

EFFICIENT HALVING FOR GENUS 3 CURVES OVER BINARY FIELDS

PETER BIRKNER

Department of Mathematics and Computer Science,
Eindhoven University of Technology,
P.O. Box 513, 5600 MB Eindhoven, The Netherlands

NICOLAS THÉRIAULT

Instituto de Matemática y Física, Universidad de Talca, Casilla 747, Talca, Chile

ABSTRACT. In this article, we deal with fast arithmetic in the Picard group of hyperelliptic curves of genus 3 over binary fields. We investigate both the optimal performance curves, where $h(x) = 1$, and the more general curves where the degree of $h(x)$ is 1, 2 or 3. For the optimal performance curves, we provide explicit halving and doubling formulas; not only for the most frequent case but also for all possible special cases that may occur when performing arithmetic on the proposed curves. In this situation, we show that halving offers equivalent performance to that of doubling when computing scalar multiples (by means of an halve-and-add algorithm) in the divisor class group.

For the other types of curves where halving may give performance gains (when the group order is twice an odd number), we give explicit halving formulas which outperform the corresponding doubling formulas by about 10 to 20 field multiplications per halving. These savings more than justify the use of halvings for these curves, making them significantly more efficient than previously thought. For halving on genus 3 curves there is no previous work published so far.

1. INTRODUCTION

Many cryptographic protocols take advantage of the difficulty of the discrete logarithm problem (DLP) to obtain their security. An essential part of these protocols is therefore the computation of scalar multiples of group elements.

The double-and-add algorithm is essential to the efficiency of cryptosystems based on elliptic and hyperelliptic curves. This algorithm (and many of its variations) is based on two basic group operations: the addition of two distinct group elements and the computation of the double of an element. An alternative that proved very successful in case of elliptic curves over binary fields is the halve-and-add algorithm, which relies on the computation of the “half” of a group element (of odd order), i.e. the computation of a pre-image of the doubling operation [14, 17].

2000 *Mathematics Subject Classification*: Primary: 94A60, 14Q05; Secondary: 11G20.

Key words and phrases: hyperelliptic curve, genus 3, divisor class, halving, doubling, binary field, explicit formulas, cryptography.

The authors would like to thank the following organizations for their support: the Fields Institute in Toronto (Canada), FONDECYT (Chile, grant no. 1070242), the Programa Reticulados y Ecuaciones (Universidad de Talca), the Danish Research Council for Technology and Production Sciences (grant no. 274-05-0151) and ECRPYT II.

Given the important savings produced by replacing doublings with halvings for elliptic curves, it is natural to ask if similar results can be obtained for hyperelliptic curves over binary fields.

In this paper, we investigate halving and doubling of divisor classes of hyperelliptic curves of genus 3 over finite fields of characteristic 2. We present complete halving and doubling formulas for many interesting curves. We investigate the optimal-performance case, i.e. we obtain the best operation counts for the explicit doubling and halving formulas. For these curves, we give a complete case study for the most frequent case and for all special cases that can occur when doubling or halving a divisor class. This provides a programmer with everything he needs for a complete implementation of high-speed scalar multiplication. We also treat other interesting cases, i.e. curves whose equation has a different form. Those cases are especially interesting since we gain comparable and sometimes even noticeable better performance for the halving compared to the appropriate doubling formulas. For these cases, we give explicit halving formulas for the most common case (the remaining formulas are available in the appendix of the extended version [7]).

In a normal double-and-add scalar multiplication, all but an almost insignificant proportion of the additions and doublings should fall in the most common cases. One can then implement explicit formulas only for the most common cases, and use Cantor's algorithm when a special case occurs. In practice, this approach does not create any measurable loss in the average performance compared to an implementation that has explicit formulas for all possible cases. The same is not true for the halve-and-add algorithm however, because the inverse operation of Cantor's doubling algorithm cannot easily be written in terms of polynomials. A halve-and-add algorithm must therefore contain explicit formulas for all possible cases of the halving operation.

In this paper, we always work with the Mumford representation of a divisor class and obtain the different cases depending on the degree of the first polynomial of the Mumford representation of the inputs and outputs.

The main results of the present paper are:

- (1) For those genus 3 curves that give the best performance we provide explicit doubling formulas for all special cases, and we thereby extend the formulas which are already published for the most common case only [4, 8, 13].
- (2) In the optimal-performance case, we also provide explicit halving formulas for all possible special cases and therefore allow a complete implementation of DLP-based cryptosystems on genus 3 curves using halving (and doubling) of divisor classes.
- (3) We look at three more general types of genus 3 hyperelliptic curves and provide halving formulas that compare extremely well to the best previously known doubling formulas. It turns out that in those cases halving is always faster. In some situations halving is almost twice as fast as the corresponding doubling operation.

The remainder of this paper is structured as follows: Section 2 contains some important terminology and mathematical background. In this section, we also discuss the arithmetic of the binary field we are working with, especially the computation of square roots and traces. Furthermore, we list the different types of curves that we will treat in this paper, and give the appropriate curve equation for each type. In Section 3, we discuss the optimal-performance case. We give a complete case

study for all possible doubling and halving cases. In Section 4, we look at more general types of curves and give explicit halving formulas for the most common case for each type of curve.

2. BASIC NOTATIONS AND PRELIMINARIES

In this section, we briefly recall the definitions of hyperelliptic curves, divisor class groups and the Mumford representation, since we will use these notions throughout the paper. A comprehensive resource for the mathematics of finite fields is [15]. For background on hyperelliptic curves we refer the interested reader to [2], from which the following definitions and notations are taken.

Definition 2.1 (Hyperelliptic curve). Let K be a field and let \overline{K} be the algebraic closure of K . A curve C , given by an equation of the form

$$C : y^2 + h(x)y = f(x),$$

where $f \in K[x]$ is a monic polynomial of degree $2g+1$ and $h \in K[x]$ is a polynomial of degree at most g , is called an imaginary *hyperelliptic curve of genus g over K* if there is no point (x, y) on the curve over \overline{K} for which both partial derivatives vanish, i.e. such that $2y + h(x) = 0$ and $f(x)' - h(x)'y = 0$.

The last condition ensures that the curve is non-singular.

Definition 2.2 (Divisor class group). Given a hyperelliptic curve C of genus g over a field K , the group of degree 0 divisors of C is denoted by Div_C^0 . The quotient group of Div_C^0 by the group of principal divisors of C is called the *divisor class group of C* and is denoted by Pic_C^0 . It is also called the *Picard group of C* .¹

Theorem 2.3 (Mumford). *Let C be a hyperelliptic curve of genus g over an arbitrary field K . Each nontrivial divisor class of C over K can be represented by a unique pair of polynomials $u, v \in K[x]$, where*

- (1) u is monic,
- (2) $\deg(v) < \deg(u) \leq g$,
- (3) $u \mid v^2 + vh - f$.

Note that the last condition will be essential in establishing some of the halving formulas. A divisor $[u, v]$ that satisfies all the conditions in Theorem 2.3 is called “reduced”. If all the conditions except $\deg(u) < g$ are satisfied, we have a “semi-reduced” divisor (i.e. the pair of polynomials u and v correspond to a divisor, but it is not the reduced representative of its class).

Algorithm 1 Cantor’s doubling algorithm for genus 3 HEC in characteristic 2

INPUT: The divisor class $\overline{D} = [u_a, v_a]$

OUTPUT: The divisor class $[u_c, v_c] = [2]\overline{D}$

- 1: $d \leftarrow \gcd(u_a, h)$, $u_0 \leftarrow u_a d^{-1}$, $v_0 \leftarrow v_a \pmod{u_0}$
- 2: $c \leftarrow h^{-1} \pmod{u_0}$, $u_1 \leftarrow u_0^2$, $v_1 \leftarrow v_0 + c(v_0^2 + v_0 h + f) \pmod{u_1}$
- 3: **if** $\deg(u_1) \leq 3$ **then**

¹In this paper we only consider imaginary hyperelliptic curves, i.e. the divisor class group is isomorphic to the Jacobian (variety) of this curve. Hence we will use the terms Jacobian and divisor class group synonymously.

```

4:    $u_c \leftarrow u_1, v_c \leftarrow v_1$ 
5: else
6:    $u_2 \leftarrow \text{monic} \left( \frac{f+v_1h+v_1^2}{u_1} \right), v_2 \leftarrow v_1 + h \pmod{u_2}$ 
7:   if  $\deg(u_2) \leq 3$  then
8:      $u_c \leftarrow u_2, v_c \leftarrow v_2$ 
9:   else
10:     $u_c \leftarrow \text{monic} \left( \frac{f+v_2h+v_2^2}{u_2} \right), v_c \leftarrow v_2 + h \pmod{u_c}$ 
11:  end if
12: end if
13: return  $[u_c, v_c]$ 

```

Since our goal is to compute pre-images of the group doubling, we refer to Algorithm 1 for a description of how this operation is performed using Cantor’s algorithm. Our proposed halving and doubling formulas expect the input divisor class to be in Mumford representation and work directly on the coefficients of the polynomials u_a and v_c of this representation. The resulting divisor class is also given in the Mumford form.

2.1. CHOICE OF THE FIELD AND DIVISOR CLASS HALVING. Throughout this paper, we will assume that the field is of the form \mathbb{F}_q , where $q = 2^n$ and n is not divisible by 2 or 3. This is mainly due to security concerns, since various versions of the Weil descent attack could be applied when n admits a factor of 2 or 3 (for example, see [10, 11, 18]). In fact, for cryptographic applications it is often assumed that n is a prime. As an added bonus, having n coprime to 6 means that we can take cube, fifth and seventh roots in the field (since the mappings $\alpha \mapsto \alpha^3$, $\alpha \mapsto \alpha^5$, and $\alpha \mapsto \alpha^7$ are all isomorphisms as 3, 5 and 7 are coprime to $2^n - 1$), which allows us to simplify the curve equations a little more.

In finite fields of characteristic 2, some operations that are computationally hard in fields of odd characteristic become efficient, in particular the computation of the square root of a field element. This observation led to the development of *halve-and-add* algorithms, a variation of the double-and-add scalar multiplication where the doubling operation is replaced with a *halving* (the representation of the scalar is adjusted accordingly). Such an approach was first used for elliptic curves [14, 17], and was recently extended to hyperelliptic curves of genus 2 (see [5, 6, 12]). In fact, some fields have the property that the computation of square roots can be faster than the computation of squares [1, 9]. It can therefore become a good strategy to “replace” squares with square roots for curve arithmetic in these fields, and this is exactly what our halving formulas do. Furthermore, since n will be odd we will have $\text{TR}(1) = 1$. In various places, we implicitly take advantage of the identity $\text{TR}(\alpha) = \text{TR}(\alpha^2)$ to simplify some trace computations.

To count the number of operations, we denote inverses by I, multiplications by M, squares by S, square roots by SR, traces by TR and half-traces by HT.

2.2. CONDITIONS ON THE ORDER OF THE PICARD GROUP. We limit ourselves to curves for which the order of the Jacobian is either odd (h constant) or 2 times an odd number. This restriction is needed to get a better performance out of the

halving. Given any hyperelliptic curve, the halve-and-add algorithm allows us to compute the scalar multiple of a divisor class, given that it is in a (sub)group of odd order. In this way, the pre-image of the doubling can always be computed and “becomes” unique (all other pre-images of the doubling have even order). The group order conditions are due to the following reasons:

- (1) To verify that the pre-image is in the subgroup of odd order, we make sure that it can be halved again as many times as we want. If the group contains divisors of order 2^s , we must ensure that we can halve the pre-image (at least) s times, which obviously affects the cost of our halving formulas. When $s \geq 2$ (i.e. when there are divisors of order 4), the increased work required for this check becomes too expensive for the halving to be interesting.
- (2) The number of pre-images of the halving depends directly on the number of divisors of order 2 in the group, which in turn depends on the factorization of $h(x)$. If $h(x)$ has r distinct irreducible factors (multiplicities do not have an impact here), then we have 2^r distinct pre-images of the doubling. Since we must identify the unique pre-image of odd order, having $r > 1$ would force us to choose between four or more reduced divisors, which increases the algorithmic cost of halving significantly. We will therefore require r to be at most 1.

Note that if $h(x)$ has r distinct irreducible factors, then the group order is divisible by (at least) 2^r , so asking the group order to be either odd or 2 times an odd number removes all curves for which $h(x)$ has 2 or 3 distinct irreducible factors.

2.3. TYPES OF CURVES. We can distinguish the genus 3 hyperelliptic curves in characteristic 2 according to the degree of $h(x)$ and the form of its factorization over \mathbb{F}_{2^n} . We find the following types:

- Type Ia: $h(x)$ is irreducible of degree 3.
- Type Ib: $h(x)$ has degree 3 and is the product of an irreducible polynomial of degree 2 and a linear factor ($r = 2$).
- Type Ic: $h(x)$ has degree 3 and is the product of 3 distinct linear factors ($r = 3$).
- Type Id: $h(x)$ has degree 3 and is the product of 2 distinct linear factors, one of which is repeated twice ($r = 2$).
- Type Ie: $h(x)$ is the cube of a linear factor (degree 3, $r = 1$).
- Type IIa: $h(x)$ is irreducible of degree 2.
- Type IIb: $h(x)$ has degree 2 and is the product of 2 distinct linear factors ($r = 2$).
- Type IIc: $h(x)$ is the square of a linear factor (degree 2, $r = 1$).
- Type III: $h(x)$ is linear (degree 1).
- Type IV: $h(x)$ is constant (degree 0).

