K. Lauter. The equivalence of the geometric and algebraic group laws for Jacobians of genus 2 curves. In *Topics in algebraic and noncommutative geometry: proceedings in memory of Ruth Michler, July 20-22, 2001, Luminy, France [and] October 25-28, 2001, Annapolis, Maryland*, volume 324, page 165. Amer Mathematical Society, 2003.
41. K. Lauter, P. L. Montgomery, and M. Naehrig. An analysis of affine coordinates for pairing computation. In M. Joye, A. Miyaji, and A. Otsuka, editors, *Pairing-Based Cryptography - Pairing 2010*, volume 6487 of *Lecture Notes in Computer Science*, pages 1–20. Springer Berlin / Heidelberg, 2010.
42. F. Leitenberger. About the group law for the Jacobi variety of a hyperelliptic curve. *Contributions to Algebra and Geometry*, 46(1):125–130, 2005.
43. K Matsuo, J. Chao, and S. Tsujii. Fast genus two hyperelliptic curve cryptosystems. Technical Report 214, IEIC, 2001.
44. V. S. Miller. Use of elliptic curves in cryptography. In H. C. Williams, editor, *CRYPTO*, volume 218 of *Lecture Notes in Computer Science*, pages 417–426. Springer, 1985.
45. Y. Miyamoto, H. Doi, K. Matsuo, J. Chao, and S. Tsujii. A fast addition algorithm of genus two hyperelliptic curve. In *Symposium on Cryptography and Information Security - SCICS*, In Japanese, 2002.
46. D. Mumford. Tata lectures on theta II. In *Progress in Mathematics*, volume 43. Birkhiauser Boston Inc., Boston, MA, 1984.
47. J.M. Pollard. Monte Carlo methods for index computation (mod p). *Mathematics of computation*, 32(143):918–924, 1978.

48. B. Smith. Isogenies and the discrete logarithm problem in Jacobians of genus 3 hyperelliptic curves. *Journal of Cryptology*, 22(4):505–529, 2009.
49. H. Sugizaki, K. Matsuo, J. Chao, and S. Tsujii. An extension of Harley addition algorithm for hyperelliptic curves over finite fields of characteristic two. Technical Report ISEC2002-9(2002-5), IEICE, 2002.
50. M Takahashi. Improving Harley algorithms for Jacobians of genus 2 hyperelliptic curves. In *Symposium on Cryptography and Information Security - SCICS*, In Japanese., 2002.
51. T. Wollinger. *Software and hardware implementation of hyperelliptic curve cryptosystems.* PhD thesis, Ruhr-University of Bochum, 2004.
52. T. Wollinger and V. Kovtun. Fast explicit formulae for genus 2 hyperelliptic curves using projective coordinates. In *Fourth International Conference on Information Technology*, pages 893 – 897, 2007.
53. T. Wollinger, J. Pelzl, and C. Paar. Cantor versus Harley: optimization and analysis of explicit formulae for hyperelliptic curve cryptosystems. *IEEE Transactions on Computers*, pages 861–872, 2005.
54. M. Yung, Y Dodis, A. Kiayias, and T. Malkin, editors. *9th international conference on theory and practice in public-key cryptography, New York, NY, USA, April 24–26, 2006, proceedings*, volume 3958 of *Lecture Notes in Computer Science*, Berlin, 2006. Springer.

