

Explicit algorithm for the arithmetic on the hyperelliptic Jacobians of genus 3

Cyril Guyot*

Kasten Chase Applied Research
Orbitor Place, 5100 Orbitor Drive
Mississauga, Ontario L4W 4Z4, Canada
cyril@zoy.org

Kiumars Kaveh*

1984 Mathematics Road
Department of Mathematics
University of British Columbia
Vancouver, British Columbia V6T 1Z2, Canada
kaveh@math.ubc.ca

Vijay M. Patankar*

100 St. George Street
Department of Mathematics
University of Toronto
Toronto, Ontario, Canada M6G 2M6
vijay@math.utoronto.ca

Abstract

We investigate efficient formulae to double and add divisors on the Jacobian of a hyperelliptic curve of genus 3. The main contributions of this paper are as follows: (1) Overall improvements in the complexity of the addition and doubling algorithms for both even and odd characteristics, (2) Algorithms applicable to almost all hyperelliptic curves of genus 3, and (3) Efficient computation of the resultant of two polynomials and of the inverse of one polynomial modulo another. This paper is specifically written in an implementation-ready format.

*Most of the work in this paper was carried out while the first and the last authors were employed by Kasten Chase Applied Research. We wish to thank the said organisation for providing us with a stimulating and encouraging work environment. All the authors wish to thank Karthika Technologies for providing the research atmosphere where the research on this topic had begun.

KEYWORDS: CRYPTOGRAPHY, EXPLICIT ALGORITHM, GENUS 3, HYPERELLIPTIC CURVES

AMS CLASSIFICATION NUMBERS: 94A60, 14G50

1 Introduction

1.1 Motivation and Earlier Work

Essential to Public Key Cryptography is the existence of a cyclic group for which the discrete logarithm (DL) problem is difficult to solve. The first Public Key Cryptosystem was introduced in 1976 by Diffie and Hellman [3] and is based on the difficulty of solving the discrete logarithm (DL) problem over finite fields. Later, Koblitz [14] and Miller [23] initiated the study of Public Key Cryptography based on groups of points of elliptic curves over finite fields, known as Elliptic Curve Cryptography (ECC). ECC has since been extensively studied from both a pure and applied perspective. Later, Koblitz [15, 16] proposed the use of higher genus hyperelliptic curves in Public Key Cryptosystems.

The following are the main lines of research towards achieving an efficient and secure implementation of hyperelliptic curve cryptography (HECC):

1. Finding hyperelliptic curves such that the order of the group of points on the Jacobian of the curve over a finite field is divisible by a large prime. Much work has been done to find efficient algorithms to count points on the Jacobian of hyperelliptic curves of higher genus. This has notably been studied by Schoof, Elkies and Atkin [32, 33, 1], Satoh (for genus 1) [35], Gaudry and Harley [9], Kedlaya [13], Weng [37], Denef-Vercauteren [4, 5, 6]. A. Weng in her work [37] has a method of constructing hyperelliptic curves of genus 3 with complex multiplication which can be used for cryptography.

2. Security of HECC and its comparison with the security of ECC. The work of Gaudry, Hess and Smart [9] have shown that Jacobians of hyperelliptic curves of genus higher than 4 are amenable to attack. This attack known as *Weil-Descent (or GHS) attack* has better complexity than the *Square Root Attack* [1]. Recently, N. Thériault has proposed a new attack in his thesis work [36], called the *Large Prime Attack* which works for genus 4 hyperelliptic curves. (Gaudry says he was aware of such an attack.) His attack also works for genus 3 curves but requires a large amount of storage. Thériault's work [36] shows that, in order to achieve cryptographic security equivalent to the one provided by an elliptic curve defined over a field of size $\log q$ it is necessary to use a hyperelliptic curve of genus 3 defined over a field of size at least $\frac{7 \log q}{20}$.

3. Algorithms for the arithmetic on the Jacobians. Seminal here is the work of Cantor [2] which gives a generic algorithm for the arithmetic on Jacobians of curves of any genus. For odd characteristic, there is an exposition of this in a short note by Murty [25]. The complexity of this algorithm of Cantor is analysed by A. Stein in [34]. Subsequently, Harley (genus 2) [12], Nagao

et al. (genus 3) [26], Kuroki et al. (genus 3) [18], Matsuo et al. (genus 2) [22], Lange (for genus 2) [19, 20, 21], Pelzl et al. (genus 2,3,4) [28] improved previous works to obtain algorithms with better complexity. Here and throughout the rest of the paper, by *Complexity of an Algorithm* (in the context of arithmetic on the Jacobians of hyperelliptic curves over finite fields \mathbb{F}_q), we mean the total number of field arithmetic operations (inversion, multiplication, squaring and addition) that are needed for the algorithm. Throughout the paper, I , M , and S will respectively stand for Inversion, Multiplication and Squaring. Since the time complexity of a field addition is negligible compared to that of the rest (inversion, multiplication and squaring), we will ignore addition.

1.2 This Work

We started working on this project with the aim of investigating whether the complexity of genus 3 HECC is comparable to that of ECC. Towards that end we have the following contributions:

(1) We present a different method for computing the resultant of two polynomials which enables us to reuse certain computations and achieve an improvement in the computation of the resultant and the inverse of a polynomial modulo another. This improves the global complexity of the doubling and adding algorithms.

For instance, our computation of the resultant and the inverse (needed for the addition algorithms) takes $16M$ while the same computation by Pelzl et al. [28] takes $16M + 2S$.

(2) Our algorithms apply to almost all hyperelliptic curves of genus 3 (up to isomorphism). Our results are general and without any of the restrictions imposed in the works of Kuroki et al. [18], Pelzl et al. [28]. For instance, if $y^2 + h(x)y = f(x)$ is an equation of a hyperelliptic curve of genus 3, Pelzl et al. [28] work under the restrictive assumption of $h \in \mathbb{F}_2[x]$, whereas we do not make such an assumption.

(3) We interleave and fine-tune the details of the algorithms to ensure that no superfluous operation is done. This improves the complexity of the algorithms by 8% for addition in odd characteristic, by 1% for doubling in odd characteristic, by 5% for addition in even characteristic (even though our algorithm is more general) and by 11% for addition and 21% for doubling in even characteristic for $h \equiv 1$. These improvements have the consequence that *HECC can be more efficient than ECC in certain cases*. This answers affirmatively, a question raised by Koblitz at a MSRI Symposium in January 2002: “Is there an explicit family of hyperelliptic curves such that, for such a family, HECC is more efficient than (an equivalently secure) ECC?”. This point is further analysed in the summary and conclusion section of this paper.

Finally, we note that preliminary versions of these algorithms have been successfully implemented.

1.3 Notation, Preliminaries and Setup

Let \mathbb{F}_q be a finite field where $q = p^n$, where p is prime. Let C be a hyperelliptic curve over \mathbb{F}_q defined by $y^2 + h(x)y = f(x)$, where $h(x)$ and $f(x)$ are polynomials over \mathbb{F}_q such that $\deg(f) = 7$ and $\deg(h) \leq 3$. Thus, $f(x) = f_7x^7 + f_6x^6 + \dots + f_0$ and $h(x) = h_3x^3 + \dots + h_0$.

Using a birational transformation of the form $(x, y) \mapsto (\lambda x + \mu, \nu y)$ one can simplify the equation of the curve. The following table lists the simplifications with the conditions under which they can be realised.

Conditions	$p = 2$ $2, 3 \nmid n$ $\deg(h) = 3$	$p = 2$ $2, 3 \nmid n$ $\deg(h) = 2$	$p = 2$ $2, 3 \nmid n$ $\deg(h) = 1$	$p = 2$ $2, 3 \nmid n$ $\deg(h) = 0$	p odd $p \neq 7$
f and h	f, h monic, $h_2 = 0$	f, h monic, $f_6 = 0$	f, h monic, $h_0 = 0$	f monic, $h \equiv 1$	f monic, $f_6 = 0$, $h \equiv 0$

Note that for $p = 7$, a Tschirnhausen transformation can not be used to make $f_6 = 0$. This fact combined with the inefficiency of the field arithmetic for \mathbb{F}_q when $q = 7^n$ makes such curves unattractive for software implementation.

Also, the assumption that n is not a multiple of 2 nor 3 is reasonable from a cryptography perspective since the existence of subfields of \mathbb{F}_{2^n} of smaller degree may facilitate attacks as pointed out by [9].

In lieu of the above, in this work, we restrict ourselves to the following assumptions:

- If p odd then $p \neq 7$, which implies $h \equiv 0$, f monic and $f_6 = 0$
- If $p = 2$ and $q = p^n$ then $2, 3 \nmid n$, which implies h and f both monic

Throughout the paper we also follow the following conventions. For a polynomial g , the polynomial obtained by making g monic by dividing by the leading non-zero coefficient will be denoted by $\text{Monic}(g)$. Also the polynomial q where $f = qg + r$, with $\deg(r) < \deg(g)$ will be denoted by $q := \left\lfloor \frac{f}{g} \right\rfloor$. In the appendices, we highlight in **bold face** the computations which contribute to the complexity of the corresponding algorithms.

Layout of the paper

Section 2 describes the addition algorithm for characteristic 2. Section 3 describes the doubling algorithm for characteristic 2. Section 4 describes the doubling algorithm for characteristic 2

when $h \equiv 1$. Section 5 and 6, respectively describe the addition and doubling algorithms for odd characteristic. The final section 7 is the summary and conclusion section. In the appendices A to F, we describe the computation of the resultant and give explicit details of the various addition and doubling algorithms.

1.4 Preliminaries

Arithmetic

Let $J(C)$ be the Jacobian of the hyperelliptic curve C of genus g defined over \mathbb{F}_q by the affine equation

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

where $\deg(f) = 2g + 1$ and $\deg(h) \leq g$. Let $P = (x, y)$ be a point on $C(\mathbb{F}_q)$. It then follows that $\tilde{P} := (x, -y - h(x))$ also lies on $C(\mathbb{F}_q)$. The point \tilde{P} will be called the *opposite* of P . (Note that $\tilde{\tilde{P}} = P$.) Let $P_\infty := (0 : 1 : 0)$ be the (unique) *point at infinity* on the projective closure of C . Let \mathcal{O} denote the additive identity (“zero”) on $Jac(C)$. We then have, $(P - P_\infty) + (\tilde{P} - P_\infty) = \mathcal{O}$ on $J(C)$.

Let $D := \sum_P m_P P$, where $P \in C \cup P_\infty$ be a degree 0 divisor on C . Then, it can be shown that $D = D_{eff} - mP_\infty$ on $J(C)$, for some *effective* divisor D_{eff} on C given by $\sum_P m_P P$ with $P \in C$ and $m_P \geq 0$. Furthermore, we can assume that:

- (i) If $P \neq \tilde{P}$ and $m_P > 0$, then $m_{\tilde{P}} = 0$,
- (ii) If $P = \tilde{P}$ and $m_P > 0$, then $m_P = 1$.

A divisor of the form $D_{eff} - mP_\infty$, with D_{eff} as above, is called a *semi-reduced* divisor on C . A semi-reduced divisor on C is called *reduced* if $m = \sum_P m_P \leq g$. It is a consequence of Riemann-Roch theorem that, every element of $J(C)$ can be represented by a *unique* reduced divisor as above (i.e. every semi-reduced divisor can be further *reduced* to a reduced divisor). This *reduced* representation of D is called the *reduction* of D and m is called the *weight* or *reduced-degree* of D .

To a semi-reduced divisor one can associate a pair of polynomials (u, v) defined over \mathbb{F}_q such that the ideal $(u, y - v)$ represents D_{eff} with u monic and $\deg(v) < \deg(u) = m$ and satisfying $u \mid (v^2 - vh - f)$. By abuse of notation, we then write $D = (u, v)$. In particular, for a reduced divisor D one can associate a pair (u, v) with the further condition that $\deg(u) = m \leq g$. This association is now one-to-one, and the associated pair (u, v) is called the *Mumford representation* [24] of D . By abuse of notation, we then write $weight(D) := m = \deg(u)$.

In particular, for a hyperelliptic curve C of genus 3, to a divisor D on $J(C)(\mathbb{F}_q)$ corresponds a unique pair of polynomials (u, v) both defined over \mathbb{F}_q with u monic, such that $\deg(u) = m \leq 3$, $\deg(v) < m = \deg(u)$, and $u \mid (v^2 + hv - f)$.

Conversely, to a pair of polynomials u_1, v_1 over \mathbb{F}_q satisfying $u_1 \mid (v_1^2 + hv_1 - f)$, and $\deg(v_1) < \deg(u_1)$ corresponds a semi-reduced divisor D on $J(C)(\mathbb{F}_q)$. This associated D need not be

reduced. Its reduction gives D_{red} and its corresponding Mumford representation (u, v) is as stated above.

Generic Divisors

- A divisor $D = (u, v)$ is said to be *generic* if it is of *weight* (or reduced-degree) 3, and $support(D)$ does not have a point with multiplicity greater than 1.
- Two generic divisors $D_A = (u_A, v_A)$ and $D_B = (u_B, v_B)$ are said to be in a *generic position for addition* if $support(D_A) \cap support(D_B) = \emptyset$.
- A generic divisor $D_A = (u_A, v_A)$ is said to be in a *generic position for doubling* if $support(D_A)$ does not contain a point of order 2.

In this paper we will deal only with divisors in a generic position. In fact, a heuristic calculation shows that the probability of randomly chosen divisors to not be in a generic position, is of the order of $1/q$. In consequence, we only need to be optimise the arithmetic for the generic cases. In the non-generic cases, one can employ the Cantor algorithm without affecting the overall performance (time complexity).

Addition and Doubling

Let $D_A = (u_A, v_A)$ and $D_B = (u_B, v_B)$ be two divisors on C in a generic position for addition. The basic idea behind the computation of $D_A + D_B$ is simple [12]. The first step is to find a semi-reduced divisor D_{comp} with Mumford representation u_{comp}, v_{comp} which represents $D_A + D_B$. One then reduces this further to get the reduced (unique) Mumford representation for $D_A + D_B$.

Let $Z := support(D_A) \cup support(D_B)$. Then, it follows that $u_{comp} = u_A u_B$. We now want to a polynomial v_{comp} (of smallest degree) such that the curve $y = v_{comp}$ passes through Z with proper multiplicity. Explicitly, this “passing through Z condition” translates into a pair of congruence conditions mod u_A and mod u_B .

$$v_{comp} \equiv \begin{cases} v_A & \text{mod } u_A \\ v_B & \text{mod } u_B \end{cases}$$

This gives us D_{comp} . Further reduction gives us the reduced Mumford representation for $D_A + D_B$. (The details are in sections 2 and 5.)