For each type of curve, we can use curve isomorphisms to “simplify” the equation of the curve. This will be handled in the next subsection.

Due to our condition on the group order, we will limit ourselves to curves of Types Ia, Ie, IIa, IIc, III and IV. Because of the structure of their 2-torsion group, curves of Types Ie and IIc have very similar properties (and essentially the same number of isomorphism classes) as curves of Type III. On the other hand, the higher degree of $h(x)$ in Type Ie and IIc makes them less efficient than curves of Type III, so the formulas for these two types of curves are presented only in the appendix of the extended version of this paper [7].

2.4. FORMS OF THE CURVE EQUATIONS. An imaginary hyperelliptic curve of genus 3 over \mathbb{F}_{2^n} is of the form

$$(1) \quad y^2 + h(x)y = f(x),$$

where $h(x) = h_3x^3 + h_2x^2 + h_1x + h_0 \neq 0$ and $f(x) = f_7x^7 + f_6x^6 + f_5x^5 + f_4x^4 + f_3x^3 + f_2x^2 + f_1x + f_0$. It is also customary to use isomorphisms to impose that f be monic, i.e. that $f_7 = 1$, but we will relax this condition for some curves types as the halving formulas are more efficient if we use isomorphisms to force a specific coefficient of $h(x)$ to be 1 (which one of the coefficients depends on the curve type). The effects on the addition formula are described in the appendix [7]. Because of the ratio between the number of additions and halvings in the scalar multiplication, the small increase this produces in the addition cost (3 multiplications) becomes interesting as soon as we can save one or more multiplications in the halving.

Since the coefficients of the curve equation (the coefficients of h and f) have a direct impact on the computations in Cantor's algorithm, it is quite natural to use isomorphisms to obtain an equivalent curve with "simpler" coefficients (i.e. getting coefficients equal to 0, restricting them to \mathbb{F}_2 etc.). For the curve (1), the possible isomorphisms are given by $x \mapsto \alpha x + \beta$ and $y \mapsto \gamma y + \delta x^3 + \varepsilon x^2 + \varrho x + \zeta$, where both α and γ are nonzero. After applying the isomorphisms, the equation is divided by γ^2 to make it monic.

Proposition 1. *Given an isomorphism that replaces f_{2i} by $f_{2i} + \omega^2 + \omega$, we can restrict f_{2i} to $\text{TR}(f_{2i}) \in \mathbb{F}_2$.*

Proof. Since $f_{2i} + \text{TR}(f_{2i})$ has trace 0, we can choose ω such that $\omega^2 + \omega = f_{2i} + \text{TR}(f_{2i})$. This choice of ω replaces f_{2i} with $\text{TR}(f_{2i})$. Note that the isomorphism does not permit us to change the trace of f_{2i} . \square

For the six types of curves where halving is interesting, we have:

(Ia) $h_3 \neq 0$ and $h(x)$ irreducible: We first use $\beta = h_2/h_3$ to remove h_2 . Once $h_2 = 0$, h_1 must be non-zero (otherwise $h(x)$ would not be irreducible), so we can set $\alpha = \sqrt{h_1/h_3}$ and $\gamma = \sqrt{h_1^3/h_3}$ to obtain $h(x) = x^3 + x + h_0$.

We can then use δ to restrict f_6 to \mathbb{F}_2 , then ε to force $f_5 = 0$, ϱ to force $f_4 = 0$ and finally ζ to have $f_3 = 0$. We are left with $f(x) = f_7x^7 + f_6x^6 + f_2x^2 + f_1x + f_0$ where $f_6 \in \mathbb{F}_2$.

(Ie) $h_3 \neq 0$ and $h(x)$ is a cube: Taking $\alpha = h_2^2/f_7$ and $\gamma = h_3^7/f_7^3$ we can force both $h(x)$ and $f(x)$ to be monic. Once $h_3 = f_7 = 1$, we can use $\beta = h_2$ to obtain $h(x) = x^3$ (since $h(x)$ is a cube).

We can then use δ to restrict f_6 to \mathbb{F}_2 , then ε to force $f_5 = 0$, ϱ to force $f_4 = 0$ and finally ζ to have $f_3 = 0$. We are left with $f(x) = x^7 + f_6x^6 + f_2x^2 + f_1x + f_0$ where $f_6 \in \mathbb{F}_2$.

(IIa) $h_3 = 0$, $h_2 \neq 0$ and $h(x)$ is irreducible: Using $\alpha = h_1/h_2$ and $\gamma = h_1^2/h_2$ we can force $h_2 = h_1 = 1$. Since $h(x)$ is irreducible, we must then have $\text{TR}(h_1) = 1$ and we can then use $\beta = \text{HT}(h_1 + 1)$ to restrict $h(x)$ to $x^2 + x + 1$.

Combining the remaining freedom on β (i.e. $\beta \in \mathbb{F}_2$) and δ , we can force $f_6 = 0$ and restrict the number of possibilities for f_5 by a factor of 2 (in general). Note that this restriction on f_5 has no impact on $\text{TR}(f_5)$.

We can then use ε to restrict f_4 to \mathbb{F}_2 , then ϱ to force $f_3 = 0$ and finally ζ to have $f_2 = 0$. We are left with $f(x) = f_7x^7 + f_5x^5 + f_4x^4 + f_1x + f_0$ where $f_4 \in \mathbb{F}_2$ and $\text{TR}(f_7) \cdot \text{TR}(f_5) = 0$.

(IIc) $h_3 = 0$, $h_2 \neq 0$ and $h(x)$ is a square: Taking $\alpha = (h_2^2/f_7)^{1/3}$ and $\gamma = (h_2^7/f_7^2)^{1/3}$ we can force both $h(x)$ and $f(x)$ to be monic. Once $h_2 = f_7 = 1$, we can use $\beta = \sqrt{h_0}$ to obtain $h(x) = x^2$ (since $h(x)$ is a square).

We can then use δ to force $f_6 = 0$, then ε to restrict f_4 to \mathbb{F}_2 , then ϱ to force $f_3 = 0$ and finally ζ to have $f_2 = 0$. We are left with $f(x) = x^7 + f_5x^5 + f_4x^4 + f_1x + f_0$ where $f_4 \in \mathbb{F}_2$.

(III) $h_3 = h_2 = 0$ and $h_1 \neq 1$: Taking $\alpha = (h_1^2/f_7)^{1/5}$ and $\gamma = (h_1^7/f_7)^{1/5}$ we can force both $h(x)$ and $f(x)$ to be monic. Once $h_1 = f_7 = 1$, we can use $\beta = h_0$ to obtain $h(x) = x$.

We can then use δ to force $f_6 = 0$, then ε to force $f_4 = 0$, ϱ to restrict f_2 to \mathbb{F}_2 and finally ζ to have $f_1 = 0$. We are left with $f(x) = x^7 + f_5x^5 + f_3x^3 + f_2x^2 + f_0$ where $f_2 \in \mathbb{F}_2$.

(IV) $h_3 = h_2 = h_1 = 0$ and $h_0 \neq 0$: Taking $\alpha = (h_0^2/f_7)^{1/7}$ and $\gamma = h_0$ we can have $h(x) = 1$ and force $f(x)$ to be monic.

Once $f_7 = 1$, we can use $\beta = \sqrt{f_5}$ to remove the term in x^5 from $f(x)$. We can then use δ to force $f_6 = 0$, then ε to force $f_4 = 0$, ϱ to force $f_2 = 0$ and finally ζ to restrict f_0 to \mathbb{F}_2 . We are left with $f(x) = x^7 + f_3x^3 + f_1x + f_0$ where $f_0 \in \mathbb{F}_2$.

Note that we did not include the non-singularity condition, nor conditions on the group order in the descriptions of the different types. In terms of isomorphism classes, Type Ia is the most common (with $\frac{2}{3}q^5 + O(q^4)$ classes), followed by Type IIa (with $q^4 + O(q^3)$ classes), then Types III, IIc, and Ie (each with $2q^3 + O(q^2)$ classes) and finally Type IV (with $2q^2 + O(q)$ classes).

3. TYPE IV: $h(x) = 1$

In this section, we consider the high-performance curves, i.e. those that are preferred when computational speed is more important than flexibility in the choice of the curve (even then, there are enough isomorphism classes available for most applications). From the results of the previous section, we can assume that curves of Type IV are of the form

$$(2) \quad C : y^2 + y = x^7 + f_3x^3 + f_1x + f_0,$$

with $f_0 \in \mathbb{F}_2$. As well as having all but two of the coefficients of the curve equation in \mathbb{F}_2 (and many of those being 0), these curves offer other advantages:

- (1) The doubling is significantly faster than for other types of curves, and also much faster than the group addition.
- (2) The curve C is not supersingular (see Theorem 1.2 in [Scholten, Zhu] with $n = 3$). This is an important advantage over the genus 1 and 2 situation where curves with $h = c$ are supersingular if c is a constant. Hence, our genus 3 curves are secure against specialized attacks as long as the order of the Jacobian is divisible by a large prime.
- (3) Since $h = 1$, the 2-rank of the curve C is 0: Given any non-zero element $\overline{D} = [u, v]$ of the Jacobian in Mumford representation, its negative is

$$-\overline{D} = [u, -v - h \pmod{u}] = [u, v + 1].$$

Since $v \not\equiv v + 1 \pmod{u}$ for all $\overline{D} \neq [1, 0]$, we have $\overline{D} \neq -\overline{D}$ and therefore $[2]\overline{D} \neq [1, 0]$ for all non-zero \overline{D} in the Jacobian of the curve, which shows that there are no non-trivial 2-torsion points (and the group order is odd).

This simple fact is extremely useful for the halving formulas. It means that the doubling map is a one-to-one function, rather than a two-to-one as it is the case for elliptic curves (and will be the case for the other curves considered in this paper). The pre-image of the doubling will be unique, removing the need for a potentially expensive verification step to find which of the pre-images has odd order.

We will now give explicit formulas for the doubling and the halving of divisor classes, and cover both the most frequent case and all other possible special cases. Combined with divisor addition formulas [2, 4, 13], this allows to program the most efficient implementation of genus 3 hyperelliptic curve group arithmetic.

3.1. EXPLICIT DOUBLING FORMULAS. In the following, we give a complete study of all cases that can occur when performing doubling of a divisor class on a genus 3 hyperelliptic curve of Type IV, i.e. we assume that we are given a curve of the form (2) over a binary field. We consider the different cases by looking at the degree of the polynomial u_a in the Mumford representation of a given divisor class $\overline{D}_a = [u_a, v_a]$, where u_a is monic of degree at most 3 and v_a is of smaller degree than u_a and such that u_a divides $v_a^2 + v_a + f$.

We will give criteria to detect which case is present, depending on the coefficients of the polynomials u_a and v_a . Therefore, we follow the steps of Cantor's algorithm to see how the degrees of the polynomials behave during the doubling. The details of Cantor's algorithm for genus 3 curves of type IV are given in Algorithm 2.

Algorithm 2 Cantor's doubling algorithm for genus 3 HEC in characteristic 2 with $h(x) = 1$ (Type IV)

INPUT: The divisor class $\overline{D} = [u_a, v_a]$

OUTPUT: The divisor class $[u_c, v_c] = [2]\overline{D}$

```

1:  $u_1 \leftarrow u_a^2, v_1 \leftarrow v_a^2 + f \pmod{u_1}$ 
2: if  $\deg(u_1) \leq 3$  then
3:    $u_c \leftarrow u_1, v_c \leftarrow v_1$ 
4: else
5:    $u_2 \leftarrow \text{monic}\left(\frac{f+v_1+v_1^2}{u_1}\right), v_2 \leftarrow v_1 + 1 \pmod{u_2}$ 
6:   if  $\deg(u_2) \leq 3$  then
7:      $u_c \leftarrow u_2, v_c \leftarrow v_2$ 
8:   else
9:      $u_c \leftarrow \text{monic}\left(\frac{f+v_2+v_2^2}{u_2}\right), v_c \leftarrow v_2 + 1 \pmod{u_c}$ 
10:  end if
11: end if
12: return  $[u_c, v_c]$ 

```

Note that from now on we will use the following notation: “Doubling $n \rightarrow m$ ” (short: $\text{DBL}nm$) stands for a doubling where the degree of the the first polynomial of the divisor class to be doubled is n and the degree of the first polynomial of the

target divisor class (in Mumford representation) is m . We will use the same syntax for halving (short: $HLVnm$).

