Jacobian Versus Infrastructure in Split Hyperelliptic Curves

Monireh Rezai Rad[1], Michael J. Jacobson Jr.[2(✉)], and Renate Scheidler[1]

[1] Department of Mathematics and Statistics, University of Calgary, 2500 University Drive NW, Calgary, AB T2N 1N4, Canada
{mrezaira,rscheidl}@ucalgary.ca

[2] Department of Computer Science, University of Calgary, 2500 University Drive NW, Calgary, AB T2N 1N4, Canada
jacobs@ucalgary.ca

Abstract. Split (also known as real) hyperelliptic curves admit two main algebraic structures: the Jacobian and the infrastructure. In this paper, we describe exactly how the infrastructure and the Jacobian are related. We show that computations in the infrastructure using a new modified notion of distance and computations in a particular subgroup of the Jacobian heuristically have exactly the same cost for curves defined over sufficiently large finite fields. We also present a novel set of explicit formulas for genus three split hyperelliptic curves that improves on the current state-of-the-art.

Keywords: Split hyperelliptic curve · Jacobian · Balanced divisor · Infrastructure · Explicit formulas

1 Introduction

The Jacobian of a hyperelliptic curve defined over a finite field is a finite abelian group at the center of a number of important open questions in algebraic geometry and number theory. Sutherland [23] surveyed some of these, including the computation of the associated L-functions and zeta functions used in his investigation of Sato-Tate distributions [15]. Many of these problems lend themselves to numerical investigation, and as emphasized by Sutherland, fast arithmetic in the Jacobian is crucial for their efficiency.

Hyperelliptic curves are categorized as ramified and split models according to their number of points at infinity. Ramified curves have one point at infinity, whereas split curves have two. There is also a third model with no infinite points called inert or unusual, but these are usually avoided in practice as they have cumbersome Jacobian arithmetic and can be transformed to a split model over a quadratic extension of the base field. Jacobian arithmetic differs on ramified and split models. The split scenario is more complicated and is most efficiently

M. J. Jacobson Jr. and R. Scheidler—Supported in part by NSERC of Canada.

realized via a divisor arithmetic framework referred to as *balanced*. As a result, optimizing divisor arithmetic on split hyperelliptic curves has received less attention from the research community. However, split models have many interesting properties; most importantly, they exist for a much larger array of hyperelliptic curves compared to ramified models. Thus, exhaustive computations such as those in [15] conduct the bulk of their work on split models by necessity.

Split hyperelliptic curves support another algebraic structure, called the infrastructure, that can also be used for numerical investigations of the Jacobian. For example, Stein and Teske [22] used arithmetic in the infrastructure for computing the order of the Jacobian by first computing the regulator, i.e. the (usually very large) order of the class of the divisor D given by the difference of the two points at infinity. One attractive aspect of the infrastructure is that it supports an especially fast operation, called a *baby step*, that can be exploited to speed up various applications, including the Jacobian order computation algorithms used in [22] and [15].

Although both structures in split models can be employed for computational problems, it was originally not clear how they are connected or which one provides better performance in practice. Mireles Morales [17] related the infrastructure to the cyclic subgroup G of the Jacobian generated by the class of D. He showed that any computation in one structure can be reduced to an analogous computation in the other. Moreover, he asserted that performing computations in the Jacobian using balanced divisor arithmetic should always be more efficient than infrastructure arithmetic. However, he did not verify his claim via proof or implementation and also did not take into account the improved infrastructure arithmetic from [12] in his analysis. Therefore, the question of which setting provides better performance has to date not been decisively settled.

This paper offers two main contributions. First, we provide a definitive answer to the aforementioned performance question through rigorous methodology. We describe exactly how the infrastructure and the Jacobian are related. We investigate the assertion by Mireles Morales, considering state-of-the-art algorithms for arithmetic in both settings, including the results of [12]. An initial numerical investigation [11] suggested that the Jacobian offers better performance than the infrastructure, raising the question of whether this deficiency was an inherent property of the infrastructure or could be overcome through arithmetic improvements. We confirm the latter answer by introducing an alternative notion of distance on the infrastructure. Using this new distance, we prove that computations in the infrastructure and in G have exactly the same cost under the assumption of some reasonable, widely accepted heuristics.

Our second contribution is a suite of new explicit formulas for arithmetic in both the Jacobian and the infrastructure of a split genus 3 hyperelliptic curve. As pointed out earlier, these formulas represent a highly useful tool for computations involving such curves. Specifically, conforming to best practices, the operations in our formulas are described completely in terms of finite field operations as opposed to polynomial arithmetic. We present two sets of formulas, one for curves defined over finite fields of odd characteristic, and a second set for characteristic 2. Our formulas for odd characteristic offer improvements over those of [23] for baby steps and addition, but not doubling. For characteristic 2, our explicit formulas are the first to appear in the literature.

Following [23], our formulas use an affine model where each operation requires one field inversion. Much of the literature on efficient Jacobian arithmetic for low genus hyperelliptic curves assumes applications to cryptography, for which inversion-free (projective) formulas are preferred because inversion is computationally expensive for finite fields of the sizes required for cryptography. In the context of computational number theoretic applications like those in [15], affine formulas are superior because group order algorithms such as baby-step giant-step require frequent equality tests. As representations of group elements using projective coordinates are not unique, this implies a non-negligible computational cost per test, and precludes the use of more efficient searchable data structures for the baby steps such as hash tables. Furthermore, as described in [23], in these types of application the inversions can often be combined using a trick due to Montgomery, allowing a batch of inversions to be computed with only one field inversion and a small number of field multiplications. Thus, in this paper, as cryptographic applications are not our motivation, we chose to only present affine formulas. If projective formulas are required for some other application, our formulas can readily be converted to that setting; such formulas can be found in [18].

2 Hyperelliptic Curves

We refer the reader to [7,18,19] for more background on hyperelliptic curves. Throughout, let $\mathbb{F}_q$ be a finite field where q is a power of a prime, $\mathbb{F}_q[x]$ the univariate polynomial ring over $\mathbb{F}_q$, and $\mathbb{F}_q(x)$ the field of rational functions over $\mathbb{F}_q$. A *hyperelliptic curve C of genus g* is a smooth projective curve with an affine equation of the form $y^2 + h(x)y = f(x)$ where $h, f \in \mathbb{F}_q[x]$ satisfy certain properties. More specifically, C is said to be

- *ramified (or imaginary)* if $\deg(f) = 2g+1$, f is monic, $h = 0$ when $\mathrm{char}(\mathbb{F}_q) \neq 2$, and h is monic of degree at most g when $\mathrm{char}(\mathbb{F}_q) = 2$;
- *split (or real)* if $\deg(f) = 2g + 2$, $h = 0$ and f is monic when $\mathrm{char}(\mathbb{F}_q) \neq 2$, h is monic of degree $g + 1$ and the leading coefficient of f is of the form $e^2 + e$ for some $e \in \mathbb{F}_q^*$ when $\mathrm{char}(\mathbb{F}_q) = 2$;

The *coordinate ring* and *function field* of C are $\mathbb{F}_q[C] = \mathbb{F}_q[x, y]$ and $\mathbb{F}_q(C) = \mathbb{F}_q(x, y)$, respectively. Ramified curves C have a unique $\mathbb{F}_q$-rational point at infinity, denoted by ∞, whereas a split curve C has two such points, denoted by ∞^+ and ∞^-, respectively. For $\alpha \in \mathbb{F}_q(C)$, we put $\deg(\alpha) = -v_{\infty^+}(\alpha)$ and define $\lfloor \alpha \rfloor$ to be the unique polynomial $A \in \mathbb{F}_q[x]$ with $\deg(\alpha - A) < 0$. Then the polynomial $\lfloor y \rfloor \in \mathbb{F}_q[x]$ is well defined and has degree $g + 1$. It will be required in several algorithms.

2.1 Jacobian

Let $Cl^0(C) = Cl^0_{\mathbb{F}_q}(C)$ denote the *Jacobian* of C over $\mathbb{F}_q$; this is the group of classes of degree zero divisors defined over $\mathbb{F}_q$ under linear equivalence. The equivalence class of a degree zero divisor D is denoted $[D]$. The inverse of a class $[D]$ in $Cl^0(C)$ is $[\,\overline{D}\,]$ where $\overline{D}$ is the image of D under the *hyperelliptic involution.* This map sends a point $P = (a, b)$ on C to the point $\overline{P} = (a, -b - h(a))$ on C. On ramified models, it leaves the point at infinity fixed, whereas on split models, it sends ∞^+ to ∞^- and vice versa.

In the case of split hyperelliptic curves, the cyclic subgroup

$$G = \langle[\infty^+ - \infty^-]\rangle$$

of $Cl^0(C)$ will be of key importance later on. Its order $R = |G|$ is the *regulator* of C. Schmidt [20] showed that $|Cl^0(C)| = Rh'$, where h' is the ideal class number of $\mathbb{F}_q[C]$. For most split hyperelliptic curves, h' is small; frequently $h' = 1$, so $Cl^0(C) = G$. Since the Hasse-Weil bounds establish $(\sqrt{q} - 1)^{2g} \leq |Cl^0(C)| \leq (\sqrt{q} + 1)^{2g}$, the regulator R is generally of magnitude q^g for large q.

A divisor on C is *affine* if it contains no infinite points. An affine divisor is *reduced* if its degree is at most g and it is not supported simultaneously at P and $\overline{P}$ for any affine point P. A degree zero divisor is reduced if its affine part is reduced. Reduced divisors are said to be *generic* if their affine part has degree g and *degenerate* otherwise.

Every affine reduced divisor D is uniquely determined by its *Mumford basis*[1] which consists of two polynomials $u, v \in \mathbb{F}_q[x]$ such that u is monic, u divides $v^2 - hv - f$ and $\deg(u) = \deg(D) \leq g$. Here, u is unique and v is unique modulo u, and we write $D = [u, v]$. In the *standard* or *adapted* Mumford basis of D, we choose v such that $\deg(v) < \deg(u)$; this is the typical choice for ramified models. For split models, we usually work with the *reduced* Mumford basis which satisfies $\deg(v-y-h) < \deg(u) < \deg(y+v)$. If $[u, v]$ is the standard basis of a reduced divisor, then the corresponding reduced basis is $[u, v']$ where $v' = \lfloor y \rfloor + h + v - \big((\lfloor y \rfloor + h) \bmod u\big)$.