The idea for doubling is similar and is as follows: for a given divisor D_A in generic position for doubling, one first computes a semi-reduced divisor D_{comp} with Mumford representation (u_{comp}, v_{comp}) representing $2D_A$. One then reduces it to get the reduced (unique) Mumford representation for $2D_A$. Let $Z := 2 \cdot support(D_A)$. We then have $u_{comp} = u_A^2$. We now want a

polynomial v_{comp} such that the curve $y = v_{comp}$ passes through Z . This “passing through Z condition” translates into a congruence condition: $v_{comp} \equiv v_A \pmod{u_A}$. The fact that (u_{comp}, v_{comp}) be a Mumford representation corresponds to another congruence condition:

$$v_{comp}^2 + hv_{comp} - f \equiv 0 \pmod{u_{comp}}$$

This gives us D_{comp} . Further reduction gives us the reduced Mumford representation for $2D_A$. (The details are in sections 3, 4, and 6.)

Resultant

The arithmetic on the Jacobian requires computation of the resultant of two polynomials. The resultant of two polynomials A and B of degrees n and m respectively, can be expressed as the determinant of the associated Sylvester matrix of size $m + n$. Thus,

$$Resultant(A, B) = \begin{vmatrix} a_n & a_{n-1} & \cdots & a_1 & a_0 & 0 & 0 & \cdots & 0 \\ 0 & a_n & a_{n-1} & \cdots & a_1 & a_0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & 0 & a_n & a_{n-1} & \cdots & a_1 & a_0 & 0 \\ 0 & \cdots & 0 & 0 & a_n & a_{n-1} & \cdots & a_1 & a_0 \\ b_m & b_{m-1} & \cdots & b_1 & b_0 & 0 & 0 & \cdots & 0 \\ 0 & b_m & b_{m-1} & \cdots & b_1 & b_0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & 0 & b_m & b_{m-1} & \cdots & b_1 & b_0 & 0 \\ 0 & \cdots & 0 & 0 & b_m & b_{m-1} & \cdots & b_1 & b_0 \end{vmatrix}$$

Our algorithms require the computation of the resultant of polynomials of degree 3 and degree 2, which we compute using the Sylvester *determinant* as above. The details are in Appendix A.

2 Addition of generic divisors in characteristic 2

Let C be a hyperelliptic curve of genus 3 defined over a field \mathbb{F}_{2^n} given by

$$y^2 + h(x)y = f(x).$$

Let $D_A = (u_A, v_A)$ and $D_B = (u_B, v_B)$, be two divisors in a generic position for addition.

(a) **Addition:** Given two divisors as above, the first step is to get a divisor

$$D_{comp} = D_A + D_B,$$

where $\text{weight}(D_{\text{comp}}) \leq 3$ and the inequality may be strict. Let $D_{\text{comp}} := (u_{\text{comp}}, v_{\text{comp}})$, where

$$\begin{aligned} u_{\text{comp}} &= u_A \cdot u_B \\ v_{\text{comp}} &\equiv \begin{cases} v_A & \text{mod } u_A \\ v_B & \text{mod } u_B \end{cases} \end{aligned}$$

Let S be such that $v_{\text{comp}} = v_A + u_A S$. The Chinese remainder theorem gives

$$S = (v_B - v_A) \left(\frac{1}{u_A} \text{ mod } u_B \right) \text{ mod } u_B.$$

This gives us D_{comp} which we need to reduce twice in succession to get a divisor of $\text{weight} \leq 3$.
(b) Reduction: The first reduction gives us a divisor $D_T := (u_T, v_T) := \text{reduced}(D_{\text{comp}})$, where

$$\begin{aligned} u_T &= \frac{v_{\text{comp}}^2 + hv_{\text{comp}} - f}{u_A u_B} \\ v_T &= -h - v_{\text{comp}} \text{ mod } u_T \end{aligned}$$

Since $\deg(u_T) = 4$ and $\deg(v_T) = 3$, D_T needs to be reduced. Let $D_E := (u_E, v_E) := \text{reduced}(D_T)$, where

$$\begin{aligned} u_E &= \frac{v_T^2 + hv_T - f}{u_T} \\ v_E &= -h - v_T \text{ mod } u_E \end{aligned}$$

It is easy to check that $\text{weight}(D_E) \leq 3$.

In practice, the first two steps are combined to compute D_T directly. A final reduction of D_T gives D_E .

The details of the algorithm and the associated table are in Appendix B.

3 Doubling of a generic divisor in characteristic 2

Let C be a hyperelliptic curve of genus 3 defined over a field \mathbb{F}_{2^n} given by $y^2 + h(x)y = f(x)$. Let $D_A = (u_A, v_A)$ be a divisor in a generic position for doubling. We want to compute $D_E = 2D_A$ such that $\text{weight}(D_E) \leq 3$. This is carried out in two main steps:

(a) Doubling: Let $D_{\text{comp}} := (u_{\text{comp}}, v_{\text{comp}})$, where

$$\begin{aligned} u_{\text{comp}} &= u_A^2 \\ v_{\text{comp}} &\equiv v_A \text{ mod } u_A \\ v_{\text{comp}}^2 + hv_{\text{comp}} - f &\equiv 0 \text{ mod } u_A^2 \end{aligned}$$

Writing $v_{comp} =: v_A + Su_A$, for a polynomial S , it follows from the Chinese remainder theorem that:

$$S = \left(\left(\frac{v_A^2 + v_A h - f}{u_A} \right) \left(\frac{1}{h} \mod u_A \right) \right) \mod u_A$$

From the above it follows that $\deg(S) \leq 2$. Let $S := s_2 x^2 + s_1 x + s_0$.

This gives us D_{comp} which needs to be reduced twice in succession to get a divisor of *weight* ≤ 3 .

(b) *Reduction*: In practice, the reduction is combined with the computation of D_{comp} to directly compute $D_T := (u_T, v_T)$, given by

$$\begin{aligned} u_T &= \text{Monic} \left(S^2 + \left[\frac{Sh}{u_A} \right] + \left[\frac{v_A^2 + v_A h - f}{u_A^2} \right] \right) \\ v_T &= -h - v_{comp} = h + v_A + Su_A \mod u_T, \end{aligned}$$

Note that, since $\deg(v_A^2 + v_A h - f) = 7$ and $\deg(u_A^2) = 6$, $\left[\frac{v_A^2 + v_A h - f}{u_A^2} \right]$ has degree 1. Since $\text{weight}(D_T) = 4$, we reduce this to $D_E := (u_E, v_E) := \text{reduced}(D_T)$ of *weight* 3, where

$$\begin{aligned} u_E &= \frac{v_T^2 + hv_T - f}{u_T} \\ v_E &= -h - v_T \mod u_T \end{aligned}$$

The details of the algorithm and the associated table are in Appendix C.

4 Doubling of a generic divisor in characteristic 2 with $h \equiv 1$

Let C be a hyperelliptic curve of genus 3 defined over a field \mathbb{F}_{2^n} given by an equation of the form $y^2 + y = f(x)$. Let $D_A = (u_A, v_A)$ be a divisor in a generic position for doubling. We want to compute $D_E = 2D_A$ such that $\text{weight}(D_E) \leq 3$. This is carried out in two main steps.

(a) *Doubling*: Let $D_{comp} := (u_{comp}, v_{comp})$, where

$$\begin{aligned} u_{comp} &= u_A^2 \\ v_{comp} &\equiv v_A \mod u_A \\ v_{comp}^2 + hv_{comp} + f &\equiv 0 \mod u_A^2 \end{aligned}$$

It is easy to check that:

$$\begin{aligned} u_{comp} &= u_A^2 \\ v_{comp} &= (v_A^2 + f) \mod u_A^2 \end{aligned}$$

This gives us D_{comp} , which needs to be reduced twice in succession to get a divisor of *weight* ≤ 3 .

(b) **Reduction:** Reduction of D_{comp} gives $D_T := (u_T, v_T)$.

$$\begin{aligned} u_T &= \text{Monic} \left(\frac{v_{comp}^2 + v_{comp} + f}{u_{comp}} \right) \\ v_T &= 1 + v_{comp} \pmod{u_T}, \end{aligned}$$

Since $\text{weight}(D_T) = 4$, we reduce this further to $D_E := (u_E, v_E) := \text{reduced}(D_T)$ of *weight* 3, where

$$\begin{aligned} u_E &= \frac{v_T^2 + v_T + f}{u_T} \\ v_E &= 1 + v_T \pmod{u_T} \end{aligned}$$

The details of the algorithm and the associated table are in Appendix D.

5 Addition of generic divisors in odd characteristic

Let C be a hyperelliptic curve of genus 3 defined over a field \mathbb{F}_q of odd characteristic given by $y^2 = f(x)$. Let $D_A = (u_A, v_A)$ and $D_B = (u_B, v_B)$ be two divisors in a generic position for addition. We want to compute $D_E = (u_E, v_E) = D_A + D_B$ such that $\text{weight}(D_E) \leq 3$. The algorithm consists of two main steps.

(a) **Addition:** The first step is to get a divisor $D_{comp} = D_A + D_B$ where $\text{weight}(D_{comp})$ is not necessarily less than or equal to 3. Let $D_{comp} := (u_{comp}, v_{comp})$, where

$$\begin{aligned} u_{comp} &= u_A u_B \\ v_{comp} &\equiv \begin{cases} v_A \pmod{u_A} \\ v_B \pmod{u_B} \end{cases} \end{aligned}$$

Let S be such that $v_{comp} = v_A + u_A S$. By the Chinese remainder theorem it follows that

$$S = \left((v_B - v_A) \left(\frac{1}{u_A} \pmod{u_B} \right) \right) \pmod{u_B}.$$

This gives us D_{comp} . We need to reduce D_{comp} twice in succession to get a divisor of *weight* ≤ 3 .

(b) **Reduction:** This is done in the following two consecutive reduction steps.

Let $D_T := (u_T, v_T)$, where

$$\begin{aligned} u_T &= \text{Monic} \left(\frac{v_{comp}^2 - f}{u_A u_B} \right) \\ v_T &\equiv -v_{comp} \pmod{u_T} \end{aligned}$$

It follows that $\deg(u_T) = 4$ and $\deg(v_T) = 3$. Thus, we need to reduce D_T . Let $D_E := \text{reduced}(D_T) := (u_E, v_E)$. Then,

$$\begin{aligned} u_E &= \frac{v_T^2 - f}{u_T} \\ v_E &= -v_T \mod u_E \end{aligned}$$

It is easy to check that $\text{weight}(D_E) = 3$. In practice, the first two steps are combined to compute D_T directly. The final reduction produces D_E .

The details of the algorithm and the associated table are in Appendix E.

6 Doubling of a generic divisor in odd characteristic

Let C be a hyperelliptic curve of genus 3 defined over a field \mathbb{F}_q of odd characteristic given by $y^2 = f(x)$. Let $D_A = (u_A, v_A)$ be a divisor in a generic position for doubling. We want to compute $D_E = 2D_A$ such that $\text{weight}(D_E) \leq 3$. The algorithm consists of two main steps.

(a) Doubling: Let $D_{\text{comp}} := (u_{\text{comp}}, v_{\text{comp}})$, where

$$\begin{aligned} u_{\text{comp}} &= u_A^2 \\ v_{\text{comp}} &\equiv v_A \mod u_A \\ v_{\text{comp}}^2 - f &\equiv 0 \mod u_A^2 \end{aligned}$$

This gives us D_{comp} . The divisor D_{comp} needs to be reduced twice in succession to get a divisor D_E of weight 3. Writing $v_{\text{comp}} = v_A + Su_A$, for a polynomial S , it follows that:

$$S = \left(\left(\frac{f - v_A^2}{u_A} \right) \left(\frac{1}{2v_A} \mod u_A \right) \right) \mod u_A$$

From the above it follows that $\deg(S) \leq 2$. Let $S := s_2x^2 + s_1x + s_0$.

(b) Reduction: In practice, the reduction is combined with the computation of D_{comp} to directly compute $D_T := (u_T, v_T)$, given by

$$\begin{aligned} u_T &= \text{Monic} \left(S^2 + \left[\frac{2Sv_A}{u_A} \right] + \left[\frac{v_A^2 - f}{u_A^2} \right] \right) \\ v_T &= -v_{\text{comp}} = -v_A - Su_A \mod u_T, \end{aligned}$$

where $\left[\frac{f}{g} \right] := q$ where $f = qg + r$, with $\deg(r) < \deg(g)$. Note that, since $\deg(v_A^2 - f) = 7$, $\deg(u_A^2) = 6$ and $f_6 = 0$, $\left[\frac{v_A^2 - f}{u_A^2} \right] = x$. Since $\deg(u_T) = 4$, it is reduced further to get $D_E := (u_E, v_E) := \text{reduced}(D_T)$ of weight 3, where

$$\begin{aligned} u_E &= \frac{v_T^2 - f}{u_T} \\ v_E &= -v_T \mod u_T \end{aligned}$$

The details of the algorithm and the associated table are in Appendix F.

7 Analysis, Summary and Conclusions

The following table compares the earlier known complexity results to those of this work.

In the table below, M/S represents either a multiplication or a square, when they are not counted as distinct operations.

authors	characteristic	curve properties	addition	doubling
Cantor [26]	odd	$h \equiv 0$	4I+200M/S	4I+207M/S
Nagao [26]	odd	$h \equiv 0$	2I+144M/S	2I+153M/S
Kuroki et al. [18]	odd	$h \equiv 0, f_6 = 0$	I+81M/S	I+74M/S
Pelzl et al. [28]	odd	$f_6 = 0$	I+70M+6S	I+61M+10S
Pelzl et al. [28]	two	$h_i \in \mathbb{F}_2, f_6 = 0$	I+65M+6S	I+53M+10S
Pelzl et al. [28]	two	$h \equiv 1, f_6 = 0$	I+65M+6S	I+14M+11S
This work	odd	$f_6 = 0$	I+64M+6S	I+61M+9S
This work	two	$\deg(h) = 3, h_2 = 0$	I+62M+5S	I+63M+9S
This work	two	$h \equiv 1, f_6 = 0$	I+58M+6S	I+11M+11S

A further comparative analysis shows that in general, ECC (with algorithms that use projective co-ordinates as in Dahab-Lopez) is always better than genus 3 HECC, except possibly, when $h \equiv 1$ and the “inversion-to-multiplication” ratio is low enough. Here, by *inversion-multiplication ratio* or in short *I/M-ratio*, we mean the ratio of the time needed for an inversion of a field element to the time needed for the multiplication of two field elements. This ratio depends on a few factors such as the characteristic of the field, the size of the field, the specific implementation of the associated field-arithmetic library, and the processor type. (*Please note that all our final calculations for the comparative graphs are for a 32-bit processor.*)

Figure 1 describes this comparative analysis in the context of our work while taking into account the various factors mentioned above. The diagram follows the approach described in Pelzl et al. [28] which aims at measuring the efficiency (or complexity) of the scalar multiplication (“ kP ” where k is of the order of the cardinality of the Jacobian) by counting the number of “Atomic Operations” required. (By an *Atomic Operation* we mean the basic processor operations such as word-shifts or XOR-s.)