3.1.1. *Distinguishing the cases.* To distinguish the different doubling cases, we start with a divisor class $\overline{D}_a = [u_a, v_a]$, and depending on the degree of u_a we see what degree can be taken by u_c (in $\overline{D}_c = [u_c, v_c] = [2]\overline{D}_a$).

If u_a has degree 3, then the first step in Algorithm 2 computes $u_1 = u_a^2$ and $v_1 \equiv v_a^2 + f \pmod{u_1}$. We obtain

$$u_1 = u_a^2 = x^6 + u_{a2}^2 x^4 + u_{a1}^2 x^2 + u_{a0}^2$$

and

$$\begin{aligned} v_1 &= v_a^2 + f \pmod{u_1} \\ (3) \quad &= u_{a2}^2 x^5 + v_{a2}^2 x^4 + (u_{a1}^2 + f_3)x^3 + v_{a1}^2 x^2 \\ &\quad + (u_{a0}^2 + f_1)x + (f_0 + v_{a0}^2). \end{aligned}$$

Since u_1 has degree 6, we must do at least one reduction step, so we compute

$$u_2 = \text{monic} \left(\frac{f + v_1 + v_1^2}{u_1} \right).$$

We now have different possibilities for the degree of u_2 depending on the degree of v_1 . Since $\deg(u_1) = 6$, the degree of v_1 is less than or equal to 5. We have the following three cases:

- (1) When $\deg(v_1)$ is equal to 1, 2 or 3, the dominating part of the numerator comes from f . The degree of u_2 is then $\deg(u_2) = \deg(f) - \deg(u_1) = 1$. Cantor's algorithm will then output $u_c \leftarrow u_2$ of degree 1. This case will be handled in Subsection 3.1.4.
- (2) When $\deg(v_1) = 4$, the numerator is dominated by v_1^2 . The degree of u_2 is then $\deg(u_2) = \deg(v_1^2) - \deg(u_1) = 2$. Cantor's algorithm outputs $u_c \leftarrow u_2$ of degree 2, which will be handled in Subsection 3.1.3.
- (3) When $\deg(v_1) = 5$, the numerator is again dominated by v_1^2 , but this time we have $\deg(u_2) = 4$. Note that we also have $\deg(v_2) \leq 3$. Cantor's algorithm will then proceed with a second reduction step, computing u_c as

$$u_c = \text{monic} \left(\frac{f + v_2 + v_2^2}{u_2} \right).$$

The numerator is once again dominated by f , and u_c has degree $\deg(f) - \deg(u_2) = 3$. This case will be handled in Subsection 3.1.2.

If u_a has degree 2, then $\deg(u_1) = 4$ and $\deg(v_1) \leq 3$. We must then do one reduction step, with

$$u_2 = \text{monic} \left(\frac{f + v_1 + v_1^2}{u_1} \right),$$

where the numerator is dominated by f . The degree of u_2 is then $\deg(u_2) = \deg(f) - \deg(u_1) = 3$ and Cantor's algorithm will then output $u_c \leftarrow u_2$ of degree 3. This case will be handled in Subsection 3.1.5.

Finally, if u_a has degree 1, then $\deg(u_1) = 2$ and $\deg(v_1) \leq 1$ and Cantor's algorithm outputs $u_c = u_1$ and $v_c = v_1$. This case will be handled in Subsection 3.1.6.

3.1.2. *Doubling 3 → 3.* This is in fact the most common case of doubling (which occurs with probability $1 - O(\frac{1}{q})$). From the previous section, we know this will happen when $\deg(v_1) = 5$, which means $u_{a2}^2 \neq 0$, i.e. when $\deg(u_a) = 3$ and $u_{a2} \neq 0$. We can now state the actual formula to double in the $3 \rightarrow 3$ case. This formula is taken from [8, Table XXVI], although we adapted the notation to the one used in this paper.

Algorithm 3 (DBL33, $h(x) = 1$, $f(x) = x^7 + f_3x^3 + f_1x + f_0$)

INPUT: $\overline{D} = [x^3 + u_{a2}x^2 + u_{a1}x + u_{a0}, v_{a2}x^2 + v_{a1}x + v_{a0}]$, $u_{a2} \neq 0$

OUTPUT: $[2]\overline{D} = [x^3 + u_{c2}x^2 + u_{c1}x + u_{c0}, v_{c2}x^2 + v_{c1}x + v_{c0}]$

1: $s_0 \leftarrow u_{a2}^2, s_1 \leftarrow u_{a1}^2, s_2 \leftarrow v_{a2}^2, s_3 \leftarrow f_3 + s_1, s_4 \leftarrow s_0^{-1}$	▷ 1I+3S
2: $s_5 \leftarrow s_4s_2, s_6 \leftarrow s_4s_3, u_{c1} \leftarrow s_5^2 + s_0, s_7 \leftarrow s_0u_{c1}$	▷ 3M+1S
3: $s_8 \leftarrow s_6^2 + s_1 + s_7, s_9 \leftarrow s_7 + s_3, s_{10} \leftarrow s_2u_{c1}, s_{11} \leftarrow s_2s_8$	▷ 2M+1S
4: $s_{12} \leftarrow s_{11} + f_0 + v_{a0}^2, s_{13} \leftarrow s_{10} + v_{a1}^2 + s_4, s_{14} \leftarrow s_4^2$	▷ 3S
5: $s_{15} \leftarrow (s_0 + s_2)(s_8 + s_{14}) + s_{11} + f_1 + u_{a0}^2 + s_4, u_{c2} \leftarrow s_9^2$	▷ 1M+2S
6: $u_{c0} \leftarrow u_{c2}u_{c1} + s_{13}^2 + s_{14}, s_{16} \leftarrow s_9u_{c0}, s_{17} \leftarrow s_9u_{c1}$	▷ 3M+1S
7: $s_{18} \leftarrow s_9u_{c2}, v_{c2} \leftarrow s_{13} + s_{18}, v_{c1} \leftarrow s_{15} + s_{17}, v_{c0} \leftarrow s_{12} + s_{16}$	▷ 1M
8: return $[x^3 + u_{c2}x^2 + u_{c1}x + u_{c0}, v_{c2}x^2 + v_{c1}x + v_{c0}]$	▷ 1I+10M+11S

3.1.3. *Doubling 3 → 2.* From Subsection 3.1.1 (2), we know this case occurs when $\deg(v_1) = 4$, and from Equation 3 this happens if and only if $\deg(u_a) = 3$, $u_{a2} = 0$, and $v_{a2} \neq 0$.

Algorithm 4 (DBL32, $h(x) = 1$, $f(x) = x^7 + f_3x^3 + f_1x + f_0$)

INPUT: $\overline{D} = [x^3 + u_{a1}x + u_{a0}, v_{a2}x^2 + v_{a1}x + v_{a0}]$, $v_{a2} \neq 0$

OUTPUT: $[2]\overline{D} = [x^2 + u_{c1}x + u_{c0}, v_{c1}x + v_{c0}]$

1: $s_0 \leftarrow v_{a2}^2, s_1 \leftarrow s_0^{-1}, u_{c1} \leftarrow s_1^2, s_2 \leftarrow f_3 + u_{a1}^2, s_3 \leftarrow s_2^2$	▷ 1I+4S
2: $u_{c0} \leftarrow u_{c1}s_3, s_4 \leftarrow s_1 + s_2, s_5 \leftarrow s_4u_{c0}$	▷ 2M
3: $s_6 \leftarrow v_{a1}^2 + (s_0 + s_4)(u_{c0} + u_{c1}) + s_5 + s_1, s_7 \leftarrow s_6u_{c0}$	▷ 2M+1S
4: $s_8 \leftarrow s_6u_{c1}, v_{c1} \leftarrow f_1 + u_{a0}^2 + s_5 + s_8, v_{c0} \leftarrow f_0 + 1 + v_{a0}^2 + s_7$	▷ 1M+2S
5: return $[x^2 + u_{c1}x + u_{c0}, v_{c1}x + v_{c0}]$	▷ 1I+5M+7S

3.1.4. *Doubling 3 → 1.* From Subsection 3.1.1 (1), we see that this case occurs when $\deg(v_1)$ is less or equal than 3, and from Equation 3 this happens when $\deg(u_a) = 3$, $u_{a2} = 0$ and $v_{a2} = 0$.

Since any divisor must satisfy $u_a \mid v_a^2 + hv_a - f$, it is easy to show that we also have $v_{a1} = 0$. We can therefore assume that the input divisor class has the form $[x^3 + u_{a1}x + u_{a0}, v_{a0}]$ and the output divisor is of the form $[x + u_{c0}, v_{c0}]$.

Algorithm 5 (DBL31, $h(x) = 1$, $f(x) = x^7 + f_3x^3 + f_1x + f_0$)

INPUT: $\overline{D} = [x^3 + u_{a1}x + u_{a0}, v_{a0}]$

OUTPUT: $[2]\overline{D} = [x + u_{c0}, v_{c0}]$

- 1: $s_0 \leftarrow u_{a1}^2, s_1 \leftarrow f_3^2, s_2 \leftarrow s_0^2, u_{c0} \leftarrow s_1 + s_2, s_4 \leftarrow u_{c0}^2$ ▷ 3S
 - 2: $v_{c0} \leftarrow u_{c0}((s_0 + f_3)s_4 + (u_{a0}^2 + f_1)) + v_{a0}^2 + f_0 + 1$ ▷ 2M+2S
 - 3: **return** $[x + u_{c0}, v_{c0}]$ ▷ 2M+5S
-

3.1.5. *Doubling 2* \rightarrow 3. As stated in Section 3.1.1, this is the only case that can occur when $\deg(u_a) = 2$. The first step of Cantor's algorithm gives us $u_1 = u_a^2 = x^4 + u_{a1}^2x^2 + u_{a0}^2$ and

$$v_1 = (f_3 + u_{a0}^2 + u_{a1}^4)x^3 + v_{a1}^2x^2 + (u_{a1}^2u_{a0}^2 + f_1)x + (f_0v_{a0}^2),$$

after which one reduction step is performed to obtain u_c and v_c . The formula is as follows:

Algorithm 6 (DBL23, $h(x) = 1$, $f(x) = x^7 + f_3x^3 + f_1x + f_0$)

INPUT: $\overline{D} = [x^2 + u_{a1}x + u_{a0}, v_{a1}x + v_{a0}]$

OUTPUT: $[2]\overline{D} = [x^3 + u_{c2}x^2 + u_{c1}x + u_{c0}, v_{c2}x^2 + v_{c1}x + v_{c0}]$

- 1: $u_{c1} \leftarrow u_{a1}^2, s_0 \leftarrow f_3 + u_{c1}^2, s_1 \leftarrow s_0 + u_{a0}^2, u_{c2} \leftarrow s_1^2, s_2 \leftarrow v_{a1}^2$ ▷ 5S
 - 2: $s_3 \leftarrow s_2 + s_1u_{a1}, u_{c0} \leftarrow s_3^2, s_4 \leftarrow s_1u_{c0}, s_5 \leftarrow s_1u_{c2}$ ▷ 3M+1S
 - 3: $v_{c2} \leftarrow s_2 + s_5, v_{c1} \leftarrow f_1 + s_0u_{c1}, v_{c0} \leftarrow f_0 + v_{a0}^2 + 1 + s_4$ ▷ 1M+1S
 - 4: **return** $[x^3 + u_{c2}x^2 + u_{c1}x + u_{c0}, v_{c2}x^2 + v_{c1}x + v_{c0}]$ ▷ 4M+7S
-

3.1.6. *Doubling 1* \rightarrow 2. This is the last case which can occur when performing a doubling. Since $\deg(u_a^2) = 2 < 3$, only the first step of Cantor's algorithm is necessary, and we obtain $u_c = u_1 = u_a^2 = x^2 + u_{a0}^2$ and

$$v_c = v_1 = (f_1 + u_{a0}^6 + f_3u_{a0}^2)x + (v_{a0}^2 + f_0).$$

We get the very short formula:

Algorithm 7 (DBL12, $h(x) = 1$, $f(x) = x^7 + f_3x^3 + f_1x + f_0$)

INPUT: $\overline{D} = [x + u_{a0}, v_{a0}]$

OUTPUT: $[2]\overline{D} = [x^2 + u_{c0}, v_{c1}x + v_{c0}]$

- 1: $u_{c0} \leftarrow u_{a0}^2, v_{c0} \leftarrow f_0 + v_{a0}^2, v_{c1} \leftarrow f_1 + u_{c0}(f_3 + u_{c0}^2)$ ▷ 1M+3S
 - 2: **return** $[x^2 + u_{c0}, v_{c1}x + v_{c0}]$ ▷ 1M+3S
-

3.2. EXPLICIT HALVING FORMULAS. Having developed formulas for all the possible cases of doubling of divisor classes, we can now look at halving these same classes of our proposed genus 3 curves over binary fields. Our general approach will consist in inverting (or “backtracking”) each one of the doubling cases to obtain the halving formulas. We will therefore have five cases of halvings:

- Halving $3 \rightarrow 3$ (from the doubling $3 \rightarrow 3$);
- Halving $2 \rightarrow 3$ (from the doubling $3 \rightarrow 2$);
- Halving $1 \rightarrow 3$ (from the doubling $3 \rightarrow 1$);
- Halving $3 \rightarrow 2$ (from the doubling $2 \rightarrow 3$);
- Halving $2 \rightarrow 1$ (from the doubling $1 \rightarrow 2$).

