Jacobian Coordinates on Genus 2 Curves

Huseyin Hisil[1] and Craig Costello[2]

[1] Yasar University, Izmir, Turkey
huseyin.hisil@yasar.edu.tr
[2] Microsoft Research, Redmond, USA
craigco@microsoft.com

Abstract. This paper presents a new projective coordinate system and new explicit algorithms which together boost the speed of arithmetic in the divisor class group of genus 2 curves. The proposed formulas generalise the use of Jacobian coordinates on elliptic curves, and their application improves the speed of performing cryptographic scalar multiplications in Jacobians of genus 2 curves over prime fields by an approximate factor of 1.25x. For example, on a single core of an Intel Core i7-3770M (Ivy Bridge), we show that replacing the previous best formulas with our new set improves the cost of generic scalar multiplications from 243,000 to 195,000 cycles, and drops the cost of specialised GLV-style scalar multiplications from 166,000 to 129,000 cycles.

Keywords: Genus 2, hyperelliptic curves, explicit formulas, Jacobian coordinates, scalar multiplication.

1 Introduction

Motivated by the popularity of low-genus curves in cryptography [29, 22, 23], we put forward a new system of projective coordinates that facilitates efficient group law computations in the Jacobians of hyperelliptic curves of genus 2. This paper combines several techniques to arrive at explicit formulas that are significantly faster than those in previous works [25, 9]. The two main ingredients we use in the derivation are:

- The generalisation of Jacobian coordinates from the elliptic curve setting to the hyperelliptic curve setting: these coordinates essentially cast affine points into projective space according to the *weights* of x and y in the defining curve equation. While applying Jacobian coordinates to elliptic curves is straightforward, their application to hyperelliptic curves requires transferring the x-y weightings into weightings for the Mumford coordinates. As it does for the x-y coordinates in genus 1, this projection naturally balances the Mumford coordinates to facilitate substantial simplifications in the projective genus 2 group law formulas.
- The adaptation of Meloni's "co-Z" idea [28] to the genus 2 setting. Although originally proposed in the context of addition-only (e.g. Fibonacci-style)

chains, this approach can also be used to gain performance in the more meaningful context of binary addition chains. Moreover, this idea is especially advantageous when used in conjunction with Jacobian coordinates.

The application of the above techniques, as well as some further optimisations discussed in the body of this paper, gives rise to the operation counts in Table 1 – the counts here include field multiplications ($\mathbf{M}$), squarings ($\mathbf{S}$), and multiplications by curve constants ($\mathbf{D}$). Here we make a brief comparison with the previous works in [25] and [9], by considering the two most common operations in the context of cryptographic scalar multiplications: a point doubling (denoted DBL), and a mixed-doubling-and-addition (denoted mDBLADD) between two points[3]. These two operations constitute the bottleneck of most state-of-the-art scalar multiplication routines, since the multiplication of a point in the Jacobian by an n-bit scalar typically requires α DBL operations and β mDBLADD operations, where $\alpha + \beta \approx n$. Thus, the improved operation counts in Table 1 give a rough idea of the speedups that we can expect when plugging these formulas into an existing genus 2 scalar multiplication routine that uses the formulas from [25] or [9]. (We give a better indication of the improvements over previous formulas by reporting concrete implementation numbers in Section 8.) As well as the reduction in field multiplications indicated in Table 1, the explicit formulas in this paper also require far fewer field additions than those in [25] and [9]. We note that the biggest relative difference occurs in the mDBLADD column: among other things, this difference results from the combination of the new coordinate system with the extension of Meloni's idea [28], which allows us to compute mDBLADD operations independently of the curve constants. On the other hand, when such curve constants are zero, certain operations in this paper become even faster (relatively speaking): for example, on the two special families exhibiting endomorphisms used in [7], the doubling formulas in [25] and [9] save $2\mathbf{D}$, while the new operation count reported for DBL in Table 1 saves $3\mathbf{S}+2\mathbf{D}$ to drop down to $21\mathbf{M}+9\mathbf{S}$.

authors	DBL	mADD	mDBLADD
Lange [25]	$32\mathbf{M}+7\mathbf{S}+2\mathbf{D}$	$36\mathbf{M}+5\mathbf{S}$	$68\mathbf{M}+12\mathbf{S}+2\mathbf{D}$
Costello-Lauter [9]	$30\mathbf{M}+9\mathbf{S}+2\mathbf{D}$	$36\mathbf{M}+5\mathbf{S}$	$66\mathbf{M}+14\mathbf{S}+2\mathbf{D}$
This work	$21\mathbf{M}+12\mathbf{S}+2\mathbf{D}$	$29\mathbf{M}+7\mathbf{S}$	$52\mathbf{M}+11\mathbf{S}$

Table 1. Field operation counts obtained in this work, versus two previous works, for the most common operations incurred during cryptographic scalar multiplications in Jacobians of genus 2 curves of the form $\mathcal{C}/K : y^2 = f(x)$, where $f(x)$ is of degree 5 and the characteristic of K is greater than 5.

[3] For genus 2 scalar multiplications, it is usually advantageous to convert precomputed (lookup table) points to their affine representation using a shared inversion – see Section 7.2. This is why the double-and-add operations involve a "mixed" addition.

While the formulas in this paper target Jacobians of imaginary genus 2 curves, Gaudry showed in [16] that one can perform cryptographic scalar multiplications much more efficiently in the special case that the Jacobian of the curve $\mathcal{C}/K$ has K-rational two-torsion, by instead working on an associated Kummer surface. To illustrate the difference between working on the Kummer surface and working in the full Jacobian group, Gaudry's analogous operation counts are a blazingly fast $6\mathbf{M} + 8\mathbf{S}$ for `DBL` and $16\mathbf{M} + 9\mathbf{S}$ for `mDBLADD`. Referring back to Table 1, it is clear that raw scalar multiplications on the Kummer surface will remain unrivalled by those in the full Jacobian group. However, there are several cryptographic caveats related to the Kummer surface that justify the continued exploration of fast algorithms for traditional arithmetic in the Jacobian. Namely, Kummer surfaces do not support *generic* additions, so while they are extremely fast in the realm of key exchange (where such additions are not necessary), it is not yet known how to efficiently use the Kummer surface in a wider realm of cryptographic settings, e.g. for general digital signatures[4]. Furthermore, the absence of generic additions complicates the application of endomorphisms [7, §8.5], and from a more pragmatic standpoint, also prevents the use of standard precomputation techniques that exploit fixed system parameters (those of which give huge speedups in practice, even over the Kummer surface [7, §7.4]). Thus, all genus 2 implementations that either target signature schemes, use endomorphisms, or optimise the use of precomputation, are currently required to work in the full Jacobian group[5]; and in all of these cases, the formulas in this paper will now offer the most efficient route. The upshot is that in popular practical scenarios the most efficient genus 2 cryptography is likely to result from a hybrid combination of operations on the Kummer surface and in the full Jacobian group. We illustrate this in Section 8 by benchmarking genus 2 curves in the context of ephemeral elliptic curve Diffie-Hellman (ECDHE) with perfect forward secrecy: to exploit the best of both worlds, Alice's multiplications of the public generator P by each one of her ephemeral scalars a can make use of our new explicit formulas (and offline precomputations on P) in the full Jacobian, and her resulting ephemeral public keys $[a]P$ can then be mapped onto the corresponding Kummer surface, whose speed can be exploited by Bob in the computation of the shared secret $[b]([a]P)$.

A set of Magma [8] scripts verifying all of the explicit formulas and operation counts can be found in the full version [21], and is also publicly available at

`http://research.microsoft.com/en-us/downloads/37730278-3e37-47eb-91d1-cf889373677a/` ;

and a complete mixed-assembly-and-C implementation of all explicit formulas and scalar multiplication routines is publicly available at

`http://hhisil.yasar.edu.tr/files/hisil20140527jacobian.tar.gz` .

[4] At least one exception here, as Gaudry points out, is the hashed version of ElGamal signatures [16, §5.3]

[5] Lubicz and Robert [27] have recently broken through the "full addition restriction" on Kummer varieties, but it is not yet clear how competitive their *compatible addition* formula are in the context of raw scalar multiplications.