This metric is calculated as follows: Recall that the scalar multiplication “ kP ” is computed by expressing k in Non-Adjacent-Form (binary expansion with ± 1 and no two consecutive terms non-zero) and using window NAF method with window of size 5, as explained in [11]. Roughly, this corresponds to $(\log_2 k)$ doublings and $(\frac{\log_2 k}{6})$ additions. Thus, the cost of the computation of “ kP ” is proportional to $(\log_2 k)D + (\frac{\log_2 k}{6})A$, where D and A denote respectively the cost of one

doubling and of one addition in the group. This can in turn be expressed in terms of the basic field operations, and then finally, in terms of the *atomic operations*. The most expensive field operations are the field inversion and the field multiplication, in that order. (In the case of fields of characteristic two, the field addition and the field squaring are not taken into consideration because of their relative low complexities.) By replacing each *inversion* by $(I/M\text{-ratio}) \cdot \text{multiplication}$, one gets an expression for the total number of atomic operations as a function of the I/M-ratio. In the finer details of the construction of this metric, one also needs to take into account the processor-word-size.

Figure 1 compares the efficiency of the scalar multiplication on the elliptic curves (in both affine and Lopez-Dahab projective coordinates), and on the Jacobians of hyperelliptic curves of genus 3, with $h \equiv 1$. Here, the security provided by the cryptosystems is chosen to be “equivalent” to the one provided by ECC over a field of size 571 bits. (In this *equivalence* between ECC and HECC, we have taken into consideration the reduction in strength stemming from the index calculus attacks on genus 3 curves, as shown by N. Thériault [36].)

Even with our improved algorithm, the generic ECC (with Dahab-Lopez Projective implementation) seems to be always better than generic genus 3 HECC. (It has been shown by Pelzl et al. [28] that generic genus 2 HECC is not as efficient as generic genus 3 HECC.) However, genus 3 HECC *with* $h \equiv 1$ is more efficient than ECC, when the I/M-ratio lies between 1.7 and 6.5. (Note that, as pointed out by Galbraith [8], all curves of genus 2 with $h \equiv 1$ are supersingular and hence weaker. This is not the case for genus 3 hyperelliptic curves with $h \equiv 1$. For example, Galbraith [8] shows that the curve given by the equation $y^2 + y = x^7$ is non-supersingular over a field of characteristic 2.)

Software implementations of the field arithmetic available to us indicated a variation of I/M-ratio between 8 and 20. Hardware implementations tend to achieve better I/M-ratio. For instance, Koc and Savas [17] present hardware designs with I/M-ratios as low as 4.

Figure 2 describes how the efficiency of the arithmetic for ECC with affine coordinates, ECC with Dahab-Lopez co-ordinates, and HECC for genus 3 with $h \equiv 1$ evolve, as the field-size and the I/M-ratio change. (Here, a field-size of n -bits corresponds to fields of size n -bits in the case of ECC, and equivalently, to fields of size $\frac{n}{20}$ -bits, in the case of genus 3 HECC, due to a result of [36].)

For “low”-security cryptosystems, genus 3 cryptosystems are particularly well suited for hardware implementation, where low I/M-ratios are possible. However, it is interesting to note that, as the security (equivalent field size in bits) increases, so does the range of I/M-ratios for which HECC (genus 3 and $h \equiv 1$) is the most efficient. Therefore, with the algorithms presented here, “high”-security hyperelliptic curve cryptosystems for $g = 3, h \equiv 1$ turn out to be more efficient than the equivalent elliptic curve cryptosystems, even for software implemented field-arithmetic.

As an example: NIST (National Institute of Standards and Technology [27]) currently recommends the use of a field of size 571 bits for exchanging a 256-bit *AES* (Advanced Encryption

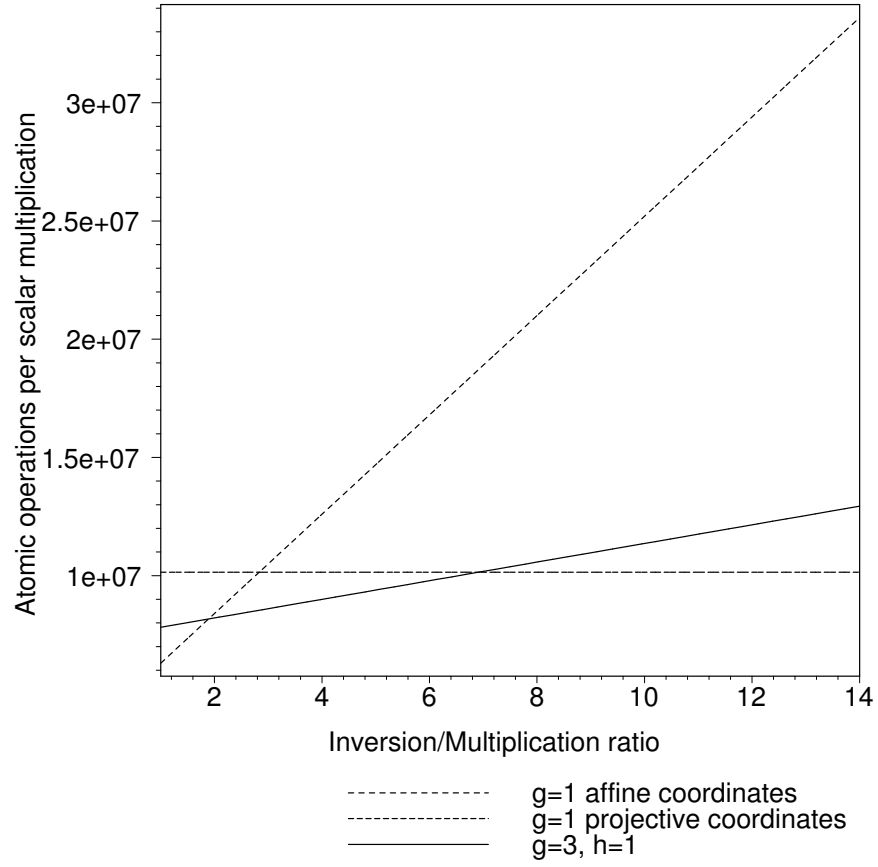


Figure 1: Cost of scalar multiplication for different cryptosystems and I/M-ratios in terms of atomic operations for a 32-bit processor and ECC equivalent security of 571 bits

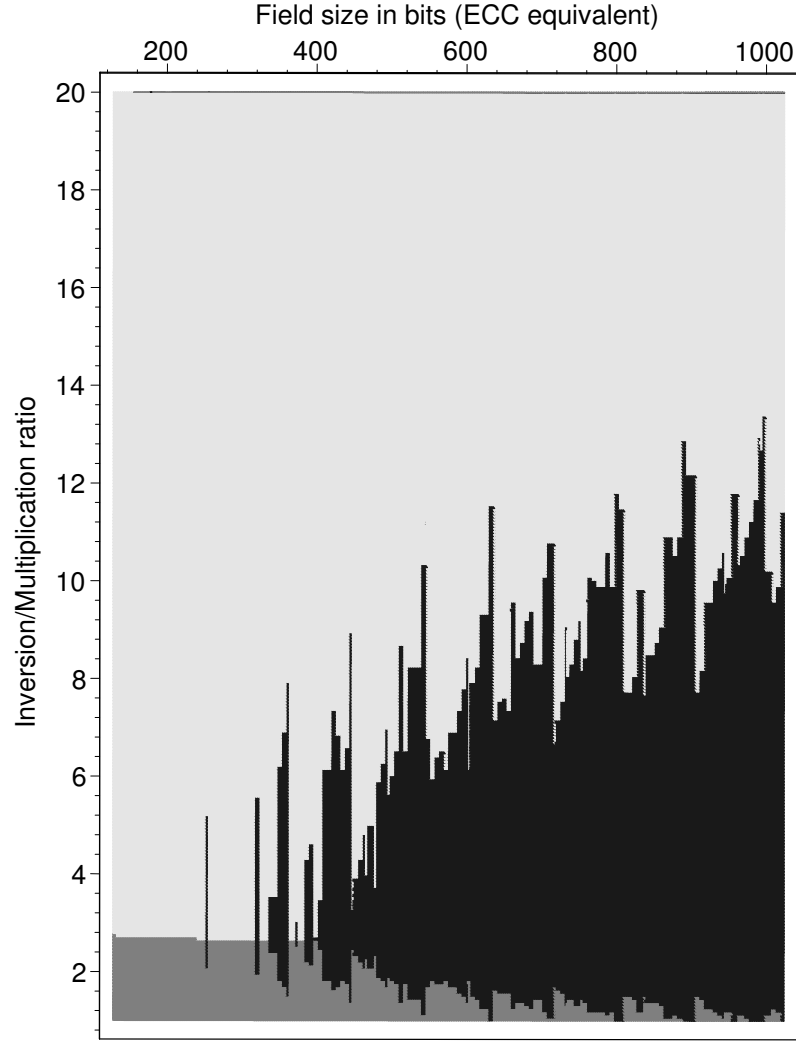


Figure 2: Depiction of the most efficient cryptosystems depending on the ECC equivalent field size (in bits) and the Inversion/Multiplication ratio for a 32-bit processor. Dark grey shows the region where ECC with affine coordinates is the most efficient. Light grey represents the region where ECC with projective (Lopez-Dahab) coordinates is the most efficient and black shows the region where genus 3 HECC with $h \equiv 1$ is the most efficient.

Standard) key with an elliptic curve cryptosystem. For an equivalent level of security, a genus 3 ($h \equiv 1$) HECC implementation would outperform an ECC implementation, provided the I/M-ratio lies between 1.7 and 6.5, as was noted earlier in relation to Figure 1.

In this context, we would like to mention a question raised by Koblitz at a MSRI Symposium in January 2002. The question was: “Is there an explicit family of hyperelliptic curves such that, for such a family, HECC is more efficient than (an equivalently secure) ECC?”. The following two examples provide an affirmative answer to this question, one for an I/M-ratio of 4 (typical for hardware implementations [17]) and one for an I/M-ratio of 8 (typical for software implementations [11]). These examples are for curves defined over fields of smallest prime degree, such that the resulting cryptosystems (ECC and HECC) provide an equivalent security of at least 637 bits.

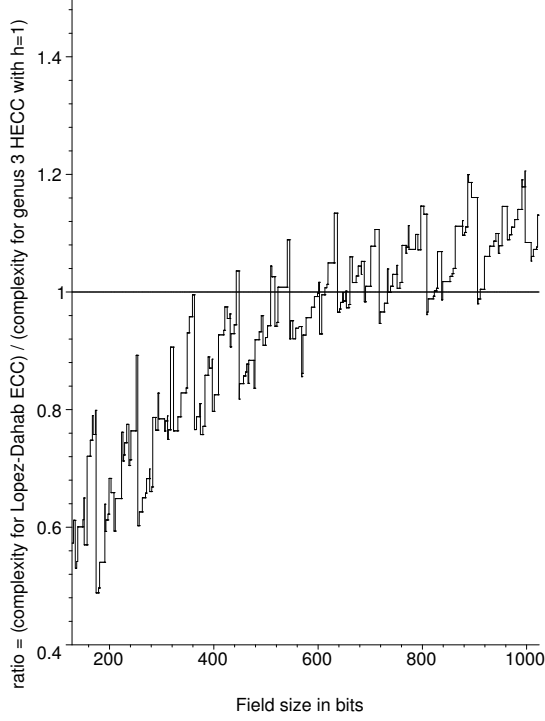
- **Example 1:** I/M-ratio = 4, ECC over $\mathbb{F}_{2^{641}}$ and genus 3 HECC with $h \equiv 1$ over $\mathbb{F}_{2^{227}}$.

In this case, the number of atomic operations for ECC is 1.37×10^7 and for HECC is 1.02×10^7 . HECC is thus roughly *34% faster* than ECC.

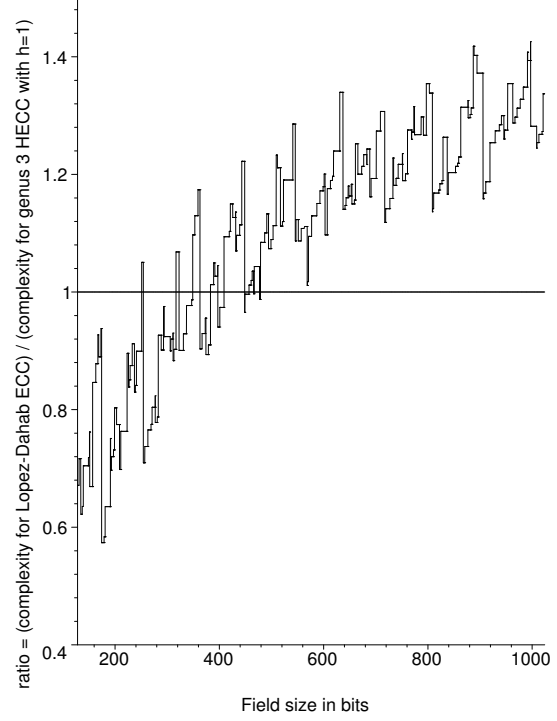
- **Example 2:** I/M-ratio = 8, ECC over $\mathbb{F}_{2^{641}}$ and genus 3 HECC with $h \equiv 1$ over $\mathbb{F}_{2^{227}}$.

In this case, the number of atomic operations for ECC is 1.37×10^7 and for HECC is 1.21×10^7 . HECC is thus roughly *15% faster* than ECC.

Figure 3 represents the evolution of the complexity of ECC versus that of genus 3 HECC with $h \equiv 1$, as the field size (ECC equivalent security) varies (for I/M-ratios 4 and 8). The graphs are generated as follows: for a security parameter d (the degree of the field), let p_E be the smallest prime number larger than d , and p_H be the smallest prime number larger than $\frac{7d}{20}$. We then compute the ratio of the complexity of ECC (over $\mathbb{F}_{2^{p_E}}$) to that of HECC (over $\mathbb{F}_{2^{p_H}}$).