Before going into the specifics of each formula, let us consider how to distinguish between the different cases. Let us consider the halving of a divisor $[u_c, v_c]$ known to come from the doubling of a divisor $[u_a, v_a]$:

- If $\deg(u_c) = 1$, then we can only be in the $1 \rightarrow 3$ case.
- If $\deg(u_c) = 2$, then $\deg(u_a)$ was either 1 (doubling $1 \rightarrow 2$)—in which case $u_c(x)$ is of the form $x^2 + u_{c0}$ —or 3 (doubling $3 \rightarrow 2$). To have a simple distinguishing condition, we would like to say that if $[u_c, v_c]$ comes from a doubling $3 \rightarrow 2$ then $u_c(x)$ is of the form $x^2 + u_{c1}x + u_{c0}$ with $u_{c1} \neq 0$, and indeed, an easy computation from the doubling formula shows that $u_{c1} = 1/v_{a2}^4$ where $v_{a2} \neq 0$ as we are coming from the $3 \rightarrow 2$ doubling case.
- If $\deg(u_c) = 3$, then $\deg(u_a)$ was either 2 (doubling $2 \rightarrow 3$) or 3 (doubling $3 \rightarrow 3$). There is no direct way to distinguish between these two cases simply by looking at the form of u_c and v_c . However, the doubling formulas do present us with a natural candidate when we notice that the $3 \rightarrow 3$ doubling contains an inversion while the $2 \rightarrow 3$ doubling does not. Not surprisingly, the same situation happens in the halving formulas. If we assume that $[u_c, v_c]$ is in the halving $3 \rightarrow 3$ case and try to work backward through the $3 \rightarrow 3$ doubling, we need to compute the inverse of $u_{c0} + v_{c1}^2 + u_{c2}(u_{c1} + u_{c2}^2)$ (or its square root), so the operation cannot be valid if this value is 0 (i.e. it must be $\neq 0$). On the other hand, if we take the result of a $2 \rightarrow 3$ doubling and substitute the values of the u_{c0}, u_{c1}, u_{c2} and v_{c1} (in terms of the coefficients of u_a and v_a) in the expression $u_{c0} + v_{c1}^2 + u_{c2}(u_{c1} + u_{c2}^2)$, then we can verify that it must always be 0. We can therefore use the value of $u_{c0} + v_{c1}^2 + u_{c2}(u_{c1} + u_{c2}^2)$ to safely distinguish between the two cases.

Now that the different cases can be identified, we can look at the formulas. To have a more “standard” look, they are written with input $[u_a, v_a]$ and output $[u_c, v_c] = [\frac{1}{2}][u_a, v_a]$, so the condition to distinguish between the cases are u_{a1} equal or not to 0 when $\deg(u_a) = 2$ and $u_{a0} + v_{a1}^2 + u_{a2}(u_{a1} + u_{a2}^2)$ equal or not to 0 when $\deg(u_a) = 3$.

3.2.1. *Halving* $3 \rightarrow 3$. We can now optimize the halving $3 \rightarrow 3$. In general, we cannot distinguish this case from the halving $3 \rightarrow 2$ until $s_4 = u_{a0} + v_{a1}^2 + u_{a2}(u_{a1} + u_{a2}^2)$ has been computed. If $s_4 = 0$, we must change to the same line of Algorithm 11. Note that both $u_{a0}\sqrt{u_{a2}}$ and $u_{a2}\sqrt{u_{a2}}$ are needed in both the $3 \rightarrow 3$ and $3 \rightarrow 2$ halvings, so they can be computed before we distinguish the two cases.

Algorithm 8 (HLV33, $h(x) = 1$, $f(x) = x^7 + f_3x^3 + f_1x + f_0$)

INPUT: $\overline{D} = [x^3 + u_{a2}x^2 + u_{a1}x + u_{a0}, v_{a2}x^2 + v_{a1}x + v_{a0}]$

OUTPUT: $[1/2]\overline{D} = [x^3 + u_{c2}x^2 + u_{c1}x + u_{c0}, v_{c2}x^2 + v_{c1}x + v_{c0}]$

1: $s_0 \leftarrow \sqrt{u_{a2}}, s_1 \leftarrow u_{a2}s_0, s_2 \leftarrow u_{a0}s_0, s_3 \leftarrow v_{a2} + s_1$ ▷ 2M+1SR
2: $s_4 \leftarrow u_{a0} + s_3^2 + u_{a1}u_{a2}, s_5 \leftarrow \sqrt{s_4}, s_6 \leftarrow s_5^{-1}$ ▷ 1I+1M+1S+1SR
3: $s_7 \leftarrow u_{a1}s_6, s_8 \leftarrow s_0 + s_7, s_9 \leftarrow s_6\sqrt{s_6 + u_{a1}}$ ▷ 2M+1SR
4: $s_{10} \leftarrow s_3 + u_{a1}s_9 + s_5, s_{11} \leftarrow s_8 + f_3, s_{12} \leftarrow s_{11} + s_4s_8^2 + s_7$ ▷ 2M+1S
5: $s_{13} \leftarrow s_{12}s_9, s_{14} \leftarrow v_{a1} + u_{a1}s_0 + (s_4 + s_{12})(s_9 + s_6) + s_{13} + s_5$ ▷ 3M
6: $s_{15} \leftarrow v_{a0} + s_2 + s_{13}, v_{c2} \leftarrow \sqrt{s_9}, v_{c1} \leftarrow \sqrt{s_{10}}$ ▷ 2SR
7: $v_{c0} \leftarrow \sqrt{s_{15} + f_0}, u_{c2} \leftarrow \sqrt{s_6}, u_{c1} \leftarrow \sqrt{s_{11}}, u_{c0} \leftarrow \sqrt{s_{14} + f_1}$ ▷ 4SR
8: **return** $[x^2 + u_{c0}, v_{c1}x + v_{c0}]$ ▷ 1I+10M+2S+9SR

3.2.2. *Halving* $2 \rightarrow 3$. Since this case of the halving is the inverse of a $3 \rightarrow 2$ doubling, we know that the output must be of the form $[x^3 + u_{c1}x + u_{c0}, v_{c2}x^2 + v_{c1}x + v_{c0}]$. However, the output has one more coefficient than the input, and it is not enough to simply reverse the doubling formula—doing so would leave us with q possible choices for the output, which is clearly impossible as the halving operation is injective.

To solve this problem, we must recall the last condition in Theorem 2.3, i.e. that u_c must divide $v_c^2 + v_c + f$ if $[u_c, v_c]$ is a divisor. Computing the coefficient of x^2 in

$$(v_{c2}x^2 + v_{c1}x + v_{c0})^2 + (v_{c2}x^2 + v_{c1}x + v_{c0}) + f \pmod{x^3 + u_{c1}x + u_{c0}},$$

we find that $v_{c2} + v_{c1}^2 + u_{c1}v_{c2}^2$ must be 0 (since the whole equation must equal 0), giving us the relation $v_{c1} = \sqrt{v_{c2} + u_{c1}v_{c2}^2}$ which allows us to complete the formula.

Algorithm 9 (HLV23, $h(x) = 1$, $f(x) = x^7 + f_3x^3 + f_1x + f_0$)

INPUT: $\overline{D} = [x^2 + u_{a1}x + u_{a0}, v_{a1}x + v_{a0}]$

OUTPUT: $[1/2]\overline{D} = [x^3 + u_{c1}x + u_{c0}, v_{c2}x^2 + v_{c1}x + v_{c0}]$

1: $s_0 \leftarrow \sqrt{u_{a1}}, s_1 \leftarrow s_0^{-1}, v_{c2} \leftarrow \sqrt{s_1}, s_2 \leftarrow \sqrt{u_{a0}}, s_3 \leftarrow s_1s_2$ ▷ 1I+1M+3SR
2: $s_6 \leftarrow s_1u_{a0}, u_{c1} \leftarrow \sqrt{s_3 + f_3}, s_4 \leftarrow v_{c2} + u_{c1}s_1, v_{c1} \leftarrow \sqrt{s_4}$ ▷ 2M+2SR
3: $s_5 \leftarrow s_3 + s_0, s_7 \leftarrow (s_4 + s_6)u_{a1}, s_8 \leftarrow u_{a0} + u_{a1}^2$ ▷ 1M+1S
4: $s_9 \leftarrow s_5s_8, s_{10} \leftarrow s_5u_{a1}, u_{c0} \leftarrow \sqrt{f_1 + s_9 + s_7 + v_{a1}}$ ▷ 2M+1SR
5: $s_{11} \leftarrow s_4 + s_6 + s_{10}, v_{c0} \leftarrow \sqrt{f_0 + 1 + s_{11}u_{a0} + v_{a0}}$ ▷ 1M+1SR
6: **return** $[x^3 + u_{c1}x + u_{c0}, v_{c2}x^2 + v_{c1}x + v_{c0}]$ ▷ 1I+7M+1S+7SR

3.2.3. *Halving* $1 \rightarrow 3$. Just as in the $2 \rightarrow 3$ case, the output has more coefficients than the input, giving us difficulties to reverse the doubling formula. This time, the output must be of the form $[x^3 + u_{c1}x + u_{c0}, v_{c0}]$, and once again the solution can be found in Theorem 2.3. We compute the coefficient of x in $v_{c0}^2 + v_{c0} + f$

mod $x^3 + u_{c1}x + u_{c0}$ to find that $f_1 + u_{c0}^2 + u_{c1}f_3 + u_{c1}^3$ must be 0, and we can complete the formula using the relation $u_{c0} = \sqrt{f_1 + u_{c1}(f_3 + u_{c1}^2)}$.

Algorithm 10 (HLV13, $h(x) = 1$, $f(x) = x^7 + f_3x^3 + f_1x + f_0$)

INPUT: $\overline{D} = [x + u_{a0}, v_{a0}]$

OUTPUT: $[1/2]\overline{D} = [x^3 + u_{c1}x + u_{c0}, v_{c0}]$

- 1: $s_0 \leftarrow \sqrt{u_{a0}} + f_3$, $u_{c1} \leftarrow \sqrt{s_0}$, $s_1 \leftarrow u_{a0}^2$ ▷ 1S+2SR
 - 2: $s_2 \leftarrow (f_3 + s_0)u_{c1}$, $s_3 \leftarrow (f_3 + s_0)s_1$, $u_{c0} \leftarrow \sqrt{f_1 + s_2}$ ▷ 2M+1SR
 - 3: $v_{c0} \leftarrow \sqrt{v_{a0} + u_{a0}(s_3 + s_2) + f_0 + 1}$ ▷ 1M+1SR
 - 4: **return** $[x^3 + u_{c1}x + u_{c0}, v_{c0}x^2 + v_{c1}x + v_{c0}]$ ▷ 3M+1S+4SR
-

3.2.4. *Halving* $3 \rightarrow 2$. Although reversing the $2 \rightarrow 3$ doubling formula can be done in 2M and 5SR, distinguishing the $3 \rightarrow 2$ halving from the $3 \rightarrow 3$ case requires a few more operations. The operation count below assumes that the first formula of Subsection 3.2.1 is used. If the implementation does not take advantage of sequential multiplications, the $3 \rightarrow 2$ halving can be done completely in 3M, 1S and 5SR.

Algorithm 11 (HLV32, $h(x) = 1$, $f(x) = x^7 + f_3x^3 + f_1x + f_0$)

INPUT: $\overline{D} = [x^3 + u_{a2}x^2 + u_{a1}x + u_{a0}, v_{a2}x^2 + v_{a1}x + v_{a0}]$

OUTPUT: $[1/2]\overline{D} = [x^2 + u_{c1}x + u_{c0}, v_{c1}x + v_{c0}]$

- 1: $s_0 \leftarrow \sqrt{u_{a2}}$, $s_1 \leftarrow u_{a2}s_0$, $s_2 \leftarrow u_{a0}s_0$, $s_3 \leftarrow v_{a2} + s_1$ ▷ 2M+1SR
 - 2: $s_4 \leftarrow u_{a0} + s_3^2 + u_{a1}u_{a2}$, $u_{c1} \leftarrow \sqrt{u_{a1}}$, $u_{c0} \leftarrow u_{a1} + \sqrt{s_0 + f_3}$ ▷ 1M+1S+2SR
 - 3: $v_{c1} \leftarrow \sqrt{v_2 + s_1}$, $v_{c0} \leftarrow \sqrt{v_{a0} + f_0 + 1 + s_2}$ ▷ 2SR
 - 4: **return** $[x^2 + u_{c1}x + u_{c0}, v_{c1}x + v_{c0}]$ ▷ 3M+1S+5SR
-

3.2.5. *Halving* $2 \rightarrow 1$. This is the final case of halving, and the simplest one.