Every degree zero divisor class on a ramified model contains a unique reduced representative of the form $D - \deg(D)\infty$ with D affine. In contrast, on split models, since there are two points at infinity, a degree zero divisor class may contain many reduced divisors. In [7], Galbraith et al. introduced a unique representative for each degree zero divisor class which is "balanced at infinity". Put $D_\infty = \lceil g/2 \rceil \infty^+ + \lfloor g/2 \rfloor \infty^-$. Then every degree zero divisor can be written as $D = D_0 - D_\infty$ where $D_0 = D_0' + n\infty^+ + m\infty^-$ with D_0' affine and $n, m \in \mathbb{Z}$. Galbraith et al. proved that every element of $Cl^0(C)$ has a unique *balanced* representative, i.e. a reduced divisor D such that $0 \leq n \leq g - \deg(D_0')$; since

[1] There is a discrepancy between [7] and [18, Definition 2.2.13] in the definition of the Mumford representation for split models. The signs of the polynomial v are opposite, which leads to slight differences in the descriptions of the algorithms for Jacobian arithmetic in the two sources. [7] is consistent with literature on ramified models, whereas [18] follows previous literature on split models such as [5,11,12,14,19,21].

$m = g - \deg(D_0') - n$, this implies $0 \le m \le g - \deg(D_0')$. We write $D = (D_0', n)$ or $D = ([u, v], n)$ where $[u, v]$ is the reduced or adapted Mumford basis of D_0'. A reduced divisor is generic if and only if $n = m = 0$; in this case, we simply write $D = [u, v]$. It is thus clear that every generic reduced divisor is balanced. Conversely, heuristically, almost all balanced divisors are generic. More specifically, we expect the probability that an element of G is represented by a degenerate balanced divisor to approach h'/q as $q \to \infty$; when $h' = 1$. i.e. $Cl^0(C) = G$, this was proved by Fontein (see, for example, [11]). Hence, heuristically, degenerate balanced divisors are extremely rare when q is of cryptographic size. This motivates the term "balanced", since generically, the infinite support of D is approximately equally balanced between the two points at infinity.

The group law on the Jacobian of a split hyperelliptic curve consists of performing the compound operation of formal addition of divisors (realized in practice as composition), divisor reduction and balancing, in this order. It is referred to as *addition* or, when the two inputs are identical, *doubling*. Adding divisor classes via divisor composition and reduction is akin to the standard arithmetic in the Jacobian of a ramified hyperelliptic curve as described, for example, by Cantor [2], and produces a reduced divisor $D = (D', n)$ that need not be balanced. A *balancing step*, denoted $D \to D_+$, decreases the value of n, whereas an *inverse balancing step*, denoted $D \to D_-$, increases this value; see Algorithm 3 of [7] and Algorithms 3 and 4 of [18]. Given any two balanced divisors D_1 and D_2 on C, the balanced representative of the class of $D_1 + D_2$ is denoted $D_1 \oplus D_2$. Thus, the group law on $Cl^0(C)$ can be expressed as $[D_1] + [D_2] = [D_1 \oplus D_2]$. Algorithm 5 of [18] and Algorithm 4 of [7] compute the divisor $D_1 \oplus D_2$. This algorithm can be considered as the main operation on the Jacobian of a split hyperelliptic curve. It is one of the fundamental operations required for many applications related to split hyperelliptic curves. See [7] and [18] for more details.

2.2 Infrastructure

Let C be a split hyperelliptic curve. In this section, we summarize the main properties of the infrastructure of C; details can be found in [18,19,21].

Every non-zero $\mathbb{F}_q[C]$-ideal $\mathfrak{a}$ is an $\mathbb{F}_q[x]$-module of rank 2 with a basis of the form $\{su, s(v + y)\}$ where u divides $v^2 - vh - f$; write $\mathfrak{a} = [su, s(v + y)]$. If we take s and u to be monic, then s and u are unique and v is unique modulo u. The ideal $\mathfrak{a}$ is *primitive* if $s = 1$ and *reduced* if additionally $\deg(u) \le g$. The basis $[u, v]$ of a primitive ideal $\mathfrak{a} = [u, y + v]$ is *adapted* if $\deg(v) < \deg(u)$, and *reduced* if $\deg(v - h - y) < \deg(u) < \deg(y + v)$.

The *divisor associated to* $\mathfrak{a} = [u, v + y]$ is the affine divisor $D = \operatorname{div}(\mathfrak{a})$ whose Mumford representation is $[u, v]$. The *degree* of $\mathfrak{a}$ is $\deg(\mathfrak{a}) = \deg(\operatorname{div}(\mathfrak{a}))$, i.e. the degree of its associated divisor. A *generic* (resp. *degenerate*) reduced ideal is one whose associated divisor is generic (resp. degenerate), so $\mathfrak{a}$ is a generic ideal if and only if the balanced divisor $D = (\operatorname{div}(\mathfrak{a}), 0)$ is generic.

Definition 1. *The* infrastructure *of C is the (finite) collection $\mathscr{R}$ of reduced principal $\mathbb{F}_q[C]$-ideals. For an ideal $\mathfrak{a} \in \mathscr{R}$, the* distance *of $\mathfrak{a}$ is defined to be $\delta(\mathfrak{a}) = \deg(\alpha)$, where α is the unique generator of $\mathfrak{a}$ with $0 \leq \deg(\alpha) < R$.*

The fact that no two infrastructure ideals have the same distance imposes an ordering on $\mathscr{R}$ by distance. Hence, if we put $\mathfrak{a}_1 = (1)$ and $\delta_i = \delta(\mathfrak{a}_i)$, then we can write

$$\mathscr{R} = \{\mathfrak{a}_1, \mathfrak{a}_2, \ldots, \mathfrak{a}_r\} \qquad 0 = \delta_1 < \delta_2 < \cdots < \delta_r < R\ .$$

Computing the distance of an infrastructure ideal given its $\mathbb{F}_q[x]$-basis is believed to be computationally infeasible; this is the *infrastructure discrete logarithm problem* on which the security of infrastructure-based cryptosystems is based [12,14,19]. However, the relative distance between two successive ideals can be efficiently computed. In [12] and [21], it was proved that if $\mathfrak{a}_i = [u_{i-1}, y + v_{i-1}]$ then

$$\delta_{i+1} = \delta_i + g + 1 - \deg(u_{i-1}) \quad \text{ for } 1 \leq i \leq r-1. \tag{1}$$

Thus, $\delta_{i+1} = \delta_i + 1$ when $\mathfrak{a}_i$ is generic. Hence, most integers between 0 and $R-1$ occur as distance values, but this is not true for all of them: for example, $\delta_2 = g+1$ shows that there are no infrastructure ideals with distance between 1 and g.

The infrastructure supports two main operations. A *baby step* computes $\mathfrak{a}_{i+1}$ from $\mathfrak{a}_i$, along with the relative distance $\delta_{i+1} - \delta_i$. Applying the same operation a sufficient number of times to a non-reduced principal $\mathbb{F}_q[C]$-ideal produces an infrastructure ideal; this process is referred to as ideal reduction. A *giant step* computes on input $\mathfrak{a}, \mathfrak{b} \in \mathscr{R}$ the first reduced ideal equivalent to the product $\mathfrak{a}\mathfrak{b}$ when applying reduction; this ideal is denoted $\mathfrak{a} \otimes \mathfrak{b}$ and is obtained after at most (and generically exactly) $\lceil g/2 \rceil$ reduction steps. The *conjugate* ideal of an ideal $\mathfrak{a} = [u, v] \in \mathscr{R}$ is the ideal $\bar{\mathfrak{a}} = [u, v - h - y] \in \mathscr{R}$; it satisfies $\mathrm{div}(\bar{\mathfrak{a}}) = \overline{\mathrm{div}(\mathfrak{a})}$ and has distance $\delta(\bar{\mathfrak{a}}) = R + \deg(u) - \delta(\mathfrak{a})$. With these notions, $\mathscr{R}$ is "almost" an abelian group under $\otimes$, where the identity is $\mathfrak{a}_1$, the "inverse" of an infrastructure ideal is its conjugate ideal, and $\mathscr{R}$ fails associativity only barely; specifically, if $\mathfrak{a}, \mathfrak{b} \in \mathscr{R}$, then

$$\delta(\mathfrak{a} \otimes \mathfrak{b}) = \delta(\mathfrak{a}) + \delta(\mathfrak{b}) - d \ \text{ with } \ 0 \leq d \leq 2g.$$

The quantity d is very small and is expected to be equal to $\lceil g/2 \rceil$ almost always as explained in Sect. 2.3; see also [12,14,19] for further details. This "shortfall" in distance is the reason that $\mathscr{R}$ is not associative under giant steps. This can be rectified for almost all giant steps by shifting distances down by $\lceil g/2 \rceil$. To that end, we define a new distance on $\mathscr{R}$ as

$$\gamma(\mathfrak{a}) = \delta(\mathfrak{a}) - \lceil g/2 \rceil\ .$$

We refer to δ as the classic distance and to $\mathscr{R}$ under δ as the classic infrastructure. Note that γ preserves the ordering on $\mathscr{R}$ defined by δ, but the first distance value is now negative; specifically, $\gamma(\mathfrak{a}_1) = -\lceil g/2 \rceil < 0$ and $\gamma(\mathfrak{a}_2) = \lfloor g/2 \rfloor + 1 > 0$. Moreover,

$$\gamma(\mathfrak{a} \otimes \mathfrak{b}) = \gamma(\mathfrak{a}) + \gamma(\mathfrak{b}) + (\lceil g/2 \rceil - d) \quad \text{with} \quad 0 \le d \le 2g. \tag{2}$$

Since we almost always expect $d = \lceil g/2 \rceil$, the new distance is additive for most inputs. The ordering on $\mathscr{R}$ determined by γ shows that for every integer N with $-\lfloor g/2 \rfloor \le N < R - \lceil g/2 \rceil$, there exists a unique ideal $\mathfrak{a}_i \in \mathscr{R}$ such that $\gamma(\mathfrak{a}_i) \le N < \gamma(\mathfrak{a}_{i+1})$. This ideal $\mathfrak{a}_i$, referred to as the infrastructure ideal *below* N (with respect to γ) and denoted $\mathfrak{a}[N]$, can be efficiently computed, along with the "error" $N - \gamma(\mathfrak{a}[N])$. We expect $\gamma(\mathfrak{a}[N]) = N$ most of the time.

2.3 Connection Between G and $\mathscr{R}$

Mireles Morales in [17] introduced an injective map from the infrastructure into the Jacobian of a split hyperelliptic curve C. In [11], a shift of $\lceil g/2 \rceil [\infty^+ - \infty^-]$ was applied to the images under this map, yielding the injection

$$\begin{aligned} &\phi : \mathscr{R} \to Cl^0(C) \\ &\phi(\mathfrak{a}) = [\mathrm{div}(\mathfrak{a}) + (g - \deg(\mathfrak{a}))\infty^- - D_\infty] = [(\mathrm{div}(\mathfrak{a}), 0)]. \end{aligned}$$

This modification of the Mireles Morales map was a precursor to the analogous shift of $\lceil g/2 \rceil$ used to define the new distance γ on $\mathscr{R}$ and ensures that the representatives of the elements in the image $\phi(\mathscr{R})$ are balanced. By Proposition 4.2 of [11], we have

$$\phi(\mathfrak{a}) = \gamma(\mathfrak{a})\,[\infty^+ - \infty^-]. \tag{3}$$

This identity shows that the image of $\mathscr{R}$ under ϕ is the subset of G consisting of precisely those classes $m[\infty^+ - \infty^-] \in G$ for which $(m \bmod R)$ is the new distance of some infrastructure ideal.