2 Preliminaries

For ease of exposition, we immediately restrict to the most cryptographically common case of genus 2 curves, where $\mathcal{C}$ is an imaginary hyperelliptic curve over a field K of characteristic greater than 5. (In terms of a general coverage of all genus 2 curves, we mention the interesting remaining scenarios in Section 9.) Every such curve can then be written as

$$\mathcal{C}/K : y^2 = f(x) := x^5 + a_3x^3 + a_2x^2 + a_1x + a_0, \tag{1}$$

where we note the absence of an x^4 term in $f(x)$; it can always be removed via a trivial substitution thanks to $\mathrm{char}(K) \neq 5$.

Let $J_{\mathcal{C}}$ denote the Jacobian of $\mathcal{C}$. We assume that we are working with a general point $P \in J_{\mathcal{C}}(K)$, whose Mumford representation[6]

$$\begin{aligned} P \leftrightarrow (u(x), v(x)) &= \left(x^2 + qx + r, sx + t\right) \in K[x] \times K[x] \\ &\leftrightarrow (q, r, s, t) \in \mathbb{A}^4(K) \end{aligned} \tag{2}$$

encodes two affine points $(x_1, y_1), (x_2, y_2) \in \mathcal{C}(\overline{K})$, where we assume that $x_1 \neq x_2$ so that these two points are not the same, nor are they the hyperelliptic involution of one another. The Mumford coordinates (q, r, s, t) of P are uniquely determined according to $u(x_1) = u(x_2) = 0$, $v(x_1) = y_1$ and $v(x_2) = y_2$. That is,

$$q = -(x_1 + x_2), \quad r = x_1x_2, \quad s = \frac{y_1 - y_2}{x_1 - x_2}, \quad t = \frac{x_1y_2 - y_1x_2}{x_1 - x_2}. \tag{3}$$

From (1), (2) and (3), it is readily seen that

$$v(x)^2 - f(x) = 0 \quad \text{in} \quad K[x]/\langle u(x) \rangle, \tag{4}$$

from which it follows that such general points P lie in the intersection of two hypersurfaces over K [9, §3], given as

$$\begin{aligned} \mathcal{S}_0 \ &: \quad r\left(s^2 + q^3 - (2r - a_3)q - a_2\right) = t^2 - a_0, \\ \mathcal{S}_1 \ &: \quad q\left(s^2 + q^3 - (3r - a_3)q - a_2\right) = 2st - r(r - a_3) - a_1. \end{aligned} \tag{5}$$

We note that a more simple relation is found by taking $r\mathcal{S}_1 - q\mathcal{S}_0$.

Our driving motivation for improving the explicit formulas for arithmetic in the Jacobian is the application of enhancing the fundamental operation in curve-based cryptosystems: the scalar multiplication $[k]P$ of an integer $k \in \mathbb{Z}$ by a general point P in $J_{\mathcal{C}}$. Such scalar multiplications are computed using a sequence of point doubling and addition operations, and so a common way of comparing different sets of addition formulas is to tally the number of field

[6] We adopted the notation (q, r, s, t) over (u_1, u_0, v_1, v_0) to avoid additional subscripts/superscripts when working with distinct elements in $J_{\mathcal{C}}$.

multiplications (**M**), field squarings (**S**), and field additions (denoted by **a**) that each point operation incurs. In cryptographic contexts, the input and output points are typically required to be in their unique affine form, whilst intermediate computations are carried out in projective space to avoid inversions. Thus, the most commonly reported operation counts include: `DBL`, which refers to the addition of a Jacobian point in projective form to itself; `ADD`, which refers to the addition between two distinct points in projective form; `mADD`, which refers to the *mixed* addition between a projective point and an affine point; and `mDBLADD`, which refers to the combined doubling of a projective point and subsequent addition of the result with an affine point.

As is done in [25, §5-6], in this paper we focus on deriving formulas for the most common cases of arithmetic in $J_{\mathcal{C}}$. This set of formulas is enough to perform and benchmark scalar multiplications in $J_{\mathcal{C}}$, since the possible input/output cases are extremely dense amongst all possible scenarios, i.e. for random input points P and scalars k, the cases not covered by these formulas have an exponentially small probability of being encountered in the scalar multiplication routine (see [25, §1.2] for a similar discussion). Nevertheless, the set of formulas we present are still far from a *complete* and cryptographically adequate coverage, so it is important to distinguish exactly which input/output cases they do apply to. We clarify this in Assumption 1 below, and return to this discussion in §7.3.

Assumption 1 (General points and operations in $J_{\mathcal{C}}$.) *Throughout this paper, we assume that all input and output points are "general" points in $J_{\mathcal{C}}$: we say that $P \in J_{\mathcal{C}}$ is general if the Mumford representation of P encodes two distinct affine points (x_1, y_1) and (x_2, y_2) on $\mathcal{C}$, where $x_1 \neq x_2$. Moreover, all operations in this paper are of the form $P_1 + P_2 = P_3$, where we assume that P_1, P_2 and P_3 are general points and that we are in one of two cases: (i) either $P_1 = P_2$, in which case we are computing the "doubling" $P_3 = [2]P_1$, where we further assume that neither of the two x-coordinates encoded by P_1 coincide with the two encoded by P_3, or (ii) that of the six points encoded by P_1, P_2 and P_3, no two share the same x-coordinate.*

3 Extending Jacobian coordinates to Jacobians

Let λ be a nonzero element in K. Over fields of large characteristic, Jacobian coordinates have proven to be a natural and efficient way to work projectively on elliptic curves in short Weierstrass form $\mathcal{E}/K : y^2 = x^3 + ax + b$. Indeed, in cryptographic contexts, using the triple $(\lambda^2 X : \lambda^3 Y : \lambda Z) \in \mathbb{P}(2,3,1)(K)$ to represent the affine point $(X/Z^2, Y/Z^3) \in \mathbb{A}^2(K)$ on $\mathcal{E}$ was suggested by Miller in his seminal 1985 paper [29, p. 424], and his comment that this representation "appears best" still holds true after decades of further exploration: Jacobian coordinates (and extended variants) remain the most efficient way to work on such general Weierstrass curves [4]. Moreover, the weightings $\mathrm{wt}(x) = 2$ and

$\mathrm{wt}(y) = 3$ are the orders of the poles of the functions x and y at the point at infinity on $\mathcal{E}$.

In the context of imaginary hyperelliptic curves of the form

$$\mathcal{C}/K : y^2 = x^5 + a_3x^3 + a_2x^2 + a_1x + a_0,$$

the analogous weightings are

$$\mathrm{wt}(x) = 2, \qquad \text{and} \qquad \mathrm{wt}(y) = 5, \tag{6}$$

under which the affine point $(X/Z^2, Y/Z^5) \in \mathbb{A}^2(K)$ is represented by the triple $(\lambda^2X : \lambda^5Y : \lambda Z) \in \mathbb{P}(2, 5, 1)(K)$, which lies on

$$\mathcal{C}/K : Y^2 = X^5 + a_3X^3Z^4 + a_2X^2Z^6 + a_1XZ^8 + a_0Z^{10}. \tag{7}$$

Indeed, the weights $\mathrm{wt}(x) = 2$ and $\mathrm{wt}(y) = 5$ are the orders of the poles of x and y at the (unique) point at infinity on $\mathcal{C}$. Since we perform arithmetic using the Mumford coordinates in $J_{\mathcal{C}}$, rather than the x-y coordinates on $\mathcal{C}$, we transfer the above weightings across to the Mumford coordinates via Equation (3), which yields

$$\mathrm{wt}(q) = \mathrm{wt}(x), \quad \mathrm{wt}(r) = 2 \cdot \mathrm{wt}(x), \quad \mathrm{wt}(s) = \mathrm{wt}(y) - \mathrm{wt}(x), \quad \mathrm{wt}(t) = \mathrm{wt}(y). \tag{8}$$

Combining (6) and (8) then gives

$$\mathrm{wt}(q) = 2, \qquad \mathrm{wt}(r) = 4, \qquad \mathrm{wt}(s) = 3, \qquad \mathrm{wt}(t) = 5, \tag{9}$$

which suggests the use of $(\lambda^2Q : \lambda^4R : \lambda^3S : \lambda^5T : \lambda Z) \in \mathbb{P}(2, 4, 3, 5, 1)(K)$ to represent the affine point

$$(q, r, s, t) \;=\; \left(\frac{Q}{Z^2}, \frac{R}{Z^4}, \frac{S}{Z^3}, \frac{T}{Z^5}\right) \in \mathbb{A}^4(K). \tag{10}$$

Equation (10) is at the heart of this paper. We found these weightings to be highly advantageous for group law computations: the Mumford coordinates balance naturally under this projection, and significant simplifications occur regularly in the derivation of the corresponding explicit formulas. This coordinate system is referred to as *Jacobian coordinates* in this paper. We note that, in line with Assumption 1, we will not work with the full projective closure of the affine part in $\mathbb{P}(2, 4, 3, 5, 1)(K)$, but rather with the affine patch where $Z \neq 0$.