A Magma scripts for affine genus 2 formulas (and projective genus 1 tripling)

```
function AffADD(u1,u0,v1,v0,u1s,u01,u1d,u0d,v1d,v0d,u1ds,u01d);
    uS:=u1+u1d; vOD:=v0-v0d; v1D:=v1-v1d; M1:=u1s-u0-u1ds+u0d; M2:=u01d-u01;
    M3:=u1-u1d; M4:=u0d-u0; t1:=(M2-vOD)*(v1D-M1); t2:=(-vOD-M2)*(v1D+M1);            //2M
    t3:=(-vOD+M4)*(v1D-M3); t4:=(-vOD-M4)*(v1D+M3);                                    //2M
    l2:=t1-t2; l3:=t3-t4; d:=t3+t4-t1-t2-2*(M2-M4)*(M1+M3);                            //1M
    A :=1/(d*l3); B :=d*A; C :=d*B; D :=l2*B;                                        //I+4M
    E :=l3^2 *A; Cs :=C^2; u1dd := 2*D-Cs-uS;                                        //1M+2S
    u0dd := D^2 + C*(v1+v1d) -((u1dd-Cs)*uS+(u1s+u1ds))/2;                           //2M+1S
    uu1dd :=u1dd^2; uu0dd:=u1dd*u0dd; v1dd := D*(u1-u1dd)+ uu1dd-u0dd-u1s+u0;        //2M+1S
    v0dd := D*(u0-u0dd) + uu0dd - u01; v1dd := -(E*v1dd + v1);   v0dd := -(E*v0dd + v0);  //3M
    Jac![x^2+u1dd*x+u0dd,v1dd*x+v0dd]; //Check
    return   u1dd,u0dd,v1dd,v0dd,uu1dd,uu0dd;                                           //Total
 end  function;                                                                   //I+17M+4S
```

Table 3. MAGMA code for a general (affine) addition $D'' = D + D'$ of two degree 2 divisors on $\mathrm{Jac}(C_2)$.

```
function AffDBL(u1,u0,v1,v0,uu1,uu0,f2,f3);
    vv:=v1^2 ; valpha:=(v1+u1)^2-vv-uu1; M1:=2*v0-2*valpha; M2:=2*v1*(u0+2*uu1);   //1M+2S
    M3:=-2*v1; M4:=valpha+2*v0; z1:=f2+2*uu1*u1+2*uu0-vv;                            //1M
     z2:=f3-2*u0+3*uu1; t1:=(M2-z1)*(z2-M1); t2:=(-z1-M2)*(z2+M1);                   //2M
    t3:=(-z1+M4)*(z2-M3); t4:=(-z1-M4)*(z2+M3); l2:=t1-t2; l3:=t3-t4;                //2M
    d:=t3+t4-t1-t2-2*(M2-M4)*(M1+M3); A :=1/(d*l3);                                 //I+2M
    B :=d*A; C :=d*B; D :=l2*B; E :=l3^2 *A; u1dd := 2*D-C^2 -2*u1;                 //4M+2S
    u0dd := (D-u1)^2 + 2*C*(v1 +C*u1); uu1dd:=u1dd^2 ;   uu0dd:=u1dd*u0dd;          //3M+2S
    v1dd := D*(u1-u1dd)+uu1dd-uu1-u0dd+u0; v0dd := D*(u0-u0dd)+(uu0dd-uu0);          //2M
    v1dd := -(E*v1dd + v1);   v0dd := -(E*v0dd + v0);                                //2M
    Jac![x^2+u1dd*x+u0dd,v1dd*x+v0dd]; //Check
    return   u1dd,u0dd,v1dd,v0dd,uu1dd,uu0dd;                                      //Total
 end  function;                                                              //I+19M+6S
```

Table 4. MAGMA code for a general (affine) doubling $D'' = [2]D$ of a degree 2 divisor on $D \in \mathrm{Jac}(C_2)$.

```
function ProjTRP(X,Y,Z,f0);
    Y2:=Y^2;    Z2:=f0*Z^2;   Y4:=Y2^2;   Z4:=Z2^2;   Y8:=Y4^2;   Z8:=Z4^2;                  //6S
    Y2Z2:=((Y2+Z2)^2-Y4-Z4)/2;    Y2Z22:=Y2Z2^2;    Y4Y2Z2:=((Y4+Y2Z2)^2-Y2Z22-Y8);          //3S
    Y2Z2Z4:=((Y2Z2+Z4)^2-Y2Z22-Z8);    Y4Y2Z2:=4*Y4Y2Z2;    Y2Z22:=18*Y2Z22;    Z8:=27*Z8;   //1S
    Z3:=27*Z*(Y8+Y4Y2Z2+Y2Z22-Z8);    Z8:=3*Z8;    Y2Z2Z4:=36*Y2Z2Z4;                        //1M
    X3:=3*X*(Y8-3*(4*Y4Y2Z2+Y2Z22-3*Y2Z2Z4+Z8)); Y3:=Y*(Y8+27*(Y4Y2Z2-5*Y2Z22+2*Y2Z2Z4-Z8));  //2M
    return   X3,Y3,Z3;                                                                      //Total
 end  function;                                                                       //3M+10S
```