Based on the presumed scarcity of degenerate divisors and ideals, [12] and [11] formulated two heuristic assumptions, denoted (H1) and (H2) in these sources. Reformulating these statements with the new distance, (H1) asserts that with heuristic probability $1 - O(q^{-1})$ as $q \to \infty$, $\mathfrak{a}_i$ is generic, or equivalently, $\gamma(\mathfrak{a}_{i+1}) = \gamma(\mathfrak{a}_i) + 1$ for $2 \le i \le r - 1$. Heuristic (H2) states that $d = \lceil g/2 \rceil$ in (2) under the same assumption, or equivalently, $\gamma(\mathfrak{a} \otimes \mathfrak{b}) = \gamma(\mathfrak{a}) + \gamma(\mathfrak{b})$ for all $\mathfrak{a}, \mathfrak{b} \in \mathscr{R} \setminus \{\mathfrak{a}_1\}$. Hence, infrastructure arithmetic using the new distance γ is more natural and less complicated than using the classic distance δ. The assertion that $d = \lceil g/2 \rceil$ in (2) also implies that the number of reduction steps required in a giant step is heuristically equal to $\lceil g/2 \rceil$; see [18, Remark 3.2.4] for a formal proof. Both heuristics also imply that $\gamma(\mathfrak{a}[N]) = N$ almost always.

Jacobson et al. in [12] obtained improvements to scalar multiplication in the infrastructure by taking advantage of the fact that these heuristics eliminate the need to keep track of all relative distances when performing infrastructure arithmetic. In cryptographic protocols such as infrastructure Diffie-Hellman, even though the respective computations of the two communicants are different, the two parties "skip over" the same degenerate ideals (i.e. exceptions to the heuristics) and are hence expected to obtain the same target ideal. Extensive numerical computations in [12] confirm this. The description of the map ϕ in (3) makes it possible to extend the improvements of [12] to computations in G by deriving analogous heuristics for divisors in $\phi(\mathscr{R}) \subset G$. These modified heuristics assert that as $q \to \infty$, the following properties hold with heuristic probability $1 - O(q^{-1})$:

(H1) $\phi(\mathfrak{a}_{i+1}) = \phi(\mathfrak{a}_i) + [\infty^+ - \infty^-]$ for $2 \le i \le r-1$.
(H2) $\phi(\mathfrak{a} \otimes \mathfrak{b}) = \phi(\mathfrak{a}) + \phi(\mathfrak{b})$ for $\mathfrak{a}, \mathfrak{b} \in \mathscr{R} \setminus \{\mathfrak{a}_1\}$.

(H2) explicitly links the arithmetic of $\mathscr{R}$ with that of G. This implies, as already noted in [17], that the infrastructure discrete logarithm problem can be reduced to the discrete logarithm problem in G and vice versa. Note that (1) shows that the cardinalities of $\mathscr{R}$ and G are of the same magnitude when g is small.

For $1 \le i \le r$, let $D_i = (\mathrm{div}(\mathfrak{a}_i), 0)$ be the balanced divisor representing the class $\phi(\mathfrak{a}_i)$. Applying Algorithm 3 of [18] (also Algorithm 3 of [7]) to a generic reduced divisor D shows that applying a balancing step to D is arithmetically the same as adding the divisor class $[\infty^+ - \infty^-]$ to $[D]$; this was already remarked in [7]. On the other hand, heuristically, $[D_{i+1}] = [D_i] + [\infty^+ - \infty^-]$ for $2 \le i \le r-1$ by (H1). As a result, we see that a balancing step in the Jacobian and a baby step in the infrastructure act identically; more exactly, the formulas for a balancing step applied to a Mumford basis $[u, v]$ of a reduced divisor are identical to those for a baby step applied to the infrastructure ideal $\mathfrak{a} = [u, v+y]$. An analogous argument applies to inverse balancing steps and backward baby steps. Moreover, if $D = (\mathrm{div}(\mathfrak{a}), 0)$ and $E = (\mathrm{div}(\mathfrak{b}), 0)$, then heuristically, the balanced representative of the class $[D] + [E] = [D + E]$ is $(\mathrm{div}(\mathfrak{a} \otimes \mathfrak{b}), 0)$. It is obtained by applying at most $\lceil g/2 \rceil$ reduction steps to $D + E$, and no subsequent balancing steps are needed. This shows that the operations $\otimes$ on $\mathscr{R}$ and $\oplus$ on G have identical cost, generically. More detailed formal proofs of these results can be found in Sect. 3.3 of [18]. Finally, if $[D] = \phi(\mathrm{div}(\mathfrak{a}))$, then heuristically, $[nD] = \phi(\mathrm{div}(\mathfrak{b}))$ where $\mathfrak{b}$ is the infrastructure ideal with $\gamma(\mathfrak{b}) = n\gamma(\mathfrak{a})$. In other words, heuristically, scalar multiplying a generic divisor in G by n results in multiplying the associated distance value on $\mathscr{R}$ by n. Therefore, using the new distance, exponentiation on $\mathscr{R}$ and scalar multiplication on G are arithmetically identical.

We conclude this section with a remark on generic inversion in G, an operation that is required for some applications. Recall that $D_\infty = \lceil g/2 \rceil \infty^+ + \lfloor g/2 \rfloor \infty^-$, so $\overline{D}_\infty = D_\infty$ when g is even, and $\overline{D}_\infty = D_\infty - (\infty^+ - \infty^-)$ when g is odd. As a result, if $D = (D', 0)$ is a generic balanced divisor, then assuming (H1), the balanced representative of $[\overline{D}]$ is $(\overline{D'}, 0)$ if g is even and $(E', 0)$ if g is odd, where E' is obtained by applying a balancing step (i.e. an addition of $[\infty^+ - \infty^-]$ by our earlier remarks) to $\overline{D'}$.

3 Explicit Formulas

All algorithms in the Jacobian and infrastructure operate on the Mumford polynomials of a divisor or its corresponding ideal. Explicit formulas describe these operations in terms of the coefficients of the polynomials, allowing better optimization. We developed novel explicit formulas for genus 3 split hyperelliptic curves over both odd characteristic and characteristic 2 fields in affine representation (requiring one field inversion). Our input divisors are all reduced and given by their reduced Mumford basis.

The first step, as is standard in the literature, is to apply curve isomorphisms to cause as many coefficients of f and h as possible to vanish. For a genus 3 split curve C with equation $y^2 + h(x)y = f(x)$, we write $h(x) = \sum_{i=0}^{4} h_i x^i$ and $f(x) = \sum_{i=0}^{8} f_i x^i$. In odd characteristic, $h(x) = 0$ and the transformation $x \to x - f_7/8$ eliminates the x^7 term of $f(x)$. In characteristic 2, when $h_3 = 0$ the substitutions $x \to x$ and

$$y \to y + f_7 x^3 + (f_6 + f_7^2)x^2 + (f_7 h_2 + f_5)x + (h_2 f_6 + h_2 f_7^2 + f_6^2 + f_7^2 + h_1 f_7 + f_4)$$

cause the x^7, x^6, x^5, x^4 terms in $f(x)$ to vanish, as can easily be verified by direct substitution and simplification. A similar transformation exists when $h_3 \neq 0$, but we restrict to the case $h_3 = 0$ because having the x^3 coefficient of y being zero results in more efficient formulas. It is straightforward to compute the coefficients of $\lfloor y \rfloor = \sum_{i=0}^{4} y_i x^i$ by equating symbolically the coefficients of $y^2 + h(x)y$ with those of $f(x)$.

When implementing addition in the Jacobian and infrastructure, following Cantor's algorithm [2] literally is not the best strategy. It is instead preferable to use the method suggested by Gaudry and Harley in [9], in which the computations are broken into sub-expressions that can be re-used as much as possible. As in [9], we also specialize to the frequently occurring cases where the input divisors $D_1 = [u_1, v_1]$ and $D_2 = [u_2, v_2]$ for addition and $D = [u, v]$ for doubling are generic (i.e. $\deg(u_1) = \deg(u_2) = \deg(u) = 3$) and satisfy $\gcd(u_1, u_2) = \gcd(u, h + 2v) = 1$. In the rare cases where this does not hold, an implementation can resort to Cantor's algorithm. Specializations of Harley's addition and doubling algorithms for split hyperelliptic curves of genus 3 are given in Appendix A as Algorithms A.3 and A.4, respectively. Furthermore, balancing and inverse balancing steps in $Cl^0(C)$, or equivalently, baby steps and backward baby steps in the infrastructure, are given as Algorithms A.1 and A.2.

We convert these algorithms to explicit formulas using standard methods from the literature as follows; see, for example, [6,8,16,24].

1. Both addition and doubling require the computation of a resultant, with both inputs of degree 3 for addition and inputs of degree 3 and 4 for doubling. We employ the method presented in [1] based on Cramer's rule to compute the quantity *inv* in step 1 of Algorithms A.3 and A.4 directly, yielding the smallest number of finite field operations required for this computation.
2. We use Karatsuba's method for polynomial multiplication and modular reduction whenever possible.
3. All four algorithms require exact polynomial division. In this context, we need not compute all the coefficients of the inputs since, as explained in [8] for example, the division of two polynomials of respective degrees d_1 and d_2 with $d_1 > d_2$ depends only on the $d_1 - d_2 + 1$ highest coefficients of the dividend.
4. As field inversions are significantly more costly than other operations (eg. estimates suggest that one field inversion in odd characteristic costs approximately the same as 100 multiplications), we seek to eliminate as many of these as possible. We use Montgomery's trick [3, Algorithm 10.3.4] to invert the product of all elements that must be inverted and recover the individual inverses at the cost of a few extra multiplications. Specifically, putting $I = (ab)^{-1}$, we have $a^{-1} = bI$ and $b^{-1} = aI$. This technique produces formulas that require only a single field inversion.

Our explicit formulas using affine coordinates for a balancing step, inverse balancing step, addition and doubling are presented in Appendix B. They can also be found in [18, Sections 4.6.1 and 4.6.2], along with further details and a more comprehensive exposition of the techniques employed.