Just as in [25, §6], we found it useful to introduce an additional coordinate (independent of Z) in the denominator of the two coordinates corresponding to the v-polynomial in the Mumford representation. So, in addition to the Jacobian coordinate Z, we include the coordinate W and use the projective six-tuple $(\lambda^2Q : \lambda^4R : \lambda^3\mu S : \lambda^5\mu T : \lambda Z : \mu W)$ to represent the affine point

$$(q, r, s, t) = \left(\frac{Q}{Z^2}, \frac{R}{Z^4}, \frac{S}{Z^3W}, \frac{T}{Z^5W}\right) \in \mathbb{A}^4(K) \tag{11}$$

for some nonzero μ in K. This coordinate system is referred to as *auxiliary Jacobian coordinates* in this paper.

Remark 1. We note the distinction between the above coordinate weightings and the weightings used by Lange, which were also said to "generalise the concept of Jacobian coordinates ... from elliptic to hyperelliptic curves" [25, §6]. In terms of the first projective coordinate Z, Lange used $(q, r, s, t) = \left(Q/Z^2, R/Z^2, S/Z^3, T/Z^3\right)$. Although these weight the u- and v-polynomials of a point with the same (Jacobian) weightings as the x- and y-coordinates on an elliptic curve, the derivation of the weightings in (10) draws a closer analogy with the use of Jacobian coordinates in genus 1. This is why we dubbed the weightings used in this work as "Jacobian coordinates".

4 Adopting the "co-Z" approach

With the aim of improving addition formulas on elliptic curves, Meloni [28] put forward a nice idea that is particularly suited to working in Jacobian coordinates. In the explicit addition of two elliptic curve points $(X_1 : Y_1 : Z_1)$ and $(X_2 : Y_2 : Z_2)$ in $\mathbb{P}(2, 3, 1)(K)$, which respectively correspond to the points $(X_1/Z_1^2, Y_1/Z_1^3)$ and $(X_2/Z_2^2, Y_2/Z_2^3)$ in $\mathbb{A}^2(K)$, Meloni observed that almost all expressions of the form $Z_1^i Z_2^j$ can completely vanish if $Z_1 = Z_2$. That is, the sum of the points $(X_1 : Y_1 : Z_1)$ and $(X_2 : Y_2 : Z_1)$ can be written as an expression of the form $(X_3 Z_1^6 : Y_3 Z_1^9 : Z_3 Z_1^3)$, which is projectively equivalent to $(X_3 : Y_3 : Z_3)$; here X_3 and Y_3 depend only on X_1, Y_1, X_2 and Y_2, so now it is only Z_3 that depends on Z_1. Since two projective points are unlikely to share the same Z-coordinate in general, the method starts by updating one or both of the input points to force this equivalence. The obvious way to do this is to respectively *cross-multiply* $(X_1 : Y_1 : Z_1)$ and $(X_2 : Y_2 : Z_2)$ into $(X_1 Z_2^2 : Y_1 Z_2^3 : Z_1 Z_2)$ and $(X_2 Z_1^2 : Y_2 Z_1^3 : Z_2 Z_1)$, but as it stands, performing this update would incur a significant overhead. The observation that is key to making this "co-Z" approach advantageous is that, in the context of scalar multiplications, these updated values (or the main subexpressions within them) are often already computed in the previous operation [28, p. 192], so this update can be performed either for free, or with a much smaller overhead.

Meloni did not apply his idea to classical "double-and-add" style addition chains, but subsequent papers [26, 19] showed how his approach could be used to enhance performance in such binary chains. In genus 2 however, successful transferral of the "co-Z" idea has not yet been achieved: the work in [24] also uses non-binary addition chains, and crucially, it was performed without access to the hyperelliptic analogue of Jacobian coordinates (those which work in stronger synergy with Meloni's idea).

Our adaptation of the "co-Z" approach requires that both the Z and W coordinates are the same, for two different input points. The first projective formulas we derive in Section 6 are for the "co-ZW" addition between the two points $P_1 = (Q_1 : R_1 : S_1 : T_1 : Z_1 : W_1)$ and $P_2 = (Q_2 : R_2 : S_2 : T_2 : Z_1 : W_1)$, and this routine is then used as a subroutine for all subsequent operations (except for standalone doublings).

5 Arithmetic in affine coordinates with new common subexpressions

The explicit formulas for arithmetic in genus 2 Jacobians are significantly more complicated than their elliptic curve counterparts, so it is especially useful to start the derivation by looking for common subexpressions and advantageous orderings in the affine versions of the formulas (i.e., before the introduction of more coordinates complicates the situation further). Our derivation follows that of [9], but it is important to point out that the resulting affine formulas have been refined by grouping new subexpressions throughout; these groupings were strategically chosen to exploit the symmetries of the q and r coordinates, and especially for the application of Jacobian coordinates that follows in Section 6.

In what follows, we give the affine formulas for general point additions and general point doublings respectively. From Section 2, recall the abbreviated notation $(q, r, s, t) \in \mathbb{A}^4(K)$ for the point in $J_{\mathcal{C}}$ with Mumford representation $(x^2 + qx + r, sx + t)$.

Let $P_1 = (q_1, r_1, s_1, t_1)$, $P_2 = (q_2, r_2, s_2, t_2)$ and $P_1 + P_2 =: P_3 = (q_3, r_3, s_3, t_3)$ be points in $J_{\mathcal{C}}$ satisfying Assumption 1. The choice of the three subexpressions

$$\begin{aligned}
A &:= (t_1 - t_2)\,(q_2\,(q_1 - q_2) - (r_1 - r_2)) - r_2\,(q_1 - q_2)\,(s_1 - s_2)\,,\\
B &:= (r_1 - r_2)\,(q_2\,(q_1 - q_2) - (r_1 - r_2)) - r_2\,(q_1 - q_2)^2\,,\\
C &:= (q_1 - q_2)\,(t_1 - t_2) - (r_1 - r_2)\,(s_1 - s_2)
\end{aligned}$$

is key to our refined derivation. The point P_3 is then given by

$$\begin{aligned}
q_3 &= (q_1 - q_2) + 2\frac{A}{C} - \frac{B^2}{C^2},\\
r_3 &= (q_1 - q_2)\,\frac{A}{C} + \frac{A^2}{C^2} + (q_1 + q_2)\,\frac{B^2}{C^2} - (s_1 + s_2)\,\frac{B}{C},\\
s_3 &= (r_1 - r_3)\,\frac{C}{B} - q_3\,(q_1 - q_3)\,\frac{C}{B} + (q_1 - q_3)\,\frac{A}{B} - s_1,\\
t_3 &= (r_1 - r_3)\,\frac{A}{B} - r_3\,(q_1 - q_3)\,\frac{C}{B} - t_1.
\end{aligned} \tag{12}$$

These formulas are used to derive the projective co-ZW addition formulas in §6.1, those which form a basis for all of the other (non-doubling) formulas in this work.