Table 5. MAGMA code for a general (projective) tripling $P'' = [3]P$ of a point $P \in E/F_q : y^2 = x^3 + a_0$.

B Magma scripts for projective genus 2 formulas

```
function ProjADD(U1,U0,V1,V0,Z,U1d,U0d,V1d,V0d,Zd);
    ZZ:=Z*Zd; U1Z:=U1*Zd; U1dZ:=U1d*Z; U1ZS:=U1Z^2; U1dZS:=U1dZ^2; U0Z:=U0*Zd; U0dZ:=U0d*Z;     //5M + 2S
    V1Z:=V1*Zd; V1dZ:=V1d*Z; M1:=U1ZS-U1dZS+ZZ*(U0dZ-U0Z); M2:=U1dZ*U0dZ-U1Z*U0Z;                //5M
     M3:=U1Z-U1dZ; M4:=U0dZ-U0Z; z1:=V0*Zd-V0d*Z; z2:=V1Z-V1dZ; t1:=(M2-z1)*(z2-M1);             //3M
    t2:=(-z1-M2)*(z2+M1); t3:=(-z1+M4)*(z2-M3); t4:=(-z1-M4)*(z2+M3); l2:=t1-t2; l3:=t3-t4;     //3M
    d:=t3+t4-t1-t2-2*(M2-M4)*(M1+M3); A:=d^2; B:=l3*ZZ;                                         //2M + 1S
    C:=l2*B; D:=d*B; E:=l3*B; F:=U1Z*E; G:=ZZ*E; H:=U0Z*G; J:=D*G; K:=Zd*J;                     //8M
    U1dd := 2*C-A-E*(U1Z+U1dZ);                                                                 //1M
    U0dd := l2^2*ZZ + D*(V1Z+V1dZ) -((U1dd-A)*(U1Z+U1dZ)+E*(U1ZS+U1dZS))/2;                     //4M + 1S
    V1dd := U1dd*(U1dd-C) + F*(C-F) +E*(H-U0dd); V0dd :=H*(C- F) + U0dd*(U1dd -C);              //5M
    V1dd := -(V1dd*ZZ + K*V1); V0dd := -(V0dd + K*V0); U1dd:=U1dd*D*ZZ; U0dd:=U0dd*D; Zdd:=ZZ*J; //7M
    return   U1dd,U0dd,V1dd,V0dd,Zdd;                                                           //Total
 end  function;                                                                                 //43M + 4S
```