To test our formulas, they were initially implemented in Sage version 5.2 using the small finite fields $\mathbb{F}_{7919}$ and $\mathbb{F}_{2^{13}}$. The testing was done in three stages. First, we implemented Cantor's algorithm with no modifications, then Harley's algorithm, and finally, our explicit formulas. We then compared all three outputs. We used the computer algebra library NTL to generate split hyperelliptic curves over the desired finite fields and reduced divisors on these curves. Our testing was done on 20 different examples. To test our explicit formulas for large finite fields, we implemented the Diffie-Hellman key exchange protocol in C++. The correctness of our formulas was confirmed by the fact that both parties always obtained the same shared key, after hundreds of thousands of iterations of the protocol over numerous randomly generated curves and base fields.

Operation counts for all our formulas are presented in Table 1. For each operation, we give the number of field squarings (S), multiplications (M), and inversions (I). We did not count multiplications or squarings involving coefficients of $\lfloor y \rfloor$, h, and f exclusively (such as $h_2 y_4$ for example), since such quantities can be precomputed for a given curve and can be optimized as multiplications by constants.

In addition to our new formulas, we compare operation counts for the best available formulas in the literature. Counts for affine formulas for genus 3 ramified curves in odd characteristic are taken from [6] We considered the explicit formulas presented in Tables B.1 and C.1 of [10] for genus 3 ramified curves in characteristic 2. The closest analogue to a balancing step on a ramified model is the addition of a generic divisor class represented by a divisor of the form $P - \infty$ where P is an $\mathbb{F}_q$-rational point on the curve. We used Tables VIII and XIII in [6] for the baby/balancing step equivalent in affine coordinates on ramified curves.

For genus 2 split hyperelliptic curves, we used the explicit formulas from [5]. For genus 3 split hyperelliptic curves in odd characteristic, we also include the operation counts from [23], for which the input divisors are represented by their adapted basis.

Table 1. Explicit Formula Operation Counts for Jacobian Arithmetic

	Operation	Split $g = 2$	Split $g = 3$	Split $g = 3$ from [23]	Ramified $g = 3$
$\mathbb{F}_q$	Baby/Bal Step	1I+4M+4S	1I+9M+1S	1I+14M	1I+21M
	Inv Baby/Bal Step	1I+4M+2S	1I+9M+1S	-	1I+21M
	Addition	1I+26M+2S	1I+74M+1S	1I+79M	1I+67M
	Doubling	1I+28M+4S	1I+84M+2S	1I+82M	1I+68M
$\mathbb{F}_{2^n}$	Baby/Bal Step	1I+5M+1S	1I+9M+1S	-	1I+20M
	Inv Baby/Bal Step	1I+5M+1S	1I+9M+1S	-	1I+20M
	Addition	1I+27M+1S	1I+81M	-	1I+64M+4S
	Doubling	1I+29M+2S	1I+89M+1S	-	1I+64M+5S

As expected, the number of operations for split models exceeds those for ramified models. The main reason is that $\deg(f) = 7$ for ramified models, but 8 for split models. Our addition and baby/balancing step formulas are more efficient than those given in [23], but our doublings are more costly. Note that for use in the baby-step giant-step algorithm, faster addition is in fact favorable, as addition is the main operation required. Note also that [23] only contains formulas for curves in odd characteristic.

4 Conclusion

Our investigations show that, computationally, there is no advantage to using Jacobian arithmetic over infrastructure arithmetic for number theoretic computations on split hyperelliptic curves. Jacobian arithmetic on balanced divisors realizes arithmetic in a group (as opposed to an ordered set) and is hence perhaps more natural and conceptually simpler than infrastructure arithmetic. Moreover, it covers the entire Jacobian rather than just a fixed (albeit large) subgroup. Hence, we recommend using Jacobian arithmetic in general. The computational advantages offered by the inexpensive infrastructure baby step operation, as recognized by Stein and Teske [22] for example, can be gained in general by computing in the subgroup G as opposed to the infrastructure.

Our new explicit formulas for genus 3 split hyperelliptic curves will be useful for researchers doing large scale computations in the Jacobian, especially for applications that rely heavily on addition and baby steps, as they require fewer field operations as compared to those of Sutherland [23]. We are currently investigating whether further improvements may be obtained by using other approaches to arithmetic such as the geometric methods described by Costello and Lauter [4] and the Shanks' NUCOMP algorithm as adapted to hyperelliptic curves by Jacobson, Scheidler and Stein [13].

A Basic Algorithms

All input divisors are assumed to be given by a reduced Mumford basis $\{u, v\}$ with $u = x^3 + u_2x^2 + u_1x + u_0$ and $v = v_4x^4 + v_3x^3 + v_2x^2 + v_1x + v_0$, with v_4 and v_3 are determined by f and h and are never computed. We use the notation $D = [u_2, u_1, u_0, v_2, v_1, v_0]$. In Algorithms A.3 and A.4, s_n and s'_n denote the leading coefficient of the polynomials s and s', respectively.

Algorithm A.1. Balancing Step

```
Input: Generic reduced divisor $D_0 = [u_0, v_0]$.
Output: $D_+ = [u_1, v_1]$.
1: $v_1 = \lfloor y \rfloor + h - (v_0 + \lfloor y \rfloor) \pmod{u_0}$,   $u_1 = \frac{v'^2 - hv' - f}{u_0}$ made monic
2: return $[u_1, v_1]$
```

Algorithm A.2. Inverse Balancing Step

```
Input: Generic reduced divisor $D_0 = [u_0, v_0]$.
Output: $D_- = [u_1, v_1]$.
1: $u_1 = \frac{v_0^2 - hv_0 - f}{u_0}$ made monic,   $v_1 = \lfloor y \rfloor + h - (v_0 + \lfloor y \rfloor) \pmod{u_1}$
2: return $[u_1, v_1]$
```

Algorithm A.3 . Harley's Algorithm for Divisor Class Addition (genus 3)

```
Input: Generic reduced divisors $D_1 = [u_1, v_1]$, $D_2 = [u_2, v_2]$ with $\gcd(u_1, u_2) = 1$.
Output: $D' = D_1 \oplus D_2$.
1: Compute the resultant $r$ of $u_1$ and $u_2$; $inv = r(u_2^{-1}) \pmod{u_1}$
2: $s' = (v_1 - v_2)inv \pmod{u_1}$, $s = s'/r$ and $s_{monic} = s/s_n$
3: $l = s_{monic}u_2$, $k = \frac{f + hv_2 - v_2^2}{u_2}$
4: $u_t = \frac{s_{monic}(s_{monic}u_2 + 2v_2w - hw) - kw^2}{u_1}$ where $w = r/s'_n$,   $v_t = (su_2 + v_2) \pmod{u_t}$
5: $v'_t = \lfloor y \rfloor + h - (v_t + \lfloor y \rfloor) \pmod{u_t}$
6: $u' = \frac{f + hv'_t - (v'_t)^2}{u_t}$ made monic,   $v' = h + \lfloor y \rfloor - (\lfloor y \rfloor + v'_t) \pmod{u'}$
7: return $[u', v']$
```

Algorithm A.4 . Harley's Algorithm for Divisor Class Doubling (genus 3)

```
Input: Generic reduced divisor $D_1 = [u_1, v_1]$ with $\gcd(u_1, h + 2v_1) = 1$.
Output: $D' = D_1 \oplus D_1$.
1: Compute the resultant $r$ of $2v_1 - h$ and $u_1$; $inv = r(h + 2v_1)^{-1} \pmod{u}$
2: $k = \frac{f - hv_1 - v_1^2}{u_1}$, $k' = k \pmod{u_1}$
3: $s' = k'inv \pmod{u_1}$, $s = s'/r$ and $s_{monic} = s/s_n$
4: $l = s_{monic}u_1$
5: $u_t = s_{monic}^2 + \frac{ws_{monic}(2v_1 - h) - w^2k}{u_1}$ where $w = r/s'_n$,   $v_t = (v_1 + su_1) \pmod{u_t}$
6: $v'_t = \lfloor y \rfloor + h - (v_t + \lfloor y \rfloor) \pmod{u_t}$
7: $u' = \frac{f + hv'_t - (v'_t)^2}{u_t}$ made monic,   $v' = h + \lfloor y \rfloor - (\lfloor y \rfloor + v'_t) \pmod{u'}$
8: return $[u', v']$
```

B Genus 3 Explicit Formulas

Balancing and Inverse Balancing

Balancing Step, Odd Characteristic		
Input	$D = [u_2, u_1, u_0, v_2, v_1, v_0]$	
Output	$D_+ = [u'_2, u'_1, u'_0, v'_2, v'_1, v'_0]$	
Step	Expression	Operations
Computing $v' = \lfloor y \rfloor + h - [(\lfloor y \rfloor + v) \pmod{u}]$		
1	$v'_2 = -v_2 + 2u_1 - 2u_2^2,\quad v'_1 = -v_1 + 2u_0 - 2u_1u_2,\quad v'_0 = -v_0 - 2u_0u_2$	2M, 1S
Precomputation for Step 2		
	$w_0 = 2y_0 - 2v'_0,\quad w_1 = 2y_1 - 2v'_1,\quad w_2 = 2y_2 - 2v'_2$	
Computing $u' = $Monic$((f + hv' - v'^2)/u)$		
General case: $w_2 \neq 0$ ($\deg(u') = 3$)		
2	$I = w_2^{-1},\quad u'_2 = Iw_1 - u_2,\quad u'_1 = Iw_0 + (y_2 + v'_2)/2 - u_1 - u_2u'_2$ $u'_0 = I(f_3 - f_6v'_1) + v'_1 - u_0 - u_1u'_2 - u_2u'_1$	1I, 7M
Special case 1: $w_2 = 0$ and $w_1 \neq 0$ ($\deg(u') = 2$)		
2	$I = w_1^{-1},\quad u'_1 = Iw_0 - u_2,\quad u'_0 = I(f_3 - f_6y_1) + y_2 - u_1 - u_2u'_1$	1I, 3M
Special case 2: $w_2 = w_1 = 0$ and $w_0 \neq 0$ ($\deg(u') = 1$)		
2	$I = w_0^{-1},\quad u'_0 = I(f_3 - f_6y_1) - u_2$	1I, 1M
Special case 3: $v'_2 = y_2,\quad v'_1 = y_1$ and $v'_0 = y_0$ ($\deg(u') = 0$)		
2	$u' = 1,\quad v' = y$	
Total	General Case	1I, 9M, 1S
	Special Case 1	1I, 5M, 1S
	Special Case 2	1I, 3M, 1S
	Special Case 3	2M, 1S