Algorithm 12 (HLV21, $h(x) = 1$, $f(x) = x^7 + f_3x^3 + f_1x + f_0$)

INPUT: $\overline{D} = [x^2 + u_{a0}, v_{a1}x + v_{a0}]$

OUTPUT: $[1/2]\overline{D} = [x + u_{c0}, v_{c0}]$

- 1: $u_{c0} \leftarrow \sqrt{u_{a0}}$, $v_{c0} \leftarrow \sqrt{v_{a0} + f_0}$
 - 2: **return** $[x + u_{c0}, v_{c0}]$ ▷ 2SR
-

4. HALVING FOR OTHER TYPES OF CURVES

In this section, we consider halving formulas for curves of genus 3 with $h(x)$ irreducible (and non-constant), i.e. curves of Type Ia, IIa, and III. Types Ie and IIc, where $h(x)$ is a perfect power (rather than irreducible) are available in the appendix of the extended version of this paper [7]. From an efficiency point of view, these cases are less interesting since they offer the same number of isomorphism classes as when $h(x)$ is linear, but at a higher cost.

In general, it would be safe to say that the cost of the halving operation increases with the degree of $h(x)$, but this is in a way offset by having a larger number of isomorphism classes (in particular when $h(x)$ is irreducible), giving us more flexibility in the choice of the curves.

Unlike the curves in the previous section (Type IV), the doubling of a divisor admits two pre-images, and we must distinguish which of the two has odd order. Because of the doubling is a 2-to-1 map, the structure of the special cases will be somewhat altered. How to deal with this altered situation will be described in Subsection 4.1

In the following subsections, we will study the three types of curves in increasing order of complexity (i.e. increasing the degree of $h(x)$). For each curve type, we will “define” the different cases (i.e. describe how to distinguish them) and obtain necessary and sufficient conditions under which a divisor can be halved, which will allow us to give a simple criteria for the curve to have order $2m$, where m is odd, after which we give the explicit formulas for the most common case (to reduce the length of this paper, the other cases are available in the appendix of the extended version [7]). This structure will be repeated in the appendix for Types Ie and IIc.

Finally, we will analyze the results in Subsection 4.5.

4.1. HALVING $3 \rightarrow 3$ VERSUS SPECIAL CASES. If we look at the doubling algorithm when $\deg(h) > 0$, the most obvious difference is that we cannot ignore the gcd of $h(x)$ and $u_a(x)$. If $\gcd(u_a, h) = d \neq 1$, we first divide u_a by d , and reduce v_a accordingly, after which the “normal” structure of special cases applies (clearly only the doublings $2 \rightarrow 3$ and $1 \rightarrow 2$ are possible if $u_a/\gcd(u_a, h)$ is different from 1).

The observation on $\gcd(u_a, h) \neq 1$ is very indicative of the problem we face with the special cases of halving, but also hints at the solution. In the curves we are interested in this section, the doubling is a 2-to-1 function, so to compute the halving we will find two possible pre-images, but these pre-images could have different degrees (which complicates the distinction between the different special cases). On the other hand, the difference between the two pre-images is always the unique divisor class of order 2, so once we can compute a pre-image the other one could be found using Cantor’s algorithm (adding the divisor class of order 2). Note that the divisor class of order 2 is of the form $[h, v_h]$ when h is irreducible, and of the form $[x, \sqrt{f_0}]$ when h is a square or a cube.

To denote the halving cases, we will base ourselves on the lowest degree of the pre-image, and then aggregate the degree of the other pre-image if it is different. For example, HLV32/33 indicates that the input has degree 3, that one of the two pre-images has degree 2 and the second one has degree 3. If both pre-images have the same degree, we keep the same notation as before (for example HLV23). The main advantage of this notation is that the pre-image of lowest degree is generally the one that closely matches the corresponding case for Type IV curves.

In fact, when the pre-images have distinct degrees, the second pre-image can often be found simply by adding the (unique) divisor class of order 2 to the first pre-image using Cantor’s algorithm without the reduction step (as long as the total degree remains less than 3), and it is usually more efficient to compute it explicitly in this way. When adding the divisor of order 2 requires a reduction, it appears more practical to go back to inverting the doubling, this time using the degree for the second pre-image. We observe that those cases are due to certain coefficients being 0 in the doubling, leading to “degenerate” quadratic equations, for example $z^2 + 0z = \alpha$ (which has a double root instead of two distinct ones).

4.2. TYPE III: $h(x) = x$. According to Section 2.4, curves of Type III are of the form

$$(4) \quad C : y^2 + xy = x^7 + f_5x^5 + f_3x^3 + f_2x^2 + f_0,$$

where $f_2 \in \mathbb{F}_2$. The Picard group of these curves has precisely one divisor class of order 2, which is of the form $[x, \sqrt{f_0}]$.

Theorem 4.1. *Let $\overline{D}_a = [u_a, v_a]$ be a divisor class in $\text{Pic}_C^0(\mathbb{F}_q)$. If $\deg(u_a) = 3$, then \overline{D}_a can be halved if and only if $\text{TR}(u_{a1}((u_{a2}^2 + f_5)u_{a2} + v_{a2}^2 + u_{a0})) = 0$. If $\deg(u_a) = 2$, then \overline{D}_a can be halved if and only if $\text{TR}(u_{a1}((u_{a0} + u_{a1}^2)(u_{a0} + f_5) + u_{a1}^4 + f_3)) = 0$. If $\deg(u_a) = 1$, then \overline{D}_a can be halved if and only if $\text{TR}(u_{a0}(u_{a0}^2(u_{a0}^2 + f_5) + f_3) + f_2) = 0$.*

Proof. To halve a divisor class $\overline{D}_a = [u_a, v_a]$, we assume that it is the image under the doubling of a divisor class $\overline{D}_c = [u_c, v_c]$. To perform the halving, we work our way backwards through the doubling of \overline{D}_c , trying to solve for the coefficients of u_c and v_c given the coefficients of u_a and v_a (the form of u_c and v_c are determined by the halving case).

In cases HLV33, HLV23 and HLV13, the halving requires us to solve an equation of the form $z^2 + z + \alpha = 0$ at some point in the computations. If \overline{D}_a is indeed equal to $[2]\overline{D}_c$ for some \mathbb{F}_q -rational divisor class \overline{D}_c , then an \mathbb{F}_q -rational root of $z^2 + z + \alpha = 0$ must exist (since all the operations in Cantor's algorithm are performed over \mathbb{F}_q). If $\text{TR}(\alpha) = 1$, then no such root can exist, so a divisor class must have $\text{TR}(\alpha) = 0$ if we want to halve it.

For cases HLV21/22 and HLV32/33, it is always possible to halve them, but there are special conditions on the coefficients of u_a and v_a and it can be shown that these conditions force $\alpha = 0$ (and obviously $\text{TR}(\alpha) = 0$). This gives us the necessity of the trace conditions.

To complete the proof, we must show that the trace conditions are also sufficient. For this, we show that if the trace condition holds for a reduced divisor, then applying one of the halving formula to this divisor will return an output that is a valid divisor. By construction (of the formula), the double of that new divisor must be the input of our halving, hence this input can be halved. Note that being able to compute two \mathbb{F}_q -rational polynomials u_c and v_c with the halving formulas is not sufficient on its own to give us a divisor. We must also verify that $v_c^2 + v_ch + f$ is divisible by u_c .

We explain how to do this in the HLV33 case, the other cases follow the same pattern. We begin with a divisor class $[u_a, v_a]$, i.e. $v_a^2 + v_ah + f \equiv 0 \pmod{u_a}$. The coefficients of x^0 , x^1 and x^2 in this equality give us 3 coefficient identities, the "divisibility conditions". To obtain the halving formulas, we compute a sequence of pairs of polynomials $[u_i, v_i]$ which should all be semi-reduced divisors if we want the output to be a reduced divisor (rather than a random pair of polynomials).

From $[u_a, v_a]$, we first compute $[u_2, v_2]$ using the polynomial equations

$$\begin{aligned} u_a &= \text{monic} \left(\frac{v_2^2 + v_2h + f}{u_2} \right), \\ v_a &\equiv v_2 + h \pmod{u_a}, \\ v_2^2 + v_2h + f &\equiv 0 \pmod{u_2} \end{aligned}$$

(working backwards through the second reduction and making sure we have a semi-reduced divisor). These equations give us 10 identities that must be satisfied by the

coefficients of u_2 and v_2 . We use 7 of these identities to compute the coefficients, and the 3 remaining identities become our new divisibility conditions. To show that $[u_c, v_c]$ is a semi-reduced divisor, we use the 7 identities of the halving formula to show that the 3 divisibility conditions of $[u_c, v_c]$ imply the 3 new divisibility conditions (once all 10 identities are satisfied, so are the 3 polynomial equations).

We then repeat the same idea to show that $[u_1, v_1]$ (first reduction) is also a semi-reduced divisor: To compute the coefficients of u_1 and v_1 , we used 9 of the 13 coefficient identities in the equations

$$\begin{aligned} u_2 &= \text{monic} \left(\frac{v_1^2 + v_1 h + f}{u_1} \right), \\ v_2 &\equiv v_1 + h \pmod{u_1}, \\ v_1^2 + v_1 h + f &\equiv 0 \pmod{u_1}. \end{aligned}$$

We are left with 4 divisibility conditions, which can be shown to be implied by the 3 divisibility conditions on u_1 and v_1 (once again using the 9 identities of the halving formula to perform the simplifications).

To finish, we have to show that $v_0^2 + v_0 h + f \equiv 0 \pmod{u_0}$, where $u_0 = \sqrt{u_1}$ and $v_0 \equiv v_1 \pmod{u_0}$ (i.e. performing the composition step backwards). This comes directly from $v_1^2 + v_1 h + f \equiv 0 \pmod{u_1}$. Since this halving case comes from $\gcd(u_c, h) = 1$ in the doubling of $[u_c, v_c]$ (the preimage of the doubling), we have $u_c = u_0$ and $v_c = v_0$ and all the divisibility conditions are already obtained.

To complete the proof, this process is repeated for the other halving cases, showing that in all cases the preimages computed are valid divisors if the trace conditions are satisfied. Note that for the HLV21/22 and HLV32/33 cases there is only one possible choice for u_0 and v_0 . The first preimage (of lower degree) corresponds to $\gcd(u_c, h) = 1$ and no further work is required. The second preimage corresponds to $\gcd(u_c, h) = x$ and the divisibility conditions come from the addition of the reduced divisor $[u_0, v_0]$ to the reduced divisor of order 2 (using Cantor's algorithm, which does not require any reduction step in this case). \square

Corollary 1. *The Picard group of the curve C over \mathbb{F}_q given by (4) has order $2m$, where m is odd, if and only if $f_2 = 1$.*

Proof. The Picard group has exactly one divisor class of order 2, namely $[x, \sqrt{f_0}]$. The order of the Picard group is divisible by 4 if and only if $[x, \sqrt{f_0}]$ can be halved. From Theorem 4.1, this is possible if and only if $\text{TR}(f_2) = 0$. Since $f_2 \in \mathbb{F}_2$, we find that $\text{Pic}_C^0(\mathbb{F}_q)$ has a divisor class of order 4 if and only if $f_2 = 0$. \square

Observe that if \overline{D}_{c_1} and \overline{D}_{c_2} are the two preimages of \overline{D}_a under the doubling, then $\overline{D}_{c_1} - \overline{D}_{c_2} = [x, \sqrt{f_0}] = \overline{D}_{c_2} - \overline{D}_{c_1}$, i.e. the difference of two preimages is the unique divisor class of order 2. This observation allows us to distinguish the different special cases.

Remark 1. Let $\overline{D}_a = [u_a, v_a]$ be a divisor class in $\text{Pic}_C^0(\mathbb{F}_q)$ that can be halved and $\overline{D}_c = [u_c, v_c] = [\frac{1}{2}]\overline{D}_a$ its preimage (under the doubling) of odd order.

- (1a) If $\deg(u_a) = 3$ and $v_{a2}^2 + u_{a2}(u_{a2}^2 + u_{a1} + f_5) + \sqrt{u_{a2}} + u_{a0} \neq 0$, then $\deg(u_c) = 3$ and we are in case HLV33.
- (1b) If $\deg(u_a) = 3$ and $v_{a2}^2 + u_{a2}(u_{a2}^2 + u_{a1} + f_5) + \sqrt{u_{a2}} + u_{a0} = 0$, then $\deg(u_c) = 2$ or 3 (with $u_{c0} = 0$ in the second case) and we are in case HLV32/33.
- (2a) If $\deg(u_a) = 2$ and $u_{a1} \neq 0$, then $\deg(u_c) = 2$ and we are in case HLV23.

- (2b) If $\deg(u_a) = 2$ and $u_{a1} = 0$, then $\deg(u_c) = 1$ or 2 (with $u_{c0} = 0$ in the second case) and we are in case HL21/22.
(3) If $\deg(u_a) = 1$, then $\deg(u_c) = 3$ and we are in case HL13.