Table 6. MAGMA code for a general addition $D'' = D \oplus D'$ of two degree 2 divisors on $\mathrm{Jac}(C_2)$ in projective coordinates.

```
function ProjDBL(U1,U0,V1,V0,Z,f2,f3);
    UU:=U1*U0; U1S:=U1^2; ZS:=Z^2; V0Z:=V0*Z; U0Z:=U0*Z; V1S:=V1^2; UV:=(V1+U1)^2-V1S-U1S; //3M + 4S
    M1:=2*V0Z-2*UV; M2:=2*V1*(U0Z+2*U1S); M3:=-2*V1; M4:=UV+2*V0Z;                          //1M
     z1:=Z*(f2*ZS-V1S)+2*U1*(U1S+U0Z); z2:=f3*ZS-2*U0Z+3*U1S;                               //2M
    t1:=(M2-z1)*(z2-M1); t2:=(-z1-M2)*(z2+M1); t3:=(-z1+M4)*(z2-M3); t4:=(-z1-M4)*(z2+M3);  //4M
    l2:=t1-t2; l3:=t3-t4; d:=t3+t4-t1-t2-2*(M2-M4)*(M1+M3);                                  //1M
    A:=l2^2; B:=l3^2; C:=((l2+l3)^2-A-B)/2; D:=B*Z; E:=B*U1; F:=d^2; G:=F*Z;                //3M + 4S
    H:=((d+l3)^2-F-B)/2; J:=H*Z; K:=V1*J; L:=U0Z*B; U1dd := 2*C-2*E-G;                      //3M + 1S
    U0dd := A+U1*(E-2*C +2*G) + 2*K; V1dd := (C-E-U1dd)*(E-U1dd)+B*(L -U0dd);               //3M
    V0dd := L*(C-E) +(U1dd-C)*U0dd; V1dd := -(V1dd*Z + K*D); V0dd := -(V0dd + V0Z*H*D);     //6M
    M:=J*Z; U1dd:=U1dd*M; U0dd:=U0dd*J; Zdd:=M*D;                                           //4M
    return   U1dd,U0dd,V1dd,V0dd,Zdd;                                                       //Total
 end  function;                                                                             //30M + 9S
```

Table 7. MAGMA code for a general doubling $D'' = [2]D$ of a degree 2 divisor on $\mathrm{Jac}(C_2)$ in projective coordinates.

```
function ProjMIXED(U1,U0,V1,V0,Z,u1,u0,v1,v0);
    u1Z:=u1*Z; U1S:=U1^2; u1ZS:=u1Z^2; u0Z:=u0*Z; M1:=u1ZS-U1S+Z*(U0-u0Z);                   //3M + 2S
    M2:=U1*U0-u1Z*u0Z; M3:=u1Z-U1; M4:=U0-u0Z; v1Z:=v1*Z; z1:=v0*Z-V0; z2:=v1Z-V1;          //4M
     t1:=(M2-z1)*(z2-M1); t2:=(-z1-M2)*(z2+M1); t3:=(-z1+M4)*(z2-M3); t4:=(-z1-M4)*(z2+M3); //4M
    l2:=t1-t2; l3:=t3-t4; d:=t3+t4-t1-t2-2*(M2-M4)*(M1+M3);                                  //1M
    A:=d^2; B:=l3*Z; C:=d*B; D:=l2*B; E:=l3*B; F:=E*u1Z; G:=B^2; H:=u0Z*G; J:=C*G;          //7M + 2S
    Zdd:=Z*J; U1dd:= 2*D-A-E*(u1Z+U1);                                                      //2M
    U0dd := l2^2*Z + C*(v1Z+V1) -((U1dd-A)*(u1Z+U1)+E*(u1ZS+U1S))/2;                        //4M + 1S
    V1dd := F*(D-F) +U1dd*(U1dd-D) +E*(H-U0dd); V0dd := H*(D - F) + (U1dd-D)*U0dd ;         //5M
    V1dd := -(Z*V1dd + Zdd*v1); V0dd := -(V0dd + Zdd*v0); U1dd:=U1dd*Z*C; U0dd:=U0dd*C;     //6M
    return   U1dd,U0dd,V1dd,V0dd,Zdd;                                                       //Total
 end  function;                                                                             //36M + 5S
```

Table 8. MAGMA code for a mixed addition $D'' = D + D'$ of two degree 2 divisors on $\mathrm{Jac}(C_2)$, where D is in projective coordinates and D' is in affine coordinates.