Balancing Step, Characteristic 2		
Input	$D = [u_2, u_1, u_0, v_2, v_1, v_0]$	
Output	$D_+ = [u'_2, u'_1, u'_0, v'_2, v'_1, v'_0]$	
Step	Expression	Operations
Computing $v' = \lfloor y \rfloor + h - [(\lfloor y \rfloor + v) \pmod{u}]$		
1	$v'_2 = h_2 + v_2 + u_1 + u_2^2,\quad v'_1 = h_1 + v_1 + u_0 + u_1u_2,\quad v'_0 = h_0 + v_0 + u_0u_2$	2M, 1S
Precomputation for Step 2		
	$w_0 = y_0 + h_0 + v'_0,\quad w_1 = y_1 + h_1 + v'_1,\quad w_2 = y_2 + h_2 + v'_2$	
Computing $u' = $Monic$((f + hv' - v'^2)/u)$		
General case: $w_2 \neq 0$ ($\deg(u') = 3$)		
2	$I = w_2^{-1},\quad u'_2 = Iw_1 + u_2,\quad u'_1 = Iw_0 + w_2 + h_2 + u_1 + u_2u'_2$ $u'_0 = I(f_3 + h_2w_1) + h_1 + u_0 + u_1u'_2 + u_2u'_1$	1I, 7M
Special case 1: $w_2 = 0$ and $w_1 \neq 0$ ($\deg(u') = 2$)		
2	$I = w_1^{-1},\quad u'_1 = Iw_0 + u_2,\quad u'_0 = If_3 + h_2 + u_1 + u_2u'_1$	1I, 3M
Special case 2: $w_2 = w_1 = 0$ and $w_0 \neq 0$ ($\deg(u') = 1$)		
2	$I = w_0^{-1},\quad u'_0 = If_3 + u_2$	1I, 1M
Special case 3: $v'_2 = y_2$, $v'_1 = y_1$ and $v'_0 = y_0$ ($\deg(u') = 0$)		
2	$u' = 1,\quad v' = y$	
Total	General Case	1I, 9M, 1S
	Special Case 1	1I, 5M, 1S
	Special Case 2	1I, 3M, 1S
	Special Case 3	2M, 1S

Inverse Balancing Step, Odd Characteristic,		
Input Output	$D = [u_2, u_1, u_0, v_2, v_1, v_0]$ $D_- = [u'_2, u'_1, u'_0, v'_2, v'_1, v'_0]$	
Step	Expression	Operations
Precomputation		
	$w_0 = 2y_0 - 2v_0, \quad w_1 = 2y_1 - 2v_1, \quad w_2 = 2y_2 - 2v_2$	
General case: $v_2 \neq y$ $(\deg(u') = 3)$		
Computing $u' = \text{Monic}((f + hv - v^2)/u)$		
1	$\underline{u' = x^3 + u'_2 x^2 + u'_1 x + u'_0}$ $I = w_2^{-1}, \quad u'_2 = Iw_1 - u_2, \quad u'_1 = Iw_0 + (y_2 + v_2)/2 - u_1 - u_2 u'_2$ $u'_0 = I(f_3 - f_6 v_1) + v_1 - u_0 - u_2 u'_1 - u_1 u'_2$	1I, 7M
Computing $v' = \lfloor y \rfloor + h - [(\lfloor y \rfloor + v) \pmod{u'}]$		
2	$\underline{v' = x^4 + v'_2 x^2 + v'_1 x + v'_0}$ $v'_2 = -v_2 + 2u'_1 - 2(u'_2)^2, \quad v'_1 = -v_1 + 2u'_0 - 2u'_1 u'_2, \quad v'_0 = -v_0 - 2u'_0 u'_2$	2M, 1S
Special case 1: $w_2 = 0$ and $w_1 \neq 0$ $(\deg(u') = 2)$		
Computing $u' = \text{Monic}((f + hv - v^2)/u)$		
1	$\underline{u' = x^2 + u'_1 x + u'_0}$ $I = w_1^{-1}, \quad u'_1 = Iw_0 - u_2, \quad u'_0 = I(f_3 - f_6 y_1) + y_2 - u_1 - u_2 u'_1$	1I, 3M
Computing $v' = \lfloor y \rfloor + h - [(\lfloor y \rfloor + v) \pmod{u'}]$		
2	$\underline{v' = x^4 + y_2 x^2 + v'_1 x + v'_0}$ $t_1 = y_2 - u'_0 + (u'_1)^2, \quad v'_1 = -v_1 + 2(t_1 - u'_0)u'_1, \quad v'_0 = -v_0 + 2t_1 u'_0$	2M, 1S
Special case 2: $w_2 = w_1 = 0$ and $w_0 \neq 0$ $(\deg(u') = 1)$		
Computing $u' = \text{Monic}((f + hv - v^2)/u)$		
1	$\underline{u' = x + u'_0}$ $I = w_0^{-1}, \quad u'_0 = I(f_3 - f_6 y_1) - u_2$	1I, 1M
Computing $v' = \lfloor y \rfloor + h - [(\lfloor y \rfloor + v) \pmod{u'}]$		
2	$\underline{v' = x^4 + y_2 x^2 + y_1 x + v'_0}$ $t_1 = (u'_0)^2, \quad v'_0 = -v_0 - 2(y_2 + t_1)t_1 + 2y_1 u'_0$	2M, 1S
Special case 3: $w_2 = w_1 = w_0 = 0$ $(\deg(u') = 0)$		
	$u' = 1, \quad v' = y + h$	
Total	General Case	1I, 9M, 1S
	Special Case 1	1I, 5M, 1S
	Special Case 2	1I, 3M, 1S

Inverse Balancing Step, Characteristic 2,		
Input Output	$D = [u_2, u_1, u_0, v_2, v_1, v_0]$ $D_- = [u'_2, u'_1, u'_0, v'_2, v'_1, v'_0]$	
Step	Expression	Operations
Precomputation		
1	$w_0 = y_0 + h_0 + v_0, \quad w_1 = y_1 + h_1 + v_1, \quad w_2 = y_2 + h_2 + v_2$	
General case: $w_2 \neq 0$ $(\deg(u') = 3)$		
Computing u' =Monic($(f + hv - v^2)/u$)		
2	$u' = x^3 + u'_2x^2 + u'_1x + u'_0$ $I = w_2^{-1}, \quad u'_2 = Iw_1 + u_2, \quad u'_1 = Iw_0 + h_2 + w_2 + u_1 + u_2u'_2$ $u'_0 = I(f_3 + h_2w_1) + h_1 + u_0 + u_1u'_2 + u_2u'_1$	1I, 7M
Computing $v' = \lfloor y \rfloor + h - [(\lfloor y \rfloor + v) \pmod{u'}]$		
3	$v' = (y_4 + 1)x^4 + (h_3 + y_3)x^3 + v'_2x^2 + v'_1x + v'_0$ $v'_2 = h_2 + v_2 + u'_1 + (u'_2)^2, \quad v'_1 = h_1 + v_1 + u'_0 + u'_1u'_2, \quad v'_0 = h_0 + v_0 + u'_0u'_2$	2M, 1S
Special case 1: $w_2 = 0$ and $w_1 \neq 0$ $(\deg(u') = 2)$		
Computing u' =Monic($(f + hv - v^2)/u$)		
2	$u' = x^2 + u'_1x + u'_0$ $I = w_1^{-1}, \quad u'_1 = Iw_0 + u_2, \quad u'_0 = If_3 + h_2 + u_1 + u_2u'_1$	1I, 3M
Computing $v' = \lfloor y \rfloor + h - [(\lfloor y \rfloor + v) \pmod{u'}]$		
3	$v' = (y_4 + 1)x^4 + (h_3 + y_3)x^3 + (h_2 + y_2)x^2 + v'_1x + v'_0$ $t_1 = (u'_1)^2 + h_2, \quad v'_1 = h_1 + v_1 + t_1u'_1, \quad v'_0 = h_0 + v_0 + (t_1 + u'_0)u'_0$	2M, 1S
Special case 2: $w_2 = w_1 = 0$ and $w_0 \neq 0$ $(\deg(u') = 1)$		
Computing u' =Monic($(f + hv - v^2)/u$)		
2	$u' = x + u'_0$ $I = w_0^{-1}, \quad u'_0 = If_3 + u_2$	1I, 1M
Computing $v' = \lfloor y \rfloor + h - [(\lfloor y \rfloor + v) \pmod{u'}]$		
3	$v' = (y_4 + 1)x^4 + (h_3 + y_3)x^3 + (h_2 + y_2)x^2 + (h_1 + y_1)x + v'_0$ $t_1 = (u'_0)^2, \quad v'_0 = h_0 + v_0 + h_1u'_0 + (h_2 + t_1)t_1$	2M, 1S
Special case 3: $w_2 = w_1 = w_0 = 0$ $(\deg(u') = 0)$		
2	$u' = 1, \quad v' = y + h$	
Total	General Case	1I, 9M, 1S
	Special Case 1	1I, 5M, 1S
	Special Case 2	1I, 3M, 1S