I/M-ratio = 8



I/M-ratio = 4

Figure 3: Graph of the relative complexity of ECC to that of genus 3 HECC with $h \equiv 1$, as a function of the field size in bits. A relative complexity higher than 1 indicates that HECC is more efficient than ECC (for the associated field sizes).

HECC over fields of odd characteristic: For fields \mathbb{F}_p with p odd, implementing HECC is easier than ECC when the field size is smaller than 64 bits, since no library for long integer arithmetic is needed. Furthermore, if the processor is capable of 64-bit integer arithmetic (multiplication and inversion in particular), it is worth investigating whether in this case, HECC would be more efficient than ECC. We plan to tackle this and other implementation related aspects in a future work.

APPENDICES

A Recurring computations

A.1 Resultant and inverse computation

Given two polynomials $u_A(x)$ and $u_B(x)$ both monic of degree 3. We want to compute the resultant $\Delta := \text{Resultant}(u_A, u_B) = \text{Resultant}(u_A - u_B, u_B)$ and $\frac{1}{u_A} \bmod u_B$ when $\Delta \neq 0$. Let $M(x)$ and $N(x)$ be such that

$$\begin{aligned} 1 &= (u_A(x) - u_B(x))M(x) + u_B(x)(M(x) + N(x)), \text{ where} \\ u_A(x) &= x^3 + a_2x^2 + a_1x + a_0, \\ u_B(x) &= x^3 + b_2x^2 + b_1x + b_0, \\ M(x) &= m_2x^2 + m_1x + m_0. \end{aligned}$$

The computation of the resultant and the inverse is explained in the table below.

Computation of $\Delta := \text{Resultant}$ and $\Delta \cdot M$	Cost
Steps	M and S
$z_0 := a_0 - b_0, z_1 := a_1 - b_1, z_2 := a_2 - b_2$	0
$w_0 := z_0 - \mathbf{b}_1\mathbf{z}_2, w_1 := z_1 - \mathbf{b}_2\mathbf{z}_2$	2
$k_0 := w_0 - \mathbf{b}_2\mathbf{w}_1, k_1 := \mathbf{b}_0\mathbf{z}_2 + \mathbf{b}_1\mathbf{w}_1$	3
$\mathbf{w}_0\mathbf{w}_1, \mathbf{k}_0\mathbf{z}_1, \mathbf{k}_1\mathbf{z}_2, \mathbf{b}_0\mathbf{w}_1, \mathbf{z}_1\mathbf{w}_1, \mathbf{k}_0\mathbf{w}_0, \mathbf{k}_1\mathbf{w}_1$	7
$\Delta \cdot m_0 := k_0w_0 + k_1w_1$	0
$\Delta \cdot m_1 := -(k_1z_2 + k_0z_1)$	0
$\Delta = (b_0z_2) \cdot (w_0w_1 - \Delta \cdot m_1) - (b_0w_1) \cdot (z_1w_1) + \mathbf{z}_0 \cdot (\Delta \cdot m_0)$	3
$\Delta \cdot m_2 := z_1w_1 - \mathbf{z}_2\mathbf{w}_0$	1
$\{\Delta, \Delta M = \Delta \cdot m_0 + \Delta \cdot m_1x + \Delta \cdot m_2x^2\}$	Total=16M
<i>Method of Pelzl et al. [28]</i>	$16M + 2S$

In the doubling algorithm for characteristic 2, we need to compute $M(x) = \frac{1}{h} \bmod u_A = \left(\frac{1}{h+u_A} \right) \bmod u_A$.

Computation of $\Delta := \text{Resultant}(u_A, h)$ and $\frac{\Delta}{h}$ mod u_A , when $\deg(h) = 3$ and $h_2 = 0$	Cost
Steps	M and S
$z_0 := h_0 + a_0, z_1 := h_1 + a_1, z_2 := a_2$	0
$w_0 := z_0 + \mathbf{a}_1 \mathbf{a}_2, w_1 := z_1 + \mathbf{a}_2^2$	1M+1S
$\mathbf{w}_1^2, \mathbf{a}_0 \mathbf{a}_2, \mathbf{z}_1 \mathbf{w}_0, \mathbf{a}_2 \mathbf{w}_1$	3M+1S
$\Delta \cdot m_0 := \mathbf{w}_0^2 + a_1 \cdot w_1^2 + (a_2 w_1) \cdot (w_0 + a_0)$	2M+1S
$\Delta \cdot m_1 := a_2 \cdot ((a_0 a_2) + w_1 \cdot (a_1 + z_1)) + z_1 w_0$	2M
$\Delta = (a_0 a_2) \cdot (\mathbf{w}_0 \mathbf{w}_1 + \Delta \cdot m_1) + (\mathbf{a}_0 \mathbf{z}_1) \cdot w_1^2 + z_0 \cdot (\Delta \cdot m_0)$	5M
$\Delta \cdot m_2 := (a_2 + z_1) \cdot (w_0 + w_1) + a_2 w_1 + z_1 w_0$	1M
$\{\Delta, \Delta M = \Delta \cdot m_0 + \Delta \cdot m_1 x + \Delta \cdot m_2 x^2\}$	Total=14M+3S

Computation of $\Delta := \text{Resultant}(u_A, h)$ and $\frac{\Delta}{h}$ mod u_A , when $\deg(h) = 2$	Cost
Steps	M and S
$z_0 := a_0, z_1 := h_0 + a_1, z_2 := h_1 + a_2$	0
$w_0 := z_0 + \mathbf{z}_2 \mathbf{h}_0, w_1 := z_1 + \mathbf{z}_2 \mathbf{h}_1$	2M
$\Delta \cdot m_0 := w_1$	0
$\Delta \cdot m_1 := w_0 + (\mathbf{h}_1 + \mathbf{z}_2) \cdot \mathbf{w}_1$	1M
$\Delta \cdot m_2 := \mathbf{w}_0 \mathbf{z}_2 + \mathbf{w}_1 \mathbf{z}_1$	2M
$\Delta = \mathbf{w}_0^2 + (\mathbf{h}_1 \mathbf{w}_0 + \mathbf{h}_0 \mathbf{w}_1) \cdot \mathbf{w}_1$	3M+1S
$\{\Delta, \Delta M = \Delta \cdot m_0 + \Delta \cdot m_1 x + \Delta \cdot m_2 x^2\}$	Total=8M+1S

Computation of $\Delta := \text{Resultant}(u_A, h)$ and $\frac{\Delta}{h}$ mod u_A , when $\deg(h) = 1$ and $h_0 = 0$	Cost
Steps	M and S
$\Delta \cdot m_0 := 1$	0
$\Delta \cdot m_1 := a_2$	0
$\Delta \cdot m_2 := a_1$	0
$\Delta = a_0$	0
$\{\Delta, \Delta M = \Delta \cdot m_0 + \Delta \cdot m_1 x + \Delta \cdot m_2 x^2\}$	Total=FREE

A.2 Other recurring computations

Let $P := p_2 x^2 + p_1 x + p_0$, and $Q := q_2 x^2 + q_1 x + q_0$ two non-monic polynomials of degree 2, and $U := x^3 + u_2 x^2 + u_1 x + u_0$, a monic polynomial of degree 3. We want to compute

$R := r_2x^2 + r_1x + r_0 := P \cdot Q \pmod{U}$, where $PQ = U \cdot (\lambda_1x + \lambda_0) + R$.

$$\lambda_1 = \mathbf{p_2q_2}$$

$$\lambda_0 = p_1q_2 + p_2q_1 - \lambda_1u_2 = (\mathbf{p_1} + \mathbf{p_2})(\mathbf{q_1} + \mathbf{q_2}) - \mathbf{p_1q_1} - p_2q_2 - \lambda_1\mathbf{u_2}$$

$$r_2 = p_0q_2 + p_2q_0 + p_1q_1 - \lambda_1u_1 - \lambda_0u_2$$

$$= (\mathbf{p_0} + \mathbf{p_2})(\mathbf{q_0} + \mathbf{q_2}) - \mathbf{p_0q_0} - p_2q_2 + p_1q_1 - \lambda_1\mathbf{u_1} - \lambda_0\mathbf{u_2}$$

$$r_1 = p_0q_1 + p_1q_0 - \lambda_1u_0 - \lambda_0u_1$$

$$= (\mathbf{p_0} + \mathbf{p_1})(\mathbf{q_0} + \mathbf{q_1}) - p_0q_0 - p_1q_1 - (\lambda_0 + \lambda_1)(\mathbf{u_0} + \mathbf{u_1}) - \lambda_0\mathbf{u_0} - \lambda_1u_1$$

$$r_0 = p_0q_0 - \lambda_0u_0$$

This takes **11M**

B Details of the addition algorithm in characteristic 2

B.1 Tables for the addition algorithm: characteristic 2

	Addition of two generic divisors			$\deg(h) = 3$ $h_2 = 0$	$\deg(h) = 3$ $f_6 = 0$	$h \equiv 1$ $f_6 = 0$
	Steps	S	I	M and S	M and S	M and S
1	$\Delta := \text{resultant}(u_A, u_B), \Delta M := \frac{\Delta}{u_A} \bmod u_B$	0	0	16M	16M	16M
2	$\Delta S := \Delta M(v_B - v_A) \bmod u_B$	0	0	11M	11M	11M
3	$\Delta^2 s_2, \frac{1}{\Delta^2 s_2}, \frac{1}{\Delta s_2}, \Delta^2 s_2^2, s_2, \frac{1}{s_2}$	1	1	4M	4M	4M
4	$\tilde{S} := \frac{\Delta S}{\Delta s_2} = \text{Monic}(\Delta S)$	0	0	2M	2M	2M
5	$\tilde{s}_1 a_2, \tilde{s}_0 a_2, \tilde{s}_1 a_1, (\tilde{s}_0 + \tilde{s}_1)(a_0 + a_1), \tilde{s}_0 a_0$	0	0	5M	5M	5M
6	$\tilde{s}_1^2, \tilde{s}_1^2 a_2, \tilde{s}_1^2 a_1, \tilde{s}_0^2, F_6, F_5, F_4, F_3$ can now be computed for free.	2	0	2M	2M	2M
7	$l_3, l_2, l_1, l_0, L := \left\lfloor \frac{\tilde{S}^2 u_A}{u_B} \right\rfloor$	0	0	3M	3M	3M
8	$P := \left\lfloor \frac{\tilde{S} h}{u_B} \right\rfloor$ or $P = 0$ when $h \equiv 1$	0	0	1M	1M	0
9	$k := \left\lfloor \left(\frac{f - h v_A - v_A^2}{u_A u_B} \right) \right\rfloor = x + (f_6 + a_2 + b_2)$	0	0	0	0	0
10	$s_2^{-1} k$	0	0	1M	1M	0
11	$s_2^{-1}(P + s_2^{-1} k)$ or $s_2^{-2} k$ when $h \equiv 1$	0	0	2M	2M	1M+1S
12	$\mathbf{u_T} := x^4 + t_3 x^3 + t_2 x^2 + t_1 x + t_0 := L + s_2^{-1}(P + s_2^{-1} k)$. This is free .	0	0	0	0	0
13	$t_0 \mu, t_1 \mu, t_2 \mu, t_3 \mu$, here $\mu = -t_3 + \tilde{s}_1 + a_2$ as in step below.	0	0	4M	4M	4M
14	Compute G where $\tilde{S} u_A = u_T(x + \mu) + G$	0	0	0	0	0
15	$v_T := s_2(\tilde{S} u_A \bmod u_T) + h + v_A$	0	0	4M	4M	4M
16	$u_E := \frac{f + v_T h + v_T^2}{u_T}$	1	0	4M+1S	6M	3M+1S
17	$v_E := (h + v_T) \bmod u_E$	0	0	3M	3M	3M
	TOTAL	4S	1I	62M+1S	64M	58M+2S

	Addition of two generic divisors			$\deg(h) = 2$ $f_6 = 0$	$\deg(h) = 1$ $h_0 = 0$
	Steps	S	I	M and S	M and S
1	$\Delta := \text{resultant}(u_A, u_B), \Delta M := \frac{\Delta}{u_A} \bmod u_B$	0	0	16M	16M
2	$\Delta S := \Delta M(v_B - v_A) \bmod u_B$	0	0	11M	11M
3	$\Delta^2 s_2, \frac{1}{\Delta^2 s_2}, \frac{1}{\Delta s_2}, \Delta^2 s_2^2, s_2, \frac{1}{s_2}$	1	1	4M	4M
4	$\tilde{S} := \frac{\Delta S}{\Delta s_2} = \text{Monic}(\Delta S)$	0	0	2M	2M
5	$\tilde{s}_1 a_2, \tilde{s}_0 a_2, \tilde{s}_1 a_1, (\tilde{s}_0 + \tilde{s}_1)(a_0 + a_1), \tilde{s}_0 a_0$	0	0	5M	5M
6	$\tilde{s}_1^2, \tilde{s}_1^2 a_2, \tilde{s}_1^2 a_1, \tilde{s}_0^2$. F_6, F_5, F_4, F_3 can now be computed for free.	2	0	2M	2M
7	$l_3, l_2, l_1, l_0, L := \frac{\tilde{S}^2 u_A}{u_B}$	0	0	3M	3M
8	$P := \frac{\tilde{S}h}{u_B}$ or $P = 1$ when $\deg(h) = 1$	0	0	0	0
9	$k := \left\lfloor \frac{(f - h v_A - v_A^2)}{u_A u_B} \right\rfloor = x + (f_6 + a_2 + b_2)$	0	0	0	0
10	$s_2^{-1} k$	0	0	1M	1S
11	$s_2^{-1}(P + s_2^{-1} k)$	0	0	1M + 1S	1M
12	$\mathbf{u_T} := x^4 + t_3 x^3 + t_2 x^2 + t_1 x + t_0 := L + s_2^{-1}(P + s_2^{-1} k)$. This is free .	0	0	0	0
13	$t_0 \mu, t_1 \mu, t_2 \mu, t_3 \mu$, here $\mu = -t_3 + \tilde{s}_1 + a_2$ as in step below.	0	0	4M	4M
14	Compute G where $\tilde{S} u_A = u_T(x + \mu) + G$	0	0	0	0
15	$v_T := s_2(\tilde{S} u_A \bmod u_T) + h + v_A$	0	0	4M	4M
16	$u_E := \frac{f + v_T h + v_T^2}{u_T}$	1	0	4M + 1S	3M + 1S
17	$v_E := (h + v_T) \bmod u_E$	0	0	3M	3M
	TOTAL	4S	1I	60M + 2S	58M + 2S

B.2 Main algorithm for the addition algorithm: characteristic 2

Algorithm B.1 (MainAlgorithm). **Input:** Two divisors $D_A = (u_A, v_A)$ and $D_B = (u_B, v_B)$, each of weight 3 defined over \mathbb{F}_q with $q = 2^n$. Let $u_A = x^3 + a_2 x^2 + a_1 x + a_0$, $u_B = x^3 + b_2 x^2 + b_1 x + b_0$. $v_A(x) = \alpha_2 x^2 + \alpha_1 x + \alpha_0$ and $v_B = \beta_2 x^2 + \beta_1 x + \beta_0$.
Output: Unique divisor $D_E = (u_E, v_E)$ of weight at most 3 such that $D_E = D_A + D_B$.