We obtain the following halving formulas:

Algorithm 13 (HL33, $h(x) = x$, $f(x) = x^7 + f_5x^5 + f_3x^3 + x^2 + f_0$)

INPUT: $\overline{D} = [x^3 + u_{a2}x^2 + u_{a1}x + u_{a0}, v_{a2}x^2 + v_{a1}x + v_{a0}]$

OUTPUT: $[1/2]\overline{D} = [x^3 + u_{c2}x^2 + u_{c1}x + u_{c0}, v_{c2}x^2 + v_{c1}x + v_{c0}]$

```

1:  $s_0 \leftarrow \sqrt{u_{a2}}, s_1 \leftarrow u_{a1} + f_5, s_2 \leftarrow s_0u_{a2} + v_{a2}, s_3 \leftarrow s_1u_{a2}$   $\triangleright$  2M+1SR
2:  $s_4 \leftarrow s_0 + s_2^2 + s_3 + u_{a0}, s_5 \leftarrow s_4u_{a1}, s_6 \leftarrow s_0u_{a1} + v_{a1} + 1$   $\triangleright$  2M+1S
3:  $s_7 \leftarrow s_0u_{a0} + v_{a0}, s_8 \leftarrow s_2 + f_3 + (s_4 + s_1)(u_{a2} + u_{a1}) + s_3 + s_5$   $\triangleright$  2M
4:  $s_9 \leftarrow s_4^{-1}, s_{10} \leftarrow \sqrt{s_9}, s_{11} \leftarrow s_{10}s_1, s_{12} \leftarrow s_{11} + s_0$   $\triangleright$  1I+1M+1SR
5:  $s_{13} \leftarrow \text{HT}(s_5), s_{14} \leftarrow s_9s_{13}$   $\triangleright$  1M+1HT
6:  $s_{15} \leftarrow s_2 + (s_{14} + s_{10})(s_4 + s_1) + s_{11} + s_{13}, s_{16} \leftarrow s_{14}s_8 + s_7$   $\triangleright$  2M
7:  $s_{17} \leftarrow s_{14} + f_5, u_{c2} \leftarrow \sqrt{s_{17}}, s_{18} \leftarrow s_{15} + f_3, u_{c1} \leftarrow \sqrt{s_{18}}$   $\triangleright$  2SR
8:  $s_{19} \leftarrow s_{16} + f_1, u_{c0} \leftarrow \sqrt{s_{19}}, s_{20} \leftarrow s_{10}s_8 + s_{13} + s_6 + 1$   $\triangleright$  1M+1SR
9:  $s_{21} \leftarrow s_{10}u_{c2}, s_{22} \leftarrow s_{14} + s_{21}, s_{23} \leftarrow s_{22}u_{c1}$   $\triangleright$  2M
10:  $s_{24} \leftarrow s_{12} + (s_{10} + s_{22})(u_{c1} + u_{c2}) + s_{21} + s_{23}, s_{25} \leftarrow s_{24}u_{c0}$   $\triangleright$  2M
11:  $v_{c2} \leftarrow s_{15} + s_{23} + (s_{10} + s_{24})(u_{c0} + u_{c2}) + s_{21} + s_{25}$   $\triangleright$  1M
12:  $s_{26} \leftarrow \text{TR}(u_{c1}(u_{c2}(s_{17} + f_5) + v_{c2}^2 + u_{c0}))$   $\triangleright$  2M+1S+1TR
13: if  $s_{26} = 1$  then
14:    $s_{20} \leftarrow s_{20} + 1, s_{27} \leftarrow s_{10}\sqrt{s_8}, s_{16} \leftarrow s_{16} + s_{27}^2$   $\triangleright$  1M+1S+1SR
15:    $u_{c2} \leftarrow u_{c2} + s_{10}, s_{28} \leftarrow s_{10}\sqrt{s_1}, u_{c1} \leftarrow u_{c1} + s_{28}$   $\triangleright$  1M+1SR
16:    $u_{c0} \leftarrow u_{c0} + s_{27}, s_{21} \leftarrow s_{21} + s_9, s_{23} \leftarrow s_{23} + s_{22}s_{28}$   $\triangleright$  1M
17:    $s_{24} \leftarrow s_{24} + s_{10}(s_{22} + s_{28}), s_{25} \leftarrow s_{24}u_{c0}$   $\triangleright$  2M
18:    $v_{c2} \leftarrow s_{15} + s_{28}^2 + s_{23} + (s_{10} + s_{24})(u_{c0} + u_{c2}) + s_{21} + s_{25}$   $\triangleright$  1M+1S
19: end if
20:  $v_{c1} \leftarrow s_{20} + (s_{24} + s_{22})(u_{c0} + u_{c1}) + s_{23} + s_{25}, v_{c0} \leftarrow s_{16} + s_{25}$   $\triangleright$  1M
21: return  $[x^3 + u_{c2}x^2 + u_{c1}x + u_{c0}, v_{c2}x^2 + v_{c1}x + v_{c0}]$ 

```

\triangleright 1I+25M+4S+7SR+1HT+1TR

We have a worst-case cost of 1I+25M+4S+7SR+1HT+1TR, which compares very well with the doubling cost of 1I+44M+6S of [13].

However, the conditional block of lines 13 to 19 is only used when the initial “choice” of the root of $z^2 + z + s_5 = 0$ (i.e. $\text{HT}(s_5)$ rather than $\text{HT}(s_5) + 1$) is incorrect and the variables computed afterwards must be corrected. This means that the 6M+2S+2SR associated to that correction in the conditional block will only be needed half of the time (on average), and the average cost of the halving operation becomes 1I+22M+3S+6SR+1HT+1TR.

4.3. TYPE IIA: $h(x) = x^2 + x + 1$. According to Section 2.4, curves of Type IIA are of the form

$$(5) \quad C : y^2 + (x^2 + x + 1)y = f_7x^7 + f_5x^5 + f_4x^4 + f_1x + f_0,$$

where $f_4 \in \mathbb{F}_2$. The Picard group of these curves has precisely one divisor class of order 2, which is of the form $[h, v_h] = [x^2 + x + 1, v_h]$.

Theorem 4.2. *Let $\overline{D}_a = [u_a, v_a]$ be a divisor class in $\text{Pic}_C^0(\mathbb{F}_q)$. If $\deg(u_a) = 3$, then \overline{D}_a can be halved if and only if $\text{TR}(f_7u_{a0} + f_4 + u_{a2}(f_7(u_{a1} + u_{a2}^2) + f_5 + f_7)) = 0$. If $\deg(u_a) = 2$, then \overline{D}_a can be halved if and only if $\text{TR}(u_{a1}(f_7(u_{a1}^2 + u_{a0}) + f_5 + f_7)) = 0$. If $\deg(u_a) = 1$, then \overline{D}_a can be halved if and only if $\text{TR}(f_4 + u_{a0}(f_7u_{a0}^2 + f_5 + f_7)) = 0$.*

Proof. We use the same approach as in Theorem 4.1. Note that some of the formulas require solving two quadratic equations. In those cases, it is easy to verify that changing the root of the first quadratic equation changes the trace of the constant term of the second quadratic equation by 1, so only one of the two roots of the first quadratic equation allows us to compute an \mathbb{F}_q -rational preimage. \square

Corollary 2. *The Picard group of the curve C given by (5) has order $2m$, where m is odd, if and only if $\text{TR}(f_7) \neq \text{TR}(f_5)$.*

Proof. The Picard group has exactly one divisor class of order 2, namely $[x^2 + x + 1, v_h]$. The order of the Picard group is divisible by 4 if and only if $[x^2 + x + 1, v_h]$ can be halved. From Theorem 4.2, this is possible if and only if $\text{TR}(f_5 + f_7) = 0$ and we find that $\text{Pic}_C^0(\mathbb{F}_q)$ has a divisor class of order 4 if and only if $\text{TR}(f_7) = \text{TR}(f_5)$. \square

Observe that if \overline{D}_{c_1} and \overline{D}_{c_2} are the two preimages of \overline{D}_a under the doubling, then $\overline{D}_{c_1} - \overline{D}_{c_2} = [x^2 + x + 1, v_h] = \overline{D}_{c_2} - \overline{D}_{c_1}$, i.e. the difference of two preimages is the unique divisor class of order 2. This observation allows us to distinguish the different special cases.

Remark 2. Let $\overline{D}_a = [u_a, v_a]$ be a divisor class in $\text{Pic}_C^0()$ that can be halved and $\overline{D}_c = [u_c, v_c] = [\frac{1}{2}]\overline{D}_a$ its preimage (under the doubling) of odd order.

- (1a) If $\deg(u_a) = 3$ and $v_{a2}^2 + v_{a2} + u_{a2}(f_5 + u_{a1}f_7 + u_{a2}^2f_7) + \sqrt{u_{a2}f_7} + f_4 + u_{a0}f_7 \neq 0$, then $\deg(u_c) = 3$ and we are in case HLV33.
- (1b) If $\deg(u_a) = 3$ and $v_{a2}^2 + v_{a2} + u_{a2}(f_5 + u_{a1}f_7 + u_{a2}^2f_7) + \sqrt{u_{a2}f_7} + f_4 + u_{a0}f_7 = 0$, then $\deg(u_c) = 2$ or 3 and we are in case HLV32/33.
- (2a) If $\deg(u_a) = 2$ and $u_{a1} \neq 0$, then $\deg(u_c) = 2$ and we are in case HLV23.
- (2b) If $\deg(u_a) = 2$ and $u_{a1} = 0$, then $\deg(u_c) = 1$ or 2 (with $u_{c2} = u_{c1} = u_{c0} + 1$ in the second case) and we are in case HLV21/23.
- (3) If $\deg(u_a) = 1$, then $\deg(u_c) = 3$ and we are in case HLV13.

Algorithm 14 (HLV33, $h(x) = x^2 + x + 1$, $f(x) = f_7x^7 + f_5x^5 + f_4x^4 + f_1x + f_0$)

INPUT: $\overline{D} = [x^3 + u_{a2}x^2 + u_{a1}x + u_{a0}, v_{a2}x^2 + v_{a1}x + v_{a0}]$

OUTPUT: $[1/2]\overline{D} = [x^3 + u_{c2}x^2 + u_{c1}x + u_{c0}, v_{c2}x^2 + v_{c1}x + v_{c0}]$

- 1: $s_0 \leftarrow u_{a2}f_7$, $s_1 \leftarrow \sqrt{s_0}$, $s_2 \leftarrow (s_1 + f_5)f_7^{-1} + u_{a1}$ $\triangleright 2M+1SR$
- 2: $s_3 \leftarrow s_1u_{a2} + v_{a2} + 1$, $s_4 \leftarrow s_1u_{a1} + v_{a1} + 1$, $s_5 \leftarrow s_1u_{a0} + v_{a0} + 1$ $\triangleright 3M$
- 3: $s_6 \leftarrow s_2s_0$, $s_7 \leftarrow (s_6 + s_3^2 + s_3 + s_1 + f_4)f_7^{-1} + u_{a0}$, $s_8 \leftarrow s_7^{-1}$ $\triangleright 1I+2M+1S$