Addition and Doubling

Addition, Odd Characteristic		
Input Output	$D_1 = [u_{12}, u_{11}, u_{10}, v_{12}, v_{11}, v_{10}], \quad D_2 = [u_{22}, u_{21}, u_{20}, v_{22}, v_{21}, v_{20}]$ $D_1 \oplus D_2 = [u'_2, u'_1, u'_0, v'_2, v'_1, v'_0]$	
Step	Expression	Operations
Computing the resultant r =resultant(u_1, u_2) and $inv = r(u_2^{-1}) \pmod{u_1}$		
1	$\underline{r = u_{10}(i_2t_3 + i_1t_2) - i_0t_0, \quad inv = i_2x^2 + i_1x + i_0}$ $t_0 = u_{10} - u_{20}, \quad t_1 = u_{11} - u_{21}, \quad t_2 = u_{12} - u_{22}, \quad t_3 = t_1 - t_2u_{12}$ $t_4 = t_0 - t_2u_{11}, \quad t_5 = t_4 - t_3u_{12}, \quad t_6 = t_2u_{10} + t_3u_{11}$ $i_0 = t_4t_5 + t_3t_6, \quad i_1 = -(t_2t_6 + t_1t_5), \quad i_2 = t_1t_3 - t_2t_4$	15M
Computing $s' = (v_1 - v_2)inv \pmod{u_1}$		
2	$\underline{s' = s'_2x^2 + s'_1x + s'_0}$ $w_0 = v_{10} - v_{20}, \quad w_1 = v_{11} - v_{21}, \quad w_2 = v_{12} - v_{22}$ $t_0 = w_0i_0, \quad t_1 = w_1i_1, \quad t_2 = w_2i_2$ $t_3 = u_{12}t_2, \quad t_4 = (w_1 + w_2)(i_1 + i_2) - t_1 - t_2 - t_3$ $t_5 = u_{10} + u_{12}, \quad t_6 = (t_5 + u_{11})(t_2 + t_4), \quad t_7 = (t_5 - u_{11})(t_2 - t_4)$ $s'_0 = t_0 - t_4u_{10}, \quad s'_1 = (w_0 + w_1)(i_0 + i_1) - (t_7 + t_6)/2 - t_0 - t_1 + t_3$ $s'_2 = (w_0 + w_2)(i_0 + i_2) + (t_7 - t_6)/2 + t_1 - s'_0 - t_2$ Call Cantor algorithm if $s'_2 = 0$	10M
Computing Required Inverses		
3	$\hat{u}_3 = s'_2(s'_2u_{11} - s'_0) - s'_1(s'_2u_{12} - s'_1)$ $t_1 = s'_2r, \quad w' = t_1\hat{u}_3, \quad I = w'^{-1}, \quad t_2 = It_1, \quad t_3 = I\hat{u}_3$ $I_r = t_3s'_2$ (the inverse of r), $\quad I_s = t_3r$ (the inverse of s'_2) $I_{u'} = t_1t_2$ (the inverse of the leading coefficient of u' in step 9), $\quad w = I_sr, \quad s_2 = I_rs'_2$	1I, 13M
Computing $s = s'/s'_2 = x^2 + s_1x + s_0$		
4	$s_1 = I_ss'_1, \quad s_0 = I_ss'_0$	2M
Computing $l = su_2$		
5	$\underline{l = x^5 + l_4x^4 + l_3x^3 + l_2x^2 + l_1x + l_0}$ $t_1 = u_{20} + u_{22}, \quad t_2 = (s_0 + s_1)(t_1 + u_{21}), \quad t_3 = (s_0 - s_1)(t_1 - u_{21}), \quad t_4 = s_1u_{22}$ $l_0 = s_0u_{20}, \quad l_1 = (t_2 - t_3)/2 - t_4, \quad l_2 = (t_2 + t_3)/2 - l_0 + u_{20}$ $l_3 = t_4 + s_0 + u_{21}, \quad l_4 = s_1 + u_{22}$	4M
Computing $k = (v_2^2 - hv_2 - f)/u_2$		
6	$k_3 = 2(y_2 - v_{22})$	
Computing $u_t = (s(l + wh + 2wv_2) - kw^2)/u_1$		
7	$\underline{u_t = x^4 + u_{t3}x^3 + u_{t2}x^2 + u_{t1}x + u_{t0}}$ $t_0 = l_4 + 2w, \quad t_1 = t_0s_1, \quad t_2 = s_0l_3, \quad t_3 = 2v_{22}w$ $u_{t3} = t_0 + s_1 - u_{12}, \quad t_4 = u_{12}u_{t3}, \quad u_{t2} = t_1 + l_3 + s_0 - u_{11} - t_4, \quad t_5 = u_{11}u_{t2}$ $u_{t1} = (s_0 + s_1)(l_3 + t_0) - (u_{11} + u_{12})(u_{t2} + u_{t3}) - t_1 - t_2 + t_3 + l_2 + t_4 + t_5 - u_{10}$ $u_{t0} = s_1(t_3 + l_2) + w(2v_{21} - k_3w) + t_2 + l_1 - t_5 - u_{12}u_{t1} - u_{10}u_{t3}$	12M
Computing $v_t = (v_2 + ls_2) \pmod{u_t}$		
8	$\underline{v_t = v_{t3}x^3 + v_{t2}x^2 + v_{t1}x + v_{t0}}$ $t_1 = s_2(u_{t3} - l_4) - 1$ $v_{t3} = t_1u_{t3} - s_2(u_{t2} - l_3), \quad v_{t2} = t_1u_{t2} - s_2(u_{t1} - l_2) + v_{22}$ $v_{t1} = t_1u_{t1} - s_2(u_{t0} - l_1) + v_{21}, \quad v_{t0} = t_1u_{t0} + s_2l_0 + v_{20}$	9M
Computing $u' = (f + hv'_t - (v'_t)^2)/u_t$ where $v'_t = h + \lfloor y \rfloor - (v_t + \lfloor y \rfloor) \pmod{u_t}$		
9	$\underline{u' = x^3 + u'_2x^2 + u'_1x + u'_0}$ $w_3 = v_{t3} - u_{t3}, \quad w_2 = v_{t2} - u_{t2}, \quad w_1 = v_{t1} - u_{t1}, \quad w_0 = v_{t0} - u_{t0} + y_0$ $u'_2 = I_{u'}(w_2 + y_2) - w_3/2 - u_{t3}, \quad u'_1 = I_{u'}(w_1 + y_1) - w_2 - u'_2u_{t3} - u_{t2}$ $u'_0 = I_{u'}((y_2^2 - w_2^2)/2 + w_0) - w_1 - u'_1u_{t3} - u'_2u_{t2} - u_{t1}$	6M, 1S
Computing $v' = h + \lfloor y \rfloor - (\lfloor y \rfloor + v'_t) \pmod{u'}$		
10	$\underline{v' = x^4 + v'_2x^2 + v'_1x + v'_0}$ $t_1 = w_3 + 2u'_2$ $v'_2 = -t_1u'_2 + 2u'_1 + w_2, \quad v'_1 = -t_1u'_1 + 2u'_0 + w_1, \quad v'_0 = -t_1u'_0 - y_0 + w_0$	3M
Total		1I, 74M, 1S

Addition, Characteristic 2		
Input Output	$D_1 = [u_{12}, u_{11}, u_{10}, v_{12}, v_{11}, v_{10}]$, $D_2 = [u_{22}, u_{21}, u_{20}, v_{22}, v_{21}, v_{20}]$ $D_1 \oplus D_2 = [u'_2, u'_1, u'_0, v'_2, v'_1, v'_0]$	
Step	Expression	Operations
Computing the resultant r =resultant(u_1, u_2) and $inv = r(u_2^{-1}) \pmod{u_1}$		
1	$\underline{r = i_0t_0 + u_{10}(i_2t_3 + i_1t_2), \quad inv = i_2x^2 + i_1x + i_0}$ $t_0 = u_{20} + u_{10}, \quad t_1 = u_{21} + u_{11}, \quad t_2 = u_{22} + u_{12}, \quad t_3 = t_1 + t_2u_{12}$ $t_4 = t_0 + t_2u_{11}, \quad t_5 = t_4 + t_3u_{12}, \quad t_6 = t_2u_{10} + t_3u_{11}$ $i_0 = t_4t_5 + t_3t_6, \quad i_1 = t_2t_6 + t_1t_5, \quad i_2 = t_2t_4 + t_1t_3$	15M
Computing $s' = (v_1 + v_2)inv \pmod{u_1}$		
2	$\underline{s' = s'_2x^2 + s'_1x + s'_0}$ $w_0 = v_{10} + v_{20}, \quad w_1 = v_{11} + v_{21}, \quad w_2 = v_{12} + v_{22}, \quad t_0 = w_0i_0, \quad t_1 = w_1i_1,$ $t_2 = w_2i_2$ $t_3 = (w_1 + w_2)(i_1 + i_2) + t_1 + t_2 + t_2u_{12}, \quad t_4 = t_2u_{11}, \quad t_5 = t_3u_{10}$ $s'_2 = (w_0 + w_2)(i_0 + i_2) + t_0 + t_1 + t_2 + t_4 + t_3u_{12}$ $s'_1 = (w_0 + w_1)(i_0 + i_1) + (t_3 + t_2)(u_{10} + u_{11}) + t_0 + t_1 + t_4 + t_5, \quad s'_0 = t_0 + t_5$ Call Cantor algorithm if $s'_2 = 0$	11M
Computing Required Inverses		
3	$\hat{u}_3 = s'_2(s'_2u_{11} + s'_0) + s'_1(s'_2u_{12} + s'_1)$ $t_1 = s'_2r, \quad w' = t_1\hat{u}_3, \quad I = w'^{-1}, \quad t_2 = It_1, \quad t_3 = I\hat{u}_3$ $I_r = t_3s'_2$ (the inverse of r), $\quad I_s = t_3r$ (the inverse of s'_2) $I_{u'} = t_1t_2$ (the inverse of the leading coefficient of u' in step 9), $\quad w = I_sr,$ $s_2 = I_rs'_2$	1I, 13M
Computing $s = s'/s'_2 = x^2 + s_1x + s_0$		
4	$s_1 = I_ss'_1, \quad s_0 = I_ss'_0$	2M
Computing $l = su_2$		
5	$\underline{l = x^5 + l_4x^4 + l_3x^3 + l_2x^2 + l_1x + l_0}$ $\tilde{w}_0 = s_0u_{21}, \quad \tilde{w}_1 = s_1u_{22}, \quad l_4 = s_1 + u_{22}, \quad l_3 = \tilde{w}_1 + s_0 + u_{21}$ $l_2 = (s_0 + s_1)(u_{21} + u_{22}) + \tilde{w}_1 + \tilde{w}_0 + u_{20}, \quad l_1 = s_1u_{20} + \tilde{w}_0, \quad l_0 = s_0u_{20}$	5M
Computing $k = (v_2^2 + hv_2 + f)/u_2$		
6	$k_3 = h_2 + y_2 + v_{22}$	
Computing $u_t = s(l + wh + 2wv_2) + kw^2)/u_1$		
7	$\underline{u_t = x^4 + u_{t3}x^3 + u_{t2}x^2 + u_{t1}x + u_{t0}}$ $t_0 = wh_2, \quad a_1 = s_1(w + l_4), \quad a_0 = s_0l_3$ $u_{t3} = w + l_4 + s_1 + u_{12}, \quad t_1 = u_{12}u_{t3}, \quad u_{t2} = a_1 + t_1 + l_3 + s_0 + u_{11},$ $t_2 = u_{11}u_{t2}$ $u_{t1} = (s_0+s_1)(w+l_3+l_4)+(u_{11}+u_{12})(u_{t2}+u_{t3})+t_0+a_0+a_1+l_2+t_1+t_2+u_{10}$ $u_{t0} = w(wk_3 + h_1) + s_1(t_0 + l_2) + a_0 + l_1 + t_2 + u_{12}u_{t1} + u_{10}u_{t3}$	12M
Computing $v_t = (v_2 + ls_2) \pmod{u_t}$		
8	$\underline{v_t = v_{t3}x^3 + v_{t2}x^2 + v_{t1}x + v_{t0}}$ $t_1 = s_2(u_{t3} + l_4) + y_4 + 1$ $v_{t3} = t_1u_{t3} + s_2(u_{t2} + l_3), \quad v_{t2} = t_1u_{t2} + s_2(u_{t1} + l_2) + v_{22}$ $v_{t1} = t_1u_{t1} + s_2(u_{t0} + l_1) + v_{21}, \quad v_{t0} = t_1u_{t0} + s_2l_0 + v_{20}$	9M
Computing $u' = \frac{f + hv'_t + (v'_t)^2}{u_t}$ where $v'_t = h + \lfloor y \rfloor - (v_t + \lfloor y \rfloor) \pmod{u_t}$		
9	$\underline{u' = x^3 + u'_2x^2 + u'_1x + u'_0}$ $w_3 = y_4u_{t3} + v_{t3}, \quad w_2 = y_4u_{t2} + v_{t2} + y_2, \quad w_1 = y_4u_{t1} + v_{t1} + y_1,$ $w_0 = y_4u_{t0} + v_{t0} + y_0$ $u'_2 = I_uw_2 + w_3 + u_{t3}, \quad u'_1 = I_uw_1 + h_2 + u_{t2} + u_{t3}u'_2$ $u'_0 = I_u(w_2(w_2 + h_2) + w_0) + h_1 + u_{t1} + u_{t3}u'_1 + u_{t2}u'_2$	11M
Computing $v' = h + \lfloor y \rfloor + (\lfloor y \rfloor + v'_t) \pmod{u'}$		
10	$\underline{v' = (h_4 + y_4)x^4 + (h_3 + y_3)x^3 + v_2x^2 + v_1x + v_0}$ $t_1 = w_3 + u'_2$ $v'_2 = t_1u'_2 + w_2 + y_2 + u'_1, \quad v'_1 = t_1u'_1 + w_1 + y_1 + u'_0, \quad v'_0 = t_1u'_0 + w_0 + y_0$	3M
Total		1I, 81M