(A) *Compute:*

$$\begin{aligned}
u_T &= \frac{s_2^{-2} \left(\frac{f-hv_A-v_A^2}{u_A} \right) - h(s_2^{-2}S) - u_A(s_2^{-1}S)^2}{u_B} \\
&= \left[\frac{\tilde{S}^2 u_A}{u_B} \right] + s_2^{-1} \left(\left[\frac{\tilde{S}h}{u_B} \right] - s_2^{-1}k \right) \\
&= L + s_2^{-1}(P - s_2^{-1}k) \\
v_T &= -h - (v_A + u_A S) \mod u_T,
\end{aligned}$$

where $\tilde{S} := \text{Monic}(S) := s_2^{-1}S$, $L := \left[\frac{\tilde{S}^2 u_A}{u_B} \right]$, $P := \left[\frac{\tilde{S}h}{u_B} \right]$ and $k := \left[\frac{f-hv_A-v_A^2}{u_A} \right]$.

(B) *Compute:*

$$\begin{aligned}
u_E &= \frac{f - hv_T - v_T^2}{u_T} \\
v_E &= -h - v_T \mod u_E
\end{aligned}$$

1. To compute Δ and $\Delta \cdot M = \frac{\Delta}{u_A} \mod u_B$. If $\Delta = 0$ abort and use Cantor algorithm instead. It is easy to check that $\Delta = 0$ if and only if $\text{support}(D_A)$ and $\text{support}(D_B)$ have a point in common or that $\text{support}(D_A)$ contains a point whose opposite lies in $\text{support}(D_B)$. This takes 16M.
2. Compute $\Delta \cdot S = \Delta \cdot M(v_B - v_A) \mod u_B$. If $\Delta s_2 = 0$ abort and use Cantor algorithm instead. This takes 11M as described in Appendix A.
3. Compute $\Delta(\Delta s_2)$, $\frac{1}{\Delta^2 s_2}$, $\frac{\Delta}{\Delta^2 s_2} = \frac{1}{\Delta s_2}$, $(\Delta s_2)^2 = \Delta^2 s_2^2$, $\frac{\Delta^2 s_2^2}{\Delta^2 s_2} = s_2$, $\frac{\Delta}{\Delta s_2} = \frac{1}{s_2}$. This takes 1I, 4M, 1S.
4. Compute $\tilde{S} := \frac{\Delta S}{\Delta s_2} = \text{Monic}(\Delta S)$. This takes 2M.
5. Compute the following values: $\tilde{s}_1 a_2, \tilde{s}_0 a_2, \tilde{s}_1 a_1, (\tilde{s}_0 + \tilde{s}_1)(a_0 + a_1), \tilde{s}_0 a_0$. This takes 5M. Note that some of these will only be used at a later step.
6. To compute: $\left[\frac{\tilde{S}^2 u_A}{u_B} \right]$. If we let $F(x) := \tilde{S}^2 u_A$ then we want to compute $\left[\frac{F}{u_B} \right]$. Let $F = L(x)u_B(x) + \text{remainder}$, where $L(x) := (x^4 + l_3 x^3 + l_2 x^2 + l_1 x + l_0)$, then it follows that:

$$\begin{aligned}
l_3 &= F_6 - b_2 & l_2 &= F_5 - b_1 - l_3 b_2 \\
l_1 &= F_4 - b_0 - (l_3 b_1 + l_2 b_2) & l_0 &= F_3 - (l_1 b_2 + l_2 b_1 + l_3 b_0)
\end{aligned}$$

Thus, we need to know F_6, F_5, F_4, F_3 to compute L .

Let us now compute $F(x) := x^7 + F_6x^6 + F_5x^5 + F_4x^4 + F_3x^3 + \dots + F_0 := \tilde{S}^2 u_A$.

$$\begin{aligned} F_6 &= a_2 & F_5 &= a_1 + \tilde{s}_1^2 \\ F_4 &= a_0 + \tilde{s}_1^2 a_2 & F_3 &= \tilde{s}_1^2 a_1 + \tilde{s}_0^2 \end{aligned}$$

This takes **2M, 2S**.

7. To compute $L := x^4 + l_3x^3 + l_2x^2 + l_1x + l_0$.

$$\begin{aligned} l_3 &= F_6 - b_2 = a_2 - b_2 = z_2 & l_2 &= F_5 - b_1 - l_3b_2 \\ & & &= F_5 - b_1 + z_2b_2, z_2 \text{ as in step 1} \\ l_1 &= F_4 - b_0 - (l_3b_1 + l_2b_2) & l_0 &= F_3 - (l_1b_2 + l_2b_1 + l_3b_0) \\ &= F_4 - b_0 - z_2b_1 - \mathbf{l}_2\mathbf{b}_2 & &= F_3 - z_2b_0 - (\mathbf{l}_1\mathbf{b}_2 + \mathbf{l}_2\mathbf{b}_1) \end{aligned}$$

Given that we know F_6, F_5, F_4, F_3 from previous step, it takes **3M**.

8. To compute $\left[\frac{\tilde{S}h}{u_B}\right]$. Let $\tilde{S}h = Pu_B + \text{remainder}$, where $P(x) = x^2 + p_1x + p_0$ when $\deg(h) = 3$. It then follows that

$$\begin{aligned} p_1 &= \tilde{s}_1 + h_2 - b_2 & p_0 &= \tilde{s}_0 + h_1 - b_1 + h_2\tilde{s}_1 - p_1b_2 \\ & & &= \tilde{s}_0 + h_1 - b_1 + (\mathbf{h}_2 + \mathbf{b}_2)(\tilde{s}_1 + \mathbf{b}_2) \end{aligned}$$

This takes **1M** when $\deg(h) = 3$. If $\deg(h) = 2$ then the P above can be computed as $P(x) = x + \tilde{s}_1 + h_1 + b_2$ which is **FREE**. If $\deg(h) = 1$, then $P(x) = 1$ which is **FREE**. Finally, if $h \equiv 1$ then $P \equiv 0$ which is **FREE**.

9. To compute $k := \left[\frac{f - hv_A - v_A^2}{u_A u_B}\right]$. A straightforward computation gives $k = x + (f_6 + a_2 + b_2)$. This is **FREE**.

10. To compute $s_2^{-1}k$, where k as in the previous step. This takes **1M**. If $h \equiv 1$ this step is unnecessary.

11. To compute $s_2^{-1}(P + s_2^{-1}k)$, where P as previously computed. This takes **2M**. If $\deg(h) = 2$ then this takes **1M+1S**. For $\deg(h) = 1$, the previous two steps can be done simultaneously in **1M+1S**. Finally, if $h \equiv 1$, this can be computed directly as $s_2^{-2}\mathbf{k}$ and takes **1M, 1S**.

12. Compute $u_T := x^4 + t_3x^3 + t_2x^2 + t_1x + t_0 = L + s_2^{-1}(P + s_2^{-1}k)$. We already have all the data. So this is **FREE**.

13. Compute $\{t_{0\mu}, t_{1\mu}, t_{2\mu}, t_{3\mu}\}$ with μ as in following step, since we now have u_T . This takes **4M**.

14. To compute $\tilde{S}u_A \bmod u_T$: Let $\tilde{S}u_A = (x + \mu)u_T + G(x)$, where $G(x) := g_3x^3 + g_2x^2 + g_1x + g_0$, and $u_T = x^4 + t_3x^3 + t_2x^2 + t_1x + t_0$. Then, we get

$$\begin{aligned} \mu &= -t_3 + \tilde{s}_1 + a_2 & g_3 &= -(t_3\mu + t_2) + \tilde{s}_0 + a_1 + a_2\tilde{s}_1 \\ g_2 &= -(t_2\mu + t_1) + a_0 + \tilde{s}_0a_2 + \tilde{s}_1a_1 & g_1 &= -(t_1\mu + t_0) + (\tilde{s}_1a_0 + \tilde{s}_0a_1) \\ g_0 &= -t_0\mu + \tilde{s}_0a_0 \end{aligned}$$

This computation requires the value of $\{t_0\mu, t_1\mu, t_2\mu, t_3\mu\}$, which can only be computed after u_T is known. Our idea is to combine the computation of $\{F_6, F_5, F_4, F_3\}$ with that of the $\{g_3, g_2, g_1, g_0\}$, and use it for the computation of G as well. This explains the extra computations done in step 5. Since we already have all the required data, the computation of G is **FREE**.

15. Compute

$$\begin{aligned} v_T &:= \tau_3x^3 + \tau_2x^2 + \tau_1x + \tau_0 \\ &= -h - (v_A + Su_A) \bmod u_T = -h - v_A - s_2(\tilde{S}u_A \bmod u_T) \end{aligned}$$

This takes **4M**.

16. To compute: $u_E = x^3 + e_2x^2 + e_1x + e_0 := \frac{f - v_T h - v_T^2}{u_T}$. Let $J := x^7 + j_6x^6 + j_5x^5 + j_4x^4 + j_3x^3 + j_2x^2 + j_1x + j_0 := f - v_T h - v_T^2 = u_T u_E$. Since this is an exact division the e_i 's can be computed as follows:

$$\begin{aligned} j_6 &= \tau_3 + f_6 + \tau_3^2 & j_5 &= \tau_2 + f_5 + \tau_3 \mathbf{h}_2 \\ j_4 &= \tau_1 + f_4 + \tau_3 \mathbf{h}_1 + \tau_2(\mathbf{h}_2 + \tau_2) & e_2 &= j_6 + t_3 \\ e_1 &= j_5 + t_2 + \mathbf{t}_3 \mathbf{e}_2 & e_0 &= j_4 + \mathbf{t}_3 \mathbf{e}_1 + \mathbf{t}_2 \mathbf{e}_2 \end{aligned}$$

When $\deg(h) = 3$, this takes **6M+1S** if $h_2 \neq 0$, and **4M+2S** otherwise.

When $\deg(h) = 2$, this takes **4M+2S**.

Finally, if $\deg(h) \leq 1$, this takes **3M+2S**.

17. To compute $v_E := \epsilon_2x^2 + \epsilon_1x + \epsilon_0 = -h - v_T \bmod u_E$.

$$\begin{aligned} h + v_T &= (\tau_3 + 1)x^3 + (\tau_2 + h_2)x^2 + (\tau_1 + h_1)x + (\tau_0 + h_0) = u_E(\tau_3 + 1) + v_E \\ v_E &= -h - v_T \bmod u_E \\ v_E &= (\tau_2 + h_2 + (\tau_3 + 1)\mathbf{e}_2)x^2 + (\tau_1 + h_1 + (\tau_3 + 1)\mathbf{e}_1)x + (\tau_0 + h_0 + (\tau_3 + 1)\mathbf{e}_0) \end{aligned}$$

This takes **3M**.

C Details of the doubling algorithm in characteristic 2

C.1 Tables for the doubling algorithm: characteristic 2

	Doubling of a generic divisor			$\deg(h) = 3$ $h_2 = 0$	$\deg(h) = 3$ $f_6 = 0$
	Steps	S	I	M and S	M and S
1	$R := \left(\frac{v_A^2 + v_A h - f}{u_A} \right) \bmod u_A$	1	0	9M	6M
2	$\Delta M := \left(\frac{\Delta}{h} \bmod u_A \right)$	0	0	14M+3S	16M
3	$\Delta S = R \Delta M \bmod u_A, \deg(\Delta S) = 2$	0	0	11M	11M
4	$\Delta(\Delta s_2), \frac{1}{\Delta^2 s_2}, \frac{1}{\Delta}, \Delta^2, \frac{1}{s_2}, s_2$	1	1	4M	4M
5	$\tilde{S} := s_2^{-1} S = x^2 + \tilde{s}_1 x + \tilde{s}_0$	0	0	2M	2M
6	For $\tilde{S} u_A : \tilde{s}_1 a_2, \tilde{s}_0 a_2, \tilde{s}_1 a_1, \tilde{s}_0 a_0, (\tilde{s}_0 + \tilde{s}_1)(a_0 + a_1)$	0	0	5M	5M
7	$L := \tilde{S}^2$	2	0	0	0
8	$P := \left\lfloor \frac{\tilde{S} h}{u_A} \right\rfloor, \deg(P) = 2$	0	0	0	1M
9	$s_2^{-1} (P + s_2^{-1} k)$	0	0	3M	2M
10	$\mathbf{u_T} := x^4 + t_3 x^3 + t_2 x^2 + t_1 x + t_0 := L + s_2^{-1} (P + s_2^{-1} k)$. This is free .	0	0	0	0
11	$t_0 \mu, t_1 \mu, t_2 \mu, t_3 \mu$, here $\mu = -t_3 + \tilde{s}_1 + a_2$ as in step below.	0	0	4M	4M
12	Compute G , where $\tilde{S} u_A = u_T(x + \mu) + G$	0	0	0	0
13	$v_T := s_2(\tilde{S} u_A \bmod u_T) + h + v_A$	0	0	4M	4M
14	$u_E := \frac{f + v_T h + v_T^2}{u_T}$. This is an exact division.	1	0	4M+1S	6M
15	$v_E := (h + v_T) \bmod u_E$	0	0	3M	3M
	TOTAL	5S	1I	63M+4S	64M