Let $P_1 = (q_1, r_1, s_1, t_1)$ and $[2]P_1 =: P_3 = (q_3, r_3, s_3, t_3)$ be points in $J_{\mathcal{C}}$ satisfying Assumption 1. Again, it is particularly useful to make use of three subexpressions:

$$\begin{aligned}
A &:= \left(\left(q_1^2 - 4r_1 + a_3\right) q_1 - a_2 + s_1^2\right) (q_1 s_1 - t_1) + \left(3q_1^2 - 2r_1 + a_3\right) r_1 s_1,\\
B &:= 2\,(q_1 s_1 - t_1)\,t_1 - 2r_1 s_1^2,\\
C &:= \left(\left(q_1^2 - 4r_1 + a_3\right) q_1 - a_2 + s_1^2\right) s_1 + \left(3q_1^2 - 2r_1 + a_3\right) t_1.
\end{aligned}$$

The point P_3 is then given by

$$\begin{aligned}
q_3 &= 2\frac{A}{C} - \frac{B^2}{C^2},\\
r_3 &= \frac{A^2}{C^2} + 2q_1\frac{B^2}{C^2} - 2s_1\frac{B}{C},\\
s_3 &= (r_1 - r_3)\frac{C}{B} - q_3(q_1 - q_3)\frac{C}{B} + (q_1 - q_3)\frac{A}{B} - s_1,\\
t_3 &= (r_1 - r_3)\frac{A}{B} - r_3(q_1 - q_3)\frac{C}{B} - t_1.
\end{aligned} \tag{13}$$

These formulas are used to derive projective doubling formulas in §6.5. The formulas in (12) and (13) agree with those of Costello-Lauter [9].

6 Projective arithmetic in extended Jacobian coordinates

In this section we derive all of the explicit formulas that are needed for the scalar multiplication routines we describe in Section 7. The formulas are summarised in Table 2 below, where we immediately note the extension of auxiliary Jacobian coordinates discussed in Section 3 to include W^2; it is advantageous to carry this additional coordinate between consecutive operations because it is often computed *en route* to the output points already, and therefore comes for free as input into the following operation. We refer to this extended version of auxiliary Jacobian coordinates as *extended Jacobian coordinates.* Table 2 reports two sets of operation counts: the "plain" count, which corresponds to our deriving sets of formulas with the aim of minimising the total number of all field operations, and the "trade-offs" count.

If W^2 is dropped from the coordinate system, and we work only with auxiliary Jacobian coordinates, $(Q : R : S : T : Z : W)$, then we note that both `DBL` and `DBLa2a3zero` would require one extra squaring (in both the "plain" and "trade-off" formulas). The only other change resulting from this abbreviated coordinate system would be in the "trade-off" version of `ADD`, where a squaring would revert back to a multiplication. All other operation counts would remain unchanged.

Following on from the discussion in Section 4, in §6.1 we start the derivations by using the affine addition formulas in (12) to develop projective formulas for `zwADD`; these are then used in the derivation of the formulas for `ADD` in §6.2, for `mADD` in §6.3, and for `mDBLADD` in §6.4. Finally, we use the affine doubling formulas in (13) to develop projective formulas for `DBL` in §6.5.

6.1 Projective co-ZW addition (`zwADD`)

Let $P_1 = (Q_1 : R_1 : S_1 : T_1 : Z_1 : W_1)$, $P_2 = (Q_2 : R_2 : S_2 : T_2 : Z_1 : W_1)$, and $P_1 + P_2 =: P_3 = (Q_3 : R_3 : S_3 : T_3 : Z_3 : W_3 : W_3^2)$ represent three points in $J_{\mathcal{C}}$ satisfying Assumption 1. We emphasize that P_1 and P_2 need not contain

operation in $J_{\mathcal{C}}$	description of operation	derived in	field operations "plain"	field operations w. "trade-offs"
zwADD	$(Q_1 : R_1 : S_1 : T_1 : Z_1 : W_1)$ $+(Q_2 : R_2 : S_2 : T_2 : Z_1 : W_1)$	§6.1	$25\mathbf{M} + 3\mathbf{S}$ $+22\mathbf{a}$	$23\mathbf{M} + 4\mathbf{S}$ $+40\mathbf{a}$
ADD	$(Q_1 : R_1 : S_1 : T_1 : Z_1 : W_1 : W_1^2)$ $+(Q_2 : R_2 : S_2 : T_2 : Z_2 : W_2 : W_2^2)$	§6.2	$41\mathbf{M} + 7\mathbf{S}$ $+22\mathbf{a}$	$35\mathbf{M} + 12\mathbf{S}$ $+56\mathbf{a}$
mADD	$(Q_1 : R_1 : S_1 : T_1 : Z_1 : W_1 : W_1^2)$ $+(Q_2 : R_2 : S_2 : T_2 : 1 : 1 : 1)$	§6.3	$32\mathbf{M} + 5\mathbf{S}$ $+22\mathbf{a}$	$29\mathbf{M} + 7\mathbf{S}$ $+44\mathbf{a}$
mDBLADD	$[2](Q_1 : R_1 : S_1 : T_1 : Z_1 : W_1 : W_1^2)$ $+(Q_2 : R_2 : S_2 : T_2 : 1 : 1 : 1)$	§6.4	$57\mathbf{M} + 8\mathbf{S}$ $+42\mathbf{a}$	$52\mathbf{M} + 11\mathbf{S}$ $+82\mathbf{a}$
DBL	$[2](Q_1 : R_1 : S_1 : T_1 : Z_1 : W_1 : W_1^2)$	§6.5	$26\mathbf{M} + 8\mathbf{S} + 2\mathbf{D}$ $+25\mathbf{a}$	$21\mathbf{M} + 12\mathbf{S} + 2\mathbf{D}$ $+52\mathbf{a}$
DBL a2a3zero	$[2](Q_1 : R_1 : S_1 : T_1 : Z_1 : W_1 : W_1^2)$ (when $a_2 a_3 = 0$)	§6.5	$25\mathbf{M} + 6\mathbf{S}$ $+22\mathbf{a}$	$21\mathbf{M} + 9\mathbf{S}$ $+48\mathbf{a}$

Table 2. A summary of the explicit formulas derived in this section for various operations in the Jacobian, $J_{\mathcal{C}}$, of an imaginary hyperelliptic curve $\mathcal{C}/K$ of genus 2, with $\mathrm{char}(K) > 5$.

W_1^2, which is why both are given in auxiliary Jacobian coordinates. However, the output P_3 is in extended Jacobian coordinates. The projective form of (12) in extended Jacobian coordinates corresponds to the following. We define the subexpressions

$$\begin{aligned}
A &:= (T_1 - T_2)\left(Q_2\left(Q_1 - Q_2\right) - (R_1 - R_2)\right) - R_2\left(Q_1 - Q_2\right)\left(S_1 - S_2\right), \\
B &:= (R_1 - R_2)\left(Q_2\left(Q_1 - Q_2\right) - (R_1 - R_2)\right) - R_2\left(Q_1 - Q_2\right)^2, \\
C &:= (Q_1 - Q_2)(T_1 - T_2) - (R_1 - R_2)(S_1 - S_2).
\end{aligned}$$

The point P_3 is then given by

$$\begin{aligned}
W_3 &= W_1[B], \\
Q_3 &= \left(Q_1[C^2] - Q_2[C^2]\right) + 2AC - W_3^2, \\
R_3 &= \left(Q_1[C^2] - Q_2[C^2] + AC\right) AC + \\
&\quad \left(Q_1[C^2] + Q_2[C^2]\right) W_3^2 - S_1[C^3 B] - S_2[C^3 B], \qquad (14) \\
S_3 &= \left(R_1[C^4] - R_3\right) + (AC - Q_3)\left(Q_1[C^2] - Q_3\right) - S_1[C^3 B], \\
T_3 &= \left(R_1[C^4] - R_3\right) AC - R_3\left(Q_1[C^2] - Q_3\right) - T_1[C^5 B], \\
Z_3 &= Z_1[C].
\end{aligned}$$

This operation, referred to as zwADD, not only computes P_3, but also produces the subexpressions $Q_1[C^2]$, $R_1[C^4]$, $S_1[C^3B]$, $T_1[C^5B]$, $Z_1[C]$, $W_1[B]$, $W_1^2[B^2]$; if desired, these can be used to update P_1 to be of the form

$$\begin{aligned}
P_1 &= (Q_1 : R_1 : S_1 : T_1 : Z_1 : W_1 : W_1^2) \\
&= (Q_1[C^2] : R_1[C^4] : S_1[C^3 B] : T_1[C^5 B] : Z_1[C] : W_1[B] : W_1^2[B^2]),
\end{aligned}$$

so that it now has the same Z, W, and W^2 coordinates as P_3. The combination of the `zwADD` operation and this update will be denoted using the syntax

$$(P_3, P_1') := P_1 + P_2 \,,$$

where P_1' is the updated (but projectively equivalent) version of P_1.