Doubling, Odd Characteristic		
Input	$D = [u_2, u_1, u_0, v_2, v_1, v_0]$	
Output	$D \oplus D = [u'_2, u'_1, u'_0, v'_2, v'_1, v'_0]$	
Step	Expression	Operations
Computing the resultant $r =$resultant$(2v - h, u)$ and $inv = r(2v - h)^{-1} \pmod{u}$		
1	$\underline{r = \hat{w}_0 i_0 - u_0(t_1 i_2 + \hat{w}_2 i_1), \quad inv = i_2 x^2 + i_1 x + i_0}$	17M, 1S
	$\hat{w}_0 = 2(v_0 + u_0 u_2), \quad \hat{w}_1 = 2(v_1 - u_0 + u_1 u_2), \quad \hat{w}_2 = 2(v_2 - u_1 + u_2^2)$	
	$t_1 = \hat{w}_1 - u_2 \hat{w}_2, \quad t_2 = \hat{w}_0 - u_1 \hat{w}_2, \quad t_3 = t_2 - u_2 t_1, \quad t_4 = u_0 \hat{w}_2 + u_1 t_1$	
	$i_0 = t_2 t_3 + t_1 t_4, \quad i_1 = -(\hat{w}_1 t_3 + \hat{w}_2 t_4), \quad i_2 = \hat{w}_1 t_1 - \hat{w}_2 t_2$	
Computing $k' = (f + hv - v^2)/u, \quad k = k' \pmod{u}$		
2	$\underline{k = k_2 x^2 + k_1 x + k_0}$	7M
	$t_1 = y_2 - v_2, \quad t_2 = u_2 t_1, \quad k'_3 = 2t_1, \quad k_2 = -4t_1 + 2y_1 - 2v_1$	
	$k_1 = 2u_2(t_2 - y_1 + v_1) + t_2(y_2 + v_2 - 4u_1) + 2y_0 - 2v_0$	
	$k_0 = f_3 - u_2 k_1 - 4u_0 t_1 - 2u_1(y_1 - v_1) - 2v_1 v_2$	
Computing $s' = k.inv \pmod{u}$		
3	$\underline{s' = s_2 x^2 + s_1 x + s_0}$	10M
	$t_0 = k_0 i_0, \quad t_1 = k_1 i_1, \quad t_2 = k_2 i_2, \quad t_3 = u_2 t_2$	
	$t_4 = (k_1 + k_2)(i_1 + i_2) - t_1 - t_2 - t_3, \quad t_5 = t_4 u_0, \quad t_6 = u_0 + u_2, \quad t_7 = t_6 + u_1$	
	$t_8 = t_6 - u_1, \quad t_9 = t_7(t_4 + t_2), \quad t_{10} = t_8(t_4 - t_2)$	
	$s'_2 = (k_0 + k_2)(i_0 + i_2) - (t_{10} + t_9)/2 - t_0 - t_2 + t_1 + t_5$	
	$s'_1 = (k_0 + k_1)(i_0 + i_1) + (t_{10} - t_9)/2 - t_0 - t_1 + t_3, \quad s'_0 = t_0 - t_5$	
	If $s'_2 = 0$ then call Cantor Algorithm	
Computing Required Inverses		
4	$\hat{u}_3 = s'_2(s'_2 u_2 - s'_0) - s'_1(s'_1 - s'_2 u_2)$	1I, 13M
	$t_1 = s'_2 r, \quad w' = t_1 \hat{u}_3, \quad I = w'^{-1}, \quad t_2 = I t_1, \quad t_3 = I \hat{u}_3$	
	$I_r = t_3 s'_2$ (the inverse of r), $\quad I_s = t_3 r$ (the inverse of s'_2)	
	$I_{u'} = t_1 t_2$ (the inverse of the leading coefficient of u' in step 9), $\quad w = I_s r$,	
	$s_2 = I_r s'_2$	
Computing $s = s'/s'_2 = x^2 + s_1 x + s_0$		
5	$s_1 = I_s s'_2, \quad s_0 = I_s s'_0$	2M
Computing $l = su$		
6	$\underline{l = x^5 + l_4 x^4 + l_3 x^3 + l_2 x^2 + l_1 x + l_0}$	4M
	$t_1 = u_0 + u_2, \quad t_2 = (s_0 + s_1)(t_1 + u_1), \quad t_3 = (s_0 - s_1)(t_1 - u_1), \quad \tilde{w}_1 = s_1 u_2$	
	$l_0 = s_0 u_0$	
	$l_1 = (t_2 - t_3)/2 - \tilde{w}_1, \quad l_2 = (t_2 + t_3)/2 - l_0 + u_0, \quad l_3 = \tilde{w}_1 + s_0 + u_{21},$	
	$l_4 = s_1 + u_{22}$	
Computing $u_t = s^2 + (sw(h + 2v) - w^2 k')/u$		
7	$\underline{u_t = x^4 + u_{t3} x^3 + u_{t2} x^2 + u_{t1} x + u_{t0}}$	13M
	$u_{t3} = 2(w + s_1), \quad t_1 = u_2 u_{t3}$	
	$u_{t2} = s_1(2w + s_1) + 2\tilde{w}_1 - t_1 + 2s_0, \quad t_2 = u_1 u_{t2}$	
	$u_{t1} = 2w(s_0 + v_2) + s_1(\tilde{w}_1 + 2s_0) - (u_{t2} + u_{t3})(u_1 + u_2) + 2l_2 - 2u_0 + t_1 + t_2$	
	$u_{t0} = w(2s_1 v_2 - 2wk'_3 + 2v_1) + s_1(l_2 + u_0) + s_0(s_0 + \tilde{w}_1 + 2u_1) - t_2 - u_2 u_{t1} - u_0 u_{t3}$	
Computing $v_t = (v + s_2 l) \pmod{u_t}$		
8	$\underline{v_t = v_{t3} x^3 + v_{t2} x^2 + v_{t1} x + v_{t0}}$	9M
	$t_1 = s_2(u_{t3} - l_4) - 1$	
	$v_{t3} = u_{t3} t_1 - s_2(u_{t2} - l_3), \quad v_{t2} = u_{t2} t_1 - s_2(u_{t1} - l_2) + v_2$	
	$v_{t1} = u_{t1} t_1 - s_2(u_{t0} + l_1) + v_1, \quad v_{t0} = u_{t0} t_1 + s_2 l_0 + v_0$	
Computing $u' = (f + hv'_t - (v'_t)^2)/u_t$ where $v'_t = h + \lfloor y \rfloor - (v_t + \lfloor y \rfloor) \pmod{u_t}$		
9	$\underline{u' = u'_3 x^3 + u'_2 x^2 + u'_1 x + u'_0}$	6M, 1S
	$w_3 = v_{t3} - u_{t3}, \quad w_2 = v_{t2} - u_{t2}, \quad w_1 = v_{t1} - u_{t1}, \quad w_0 = v_{t0} - u_{t0} + y_0$	
	$u'_2 = I_{u'}(w_2 + y_2) - w_3/2 - u_{t3}, \quad u'_1 = I_{u'}(w_1 + y_1) - w_2 - u_{t3} u'_2 - u_{t2}$	
	$u'_0 = I_{u'}((y_2^2 - w_2^2)/2 + w_0) - w_1 - u_{t3} u'_1 - u_{t2} u'_2 - u_{t1}$	
Computing $v' = h + \lfloor y \rfloor - (\lfloor y \rfloor + v'_t) \pmod{u}$		
10	$\underline{v' = (h_4 + y_4) x^4 + (h_3 + y_3) x^3 + v'_2 x^2 + v'_1 x + v'_0}$	3M
	$t_1 = w_3 + 2u'_2$	
	$v'_2 = -t_1 u'_2 + 2u'_1 + w_2, \quad v'_1 = -t_1 u'_1 + 2u'_0 + w_1, \quad v'_0 = -t_1 u'_0 + w_0 - y_0$	
Total		1I, 84M, 2S