	Doubling of a generic divisor			$\deg(h) = 2$ $f_6 = 0$	$\deg(h) = 1$ $h_0 = 0$
	Steps	S	I	M and S	M and S
1	$R := \left(\frac{v_A^2 + v_A h - f}{u_A} \right) \bmod u_A$	1	0	6M	9M
2	$\Delta M := \left(\frac{\Delta}{h} \bmod u_A \right)$	0	0	8M+1S	0
3	$\Delta S = R \Delta M \bmod u_A, \deg(\Delta S) = 2$	0	0	11M	9M
4	$\Delta(\Delta s_2), \frac{1}{\Delta^2 s_2}, \frac{1}{\Delta}, \Delta^2, \frac{1}{s_2}, s_2$	1	1	4M	4M
5	$\tilde{S} := s_2^{-1} S = x^2 + \tilde{s}_1 x + \tilde{s}_0$	0	0	2M	2M
6	For $\tilde{S} u_A : \tilde{s}_1 a_2, \tilde{s}_0 a_2, \tilde{s}_1 a_1, \tilde{s}_0 a_0, (\tilde{s}_0 + \tilde{s}_1)(a_0 + a_1)$	0	0	5M	5M
7	$L := \tilde{S}^2$	2	0	0	0
8	$P := \left\lfloor \frac{\tilde{S} h}{u_A} \right\rfloor, \deg(P) = 1$	0	0	0	0
9	$s_2^{-1} (P + s_2^{-1} k)$	0	0	1M+1S	1M
10	$\mathbf{u_T} := x^4 + t_3 x^3 + t_2 x^2 + t_1 x + t_0 := L + s_2^{-1} (P + s_2^{-1} k)$. This is free .	0	0	0	0
11	$t_0 \mu, t_1 \mu, t_2 \mu, t_3 \mu$, here $\mu = -t_3 + \tilde{s}_1 + a_2$ as in step below.	0	0	4M	4M
12	Compute G , where $\tilde{S} u_A = u_T(x + \mu) + G$	0	0	0	0
13	$v_T := s_2(\tilde{S} u_A \bmod u_T) + h + v_A$	0	0	4M	4M
14	$u_E := \frac{f + v_T h + v_T^2}{u_T}$. This is an exact division.	1	0	4M+1S	3M+1S
15	$v_E := (h + v_T) \bmod u_E$	0	0	3M	3M
	TOTAL	5S	1I	52M+3S	44M+1S

C.2 Main algorithm for the doubling algorithm: characteristic 2

Algorithm C.1 (MainAlgorithm). **Input:** A divisor $D_A = (u_A, v_A)$ of weight 3 defined over \mathbb{F}_q with $q = 2^n$. $u_A = x^3 + a_2 x^2 + a_1 x + a_0$, and $v_A(x) = \alpha_2 x^2 + \alpha_1 x + \alpha_0$.

Output: A divisor $D_E = (u_E, v_E)$ of weight at most 3 such that $D_E = 2D_A$.

(A) Compute $D_T := (u_T, v_T)$.

$$\begin{aligned}
u_T &:= \text{Monic} \left(\frac{f - v_{\text{comp}}h - v_{\text{comp}}^2}{u_{\text{comp}}} \right) \\
&= s_2^{-2} \left(\frac{f + (v_A + Su_A)h + (v_A + Su_A)^2}{u_A^2} \right) \\
&= \tilde{S}^2 + s_2^{-1} \left(\left[\frac{\tilde{S}h}{u_A} \right] + s_2^{-1} \left[\frac{f - v_Ah - v_A^2}{u_A^2} \right] \right) \\
&= L + s_2^{-1}(P + s_2^{-1}k) \\
v_T &:= -h - v_A + Su_A \mod u_T \\
&= -h - v_A + s_2(\tilde{S}u_A) \mod u_T
\end{aligned}$$

Here, $S := s_2x^2 + s_1x + s_0$ is as explained in Section 3,

$$\tilde{S} := \text{Monic}(S) = s_2^{-1}S = \frac{\Delta S}{\Delta s_2}, \quad L := \tilde{S}^2, \quad P := \left[\frac{\tilde{S}h}{u_A} \right] \quad \text{and} \quad k := \left[\frac{f - v_Ah - v_A^2}{u_A^2} \right].$$

(B) Compute:

$$\begin{aligned}
u_E &= \frac{f - hv_T - v_T^2}{u_T} \\
v_E &= -h - v_T \mod u_E
\end{aligned}$$

1. We want to compute $Q := \left(\frac{v_A^2 + v_Ah - f}{u_A} \right)$ and then $R := Q \mod u_A$. Let $F := v_A^2 + v_Ah - f := x^7 + F_6x^6 + F_5x^5 + F_4x^4 + F_3x^3 + F_2x^2 + F_1x + F_0$. We want to compute $R := Q \mod u_A = \frac{F}{u_A} \mod u_A$. Computing R only requires the knowledge of F_6, F_5, F_4 and F_3 . Let $H(x) := x^3 + H_2x^2 + H_1x + H_0 := h + v_A$. From the equation $F = v_AH + f$, we get

$$\begin{aligned}
F_6 &= f_6 & F_5 &= f_5 + \alpha_2 \\
F_4 &= f_4 + \alpha_1 + \alpha_2H_2 & F_3 &= f_3 + \alpha_0 + \alpha_2H_1 + \alpha_1H_2
\end{aligned}$$

Given H , this takes **3M**.

Let $Q := x^4 + q_3x^3 + q_2x^2 + q_1x + q_0$. The equation $F = Qu_A$ gives:

$$\begin{aligned}
q_3 &= F_6 + a_2 & q_2 &= F_5 + a_1 + a_2q_3 = F_5 + a_1 + \mathbf{a}_2\mathbf{f}_6 + \mathbf{a}_2^2 \\
q_1 &= F_4 + a_0 + a_1q_3 + a_2q_2 & q_0 &= F_3 + a_0q_3 + a_1q_2 + a_2q_1 \\
&= F_4 + a_0 + \mathbf{a}_1\mathbf{f}_6 + \mathbf{a}_2(\mathbf{a}_1 + \mathbf{q}_2) & &= F_3 + \mathbf{a}_0\mathbf{f}_6 + \mathbf{a}_2(\mathbf{q}_1 + \mathbf{a}_0) + \mathbf{a}_1\mathbf{q}_2
\end{aligned}$$

This takes **6M+1S** if $f_6 \neq 0$, and **3M+1S** otherwise.

We can now compute $R := r_2x^2 + r_1x + r_0 := Q \bmod u_A$. Let $\lambda \in \mathbb{F}_q$ be such that $Q = u_A(x + \lambda) + R$. This gives:

$$\begin{aligned} \lambda &= q_3 + a_2 = f_6 & r_1 &= q_1 + a_0 + \lambda a_1 \\ r_2 &= q_2 + a_1 + \lambda a_2 & r_0 &= q_0 + \lambda a_0 \end{aligned}$$

This is **FREE**.

Thus, computation of R takes **9M+1S** if $f_6 \neq 0$ and **6M+1S** otherwise.

2. Compute Δ and $\Delta \cdot M := \frac{\Delta}{h} \bmod u_A$. Here, Δ is the resultant of h and u_A . (For details on the computation of the resultant please refer to Appendix A.) (If $\Delta = 0$ abort and use Cantor algorithm instead. It can be checked that $\Delta = 0$ if and only if the $\text{Support}(D_A)$ contains a point of order 2.) This computation takes **14M+3S** when $\deg(h) = 3$ and $h_2 = 0$, **16M** when $\deg(h) = 3$ with no assumption on h_2 , **8M+1S** when $\deg(h) = 2$, and is **FREE** when $\deg(h) = 1$.
3. Compute $\Delta \cdot S := R \cdot (\Delta \cdot M) \bmod u_A$. (If $\Delta_{s_2} = 0$ abort, and use Cantor algorithm instead.) This takes **11M** as described in Appendix A when $\deg(h)$ is 3 or 2, and **9M** when $\deg(h) = 1$.
4. Compute $\Delta(\Delta s_2)$, $\frac{1}{\Delta^2 s_2}$, $\frac{\Delta}{\Delta^2 s_2} = \frac{1}{\Delta s_2}$, $(\Delta s_2)^2 = \Delta^2 s_2^2$, $\frac{\Delta^2 s_2^2}{\Delta^2 s_2} = s_2$, $\frac{\Delta}{\Delta s_2} = \frac{1}{s_2}$. This takes **1I**, **4M**, **1S**.
5. $\tilde{S} := \frac{\Delta S}{\Delta s_2} = \text{Monic}(\Delta S)$. This takes **2M**.
6. Compute the following values: $\tilde{s}_1 a_2, \tilde{s}_0 a_2, \tilde{s}_1 a_1, (\tilde{s}_0 + \tilde{s}_1)(a_0 + a_1), \tilde{s}_0 a_0$. This takes **5M**. Note that some of these will only be used at a later step.
7. Compute $L := \tilde{S}^2 = x^4 + \tilde{s}_1^2 x^2 + \tilde{s}_0^2$. This takes **2S**.
8. Compute $P(x)$, where $\tilde{S}h = Pu_A + \text{remainder}$. If $\deg(h) = 3$ then $P(x) := x^2 + p_1x + p_0$.

$$\begin{aligned} p_1 &= h_2 + \tilde{s}_1 - a_2 & p_0 &= h_1 + h_2 \tilde{s}_1 + \tilde{s}_0 - (p_1 a_2 + a_1) \\ & & &= \begin{cases} \tilde{s}_0 + h_1 - a_1 + (h_2 + a_2)(\tilde{s}_1 + a_2), & h_2 \neq 0 \\ \tilde{s}_0 + h_1 - a_1 + a_2 \tilde{s}_1 + a_2^2, & h_2 = 0 \end{cases} \end{aligned}$$

This takes **1M** if $h_2 \neq 0$, and is **FREE** otherwise. When $\deg(h) = 2$, $P(x) = x + \tilde{s}_1 + h_1 + a_2$ which is **FREE**. If $\deg(h) = 1$ then $P(x) = 1$ which is **FREE**.

9. Compute $s_2^{-1}(P + s_2^{-1}k) = s_2^{-1}(P + s_2^{-1}(x + f_6))$. This takes **3M** if $f_6 \neq 0$ and **2M** otherwise. If $\deg(h) = 2$ then it takes **1M+1S**. For $\deg(h) = 1$, this computation takes **1M**.

10. Compute $u_T := x^4 + t_3x^3 + t_2x^2 + t_1x + t_0 := \tilde{S}^2 + s_2^{-1}(P + s_2^{-1}x)$ This is **FREE**.
11. Compute $\{\mathbf{t}_0\mu, \mathbf{t}_1\mu, \mathbf{t}_2\mu, \mathbf{t}_3\mu\}$, where $\mu = -t_3 + \tilde{s}_1 + a_2$ is as in step below. This takes **4M**.
12. To compute $\tilde{S}u_A \bmod u_T$: Let $\tilde{S}u_A = (x + \mu)u_T + G(x)$, where $G(x) := g_3x^3 + g_2x^2 + g_1x + g_0$, and $u_T = x^4 + t_3x^3 + t_2x^2 + t_1x + t_0$. Then, we get

$$\begin{aligned}\mu &= -t_3 + \tilde{s}_1 + a_2 \\ g_3 &= -(t_3\mu + t_2) + \tilde{s}_0 + a_1 + a_2\tilde{s}_1 \\ g_2 &= -(t_2\mu + t_1) + a_0 + \tilde{s}_0a_2 + \tilde{s}_1a_1 \\ g_1 &= -(t_1\mu + t_0) + (\tilde{s}_1a_0 + \tilde{s}_0a_1) \\ &= -(t_1\mu + t_0) + (\tilde{s}_0 + \tilde{s}_1)(a_0 + a_1) - (\tilde{s}_1a_1 + \tilde{s}_0a_0) \\ g_0 &= -t_0\mu + \tilde{s}_0a_0\end{aligned}$$

The computation of G is **FREE**, since we already have all the data.

13. To Compute

$$\begin{aligned}v_T &:= \tau_3x^3 + \tau_2x^2 + \tau_1x + \tau_0 = -h - (v_A + Su_A) \bmod u_T \\ &= -h - (v_A + s_2(\tilde{S}u_A \bmod u_T))\end{aligned}$$

This takes **4M**.

14. To compute: $u_E = x^3 + e_2x^2 + e_1x + e_0 := \frac{f - v_T h - v_T^2}{u_T}$. Let $J := x^7 + j_6x^6 + j_5x^5 + j_4x^4 + j_3x^3 + j_2x^2 + j_1x + j_0 := f - v_T h - v_T^2 = u_T u_E$. Since this is an exact division the e_i 's can be computed as follows:

$$\begin{aligned}j_6 &= \tau_3 + f_6 + \tau_3^2 & e_2 &= j_6 + t_3 \\ j_5 &= \tau_2 + f_5 + \tau_3 \mathbf{h}_2 & e_1 &= j_5 + t_2 + \mathbf{t}_3 \mathbf{e}_2 \\ j_4 &= \tau_1 + f_4 + \tau_3 \mathbf{h}_1 + \tau_2(\mathbf{h}_2 + \tau_2) & e_0 &= j_4 + \mathbf{t}_3 \mathbf{e}_1 + \mathbf{t}_2 \mathbf{e}_2\end{aligned}$$

When $\deg(h) = 3$ this takes **6M+1S** if $h_2 \neq 0$, and **4M+2S** otherwise.

If $\deg(h) = 2$, then this takes **4M+2S**. Finally, if $\deg(h) = 1$, this takes **3M+2S**.

15. To compute $v_E := \epsilon_2x^2 + \epsilon_1x + \epsilon_0 = -h - v_T \bmod u_E$.

$$\begin{aligned}h + v_T &= (\tau_3 + 1)x^3 + (\tau_2 + h_2)x^2 + (\tau_1 + h_1)x + (\tau_0 + h_0) \\ &= u_E(\tau_3 + 1) + v_E \\ v_E &= -h - v_T \bmod u_E \\ v_E &= (\tau_2 + h_2 + (\tau_3 + \mathbf{1})\mathbf{e}_2)x^2 + (\tau_1 + h_1 + (\tau_3 + \mathbf{1})\mathbf{e}_1)x + (\tau_0 + h_0 + (\tau_3 + \mathbf{1})\mathbf{e}_0)\end{aligned}$$

This takes **3M**.

D Details of the doubling algorithm in characteristic 2 and $h = 1$

D.1 Table for the doubling algorithm: characteristic 2 and $h = 1$

	Doubling of a generic divisor		$f_6=0, h=1$
	Steps	I	M and S
1	$u_{comp} := u_A^2$	0	3S
2	$v_{comp} = (v_A^2 + f) \bmod u_{comp}$	0	3S
3	$u_T = \text{Monic}\left(\frac{f+v_{comp}+v_{comp}^2}{u_{comp}}\right) = \frac{\text{Monic}(f+v_{comp}+v_{comp}^2)}{u_{comp}}$	1	3M+3S
4	$v_T = 1 + v_{comp} \bmod u_T$	0	4M
	$(u_E, v_E) := \text{Reduction of } (u_T, v_T)$		
5	$u_E := \frac{f+v_T+v_T^2}{u_T}$. (exact division)	0	1M+2S
6	$v_E := 1 + v_T \bmod u_E$	0	3M
	Total	1I	11M+11S

D.2 Main algorithm for the doubling algorithm: characteristic 2 and $h = 1$

Algorithm D.1 (MainAlgorithm). **Input:** A divisor $D_A = (u_A, v_A)$ of weight 3 defined over \mathbb{F}_q with $q = 2^n$. $u_A = x^3 + a_2x^2 + a_1x + a_0$, and $v_A(x) = \alpha_2x^2 + \alpha_1x + \alpha_0$.