6.2 Projective addition (`ADD`)

Rather than producing lengthy formulas for additions, we use a simple construction that exploits `zwADD`. Let $P_1 = (Q_1 : R_1 : S_1 : T_1 : Z_1 : W_1 : W_1^2)$, $P_2 = (Q_2 : R_2 : S_2 : T_2 : Z_2 : W_2 : W_2^2)$, and $P_1 + P_2 =: P_3 = (Q_3 : R_3 : S_3 : T_3 : Z_3 : W_3 : W_3^2)$ represent three points in $J_{\mathcal{C}}$ satisfying Assumption 1. We can then cross-multiply to define the points in auxiliary Jacobian coordinates

$$\begin{aligned} P_1' &:= \left(Q_1\left[Z_2^2\right] : R_1\left[Z_2^4\right] : S_1\left[Z_2^3W_2\right] : T_1\left[Z_2^5W_2\right] : Z_1[Z_2] : W_1[W_2]\right), \\ P_2' &:= \left(Q_2\left[Z_1^2\right] : R_2\left[Z_1^4\right] : S_2\left[Z_1^3W_1\right] : T_2\left[Z_1^5W_1\right] : Z_2[Z_1] : W_2[W_1]\right). \end{aligned}$$

Observe that $P_1' = P_1$ and $P_2' = P_2$, but that P_1' and P_2' now share the same Z and W coordinates. This means that we can use the `zwADD` operation defined in §6.1 to compute $P_3 = P_1 + P_2$ as $(P_3, P_1'') := P_1' + P_2'$. Observe that $P_1'' = P_1$, and that P_1'' will share the same Z, W, and W^2 coordinates as P_3. We note that this update of P_1 into P_1'' can be useful in the generation of lookup tables [26], but is generally not useful during the main loop.

6.3 Projective mixed addition (`mADD`)

In a similar way, let $P_1 = (Q_1 : R_1 : S_1 : T_1 : Z_1 : W_1 : W_1^2)$, $P_2 = (Q_2 : R_2 : S_2 : T_2 : 1 : 1 : 1)$, and $P_1 + P_2 =: P_3 = (Q_3 : R_3 : S_3 : T_3 : Z_3 : W_3 : W_3^2)$ represent three points in $J_{\mathcal{C}}$ satisfying Assumption 1. This time we only need to update P_2 into P_2', which is performed in auxiliary Jacobian coordinates as

$$P_2' := \left(Q_2\left[Z_1^2\right] : R_2\left[Z_1^4\right] : S_2\left[Z_1^3W_1\right] : T_2\left[Z_1^5W_1\right] : [Z_1] : [W_1]\right),$$

where we observe that P_1 and P_2' now have the same Z and W coordinates. Subsequently, using the `zwADD` operation from §6.1 allows $P_3 = P_1 + P_2$ to be computed by $(P_3, P_1') := P_1 + P_2'$.

6.4 Projective mixed doubling-and-addition (`mDBLADD`)

Let $P_1 = (Q_1 : R_1 : S_1 : T_1 : Z_1 : W_1 : W_1^2)$, $P_2 = (Q_2 : R_2 : S_2 : T_2 : 1 : 1 : 1)$, and $[2]P_1 + P_2 =: P_3 = (Q_3 : R_3 : S_3 : T_3 : Z_3 : W_3 : W_3^2)$, represent three points in $J_{\mathcal{C}}$ satisfying Assumption 1. To compute $[2]P_1 + P_2$, we schedule the higher level operations in the form $(P_1 + P_2) + P_1$ (see [11] and [26] for the same high level scheduling). This means that `mDBLADD` can be computed using an `mADD` operation before a `zwADD` operation. (Subsequently, we must also assume that P_1, the intermediate point $P_1 + P_2$, and the output point $[2]P_1 + P_2 =: P_3 = (Q_3 : R_3 : S_3 : T_3 : Z_3 : W_3 : W_3^2)$ represent three points in $J_{\mathcal{C}}$ satisfying Assumption 1.)

6.5 Projective doubling (DBL)

Let $P_1 = (Q_1 : R_1 : S_1 : T_1 : Z_1 : W_1 : W_1^2)$ and $[2]P_1 =: P_3 = (Q_3 : R_3 : S_3 : T_3 : Z_3 : W_3 : W_3^2)$ represent two points in $J_{\mathcal{C}}$ satisfying Assumption 1. The projective form of (13) in extended Jacobian coordinates corresponds to the following. We define the subexpressions

$$\begin{aligned}
A &:= \big(\big(Q_1\left(Q_1^2-4R_1\right)+\left(Q_1-\left(a_2/a_3\right)Z_1^2\right)a_3Z_1^4\big)W_1^2+S_1^2\big)\left(Q_1S_1-T_1\right)+\\
&\quad\;\big(3Q_1^2-2R_1+a_3Z_1^4\big)W_1^2R_1S_1,\\
B &:= 2\left(Q_1S_1-T_1\right)T_1-2R_1S_1^2,\\
C &:= \big(\big(Q_1\left(Q_1^2-4R_1\right)+\left(Q_1-\left(a_2/a_3\right)Z_1^2\right)a_3Z_1^4\big)W_1^2+S_1^2\big)S_1+\\
&\quad\;\big(3Q_1^2-2R_1+a_3Z_1^4\big)W_1^2T_1.
\end{aligned}$$

We can then write P_3 as

$$\begin{aligned}
W_3 &= W_1[B],\\
Q_3 &= 2AC-W_3^2,\\
R_3 &= (AC)^2+2Q_1\left[C^2\right]W_3^2-2S_1\left[C^3B\right],\\
S_3 &= \left(R_1\left[C^4\right]-R_3\right)+\left(AC-Q_3\right)\left(Q_1\left[C^2\right]-Q_3\right)-S_1\left[C^3B\right],\\
T_3 &= \left(R_1\left[C^4\right]-R_3\right)AC-R_3\left(Q_1\left[C^2\right]-Q_3\right)-T_1\left[C^5B\right],\\
Z_3 &= Z_1[C].
\end{aligned} \tag{15}$$

The DBL operation not only computes P_3, but also produces the subexpressions $Q_1\left[C^2\right]$, $R_1\left[C^4\right]$, $S_1\left[C^3B\right]$, $T_1\left[C^5B\right]$, $Z_1[C]$, $W_1[B]$, $W_1^2\left[B^2\right]$; if desired, these can be used to update P_1 into

$$\begin{aligned}
P_1 &= (Q_1 : R_1 : S_1 : T_1 : Z_1 : W_1 : W_1^2)\\
&= \left(Q_1\left[C^2\right] : R_1\left[C^4\right] : S_1\left[C^3B\right] : T_1\left[C^5B\right] : Z_1[C] : W_1[B] : W_1^2\left[B^2\right]\right),
\end{aligned}$$

in order to share the same Z, W, and W^2 coordinates with P_3. We define the operation DBLa2a3zero to be a doubling in the special case that the curve constants a_2 and a_3 are zero.

7 Implementation

We chose two different curves to showcase the explicit formulas derived in the previous section, both of which target the 128-bit security level.

The first curve was found in the colossal point counting effort undertaken by Gaudry and Schost [18]. From a security standpoint, it is both twist-secure and it is not considered to be special (e.g. it has a large discriminant); from a performance standpoint, it was chosen over the arithmetically advantageous field $\mathbb{F}_p$ with $p = 2^{127} - 1$, and with optimal cofactors such that the curve supports a Gaudry-style Kummer surface implementation [16]. This is the same Kummer surface that was used to set speed records in [7] and [3]. We chose the Jacobian of this curve to illustrate the performance that is gained when using our new formulas inside a general "double-and-add" scalar multiplication routine.