4: $s_9 \leftarrow (s_7 s_0 + s_4 + s_3 + s_1) f_7^{-1} + s_2 u_{a1}$, $s_{10} \leftarrow f_7 s_7$, $s_{11} \leftarrow \text{HT}(s_{10})$ \triangleright 4M+1HT
5: $s_{12} \leftarrow s_{11} s_8$, $s_{13} \leftarrow (f_7 s_2 + s_1 + f_5 + s_{12}) s_7$, $s_{14} \leftarrow \text{TR}(s_{13})$ \triangleright 3M+1TR
6: **if** $s_{14} = 1$ **then**
7: $s_{11} \leftarrow s_{11} + 1$, $s_{12} \leftarrow s_{12} + s_8$, $s_{13} \leftarrow s_{13} + 1$
8: **end if**
9: $s_{15} \leftarrow \text{HT}(s_{13})$, $s_{16} \leftarrow s_{15} s_8$, $s_{17} \leftarrow s_{12} s_2 + s_1$ \triangleright 2M+1HT
10: $s_{18} \leftarrow s_{16} s_2 + s_3 + 1 + s_{11}$, $s_{19} \leftarrow s_{12} s_9 + s_4 + 1 + s_{15}$ \triangleright 2M
11: $s_{20} \leftarrow (s_{12} + s_8 + f_7) f_7^{-2}$, $s_{21} \leftarrow (s_{17} + s_{16} + s_{12} + f_5) s_{20}$ \triangleright 2M
12: $u_{c2} \leftarrow \sqrt{s_{21}}$, $s_{22} \leftarrow (s_{19} + s_{18} + s_{17}) s_{20}$, $u_{c1} \leftarrow \sqrt{s_{22}}$ \triangleright 1M+2SR
13: $s_{24} \leftarrow (s_{23} + s_{19} + f_1) s_{20}$, $u_{c0} \leftarrow \sqrt{s_{24}}$ \triangleright 1M+1SR
14: $s_{25} \leftarrow \text{TR}(u_{c0} f_7 + f_4 + u_{c2} (f_5 + f_7 (u_{c1} + s_{21} + 1)))$ \triangleright 3M+1TR
15: **if** $s_{25} = 1$ **then**
16: $s_{16} \leftarrow s_{16} + s_8$, $s_{26} \leftarrow s_8 s_2$, $s_{18} \leftarrow s_{18} + s_{26}$, $s_{19} \leftarrow s_{19} + 1$ \triangleright 1M
17: $s_{27} \leftarrow s_8 s_9$, $s_{23} \leftarrow s_{23} + s_{27}$, $u_{c0} \leftarrow u_{c0} + \sqrt{(s_{27} + 1) s_{20}}$ \triangleright 2M+1SR
18: $u_{c2} \leftarrow u_{c2} + \sqrt{s_8 s_{20}}$, $u_{c1} \leftarrow u_{c1} + \sqrt{(s_{26} + 1) s_{20}}$ \triangleright 2M+2SR
19: **end if**
20: $s_{28} \leftarrow s_{12} u_{c2}$, $s_{29} \leftarrow s_{16} + s_{28}$, $s_{30} \leftarrow s_{29} u_{c1}$ \triangleright 2M
21: $s_{31} \leftarrow s_{17} + (s_{12} + s_{29})(u_{c2} + u_{c1}) + s_{28} + s_{30}$, $s_{32} \leftarrow s_{31} u_{c0}$ \triangleright 2M
22: $v_{c0} \leftarrow s_{23} + s_{32}$, $v_{c1} \leftarrow s_{19} + (s_{29} + s_{31})(u_{c1} + u_{c0}) + s_{30} + s_{32}$ \triangleright 1M
23: $v_{c2} \leftarrow s_{18} + (s_{12} + s_{31})(u_{c2} + u_{c0}) + s_{28} + s_{32} + s_{30}$ \triangleright 1M
24: **return** $[x^3 + u_{c2} x^2 + u_{c1} x + u_{c0}, v_{c2} x^2 + v_{c1} x + v_{c0}]$
 \triangleright 1I+36M+1S+7SR+2HT+2TR

We note that in these formulas a division by $s_{12} + f_7$ would normally be required to compute s_{21} , s_{22} and s_{24} . However, s_{12} is a root of $s_7 z^2 + z + f_7 = 0$ (since $s_{12} = s_{11}/s_7$), so $s_7(s_{12} + f_7)(s_{12} + f_7 + 1/s_7) = s_7 s_{12}^2 + s_7 f_7^2 + s_{12} + f_7 = s_7 f_7^2$. We can therefore replace divisions by $s_{12} + f_7$ with multiplications by $(s_{12} + f_7 + 1/s_7) f_7^2 = s_{20}$, replacing the inverse by a single multiplication.

We therefore have a worst-case cost of 1I+36M+1S+7SR+2HT+2TR, which compares very well with the doubling cost of 1I+52M+8S of [13].

Conditional line 7 has very little impact on the overall cost, but the conditional block of lines 15 to 19 has a noticeable cost. However, it is only used when the initial ‘‘choice’’ of the root of $z^2 + z + s_{13} = 0$ (i.e. $\text{HT}(s_{13})$ rather than $\text{HT}(s_{13}) + 1$) is incorrect and the variables computed afterwards must be corrected. This means that the cost of 5M+3SR associated to that correction will only be needed half of the time (on average). The average cost of the halving operation becomes 1I+33.5M+1S+5.5SR+2HT+2TR.

4.4. TYPE IA: $h(x) = x^3 + x + h_0$ IRREDUCIBLE. According to Section 2.4, curves of Type Ia are of the form

$$(6) \quad C : y^2 + (x^3 + x + h_0)y = f_7 x^7 + x^6 + f_2 x^2 + f_1 x + f_0,$$

where $x^3 + x + h_0$ is irreducible over \mathbb{F}_q and $f_6 \in \mathbb{F}_2$. The Picard group of these curves has precisely one divisor class of order 2, which is of the form $[h, v_h] = [x^3 + x + h_0, v_h]$.

Proposition 2. *If the polynomial $x^3 + x + h_0$ is irreducible over \mathbb{F}_q , then the equation $x^4 + x^2 + h_0x + a = 0$ has exactly one root in \mathbb{F}_q for each $a \in \mathbb{F}_q$.*

Proof. Since x^4 and x^2 act linearly in fields of characteristic 2, the operator $T(x) = x^4 + x^2 + h_0x$ is linear. We also note that the roots of $T(x) = 0$ are 0, ζ_1 , ζ_2 and ζ_3 , where the ζ_i are the roots of $x^3 + x + h_0 = 0$ in $\mathbb{F}_{q^3} \setminus \mathbb{F}_q$ (since $x^3 + x + h_0$ is irreducible). Because of this, for any $a \in \mathbb{F}_q$ there cannot exist more than one \mathbb{F}_q -rational root, otherwise we would have two \mathbb{F}_q -rational roots of $T(x) = 0$. To each element $\alpha \in \mathbb{F}_q$ we can associate a polynomial of the form $x^4 + x^2 + h_0x + a$, namely with $a = T(\alpha)$, all of which have exactly one \mathbb{F}_q -rational root. \square

Note that $T(x) = x^4 + x^2 + h_0x$ being a linear operator also allows us to compute the \mathbb{F}_q -rational root. We first compute the images of $T(e_i)$ for every e_i in the basis used to represent field elements, which gives us a system of linear equations (that can be used to describe the image of every field element). By inverting this system, we can precompute the roots x_i of $x^4 + x^2 + h_0x + e_i = 0$. For any given $a \in \mathbb{F}_q$, $a = \sum_{i=0}^{n-1} a_i e_i$ (with $a_i \in \mathbb{F}_2$), we can then compute the root x_a of $x^4 + x^2 + h_0x + a = 0$ as $x_a = \sum_{i=0}^{n-1} a_i x_i$. With a little more work (computing the roots for all blocks of w bits), it becomes possible to compute roots of the quartic in time QR at least as fast as a multiplication. In fact, this method is equivalent to what is used to compute half-traces, so $\text{QR} \approx \text{HT} \leq \text{M}$.

Remark 3. In the case where $h_0 = 1$, we can express the root of $x^4 + x^2 + x + a = 0$ as the “two-third-trace” of a : if $n \equiv 1 \pmod{3}$, we let $x_a = \text{TR}(a) - \sum_{i=0}^{\frac{n-4}{3}} a^{2^{3i+1}}$, and if $n \equiv 2 \pmod{3}$, we let $x_a = \text{TR}(a) - \sum_{i=0}^{\frac{n-2}{3}} a^{2^{3i}}$.

Theorem 4.3. *Let $\overline{D}_a = [u_a, v_a]$ be a divisor class in $\text{Pic}_C^0(\mathbb{F}_q)$. If $\deg(u_a) = 3$, then \overline{D}_a can be halved if and only if $\text{TR}(f_7 u_{a2} + f_6) = 0$. If $\deg(u_a) = 2$, then \overline{D}_a can be halved if and only if $\text{TR}(f_7 u_{a1}) = 0$. If $\deg(u_a) = 1$, then \overline{D}_a can be halved if and only if $\text{TR}(f_7 u_{a0} + f_6) = 0$.*

Proof. As in Theorem 4.1. \square

Corollary 3. *The Picard group of the curve C over \mathbb{F}_q given by (6) has order $2m$, where m is odd, if and only if $f_6 = 1$.*

Proof. The Picard group has exactly one divisor class of order 2, namely $[x^3 + x + h_0, v_h]$. The order of the Picard group is divisible by 4 if and only if $[x^3 + x + h_0, v_h]$ can be halved. From Theorem 4.3, this is possible if and only if $\text{TR}(f_6) = 0$. Since $f_6 \in \mathbb{F}_2$, we find that $\text{Pic}_C^0(\cdot)$ has a divisor class of order 4 if and only if $f_6 = 0$. \square

Observe that if \overline{D}_{c_1} and \overline{D}_{c_2} are the two preimages of \overline{D}_a under the doubling, then $\overline{D}_{c_1} - \overline{D}_{c_2} = [x^3 + x + h_0, v_h] = \overline{D}_{c_2} - \overline{D}_{c_1}$, i.e. the difference of two preimages is the unique divisor class of order 2. This observation allows us to distinguish the different special cases.

Remark 4. Let $\overline{D}_a = [u_a, v_a]$ be a divisor class in $\text{Pic}_C^0(\mathbb{F}_q)$ that can be halved and $\overline{D}_c = [u_c, v_c] = [\frac{1}{2}]\overline{D}_a$ its preimage (under the doubling) of odd order.

(1a) If $\deg(u_a) = 3$ and

$$(1 + u_{a1})(v_{a1} + u_{a0}f_7 + v_{a2}^2 + u_{a2}v_{a2} + (u_{a2}^2 + u_{a1})(1 + u_{a2}f_7)) \\ + (v_{a1} + u_{a0}f_7 + v_{a2}^2 + u_{a2}v_{a2} + (u_{a2}^2 + u_{a1})(1 + u_{a2}f_7))^2 \\ + u_{a1} + (1 + u_{a2}f_7)(1 + u_{a1}^2) \neq 0,$$

then $\deg(u_c) = 3$ and we are in case HL33.

(1b) If $\deg(u_a) = 3$ and

$$(1 + u_{a1})(v_{a1} + u_{a0}f_7 + v_{a2}^2 + u_{a2}v_{a2} + (u_{a2}^2 + u_{a1})(1 + u_{a2}f_7)) \\ + (v_{a1} + u_{a0}f_7 + v_{a2}^2 + u_{a2}v_{a2} + (u_{a2}^2 + u_{a1})(1 + u_{a2}f_7))^2 \\ + u_{a1} + (1 + u_{a2}f_7)(1 + u_{a1}^2) = 0,$$

then $\deg(u_c) = 2$ or 3 and we are in case HL32/33.

(2a) If $\deg(u_a) = 2$ and $u_{a1} \neq 0$, then $\deg(u_c) = 2$ and we are in case HL23.

(2b) If $\deg(u_a) = 2$ and $u_{a1} = 0$, then $\deg(u_c) = 1$ or 2 and we are in case HL21/23.

(3) If $\deg(u_a) = 1$, then $\deg(u_c) = 3$ and we are in case HL13.

We obtain the following halving formulas:

Algorithm 15 (HL33, $h(x) = x^3 + x + h_0$ irreducible, $f(x) = f_7x^7 + x^6 + f_2x^2 + f_1x + f_0$)

INPUT: $\overline{D} = [x^3 + u_{a2}x^2 + u_{a1}x + u_{a0}, v_{a2}x^2 + v_{a1}x + v_{a0}]$

OUTPUT: $[1/2]\overline{D} = [x^3 + u_{c2}x^2 + u_{c1}x + u_{c0}, v_{c2}x^2 + v_{c1}x + v_{c0}]$

```

1:  $s_0 \leftarrow \text{HT}(f_7u_{a2} + 1)$ ,  $s_1 \leftarrow v_{a2} + u_{a2}(s_0 + 1)$  ▷ 2M+1HT
2:  $s_2 \leftarrow v_{a1} + 1 + u_{a1}(s_0 + 1)$ ,  $s_3 \leftarrow v_{a0} + h_0 + u_{a0}(s_0 + 1)$  ▷ 2M
3:  $s_4 \leftarrow s_1f_7^{-1} + u_{a1}$ ,  $s_5 \leftarrow (s_2 + s_0 + s_1^2)f_7^{-1} + u_{a0} + s_4u_{a2}$  ▷ 3M+1S
4:  $s_6 \leftarrow (s_1 + (s_4 + 1)f_7)s_5 + (s_5f_7)^2$ ,  $s_7 \leftarrow \text{QR}(s_6)$  ▷ 3M+1S+1QR
5:  $s_8 \leftarrow s_7^2$ ,  $s_9 \leftarrow s_8s_5$ ,  $s_{10} \leftarrow s_9^{-1}$ ,  $s_{11} \leftarrow s_{10}s_8$ ,  $s_{12} \leftarrow s_{10}s_5^2$  ▷ 1I+3M+2S
6:  $s_{13} \leftarrow s_7s_{11}$ ,  $s_{14} \leftarrow s_{13}^2s_5 + f_7$ ,  $s_{15} \leftarrow (s_4 + 1)s_{14} + (h_0 + s_5)s_{13} + s_1$  ▷ 4M+1S
7:  $s_{16} \leftarrow s_{12}s_{15}$ ,  $s_{17} \leftarrow \text{TR}(s_{16}f_7^2)$  ▷ 2M+1TR
8: if  $s_{17} = 0$  then
9:    $s_0 \leftarrow s_0 + 1$ ,  $s_1 \leftarrow s_1 + u_{a2}$ ,  $s_2 \leftarrow s_2 + u_{a1}$ ,  $s_3 \leftarrow s_3 + u_{a0}$ 
10:   $s_4 \leftarrow s_4 + u_{a2}f_7^{-1}$ ,  $s_5 \leftarrow s_5 + (u_{a1} + 1)f_7^{-1}$  ▷ 2M
11:   $s_6 \leftarrow (s_1 + (s_4 + 1)f_7)s_5 + (s_5f_7)^2$ ,  $s_7 \leftarrow \text{QR}(s_6)$  ▷ 3M+1S+1QR
12:   $s_8 \leftarrow s_7^2$ ,  $s_9 \leftarrow s_8s_5$ ,  $s_{10} \leftarrow s_9^{-1}$ ,  $s_{11} \leftarrow s_{10}s_8$  ▷ 1I+2M+1S
13:   $s_{12} \leftarrow s_{10}s_5^2$ ,  $s_{13} \leftarrow s_7s_{11}$ ,  $s_{14} \leftarrow s_{13}^2s_5 + f_7$  ▷ 3M+2S
14:   $s_{15} \leftarrow (s_4 + 1)s_{14} + (h_0 + s_5)s_{13} + s_1$ ,  $s_{16} \leftarrow s_{12}s_{15}$  ▷ 3M
15: end if
16:  $s_{18} \leftarrow (s_3 + s_1 + s_0h_0)f_7^{-1} + s_4u_{a1} + s_5u_{a2}$ ,  $s_{19} \leftarrow s_{13}s_4$  ▷ 5M
17:  $s_{20} \leftarrow s_0 + 1 + s_{19}$ ,  $s_{21} \leftarrow (s_{14} + s_{13})(s_5 + s_4)$ ,  $s_{22} \leftarrow s_{14}s_5$  ▷ 2M
18:  $s_{23} \leftarrow s_1 + s_{19} + s_{21} + s_{22}$ ,  $s_{24} \leftarrow s_2 + 1 + s_{13}s_{18} + s_{22}$  ▷ 1M