Output: A divisor $D_E = (u_E, v_E)$ of weight at most 3 such that $D_E = 2D_A$.

(A) Compute $D_{comp} := (u_{comp}, v_{comp})$.

$$\begin{aligned} u_{comp} &:= u_A^2 \\ v_{comp} &:= v_A^2 + f \bmod u_A^2 \end{aligned}$$

(B) Compute $D_T := (u_T, v_T)$.

$$\begin{aligned} u_T &:= \text{Monic}\left(\frac{f + v_{comp} + v_{comp}^2}{u_{comp}}\right) \\ v_T &:= 1 + v_{comp} \bmod u_T \end{aligned}$$

(C) Compute $D_E := (u_E, v_E)$.

$$\begin{aligned} u_E &= \frac{f - v_T - v_T^2}{u_T} \\ v_E &= -1 - v_T \bmod u_E \end{aligned}$$

1. Compute $u_{comp} := u_A^2 = x^6 + c_5x^5 + c_4x^4 + c_3x^3 + c_2x^2 + c_1x + c_0$. $c_5 = c_3 = c_1 = 0$, $c_4 = a_2^2$, $c_2 = a_1^2$, $c_0 = a_0^2$. This takes 3S.

2. To Compute $v_{comp} := (v_A^2 + f) \bmod u_{comp} = \gamma_5 x^5 + \gamma_4 x^4 + \gamma_3 x^3 + \gamma_2 x^2 + \gamma_1 x + \gamma_0$.

It turns out that:

$\gamma_5 = f_5 + a_2^2$. (If $\gamma_5 = 0$ then abort, and use the standard Cantor algorithm instead.)

$\gamma_4 = f_4 + \alpha_2^2$, $\gamma_3 = f_3 + a_1^2$, $\gamma_2 = f_2 + \alpha_1^2$, $\gamma_1 = f_1 + a_0^2$, $\gamma_0 = f_0 + \alpha_0^2$.

This takes **3S**.

3. To compute: $u_T := x^4 + t_3 x^3 + t_2 x^2 + t_1 x + t_0 = \text{Monic} \left(\frac{f + v_{comp} + v_{comp}^2}{u_{comp}} \right) = \frac{\text{Monic}(f + v_{comp} + v_{comp}^2)}{u_{comp}}$.

Let $L := x^{10} + l_9 x^9 + l_8 x^8 + l_7 x^7 + l_6 x^6 + l_5 x^5 + l_4 x^4 + l_3 x^3 + l_2 x^2 + l_1 x + l_0 = \text{Monic}(f + v_{comp} + v_{comp}^2)$.

We have:

$$\begin{aligned} l_9 &= 0 & l_8 &= \gamma_5^{-2} \gamma_4^2 = (\gamma_5^{-1} \gamma_4)^2 \\ l_7 &= (\gamma_5^{-1})^2 & l_6 &= \gamma_5^{-2} \gamma_3^2 = (\gamma_5^{-1} \gamma_3)^2 \end{aligned}$$

This takes **1I+2M+3S**. Also,

$$\begin{aligned} t_3 &= 0 & t_2 &= c_4 + l_8 \\ t_1 &= c_4 t_3 + l_7 = l_7 & t_0 &= \mathbf{c_4 t_2} + c_2 + l_6 \end{aligned}$$

This takes **1M**. Observe that we did not need $l_5, l_4, l_3, l_2, l_1, l_0$ for the computation of u_T . Thus, it takes **1I+3M+3S** to compute u_T .

4. To Compute $v_T := \tau_3 x^3 + \tau_2 x^2 + \tau_1 x + \tau_0 = (1 + v_{comp}) \bmod u_T$.

Let $1 + v_{comp} = (\lambda_1 x + \lambda_0) u_T + (\tau_3 x^3 + \tau_2 x^2 + \tau_1 x + \tau_0)$. Then, $\lambda_1 = \gamma_5$, $\lambda_0 = \gamma_4$ and

$$\begin{aligned} \tau_3 &= \gamma_3 + \gamma_5 \mathbf{t_2} & \tau_2 &= \gamma_2 + \gamma_4 t_2 + \gamma_5 t_1 = \gamma_2 + \gamma_4 \mathbf{t_2} + \gamma_5^{-1} \\ \tau_1 &= \gamma_1 + (\gamma_4 t_1 + \gamma_5 t_0) & \tau_0 &= \gamma_0 + 1 + \gamma_4 t_0 \\ &= \gamma_1 + (\gamma_4 + \gamma_5)(\mathbf{t_0} + \mathbf{t_1}) - \gamma_4 \mathbf{t_0} - \gamma_5 t_1 \end{aligned}$$

This takes **4M**.

5. To compute: $u_E = x^3 + e_2 x^2 + e_1 x + e_0 := \frac{f + v_T + v_T^2}{u_T}$. Let

$J := x^7 + j_6 x^6 + j_5 x^5 + j_4 x^4 + j_3 x^3 + j_2 x^2 + j_1 x + j_0 := f + v_T + v_T^2 = u_T u_E$. Since this is an exact division the e_i 's can be computed as follows:

$$\begin{aligned} j_6 &= \tau_3^2 & j_5 &= f_5 & j_4 &= f_4 + \tau_2^2 \\ e_2 &= j_6 & e_1 &= j_5 + t_2 & e_0 &= j_4 + t_1 + \mathbf{t_2 e_2} \end{aligned}$$

This takes **1M+2S**.

6. To compute $v_E := e_2 x^2 + e_1 x + e_0 = 1 + v_T \bmod u_E$.

$$\begin{aligned} 1 + v_T &= \tau_3 x^3 + \tau_2 x^2 + \tau_1 x + \tau_0 + 1 = \tau_3 u_E + v_E \\ v_E &= 1 + v_T \bmod u_E \\ v_E &= (\tau_2 + \tau_3 \mathbf{e_2}) x^2 + (\tau_1 + \tau_3 \mathbf{e_1}) x + (\tau_0 + 1 + \tau_3 \mathbf{e_0}) \end{aligned}$$

This takes **3M**.

E Details of the addition algorithm in odd characteristic

E.1 Table for the addition algorithm: odd characteristic

	Addition of two generic divisors		$f_6=0$
	Steps	I	M and S
1	$\Delta := \text{resultant}(u_A, u_B), \Delta M := \frac{\Delta}{u_A} \bmod u_B$	0	16M
2	$\Delta S := \Delta M(v_B - v_A) \bmod u_B$	0	11M
3	$\Delta^2 s_2, \frac{1}{\Delta^2 s_2}, \frac{1}{\Delta s_2}, \Delta^2 s_2^2, s_2, \frac{1}{s_2}$	1	4M+1S
4	$\tilde{S} := \frac{\Delta S}{\Delta s_2} = \text{Monic}(\Delta S)$	0	2M
5	$\tilde{s}_1 a_2, \tilde{s}_0 a_2, \tilde{s}_1 a_1, (\tilde{s}_0 + \tilde{s}_1)(a_0 + a_1), \tilde{s}_0 a_0$	0	5M
6	$\tilde{s}_1^2, \tilde{s}_1(2\tilde{s}_0 + \tilde{s}_1 a_2), \tilde{s}_1(\tilde{s}_1 a_1 + 2\tilde{s}_0 a_2), \tilde{s}_0^2$. We can now compute F_6, F_5, F_4, F_3 for free.	0	2M+2S
7	$l_3, l_2, l_1, l_0, L := \left\lfloor \frac{\tilde{S}^2 u_A}{u_B} \right\rfloor$	0	5M
8	$s_2^{-1} k := s_2^{-1}(x + (a_2 + b_2))$	0	1M
9	$P := \left\lfloor \frac{\tilde{S} u_A}{u_B} \right\rfloor$	0	1M
10	$s_2^{-1}(2P + s_2^{-1} k)$	0	2M
11	$u_T := L + s_2^{-1}(P + s_2^{-1} k), \deg(u_T) = 4$. (we have everything.)	0	0
12	$t_0 \mu, t_1 \mu, t_2 \mu, t_3 \mu$	0	4M
13	$\tilde{S} u_A \bmod u_T$	0	0
14	$v_T := -s_2(\tilde{S} u_A \bmod u_T) - v_A$	0	4M
15	$u_E := \frac{f - v_T^2}{u_T}$	0	4M+3S
16	$v_E := -v_T \bmod u_E$	0	3M
	TOTAL	1	64M+6S

E.2 Main algorithm for the addition algorithm: odd characteristic

Algorithm E.1 (MainAlgorithm). **Input:** Two divisors $D_A = (u_A, v_A)$ and $D_B = (u_B, v_B)$, each of weight 3 defined over \mathbb{F}_q with $q = p$, where p is an odd prime, $u_A = x^3 + a_2 x^2 + a_1 x + a_0$,

$u_B = x^3 + b_2 x^2 + b_1 x + b_0$, $v_A(x) = \alpha_2 x^2 + \alpha_1 x + \alpha_0$, and $v_B = \beta_2 x^2 + \beta_1 x + \beta_0$.

Output: Unique divisor $D_E = (u_E, v_E)$ of weight at most 3 such that $D_E = D_A + D_B$.

(A) *Compute:*

$$\begin{aligned}
u_T &= s_2^{-2} \left(\frac{S^2 u_A + 2Sv_A}{u_B} - \frac{f - v_A^2}{u_A u_B} \right) \\
&= \left[\frac{\tilde{S}^2 u_A}{u_B} \right] + s_2^{-1} \left(2 \left[\frac{\tilde{S} v_A}{u_B} \right] - s_2^{-1} \left[\frac{f - v_A^2}{u_A u_B} \right] \right) \\
v_T &= -(v_A + S u_A) \bmod u_T \\
&= -v_A - s_2(\tilde{S} u_A \bmod u_T)
\end{aligned}$$

(B) *Compute:*

$$\begin{aligned}
u_E &= \frac{f - v_T^2}{u_T} \\
v_E &= -v_T \bmod u_E
\end{aligned}$$

1. *Compute Δ and $\Delta \cdot M = \frac{\Delta}{u_A} \bmod u_B$. If $\Delta = 0$ abort and use Cantor algorithm instead. It is easy to check that $\Delta = 0$ if and only if $\text{support}(D_A)$ and $\text{support}(D_B)$ have a point in common or that $\text{support}(D_A)$ contains a point whose opposite lies in $\text{support}(D_B)$. This needs the resultant computation as done in Appendix A. This takes 16M.*
2. *Compute $\Delta \cdot S = \Delta \cdot M(v_B - v_A) \bmod u_B$. (If $\Delta s_2 = 0$ abort, and use Cantor algorithm instead.) This takes 11M as described in Appendix A.*
3. *Compute $\Delta(\Delta s_2)$, $\frac{1}{\Delta^2 s_2}$, $\frac{\Delta}{\Delta^2 s_2} = \frac{1}{\Delta s_2}$, $(\Delta s_2)^2 = \Delta^2 s_2^2$, $\frac{\Delta^2 s_2^2}{\Delta^2 s_2} = s_2$, $\frac{\Delta}{\Delta s_2} = \frac{1}{s_2}$. This takes 1I, 4M, 1S.*
4. *$\frac{\Delta S}{\Delta s_2} = \tilde{S} = \text{Monic}(\Delta S) = \text{Monic}(S)$. This takes 2M.*
5. *Compute $\tilde{s}_1 a_2, \tilde{s}_0 a_2, \tilde{s}_1 a_1, (\tilde{s}_0 + \tilde{s}_1)(a_0 + a_1), \tilde{s}_0 a_0$. This takes 5M.*
6. *Compute $\tilde{s}_1^2, \tilde{s}_1(2\tilde{s}_0 + \tilde{s}_1 a_2), \tilde{s}_1(\tilde{s}_1 a_1 + 2\tilde{s}_0 a_2), \tilde{s}_0^2$. This takes 2M+2S. Using this and previous computations, we can now compute F_6, F_5, F_4, F_3 for **FREE** as explained below.*
7. *To compute: $L := \left[\frac{\tilde{S}^2 u_A}{u_B} \right]$. Let $F(x) := \tilde{S}^2 u_A$. We want to compute $\left[\frac{F}{u_B} \right]$. Let $F = L(x)u_B(x) + \text{remainder}$, where $L(x) := x^4 + l_3 x^3 + l_2 x^2 + l_1 x + l_0$. It then follows that:*

$$\begin{aligned}
l_3 &= F_6 - b_2 & l_2 &= F_5 - b_1 - \mathbf{l}_3 \mathbf{b}_2 \\
l_1 &= F_4 - b_0 - (l_3 b_1 + l_2 b_2) & l_0 &= F_3 - (\mathbf{l}_1 \mathbf{b}_2 + l_2 b_1 + \mathbf{l}_3 \mathbf{b}_0) \\
&= F_4 - b_0 - ((\mathbf{l}_3 + \mathbf{l}_2)(\mathbf{b}_1 + \mathbf{b}_2) - l_3 b_2 - \mathbf{l}_2 \mathbf{b}_1)
\end{aligned}$$

The computation of L takes **5M** because the computation of $\{F_6, F_5, F_4, F_3\}$ in $F(x) := x^7 + F_6x^6 + F_5x^5 + F_4x^4 + F_3x^3 + \dots + F_0 := \tilde{S}^2u_A$ can be done for **FREE** as follows:

$$\begin{aligned}
F_6 &= a_2 + 2\tilde{s}_1 \\
F_5 &= a_1 + 2\tilde{s}_0 + 2\tilde{s}_1a_2 + \tilde{s}_1^2 \\
&= a_1 + 2\tilde{s}_0 + \tilde{s}_1(2a_2 + \tilde{s}_1) \\
F_4 &= a_0 + 2\tilde{s}_0\tilde{s}_1 + 2\tilde{s}_1a_1 + (2\tilde{s}_0 + \tilde{s}_1^2)a_2 \\
&= a_0 + 2\tilde{s}_1a_1 + 2\tilde{s}_0a_2 + \tilde{s}_1(2\tilde{s}_0 + \tilde{s}_1a_2) \\
F_3 &= 2\tilde{s}_1a_0 + (2\tilde{s}_0 + \tilde{s}_1^2)a_1 + 2\tilde{s}_0\tilde{s}_1a_2 + \tilde{s}_0^2 \\
&= \tilde{s}_1(\tilde{s}_1a_1 + 2\tilde{s}_0a_2) + \tilde{s}_0^2 + 2((\tilde{s}_0 + \tilde{s}_1)(a_0 + a_1) - (\tilde{s}_1a_1 + \tilde{s}_0a_0))
\end{aligned}$$