C Magma scripts for arbitrary genus composition

```
clear; q:=NextPrime(2^30); g:=6;                          /* Input prime characteristic and genus */
Fq:=GF(q); Poly<x>:=PolynomialRing(Fq);
coeffs:=[];
for i:=1 to 2*g do     coeffs:=Append(coeffs,Random(0,q));     end for;
f:=x^(2*g+1);                                             /* Create Random Hyperelliptic Curve */
for i:=1 to 2*g do
    f+:=coeffs[i]*x^(i-1);
end for;
C:=HyperellipticCurve(f); g:=Genus(C); Jac:=Jacobian(C); Inf:=PointsAtInfinity(C)[1];
PointsVec1:=[]; PointsVec2:=[];                                  /* Create full degree divisors */
for i:=1 to g do
    PointsVec1:=Append(PointsVec1,Random(C)); PointsVec2:=Append(PointsVec2,Random(C));
end for;
J1:=Jac![[PointsVec1[i]: i in [1..g]],[Inf: i in [1..g]]];
J2:=Jac![[PointsVec2[i]: i in [1..g]],[Inf: i in [1..g]]];
MumfordTuple1:=[]; MumfordTuple2:=[];                     /* Put 2g Mumford coordinates into lists */
for i:=1 to g do
    MumfordTuple1:=Append(MumfordTuple1, Coefficients(J1[1])[g+1-i]);
    MumfordTuple2:=Append(MumfordTuple2, Coefficients(J2[1])[g+1-i]);
end for;
for i:=1 to g do
    MumfordTuple1:=Append(MumfordTuple1, Coefficients(J1[2])[g+1-i]);
    MumfordTuple2:=Append(MumfordTuple2, Coefficients(J2[2])[g+1-i]);
end for;
U1:=ZeroMatrix(Fq,g,g); U2:=ZeroMatrix(Fq,g,g);
for i:=1 to g do
    U1[g+1-i,1]:=-MumfordTuple1[i]; U2[g+1-i,1]:=-MumfordTuple2[i];
end for;
for j:=2 to g do
    U1[1,j]:=U1[g,j-1]*U1[1,1]; U2[1,j]:=U2[g,j-1]*U2[1,1];
    for i:=2 to g do
        U1[i,j]:=U1[i,j]+U1[g,j-1]*U1[i,1]+U1[i-1,j-1];
        U2[i,j]:=U2[i,j]+U2[g,j-1]*U2[i,1]+U2[i-1,j-1];
    end for;
end for;
M:=U1-U2; z:=[];                                             /* Construct right hand side vector z */
for i:=1 to g do
    z:=Append(z,MumfordTuple1[2*g+1-i]-MumfordTuple2[2*g+1-i]);
end for;                                                       /* Magmas solve needs transposes */
M:=Transpose(M);z:=Vector(Fq,z); sols:=Solution(M,z); solVec:=ZeroMatrix(Fq,g,1);
for i:=1 to g do                                         /* Solve linear system for li (i > g − 1) */
    solVec[i,1]:=sols[i];
end for;
solVec2:=U1*solVec;                                                      /* Get remaining li */
for i:=1 to g do
    solVec2[g+1-i][1]:= MumfordTuple1[g+i]-solVec2[g+1-i][1];
end for;
Y:=Poly!0;
for i:=1 to g do
    Y+:=solVec2[i][1]*x^(i-1); Y+:=solVec[i][1]*x^(g+i-1);
end for;
IsDivisibleBy(Y^2-f,J1[1]*J2[1]);  /* Construct polynomial and check intersection */
```

Table 9. Script for composition between two unique divisors (Algorithm 1) on arbitrary genus curves.