```

19: $s_{25} \leftarrow h_0 + s_3 + s_{14}s_{18}, s_{26} \leftarrow (s_{23} + s_{20}h_0 + s_{25})s_{12}$	▷ 3M
20: $s_{27} \leftarrow (f_1 + s_{24}h_0 + s_{25})s_{12}, u_{c2} \leftarrow \sqrt{s_{16}}, u_{c1} \leftarrow \sqrt{s_{26}}$	▷ 2M+2SR
21: $u_{c0} \leftarrow \sqrt{s_{27}}, s_{28} \leftarrow s_{13}u_{c2}, s_{29} \leftarrow s_{14} + s_{28}, s_{30} \leftarrow s_{29}u_{c1}$	▷ 2M+1SR
22: $s_{31} \leftarrow s_{20} + (s_{13} + s_{29})(u_{c2} + u_{c1}) + s_{28} + s_{30}, s_{32} \leftarrow s_{31}u_{c0}$	▷ 2M
23: $v_{c0} \leftarrow s_{25} + s_{32}, v_{c1} \leftarrow s_{24} + (s_{29} + s_{31})(u_{c1} + u_{c0}) + s_{30} + s_{32}$	▷ 1M
24: $v_{c2} \leftarrow s_{23} + (s_{13} + s_{31})(u_{c2} + u_{c0}) + s_{28} + s_{32} + s_{30}$	▷ 1M
25: return $[x^3 + u_{c2}x^2 + u_{c1}x + u_{c0}, v_{c2}x^2 + v_{c1}x + v_{c0}]$	
▷ 2I+51M+9S+3SR+1TR+1HT+2QR	

We note that in these formulas a division by $s_{14} + f_7$ would normally be required to compute s_{16} , s_{26} and s_{27} . However, $s_{14} + f_7 = s_{13}^2 s_5$, which we can compute as $s_5 / (s_{13} s_5)^2$. Since $s_{13} = s_7 / s_5$, $1 / (s_{13} s_5) = 1 / s_7$ and we can combine this inverse with the computation of $1 / s_5$. As a result, we can compute both inverses using only 1I+3M+1S.

We therefore have a worst-case cost of 2I+49M+9S+3SR+1HT+1TR+2QR, which compares well with the doubling cost of 1I+63M+9S of Guyot, Kaveh and Patankar [13], as long as inversion costs are not too high.

However, the conditional block of lines 10 to 17 is only used when the initial “choice” for the root of $z^2 + z + u_{a2} + 1 = 0$ (i.e. $\text{HT}(u_{a2} + 1)$ rather than $\text{HT}(u_{a2} + 1) + 1$) is incorrect and the variables computed afterwards must be corrected. This means that the 1I+13M+4S+1QR associated to that correction will only be needed half of the time (on average). The average cost of the halving operation becomes 1.5I+42.5M+7S+3SR+1HT+1TR+1.5QR.

Remark 5. There is another approach to “optimise” the formulas, limiting ourselves to no more than one inversion per halving. The idea consists of doing the computations for both roots of $z^2 + z + u_{a2} + 1 = 0$ together until the computations of the inverses, at which points the two inverses can be combined into one using Montgomery’s trick (doing both in 1I+3M), after which we can use the normal branching approach. In this way, we get a worst-case cost of 1I+52M+9S+3SR+1HT+1TR+2QR, from which we expect to save 7M+2S when the first choice of the root is correct (half of the time). The final cost increases when an inversion costs less than 12M+2S+1QR, making this approach unlikely to be useful with many implementations of the field arithmetic (for field sizes used on genus-3 curves at standard cryptographic security levels).

4.5. DISCUSSION OF THE HALVING APPROACH. To obtain the formulas in this section, we inverted Cantor’s doubling algorithm rather than inverting the corresponding explicit formulas. Even though we used the general algorithm rather than the highly optimised version to obtain our formulas, we obtained operations that are more efficient.

At first glance, this could seem contradictory. After all, one of the main methods used in explicit formulas to produce such savings in comparison with Cantor’s algorithm is through the merging of the composition and the first reduction step. This merging is completely ignored in our approach, but the resulting formulas are still faster.

At the same time, the halving formulas must include the cost coming from choosing the “wrong” roots of quadratic equations, which naturally increases as the

quadratic equation is encountered earlier in the formula (since the condition to determine the “correct root” comes from whether or not the computed preimage can be halved again). As a consequence, one would expect the correction cost to increase as the degree of h also does, and in fact this is more or less what we observe ($h = x^2 + x + 1$ seems to be an exception to this rule of thumb). This means that the halving formulas get a higher penalty for selecting the correct preimage when the degree of h increases, and we do see this very clearly for the curves with h irreducible of degree 3.

Nevertheless, the absolute saving when comparing with the doubling seems to remain almost constant between the different curve types. In fact these apparent discrepancies come from the inherent difference between doubling and halving.

In the doubling, even with optimised explicit formulas, the composition step requires the computation of h^{-1} modulo u_a , or at least its almost-inverse, after which the reduction steps are relatively simple and straightforward. In fact, simply looking at the distribution of the cost in the different steps of the algorithm makes it quite clear that the composition, and in particular the computation of the almost-inverse, is one of the dominant factors.

For the halving, we work our way backwards through the reductions steps until we obtain $[u_0, v_0]$. In general, the cost of an “un-reduction” step may be higher than for the corresponding reduction step, but this increase is usually small. Once $[u_0, v_0]$ is known, computing u_c (the first polynomial of the output) only requires computing the square-root of u_0 , while v_c is obtained by reducing v_0 modulo u_c .

These last two operations are quite inexpensive, requiring a total of 6M and 3SR, no matter what form the curve has. In comparison, the composition step, even when merged with the first reduction, requires the computation of h^{-1} modulo u_a (or an almost-inverse), which becomes much more costly as the degree of h increases. The savings obtained by switching from almost-inverse to modular reduction (from doubling to halving) are therefore much greater when h becomes more complicated, and easily compensate for any of the “inconveniences” of halving that we just described.

This also explains why the costs of doubling and halving are essentially identical when $h = 1$: in that case, h^{-1} comes for free and the optimised doubling does not merge the composition and first reduction, making doubling and halving perfect mirror images of each other.

5. CONCLUSION

We have investigated doubling and halving of divisor classes of hyperelliptic curves of genus 3 over binary fields. In case $h(x) = 1$ we get best performance for both doubling and halving of a divisor class. In this “optimal performance” case we provide halving formulas that are as efficient as the appropriate doubling ones. Previously, explicit doubling formulas were known only for the most frequent case. We extended this by adding explicit doubling formulas for all special cases, and also halving formulas for all possible cases. This allows a complete implementation of a DLP-based cryptosystem using genus 3 curves.

For three further (and more general) classes of genus 3 curves we provide halving formulas that are noticeable faster than the associated doubling ones; We achieve a speed-up of 10 to 20 field multiplications in each case.

Those explicit formulas were found by a new method. We did not invert the doubling formulas to get the halving ones, but we reversed the combination and the

reduction step of Cantor’s algorithm. This turned out to be a better way to get more efficient operations.

To our knowledge, no explicit halving formulas for genus 3 curves have been available until now. Thus, this paper can be considered the first result on efficient halving on the divisor class group of genus 3 curves over finite fields of characteristic 2. As for the doubling case there are only formulas for the most frequent case published so far. In the present paper we extended this by all the missing special cases, where the first polynomial $u(x)$ in the Mumford representation is not of degree 3.

REFERENCES

- [1] Roberto M. Avanzi, *A note on square roots in binary fields*, preprint.
- [2] Roberto M. Avanzi, Henri Cohen, Christophe Doche, Gerhard Frey, Tanja Lange, Kim Nguyen and Frederik Vercauteren, “Handbook of Elliptic and Hyperelliptic Curve Cryptography”, Chapman & Hall/CRC, 2006.
- [3] Roberto M. Avanzi and Nicolas Thériault, *Effects of optimizations for software implementations of small binary field arithmetic*, in “International Workshop on the Arithmetic of Finite Fields – WAIFI 2007”, Lecture Notes in Computer Science, **4547** (2007), 69–84.
- [4] Roberto M. Avanzi, Nicolas Thériault and Zheng Wang, *Rethinking low genus hyperelliptic Jacobian arithmetic over binary fields: Interplay of field arithmetic and explicit formulæ*, J. Math. Crypt. **2** (2008), 227–255.
- [5] Peter Birkner, *Efficient divisor class halving on genus two curves*. in “13th Annual Workshop on Selected Areas in Cryptography – SAC 2006”, Lecture Notes in Computer Science, **4356** (2007), 317–326.
- [6] Peter Birkner and Nicolas Thériault, *Faster halvings in genus 2*, in: “15th Annual Workshop on Selected Areas in Cryptography – SAC 2008”, Lecture Notes in Computer Science, **5381** (2008), 1–16.
- [7] Peter Birkner and Nicolas Thériault, *Efficient Halving for Genus 3 Curves over Binary Fields: Extended Version*, available from <http://www.homepage.mac.com/ntheriault> and from <http://www.peter-birkner.de>
- [8] Xinxin Fan, Thomas J. Wollinger and Yumin Wang, *Efficient doubling on genus 3 curves over binary fields*, in “Topics in Cryptology – CT-RSA 2006”, Lecture Notes in Computer Science, **3860** (2006), 64–81.
- [9] Kenny Fong, Darrel Hankerson, Julio López and Alfred Menezes, *Field inversion and point halving revisited*, IEEE Trans. Computers, *53* no. 8 (2004), 1047–1059.
- [10] Pierrick Gaudry, *Index calculus for abelian varieties and the elliptic curve discrete logarithm problem*, preprint, IACR ePrint 2004/073.
- [11] Pierrick Gaudry, Florian Hess and Nigel P. Smart, *Constructive and destructive facets of Weil descent on elliptic curves*, J. Cryptology, **15** no. 1 (2002), 19–46.
- [12] Izuru Kitamura, Masanobu Katagi, and Tsuyoshi Takagi, *A complete divisor class halving algorithm for hyperelliptic curve cryptosystems of genus two*, in “Information Security and Privacy – ACISP 2005”, Lecture Notes in Computer Science, **3574** (2005), 146–157.
- [13] Cyril Guyot, Kiumars Kaveh and Vijay M. Patankar, *Explicit algorithm for the arithmetic on the hyperelliptic Jacobians of genus 3*, J. Ramanujan Math. Soc, **19** (2004), 119–159.
- [14] Erik Woodward Knudsen, *Elliptic scalar multiplication using point halving*, in “Advances in Cryptology – Asiacrypt 1999”, Lecture Notes in Computer Science, **1716** (1999), 135–149.
- [15] Rudolf Lidl and Harald Niederreiter, “Finite Fields”, 2nd ed., Cambridge University Press, 1997.
- [16] Jasper Scholten and Hui June Zhu, *Hyperelliptic curves in characteristic 2*, Inter. Math. Research Notices, **17** (2002), 905–917.
- [17] Richard Schroepel, *Elliptic curves: Twice as fast!*, Crypto 2000 Rump Session.
- [18] Nicolas Thériault, *Weil descent for Artin-Schreier curves*, preprint, available from: <http://www.homepage.mac.com/ntheriault>

E-mail address: p.birkner@tue.nl

E-mail address: ntheriau@inst-mat.utalca.cl