Doubling, Characteristic 2		
Input	$D = [u_2, u_1, u_0, v_2, v_1, v_0]$	
Output	$D \oplus D = [u'_2, u'_1, u'_0, v'_2, v'_1, v'_0]$	
Step	Expression	Operations
Precomputations		
Computing the resultant r =resultant$(2v + h, u)$ and $inv = r(2v + h)^{-1} \pmod{u}$		
1	$\underline{r = \hat{w}_0 i_0 + u_0(t_1 i_2 + \hat{w}_2 i_1), \quad inv = i_2 x^2 + i_1 x + i_0}$	17M, 1S
	$\hat{w}_0 = h_0 + u_0 u_2, \quad \hat{w}_1 = h_1 - u_0 + u_1 u_2, \quad \hat{w}_2 = h_2 - u_1 + u_2^2$	
	$t_1 = \hat{w}_1 + u_2 \hat{w}_2, \quad t_2 = \hat{w}_0 + u_1 \hat{w}_2, \quad t_3 = t_2 + u_2 t_1, \quad t_4 = u_0 \hat{w}_2 + u_1 t_1$	
	$i_0 = t_2 t_3 + t_1 t_4, \quad i_1 = \hat{w}_1 t_3 + \hat{w}_2 t_4, \quad i_2 = \hat{w}_1 t_1 + \hat{w}_2 t_2$	
Computing $k' = (f + hv + v^2)/u, \quad k = k' \pmod{u}$		
2	$\underline{k = k_2 x^2 + k_1 x + k_0}$	6M
	$k'_3 = h_2 + y_2 + v_2, \quad k_2 = h_1 + y_1 + v_1$	
	$k_1 = u_2(k'_3 u_2 + k_2) + v_2(h_2 + v_2) + h_0 y_4 + h_0 + v_0$	
	$k_0 = f_3 + k_2(h_2 + u_1) + k'_3 h_1 + k_1 u_2$	
Computing $s' = k.inv \pmod{u}$		
3	$\underline{s' = s'_2 x^2 + s'_1 x + s'_0}$	11M
	$t_0 = k_0 i_0, \quad t_1 = k_1 i_1, \quad t_2 = k_2 i_2, \quad t_3 = (k_1 + k_2)(i_1 + i_2) + t_1 + t_2 + t_2 u_2,$	
	$t_4 = t_2 u_1, \quad t_5 = t_3 u_0$	
	$s'_2 = (k_0 + k_2)(i_0 + i_2) + t_0 + t_1 + t_2 + t_4 + t_3 u_2$	
	$s'_1 = (k_0 + k_1)(i_0 + i_1) + (t_2 + t_3)(u_0 + u_1) + t_0 + t_1 + t_4 + t_5, \quad s'_0 = t_0 + t_5$	
	If $s'_2 = 0$ then call Cantor Algorithm	
Computing Required Inverses		
4	$\hat{u}_3 = s'_1(s'_1 + s'_2 u_2) + s'_2(s'_0 + s'_2 u_1), \quad t_1 = s'_2 r, \quad w' = t_1 \hat{u}_3, \quad I = w'^{-1},$	1I, 13M
	$t_2 = I t_1, \quad t_3 = I \hat{u}_3$	
	$I_r = t_3 s'_2$ (the inverse of r, $\quad I_s = t_3 r$ (the inverse of s'_2)	
	$I_{u'} = t_1 t_2$ (the inverse of the leading coefficient of u' in step 9), $\quad w = I_s r,$	
	$s_2 = I_r s'_2$	
Computing $s = s'/s'_2 = x^2 + s_1 x + s_0$		
5	$s_1 = I_s s'_1, \quad s_0 = I_s s'_0$	2M
Computing $l = su$		
6	$\underline{l = x^5 + l_4 x^4 + l_3 x^3 + l_2 x^2 + l_1 x + l_0}$	5M
	$\tilde{w}_1 = s_1 u_2, \quad \tilde{w}_0 = s_0 u_1, \quad l_4 = s_1 + u_2, \quad l_3 = \tilde{w}_1 + s_0 + u_1$	
	$l_2 = (s_0 + s_1)(u_1 + u_2) + \tilde{w}_0 + \tilde{w}_1 + u_0, \quad l_1 = \tilde{w}_0 + s_1 u_0, \quad l_0 = s_0 u_0$	
Computing $u_t = s^2 + (ws(h + 2v) + w^2 k')/u$		
7	$\underline{u_t = x^4 + u_{t3} x^3 + u_{t2} x^2 + u_{t1} x + u_{t0}}$	12M
	$t_1 = w u_2, \quad u_{t3} = w, \quad u_{t2} = s_1(w + s_1) + t_1, \quad t_2 = u_1 u_{t2}$	
	$u_{t1} = w(s_0 + h_2) + s_1 \tilde{w}_1 + (w + u_{t2})(u_1 + u_2) + t_1 + t_2$	
	$u_{t0} = w(s_1 h_2 + w k'_3 + h_1 + u_0) + s_1(l_2 + u_0) + s_0(s_0 + w_1) + u_2 u_{t1} + t_2$	
Computing $v_t = (v + s_2 l) \pmod{u_t}$		
8	$\underline{v_t = v_{t3} x^3 + v_{t2} x^2 + v_{t1} x + v_{t0}}$	9M
	$t_1 = s_2(w + l_4) + y_4 + 1$	
	$v_{t3} = w t_1 + s_2(u_{t2} + l_3), \quad v_{t2} = u_{t2} t_1 + s_2(u_{t1} + l_2) + v_2$	
	$v_{t1} = u_{t1} t_1 + s_2(u_{t0} + l_1) + v_1, \quad v_{t0} = u_{t0} t_1 + s_2 l_0 + v_0$	
Computing $u' = (f + h v'_t - (v'_t)^2)/u_t$ where $v'_t = h + \lfloor y \rfloor - (v_t + \lfloor y \rfloor) \pmod{u_t}$		
9	$\underline{u' = u'_3 x^3 + u'_2 x^2 + u'_1 x + u'_0}$	11M
	$w_3 = v_{t3} + y_4 w, \quad w_2 = y_2 + v_{t2} + y_4 u_{t2}, \quad w_1 = y_1 + v_{t1} + y_4 u_{t1},$	
	$w_0 = y_0 + v_{t0} + y_4 u_{t0}$	
	$u'_2 = I_{u'} w_2 + w_3 + w, \quad u'_1 = I_{u'} w_1 + h_2 + w u'_2 + u_{t2}$	
	$u'_0 = I_{u'}(w_2(w_2 + h_2) + w_0) + h_1 + w u'_1 + u_{t2} u'_2 + u_{t1}$	
Computing $v' = h + \lfloor y \rfloor + (\lfloor y \rfloor + v'_t) \pmod{u'}$		
10	$\underline{v' = (h_4 + y_4)x^4 + (h_3 + y_3)x^3 + v'_2 x^2 + v'_1 x + v'_0}$	3M
	$t_1 = u'_2 + w_3$	
	$v'_2 = u'_2 t_1 + u'_1 + w_2 + y_2, \quad v'_1 = u'_1 t_1 + u'_0 + w_1 + y_1, \quad v'_0 = u'_0 t_1 + w_0 + y_0$	
Total		1I, 89M, 1S

References

1. Avanzi, R., Thériault, N., Wang, Z.: Rethinking low genus hyperelliptic Jacobian arithmetic over binary fields: interplay of field arithmetic and explicit formuae. J. Math. Cryptol. **2**(3), 227–255 (2008)
2. Cantor, D.G.: Computing in the Jacobian of a hyperelliptic curve. Math. Comp. **48**(177), 95–101 (1987)
3. Cohen, H.: A Course in Computational Algebraic Number Theory. Graduate Texts in Mathematics, vol. 138. Springer, Berlin (1993)
4. Costello, C., Lauter, K.: Group law computations on Jacobians of hyperelliptic curves. In: Miri, A., Vaudenay, S. (eds.) SAC 2011. LNCS, vol. 7118, pp. 92–117. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28496-0_6
5. Erickson, S., Jacobson Jr., M.J., Stein, A.: Explicit formulas for real hyperelliptic curves of genus 2 in affine representation. Adv. Math. Commun. **5**(4), 623–666 (2011)
6. Fan, X., Wollinger, T.J., Gong, G.: Efficient explicit formulae for genus 3 hyperelliptic curve cryptosystems over binary fields. IET Inf. Secur. **1**(2), 65–81 (2007)
7. Galbraith, S.D., Harrison, M., Mireles Morales, D.J.: Efficient hyperelliptic arithmetic using balanced representation for divisors. In: van der Poorten, A.J., Stein, A. (eds.) ANTS 2008. LNCS, vol. 5011, pp. 342–356. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-79456-1_23
8. Gathen, J.V.Z., Gerhard, J.: Modern Computer Algebra, 3rd edn. Cambridge University Press, Cambridge (2013)
9. Gaudry, P., Harley, R.: Counting points on hyperelliptic curves over finite fields. In: Bosma, W. (ed.) ANTS 2000. LNCS, vol. 1838, pp. 313–332. Springer, Heidelberg (2000). https://doi.org/10.1007/10722028_18
10. Guyot, C., Kaveh, K., Patankar, V.M.: Explicit algorithm for the arithmetic on the hyperelliptic Jacobians of genus 3. J. Ramanujan Math. Soc. **19**, 75–115 (2004)
11. Jacobson Jr., M.J., Rezai Rad, M., Scheidler, R.: Comparison of scalar multiplication on real hyperelliptic curves. Adv. Math. Commun. **8**(4), 389–406 (2014)
12. Jacobson Jr., M.J., Scheidler, R., Stein, A.: Cryptographic protocols on real hyperelliptic curves. Adv. Math. Commun. **1**(2), 197–221 (2007)
13. Jacobson Jr., M.J., Scheidler, R., Stein, A.: Fast arithmetic on hyperelliptic curves via continued fraction expansions. Advances in Coding Theory and Cryptography. Coding Theory Cryptology, vol. 3, pp. 200–243. World Scientific Publishing, Hackensack (2007)
14. Jacobson Jr., M.J., Scheidler, R., Stein, A.: Cryptographic aspects of real hyperelliptic curves. Tatra Mountains Math. Publ. **47**(1), 31–65 (2010)
15. Kedlaya, K.S., Sutherland, A.V.: Computing *L*-series of hyperelliptic curves. In: van der Poorten, A.J., Stein, A. (eds.) ANTS 2008. LNCS, vol. 5011, pp. 312–326. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-79456-1_21
16. Lange, T.: Formula for arithmetic on genus 2 hyperelliptic curves. Appl. Algebra Eng. Commun. Comput. **15**(5), 295–328 (2005)
17. Mireles Morales, D.J.: An analysis of the infrastructure in real function fields. Cryptology ePrint Archive no. 2008/299 (2008)
18. Rezai Rad, M.: A complete evaluation of arithmetic in real hyperelliptic curves. Ph.D. thesis, University of Calgary (2016). https://prism.ucalgary.ca/bitstream/handle/11023/3293/ucalgary_2016_rezairad_monireh.pdf
19. Scheidler, R., Stein, A., Williams, H.C.: Key-exchange in real quadratic congruence function fields. Des. Codes Crypt. **7**(1), 153–174 (1996). Special issue dedicated to Gustavus J. Simmons

20. Schmidt, F.K.: Analytische Zahlentheorie in Körpern der Charakteristik p. Math. Z. **33**(1), 1–32 (1931)
21. Stein, A.: Explicit infrastructure for real quadratic function fields and real hyperelliptic curves. Glasnik Matematički. Serija III **44**(1), 89–126 (2009)
22. Stein, A., Teske, E.: Optimized baby step-giant step methods. J. Ramanujan Math. Soc. **20**(1), 27–58 (2005)
23. Sutherland, A.V.: Fast Jacobian arithmetic for hyperelliptic curves of genus 3. In: ANTS XIII–Proceedings of the Thirteenth Algorithmic Number Theory Symposium. Open Book Series, vol. 2, pp. 425–442. Mathematical Sciences Publishers, Berkeley (2019)
24. Wollinger, T.J., Pelzl, J., Paar, C.: Cantor versus Harley: optimization and analysis of explicit formulae for hyperelliptic curve cryptosystems. IEEE Trans. Comput. **54**(7), 861–872 (2005)