Once the characteristic q and the genus g have been specified, the algorithms above and below generate an arbitrary imaginary hyperelliptic curve over $\mathbb{F}_q$ of genus g, and respectively perform the geometric composition between two unique divisors (addition) and a divisor and itself (doubling).

```
clear; q:=NextPrime(2^30); g:=6;                        /* Input prime characteristic and genus */
Fq:=GF(q); Poly<x>:=PolynomialRing(Fq);
coeffs:=[];
for i:=1 to 2*g do
coeffs:=Append(coeffs,Random(0,q));
end for;
f:=x^(2*g+1);                                           /* Create Random Hyperelliptic Curve */
for i:=1 to 2*g do
    f+:=coeffs[i]*x^(i-1);
end for;
C:=HyperellipticCurve(f); g:=Genus(C); Jac:=Jacobian(C); Inf:=PointsAtInfinity(C)[1];
PointsVec:=[];                                              /* Create full degree divisor */
for i:=1 to g do
    PointsVec:=Append(PointsVec,Random(C));
end for;
J1:=Jac![[PointsVec[i]: i in [1..g]],[Inf: i in [1..g]]];
MumfordTuple:=[];                                       /* Put 2g Mumford coordinates into list */
for i:=1 to g do
    MumfordTuple:=Append(MumfordTuple, Coefficients(J1[1])[g+1-i]);
end for;
for i:=1 to g do
MumfordTuple:=Append(MumfordTuple, Coefficients(J1[2])[g+1-i]);
end for;                                                              /* Initialize */
U:=ZeroMatrix(Fq,g,g); M:=ZeroMatrix(Fq,g,g); v:=ZeroMatrix(Fq,g-1,1); z:=ZeroMatrix(Fq,g,1);
for i:=1 to g do
    U[g+1-i,1]:=-MumfordTuple[i];
end for;                                                    /* Form U (same as addition) */
for j:=2 to g do
    U[1,j]:=U[g,j-1]*U[1,1];
    for i:=2 to g do
        U[i,j]:=U[g,j-1]*U[i,1]; U[i,j]+:=U[i-1,j-1];
    end for;
end for;
uExtra:=U[g,1]*U[g,g]+U[g-1,g];                            /* Extra element required for M */
for i:=1 to g do
  M[g+1-i,1]:=MumfordTuple[i+g];
end for;                                                        /* Construct matrix M */
for j:=2 to g do
    M[1,j]:=M[1,j]+U[g,j-1]*M[1,1]+M[g,j-1]*U[1,1];
    for i:=2 to g do
        M[i,j]:=M[i,j]+U[g,j-1]*M[i,1]+M[i-1,j-1]+M[g,j-1]*U[i,1];
    end for;
end for;
for i:=1 to g-1 do                                    /* Construct right hand side vector z */
    z[g+1-i,1]+:=2*U[g,1]*U[g+1-i,1] + U[g-i,1]+U[g,i+1] + coeffs[2*g+1-i];
    for j:=1 to i do
        z[g-i,1]+:=coeffs[2*g-i+j]*U[g,j]; v[i,1]+:=-M[g+1-j,1]*M[g-i+j,1];
    end for;
end for;
z[1,1]+:=2*U[g,1]*U[1,1] + coeffs[g+1]; z[g-1,1]+:=v[1,1];
for i:=3 to g do
    for j:=2 to i-1 do
        z[g+1-i,1]+:=v[i-j,1]*U[g,j-1];
    end for;
    z[g+1-i,1]+:=v[i-1,1];
end for;
z[1,1]+:=uExtra;
for i:=1 to g do
    z[i,1]/:=2;
end for;
M:=Transpose(M); z:=Vector(Fq,Transpose(z));               /* Magmas solve needs transposes */
sols:=Solution(M,z); solVec:=ZeroMatrix(Fq,g,1);        /* Solve linear system for b_i (i > g − 1) */
for i:=1 to g do
    solVec[i,1]:=sols[i];
end for;
solVec2:=-U*solVec;                                             /* Get remaining b_i */
for i:=1 to g do
    solVec2[i,1]:=MumfordTuple[2*g+1-i]+solVec2[i,1];
end for;
Y:=Poly!0;
for i:=1 to g do
    Y+:=solVec2[i][1]*x^(i-1); Y+:=solVec[i][1]*x^(g+i-1);
end for;
IsDivisibleBy(Y^2-f,J1[1]^2);   /* Construct polynomial and check intersection */
```

Table 10. Script for geometric composition (Algorithm 2) between a divisor and itself on arbitrary genus curves.