The second curve supports a 4-dimensional Gallant-Lambert-Vanstone (GLV) decomposition [15]. Over prime fields, requiring 4-dimensional GLV imposes that the Jacobian has complex multiplication (CM) by a special field – in this case it is $\mathbb{Q}(\zeta_5)$. This (specialness) means that we cannot hope to find a twist-secure curve over a particular prime, but rather that we must search over many primes. In the same vein as [7, §8.3], we also wanted this curve to support a rational Gaudry-style Kummer surface. This curve is defined over the prime field $p = 2^{128} - c$ with $c = 7689975$, which is the *smallest* $c > 0$ such that a curve with CM by $\mathbb{Q}(\zeta_5)$ over $\mathbb{F}_p$ is twist-secure with optimal cofactors[7]. This curve was chosen to exhibit the performance that is gained when using our new formulas inside a GLV-style multiexponentation; in particular, each step of the multiexponentation requires only an `mDBLADD` operation, and this is where our explicit formulas offer the largest relative speedup over the previous ones.

7.1 Working on the Gaudry-Schost Jacobian

Let $p = 2^{127} - 1$, and define the following constants in $\mathbb{F}_p$: $a := 11$, $b := -22$, $c := -19$, $d := -3$, $e := 1 + \sqrt{-833/363}$ and $f := 1 - \sqrt{-833/363}$. For the Rosenhain invariants $\lambda = \frac{ac}{bd}$, $\mu = \frac{ce}{df}$, $\nu = \frac{ae}{bf}$, the curve

$$\mathcal{C}_{Ros}/\mathbb{F}_p \;:\; y^2 = x(x-1)(x-\lambda)(x-\mu)(x-\nu)$$

is such that $\#J_{\mathcal{C}_{Ros}} = 2^4 \cdot r$ and $\#J_{\mathcal{C}'_{Ros}} = 2^4 \cdot r'$, where r and r' are 250- and 251-bit primes respectively [18], and where $\mathcal{C}'_{Ros}$ is the quadratic twist of $\mathcal{C}_{Ros}$. The coefficient of x^4 in $\mathcal{C}_{Ros}$ is $\alpha = -(1 + \lambda + \mu + \nu)$, and we choose to zero it under the transformation $\varphi \;:\; \mathcal{C}_{Ros} \to \tilde{\mathcal{C}}, (x, y) \mapsto (x - \alpha/5, y)$. The resulting curve, $\tilde{\mathcal{C}}$, has a coefficient of x^3 which is a fourth power in $\mathbb{F}_p$; let it be u^{-4}, where we chose $u = 19859741192276546142105456991319328298$. We can then use the map $\psi \;:\; \tilde{\mathcal{C}} \to \mathcal{C}, (x, y) \mapsto (x \cdot u^2, y \cdot u^5)$ to work with the isomorphic curve $\mathcal{C}/\mathbb{F}_p \;:\; y^2 = x^5 + x^3 + a_2x^2 + a_1x + a_0$, where the coefficient of x^3 being 1 saves a multiplication inside every point doubling[8]. We use the name `Jac1271` for the Jacobian $J_{\mathcal{C}}$, and use the name `Kum1271` for the associated Kummer surface $\mathcal{K}$ – this is defined by the above constants a, b, c, d (see [16]).

In Section 8 we report two new sets of implementation numbers on `Jac1271`. First, we benchmark a generic scalar multiplication, using both the old and the new formulas, to illustrate the performance boost given by this work in the

[7] It is relatively straightforward to show that if $J_{\mathcal{C}}$ has CM by $\mathbb{Q}(\zeta_5)$ and full rational two-torsion, then either $J_{\mathcal{C}}$ or $J_{\mathcal{C}'}$ must contain a point of order 5; thus, the optimal cofactors are 16 and 80.

[8] If the coefficient of x^3 in $\tilde{\mathcal{C}}$ was not a fourth power, one could still use this form of transformation to achieve another "small" coefficient, or in this case, work on the twist instead.

general case. In addition, we benchmark a *fixed-base* scalar multiplication, which uses the new formulas and takes advantage of precomputations on a public generator to give large speedups on `Jac1271`. In the context of ECDHE, this second benchmark corresponds to the "`key_gen`" phase, which compliments the performance numbers for the "`shared_secret`" scalar multiplications on `Kum1271` in [7] and [3]. (We discuss some caveats related to this Jacobian/Kummer combination in §7.3.) To tie these two sets of performance numbers together, we also benchmark the numbers for computing the map from `Jac1271` onto `Kum1271`, which was made explicit in the AVIsogenies library [6], and for general points in $J_{\mathcal{C}_{Ros}}$ is given as

$$\Psi \ : \ J_{\mathcal{C}_{Ros}} \to \mathcal{K}, \qquad (x^2+qx+r, sx+t) \mapsto (X : Y : Z : T),$$

where

$$\begin{aligned} X &= a\left(r(\mu - r)(\lambda+q+\nu) - t^2\right), \quad & Y &= b\left(r(\nu\lambda - r)(1+q+\mu) - t^2\right), \\ Z &= c\left(r(\nu - r)(\lambda+q+\mu) - t^2\right), \quad & T &= d\left(r(\mu\lambda - r)(1+q+\nu) - t^2\right). \end{aligned} \tag{16}$$

For practical scenarios like ECDHE, it is fortunate that we only need the map in this direction, as the pullback map from $\mathcal{K}$ to $J_{\mathcal{C}_{Ros}}$ is much more complicated [16, §4.3]. Since we compute in $J_{\mathcal{C}}$ (rather than $J_{\mathcal{C}_{Ros}}$), we actually need to compute the composition of Ψ with $(\psi\varphi)^{-1}$, which when extended to general points in $J_{\mathcal{C}}$ is

$$(\psi\varphi)^{-1} \ : \ J_{\mathcal{C}} \to J_{\mathcal{C}_{Ros}} \quad , \quad (x^2+qx+r, sx+t) \mapsto (x^2+q'x+r', s'x+t'),$$

with $q' = u^{-2}q + 2\alpha/5$, $r' = u^{-4}r + \alpha/5q' - (\alpha/5)^2$, $s' = u^{-3}s$, $t' = u^{-5}t + \alpha/5s'$. Assuming that the input point in $J_{\mathcal{C}}$ is in extended Jacobian coordinates, the operation count for the full map $\Psi' = \Psi(\psi\varphi)^{-1}$ from $J_{\mathcal{C}}$ to $\mathcal{K}$ is $1\mathbf{I} + 31\mathbf{M} + 2\mathbf{S} + 19\mathbf{a}$; we benchmark it alongside the scalar multiplications in Section 8.

To draw a fair comparison against prior works, we inserted our formulas into the software made publicly available by Bos *et al.* [7], which itself employed the previous best formulas. (We tweaked both sets of formulas for `Jac1271` to take advantage of the constant $a_3 = 1$.) This software computes the scalar multiplications on `Jac1271` using an adaptation of the left-to-right signed sliding window recoding from [1] with a window size of $w = 5$, where the lookup table consists of 8 points and is constructed using the same approach as in [26, §4]. The timings are presented in Section 8.

7.2 Working on the Jacobian of a GLV curve

Let $p = 2^{128} - 7689975$ and define $\mathcal{C}/\mathbb{F}_p \ : \ y^2 = x^5 + 7^{10}$. The Jacobian groups $J_{\mathcal{C}}$ and $J_{\mathcal{C}'}$ have cardinalities $\#J_{\mathcal{C}} = 2^4 \cdot 5 \cdot r$ and $\#J_{\mathcal{C}'} = 2^4 \cdot r'$, where

$$\begin{aligned} r &= (2^{252} + 3755769283312336917821467926777982672135841316517644041 59)/5, \\ r' &= 2^{252} - 37557692833188788247584622603853339708921867977722348 2485 \end{aligned}$$

are both prime.

The implementation of a 4-dimensional GLV scalar multiplication in $J_{\mathcal{C}}$ follows that which is described in [7, §6]; again, we wrapped their GLV software around both their old and our new formulas for a fair comparison – we note that both instances were made to use the above curve, which we refer to as `GLV128c`.

Practically speaking, it does not make as much sense to benchmark `GLV128c` in the same ECDHE style as we discussed for `Jac1271` and `Kum1271`. If there is enough storage to exploit a long-term public generator P, then the presence of endomorphisms is essentially redundant in the `key_gen` phase, since multiples of P can then be precomputed offline without using an endomorphism. On the `shared_secret` side, where *variable-base* scalar multiplications are performed on fresh inputs, our implementations show that a 4-dimensional decomposition on `GLV128c` is still slightly slower than a Kummer surface scalar multiplication, so in the case of ECDHE, it is likely to be faster on both sides to stick with the combination of `Jac1271` and `Kum1271`. Nevertheless, there could be scenarios where it makes sense to use the endomorphism on `GLV128c` (e.g. for a signature verification), and still make use of the maps between the full Jacobian group and the associated Kummer surface. In this case, the map in (16) and the pullback map in [16, §4.3] can be exploited analogously to the case of `Jac1271`, keeping in mind that the maps would pass through the Jacobian of the Rosenhain form of $\mathcal{C}$.

Timings for a 4-dimensional GLV variable-base scalar multiplication on `GLV128c` using both the old and the new explicit formulas are given in Section 8.

We note that in all scalar multiplication routines, i.e. in both fixed- and variable-base scalar multiplications on `Jac1271` and in 4-dimensional multiexponentiations on `GLV128c`, we always found it advantageous to convert the lookup table elements from extended Jacobian coordinates to affine coordinates using Montgomery's simultaneous inversion method [30]. This "decision" is generally made easier in genus 2, where the difference between mixed additions and full additions is greater, and the relative cost of a field inversion (compared to the rest of the scalar multiplication routine) is much less than it is in the elliptic curve case. Finally, we note that the single conversion of the output point from Jacobian to affine coordinates comes at a cost of $1\mathbf{I} + 10\mathbf{M} + 1\mathbf{S}$.

7.3 A disclaimer: the difficulties facing constant-time, exception-free scalar multiplications in $J_{\mathcal{C}}$

We must point out that none of the scalar multiplications on `Jac1271` or `GLV128c` that we report in this paper run in constant time, and that the difficulties of achieving such a routine in genus 2 Jacobians is closely related to Assumption 1. We note that these are not the same implementation-level difficulties pointed out in [3, §1.2]; indeed, while the Kummer surface implementations reported in [3] and [7] run in constant time, a truly constant-time genus 2 implementation that does not use the Kummer surface is yet to be documented in the literature.

More specifically, there are scalar recoding algorithms (cf. [20, 12]) that make it possible to implement the `Jac1271` or `GLV128c` routines such that scalar multiplications on random inputs will run in constant time with probability exponentially close to 1. However, in order to guard against active adversaries and to be considered *truly* constant-time, the routines should be guaranteed to execute identically and run correctly for *all* combinations of integer scalars and input points; this means the explicit formulas must be able to handle input combinations in J_C that are not "general" in the sense of Assumption 1. Although explicit formulas can be developed for each of these special cases, their culmination into an efficient and truly constant-time scalar multiplication algorithm remains an important open problem.

8 Results

In this section we present the timings of the routines described in the previous section. All of the benchmarks were performed on an Intel Core i7-3770M (Ivy Bridge) processor at 3.4 GHz with hyperthreading turned off and over-clocking ("turbo-boost") disabled, and all-but-one of the cores switched off in BIOS. The implementations were compiled with gcc 4.6.3 with the `-O2` flag set and tested on a 64-bit Linux environment. Cycles were obtained using the SUPERCOP [5] toolkit and then rounded to the nearest 1,000 cycles.

The primary purpose of our benchmarks is to compare the performance of scalar multiplications in genus 2 Jacobians using both the old and new sets of explicit formulas. Table 3 reports that a generic scalar multiplication on `Jac1271` using the explicit formulas in this paper gives a factor 1.25x improvement over one that uses the previous best formulas; this is the approximate speedup that one can expect when adopting extended Jacobian coordinates on any imaginary hyperelliptic curve of genus 2 over a large prime field. Table 4 reports that a 4-dimensional GLV multiexponentiation routine using the explicit formulas in this paper gives a factor 1.29x improvement over the same routine that calls the previous explicit formulas. We note that the benchmarked implementations of the new formulas always used the "plain" versions (see Table 2), since these proved to be more efficient than the "trade-off" versions in our implementations.

curve	coordinates	formulas from	cycles
`Jac1271`	homogeneous	[9, 7]	243,000
`Jac1271`	ext. Jacobian	this work	195,000

Table 3. Benchmarking the old and new explicit formulas in the context of a generic scalar multiplication on `Jac1271`.

As a secondary set of benchmarks, in Table 5 we give summary performance numbers for the Gaudry-Schost curve in §7.1 in the context of ECDHE. Using

curve	coordinates	formulas from	cycles
GLV128c	homogeneous	[9, 7]	166,000
GLV128c	ext. Jacobian	this work	129,000

Table 4. Benchmarking the old and new explicit formulas in the context of a 4-GLV scalar multiplication on GLV128c.

extended Jacobian coordinates and precomputing a lookup table of size 256KB, each key_gen operation takes around 40,000 cycles in total. (Note that this cycle count excludes the cycles required to transfer the lookup table from main memory to the cache.) Together with the recent Kummer surface performance numbers of Bernstein *et al.* [3], this gives an idea of the performance that is possible when space permits a significant precomputation in genus 2 ECDHE. Note, however, that until an efficient remedy to the issues discussed in §7.3 is known, this style of key_gen in genus 2 is unprotected against side-channel attacks. We also benchmarked a fixed-base scalar multiplication with a much smaller 1KB lookup table, but it ran in 87,000, which when combined with the Ψ' map, is not faster than the scalar multiplication on Kum1271 from [3].

ECDHE operation	details	curve	implementation	cycles
key_gen	fixed-base scalar mul	Jac1271	this work	36,000
	Ψ' map	-	this work (and [6])	4,000
shared_secret	variable-base scalar mul	Kum1271	Bernstein *et al.* [3]	91,000

Table 5. The performance of genus 2 in ECDHE on the Gaudry-Schost curve [18].

We reiterate that, to get the performance numbers in Tables 3 and 4, and those for key_gen in Table 5, we modified the software made publicly available by Bos *et al.* [7] to be able to call both sets of explicit formulas. This software already included routines for general scalar multiplications, 4-GLV scalar multiplications, and the fixed-base scenario. To complete the benchmarks in Table 5, we ran the publicly available software from [3] on our hardware.

9 Related scenarios

We conclude by mentioning some related cases of interest, for which the analogue of (extended) Jacobian coordinates and/or the co-Z idea could also be applied.

The takeaway message of this section is that, while we focussed on the most common instance of genus 2 curves, the ideas in this work have the potential to boost the speed of arithmetic in other scenarios too.

- **Real hyperelliptic curves.** In Section 2 we immediately specialised to the *imaginary* case, where $\mathcal{C}/K$ is hyperelliptic of degree 5 with one point at infinity. The other case in genus 2, where the curve is of degree 6 and has two points at infinity [13], has received less attention in papers pursuing high performance, since it is slightly slower than the imaginary case [10]. Moreover, it is often the case (at least among the scenarios of practical interest) that a degree 6 model contains a rational Weierstrass point and can therefore be transformed to a degree 5 model (e.g. the family in [17, §4.4]). On the other hand, there are some scenarios where this transformation is not always possible, so it is of interest to see how efficient projective arithmetic can be made in the real case, and whether analogues of the ideas in this work can be carried across successfully.
- **Pairings.** Genus 2 pairings are also likely to benefit from Jacobian coordinates. Roughly speaking, the explicit formulas in this paper inherently compute the additional components (i.e. the Miller functions) that are required in a pairing computation. However, the resulting savings would not be as drastic, as the operations in $J_{\mathcal{C}}$ are dominated by extension field operations in a pairing computation. In addition, genus 2 has not been as competitive in the realm of pairings as it has as a standard discrete logarithm primitive, largely because the construction of competitive ordinary, pairing-friendly hyperelliptic curves has been very limited. On the other hand, there are attractive constructions of supersingular genus 2 curves [14], which may be of interest in the "Type 1" setting, especially given that the fastest instantiations of such pairings are (in recent times) considered broken [2]. Interestingly, the construction in [14, §7] is one example of a scenario where the real model cannot be converted into an imaginary one in general.
- **Low characteristic / higher genus.** The specialisation of Jacobian coordinates to low characteristic genus 2 curves and the extension to higher genus imaginary hyperelliptic curves follows analogously. However, the motivation in both directions is nowadays stunted by their respective security concerns. Nevertheless, it could be worthwhile to see how much faster the arithmetic in these cases can become when using Jacobian coordinates.
- **The RM families.** We benchmarked the new explicit formulas in two scenarios; on a non-special "generic" curve, and on a curve with very special CM that subsequently comes equipped with an endomorphism. A third option comes from the families with explicit RM in [17], which perhaps achieves the best of both worlds in genus 2: they also come equipped with an endomorphism, but are much more general than the CM curve we used. This generality dispels any security concerns associated with special curves, and moreover allows them to be found over a fixed prime field. Thus, at the 128-bit security level, one could find such a curve over $p = 2^{127} - 1$ that

facilitates both 2-dimensional GLV decomposition on its Jacobian and which supports a (twist-secure) Kummer surface. It would then be interesting to benchmark the new explicit formulas on one of these families, where the GLV routine would again make a higher relative frequency of calls to the fast `mDBLADD` routine.

Acknowledgements. We thank Joppe Bos, Michael Naehrig, Benjamin Smith, and Osmanbey Uzunkol for their useful comments on an early draft of this work. We also thank the anonymous referees for their valuable comments.

References

1. R. M. Avanzi. A note on the signed sliding window integer recoding and a left-to-right analogue. In H. Handschuh and M. A. Hasan, editors, *Selected Areas in Cryptography*, volume 3357 of *Lecture Notes in Computer Science*, pages 130–143. Springer, 2004.
2. R. Barbulescu, P. Gaudry, A. Joux, and E. Thomé. A heuristic quasi-polynomial algorithm for discrete logarithm in finite fields of small characteristic. In P. Q. Nguyen and E. Oswald, editors, *EUROCRYPT*, volume 8441 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2014.
3. D. J. Bernstein, C. Chuengsatiansup, T. Lange, and P. Schwabe. Kummer strikes back: new DH speed records. *IACR Cryptology ePrint Archive*, 2014:134, 2014.
4. D. J. Bernstein and T. Lange. Explicit-formulas database, accessed 2 Jan, 2014. http://www.hyperelliptic.org/EFD/.
5. D. J. Bernstein and T. Lange. eBACS: ECRYPT Benchmarking of Cryptographic Systems, accessed 28 September, 2013. http://bench.cr.yp.to.
6. G. Bisson, R. Cosset, and D. Robert. AVIsogenies – a library for computing isogenies between abelian varieties, November 2012. URL: http://avisogenies.gforge.inria.fr.
7. J. W. Bos, C. Costello, H. Hisil, and K. Lauter. Fast cryptography in genus 2. In T. Johansson and P. Q. Nguyen, editors, *EUROCRYPT*, volume 7881 of *Lecture Notes in Computer Science*, pages 194–210. Springer, 2013, full version available at: http://eprint.iacr.org/2012/670.
8. W. Bosma, J. Cannon, and C. Playoust. The Magma algebra system. I. The user language. *J. Symbolic Comput.*, 24(3-4):235–265, 1997. Computational algebra and number theory (London, 1993).
9. C. Costello and K. Lauter. Group law computations on Jacobians of hyperelliptic curves. In A. Miri and S. Vaudenay, editors, *Selected Areas in Cryptography*, volume 7118 of *Lecture Notes in Computer Science*, pages 92–117. Springer, 2011.
10. S. Erickson, T. Ho, and S. Zemedkun. Explicit projective formulas for real hyperelliptic curves of genus 2. *Advances for Mathematics of Communications*, To appear, 2014.
11. X. Fan and G. Gong. Efficient explicit formulae for genus 2 hyperelliptic curves over prime fields and their implementations. In C. Adams, A. Miri, and M. Wiener, editors, *Selected Areas in Cryptography*, volume 4876 of *Lecture Notes in Computer Science*, pages 155–172. Springer Berlin Heidelberg, 2007.

12. A. Faz-Hernández, P. Longa, and A. H. Sanchez. Efficient and secure algorithms for GLV-based scalar multiplication and their implementation on GLV-GLS curves. In J. Benaloh, editor, *CT-RSA*, volume 8366 of *Lecture Notes in Computer Science*, pages 1–27. Springer, 2014.
13. S. D. Galbraith, M. Harrison, and D. J. Mireles Morales. Efficient hyperelliptic arithmetic using balanced representation for divisors. In A. J. van der Poorten and A. Stein, editors, *ANTS*, volume 5011 of *Lecture Notes in Computer Science*, pages 342–356. Springer, 2008.
14. S. D. Galbraith, J. Pujolàs, C. Ritzenthaler, and B. A. Smith. Distortion maps for supersingular genus two curves. *J. Mathematical Cryptology*, 3(1):1–18, 2009.
15. R. P. Gallant, R. J. Lambert, and S. A. Vanstone. Faster point multiplication on elliptic curves with efficient endomorphisms. In J. Kilian, editor, *CRYPTO*, volume 2139 of *Lecture Notes in Computer Science*, pages 190–200. Springer, 2001.
16. P. Gaudry. Fast genus 2 arithmetic based on Theta functions. *Journal of Mathematical Cryptology JMC*, 1(3):243–265, 2007.
17. P. Gaudry, D. R. Kohel, and B. A. Smith. Counting points on genus 2 curves with real multiplication. In D. H. Lee and X. Wang, editors, *ASIACRYPT*, volume 7073 of *Lecture Notes in Computer Science*, pages 504–519. Springer, 2011.
18. P. Gaudry and E. Schost. Genus 2 point counting over prime fields. *J. Symb. Comput.*, 47(4):368–400, 2012.
19. R. R. Goundar, M. Joye, A. Miyaji, M. Rivain, and A. Venelli. Scalar multiplication on Weierstraß elliptic curves from Co-Z arithmetic. *J. Cryptographic Engineering*, 1(2):161–176, 2011.
20. M. Hamburg. Fast and compact elliptic-curve cryptography. Cryptology ePrint Archive, Report 2012/309, 2012. http://eprint.iacr.org/.
21. H. Hisil and C. Costello. Jacobian coordinates on genus 2 curves. *IACR Cryptology ePrint Archive*, 2014:385, 2014.
22. N. Koblitz. Elliptic curve cryptosystems. *Mathematics of computation*, 48(177):203–209, 1987.
23. N. Koblitz. Hyperelliptic cryptosystems. *Journal of cryptology*, 1(3):139–150, 1989.
24. V. Kovtun and S. Kavun. Co-Z divisor addition formulae in Jacobian of genus 2 hyperelliptic curves over prime fields. Cryptology ePrint Archive, Report 2010/498, 2010. http://eprint.iacr.org/.
25. T. Lange. Formulae for arithmetic on genus 2 hyperelliptic curves. *Appl. Algebra Eng. Commun. Comput.*, 15(5):295–328, 2005.
26. P. Longa and A. Miri. New composite operations and precomputation scheme for elliptic curve cryptosystems over prime fields. In R. Cramer, editor, *Public Key Cryptography PKC 2008*, volume 4939 of *Lecture Notes in Computer Science*, pages 229–247. Springer Berlin Heidelberg, 2008.
27. D. Lubicz and D. Robert. A generalisation of Miller's algorithm and applications to pairing computations on abelian varieties. Cryptology ePrint Archive, Report 2013/192, 2013. http://eprint.iacr.org/.
28. N. Meloni. New point addition formulae for ECC applications. In C. Carlet and B. Sunar, editors, *WAIFI*, volume 4547 of *Lecture Notes in Computer Science*, pages 189–201. Springer, 2007.
29. 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.
30. P. L. Montgomery. Speeding the Pollard and elliptic curve methods of factorization. *Mathematics of computation*, 48(177):243–264, 1987.