8. To compute $k := \left[\frac{f-v_A^2}{u_A u_B} \right]$. A straightforward computation gives $k = x + (a_2 + b_2)$. Compute $s_2^{-1}k$. This takes **1M**.
9. To compute $\left[\frac{\tilde{S}v_A}{u_B} \right]$: Let $\tilde{S}v_A = Pu_B + \text{remainder}$, where $P(x) := p_1x + p_0$, then it follows that: $p_1 = \alpha_2$, $p_0 = \alpha_1 + \alpha_2(\tilde{s}_1 - b_2)$. This takes **1M**.
10. To compute $s_2^{-1}(2P + s_2^{-1}k)$, where P as in one of the previous steps. Since the degree is 1, this takes **2M**.
11. Compute $u_T := x^4 + t_3x^3 + t_2x^2 + t_1x + t_0 = L + s_2^{-1}(P + s_2^{-1}k)$. We already have all the data. So this is **FREE**.
12. Compute $\{t_0\mu, t_1\mu, t_2\mu, t_3\mu\}$, since we now have u_T . This takes **4M**.
13. To compute $\tilde{S}u_A \bmod u_T$: Let $\tilde{S}u_A = (x + \mu)u_T + G(x)$, where $G(x) := g_3x^3 + g_2x^2 + g_1x + g_0$, and $u_T = x^4 + t_3x^3 + t_2x^2 + t_1x + t_0$. Then, we get

$$\begin{aligned}
\mu &= -t_3 + \tilde{s}_1 + a_2 \\
g_3 &= -(t_3\mu + t_2) + \tilde{s}_0 + a_1 + a_2\tilde{s}_1 \\
g_2 &= -(t_2\mu + t_1) + a_0 + \tilde{s}_0a_2 + \tilde{s}_1a_1 \\
g_1 &= -(t_1\mu + t_0) + (\tilde{s}_1a_0 + \tilde{s}_0a_1) \\
&= -(t_1\mu + t_0) + (\tilde{s}_0 + \tilde{s}_1)(a_0 + a_1) - (\tilde{s}_1a_1 + \tilde{s}_0a_0) \\
g_0 &= -t_0\mu + \tilde{s}_0a_0
\end{aligned}$$

To compute this we need $\{t_0\mu, t_1\mu, t_2\mu, t_3\mu\}$, which we now have, thanks to the computation of u_T . Our idea is to combine the computation of $\{F_6, F_5, F_4, F_3\}$ with that of the $\{g_3, g_2, g_1, g_0\}$. This is **FREE**, since we already have all the data.

14. Compute $v_T := \tau_3 x^3 + \tau_2 x^2 + \tau_1 x + \tau_0 = -v_A - s_2(\tilde{S}u_A \bmod u_T)$. This takes 4M.

15. To compute: $u_E = x^3 + e_2 x^2 + e_1 x + e_0 := \frac{f - v_T^2}{u_T}$. Let $J := x^7 + j_6 x^6 + j_5 x^5 + j_4 x^4 + j_3 x^3 + j_2 x^2 + j_1 x + j_0 := v_T^2 - f = u_T u_E$. Since this is an exact division the e_i 's can be computed as follows:

$$\begin{array}{ll} j_6 &= -\tau_3^2 & e_2 &= j_6 - t_3 \\ j_5 &= f_5 - 2\tau_2\tau_3 & e_1 &= j_5 - t_2 - \mathbf{t}_3\mathbf{e}_2 \\ &= f_5 - ((\tau_2 + \tau_3)^2 - (\tau_2^2 + \tau_3^2)) & e_0 &= j_4 - (t_1 + \mathbf{t}_2\mathbf{e}_2 + \mathbf{t}_3\mathbf{e}_1) \\ j_4 &= f_4 - (\tau_2^2 + 2\tau_1\tau_3) \end{array}$$

This takes 4M+3S.

16. To compute $v_E := \epsilon_2 x^2 + \epsilon_1 x + \epsilon_0 = -v_T \bmod u_E$.

$$\begin{array}{ll} -v_T &= -\tau_3 x^3 - \tau_2 x^2 - \tau_1 x - \tau_0 = -\tau_3 u_E + v_E \\ v_E &= (\tau_3 \mathbf{e}_2 - \tau_2)x^2 + (\tau_3 \mathbf{e}_1 - \tau_1)x + (\tau_3 \mathbf{e}_0 - \tau_0) \end{array}$$

This takes 3M.

F Details of the doubling algorithm in odd characteristic

F.1 Table for the doubling algorithm: odd characteristic

	Doubling of a generic divisor		$f_6=0$
	Steps	I	M and S
1	$R := \left(\frac{v_A^2 - f}{u_A} \right) \bmod u_A$	0	5M+2S
2	$\Delta M := \left(\frac{\Delta}{2v_A} \bmod u_A \right)$	0	16M
3	$\Delta S = R\Delta M \bmod u_A, \deg(\Delta S) = 2$	0	11M
4	$\Delta(\Delta s_2), \frac{1}{\Delta^2 s_2}, \frac{1}{\Delta}, \Delta^2, \frac{1}{s_2}, s_2$	1	4M+1S
5	$\tilde{S} := s_2^{-1} S = x^2 + \tilde{s}_1 x + \tilde{s}_0$	0	2M
6	For $\tilde{S}u_A$: $\tilde{s}_1 a_2, \tilde{s}_0 a_2, \tilde{s}_1 a_1, (\tilde{s}_0 + \tilde{s}_1)(a_0 + a_1), \tilde{s}_0 a_0$	0	5M
7	$L := \tilde{S}^2, \tilde{s}_1^2, (\tilde{s}_0 + \tilde{s}_1)^2, \tilde{s}_0^2$	0	3S
8	$P := \left[\frac{\tilde{S}h}{u_A} \right], \deg(P) = 2$	0	1M
9	$s_2^{-1} (P + s_2^{-1} x)$	0	2M
10	$u_T = \tilde{S}^2 + s_2^{-1} (P + s_2^{-1} x)$	0	0
11	$t_0 \mu, t_1 \mu, t_2 \mu, t_3 \mu$	0	4M
12	Compute G where $\tilde{S}u_A = (x + \mu)u_T + G$	0	0
13	$v_T := -v_A - s_2 G$	0	4M
14	$u_E := \frac{-f + v_T^2}{u_T}$. (exact division)	0	4M+3S
15	$v_E := -v_T \bmod u_E$	0	3M
	Total	1I	61M+9S

F.2 Main algorithm for the doubling algorithm: odd characteristic

Algorithm F.1 (MainAlgorithm). **Input:** A divisor $D_A = (u_A, v_A)$ of weight 3 defined over \mathbb{F}_q with $q = p$. $u_A = x^3 + a_2 x^2 + a_1 x + a_0$, and $v_A(x) = \alpha_2 x^2 + \alpha_1 x + \alpha_0$.

Output: A divisor $D_E = (u_E, v_E)$ of weight at most 3 such that $D_E = 2D_A$.

(A) Compute $D_T := (u_T, v_T)$.

$$\begin{aligned}
u_T &:= \text{Monic} \left(\frac{v_{\text{comp}}^2 - f}{u_{\text{comp}}} \right) \\
&= s_2^{-2} \left(\frac{v_A^2 - f + 2Su_A v_A + (Su_A)^2}{u_A^2} \right) \\
&= \tilde{S}^2 + s_2^{-1} \left(\left[\frac{2\tilde{S}v_A}{u_A} \right] - s_2^{-1}x \right) \\
&= L + s_2^{-1}(P - s_2^{-1}x) \\
v_T &:= -v_A - Su_A \pmod{u_T} \\
&= -v_A - s_2(\tilde{S}u_A) \pmod{u_T}
\end{aligned}$$

Here, $S := s_2x^2 + s_1x + s_0$ is as explained in Section 6, and $\tilde{S} := \text{Monic}(S) = s_2^{-1}S = \frac{\Delta S}{\Delta s_2}$.

(B) Compute $D_E := (u_E, v_E)$.

$$\begin{aligned}
u_E &= \frac{f - v_T^2}{u_T} \\
v_E &= -v_T \pmod{u_E}
\end{aligned}$$

1. To compute $Q := \left(\frac{f - v_A^2}{u_A} \right)$. Let

$F := x^7 + F_6x^6 + F_5x^5 + F_4x^4 + F_3x^3 + F_2x^2 + F_1x + F_0 := f - v_A^2$. We want to compute $\frac{F}{u_A} \pmod{u_A}$, which only requires the knowledge of F_6, F_5, F_4, F_3 .

$$\begin{aligned}
F_6 &= f_6 = 0 & F_5 &= f_5 \\
F_4 &= f_4 - \alpha_2^2 & F_3 &= f_3 - 2\alpha_1\alpha_2
\end{aligned}$$

This takes 1M+1S.

As defined earlier, $Q := x^4 + q_3x^3 + q_2x^2 + q_1x + q_0 := \frac{F}{u_A}$. The equation $F = Qu_A$ gives us

$$\begin{aligned}
q_3 &= -a_2 & q_1 &= F_4 - a_0 - a_1q_3 - a_2q_2 \\
q_2 &= F_5 - a_1 - a_2q_3 & &= F_4 - a_0 - a_2q_2 + a_2a_1 \\
&= F_5 - a_1 + \mathbf{a}_2^2 & &= F_4 - a_0 + \mathbf{a}_2(\mathbf{a}_1 - \mathbf{q}_2)
\end{aligned}$$

(Note that, $q_0 = F_3 - a_0q_3 - a_1q_2 - a_2q_1 = F_3 + a_2(a_0 - q_1) - a_1q_2$. Although this expression will be used in the next step, there is actually no need to evaluate it.)

This takes 1M+1S.

We want to compute $R := r_2x^2 + r_1x + r_0 := Q \bmod u_A$. Let $\lambda \in \mathbb{F}_q$ be such that $Q = u_A(x + \lambda) + R$. This gives:

$$\begin{aligned} \lambda &= q_3 - a_2 & r_2 &= q_2 - a_1 - \lambda a_2 \\ &= -2a_2 & &= q_2 - a_1 + 2a_2^2 \\ r_1 &= q_1 - a_0 - \lambda a_1 & r_0 &= q_0 - \lambda a_0 \\ &= q_1 - a_0 + 2\mathbf{a}_1\mathbf{a}_2 & &= F_3 + \mathbf{a}_2(3\mathbf{a}_0 - \mathbf{q}_1) - \mathbf{a}_1\mathbf{q}_2 \end{aligned}$$

This takes **3M**.

In total, the computation of R takes **5M+2S**.

2. Compute Δ and $\Delta \cdot M := \frac{\Delta}{2v_A} \bmod u_A$. If $\Delta = 0$ abort and use Cantor algorithm instead. It can be checked that $\Delta = 0$ if and only if $\text{support}(D_A)$ contains a point of order 2. Here, Δ is the resultant of $2v_A$ and u_A . (Details are in Appendix A.) This takes **16M**.
3. Compute $\Delta \cdot S := R \cdot (\Delta \cdot M) \bmod u_A$. If $\Delta s_2 = 0$ abort and use Cantor algorithm instead. This takes **11M** as described in Appendix A.
4. Compute $\Delta(\Delta s_2)$, $\frac{1}{\Delta^2 s_2}$, $\frac{\Delta}{\Delta^2 s_2} = \frac{1}{\Delta s_2}$, $(\Delta s_2)^2 = \Delta^2 s_2^2$, $\frac{\Delta^2 s_2^2}{\Delta^2 s_2} = s_2$, $\frac{\Delta}{\Delta s_2} = \frac{1}{s_2}$. This takes **1I, 4M, 1S**.
5. $\tilde{S} := \frac{\Delta S}{\Delta s_2} := \text{Monic}(\Delta S)$. This takes **2M**.
6. Compute the following values: $\tilde{s}_1\mathbf{a}_2, \tilde{s}_0\mathbf{a}_2, \tilde{s}_1\mathbf{a}_1, (\tilde{s}_0 + \tilde{s}_1)(\mathbf{a}_0 + \mathbf{a}_1), \tilde{s}_0\mathbf{a}_0$. This takes **5M**. Note that some of these will only be used at a later step.
7. Compute:

$$\begin{aligned} \tilde{S}^2 &= x^4 + 2\tilde{s}_1x^3 + (\tilde{s}_1^2 + 2\tilde{s}_0)x^2 + 2\tilde{s}_0\tilde{s}_1x + \tilde{s}_0^2. \\ &= x^4 + 2\tilde{s}_1x^3 + (\tilde{s}_1^2 + 2\tilde{s}_0)x^2 + ((\tilde{s}_0 + \tilde{s}_1)^2 - (\tilde{s}_0^2 + \tilde{s}_1^2))x + \tilde{s}_0^2. \end{aligned}$$

This takes **3S**.

8. Compute $P(x) := p_1x + p_0$, where $2\tilde{S}v_A = Pu_A + \text{remainder}$.

$$\begin{aligned} p_1 &= 2\alpha_2 & p_0 &= 2\alpha_2\tilde{s}_1 + 2\alpha_1 - 2\alpha_2a_2 \\ & & &= 2\alpha_1 + 2\alpha_2(\tilde{s}_1 - \mathbf{a}_2) \end{aligned}$$

This takes **1M**.

9. Compute $s_2^{-1}(P - s_2^{-1}x)$. This takes **2M**.
10. Compute $u_T := x^4 + t_3x^3 + t_2x^2 + t_1x + t_0 := \tilde{S}^2 + s_2^{-1}(P - s_2^{-1}x)$. This is **FREE**.

11. Compute $\{\mathbf{t}_0\mu, \mathbf{t}_1\mu, \mathbf{t}_2\mu, \mathbf{t}_3\mu\}$, where $\mu = -t_3 + \tilde{s}_1 + a_2$ is as in step below. This takes 4M.

12. To compute $\tilde{S}u_A \bmod u_T$: Let $\tilde{S}u_A = (x+\mu)u_T + R(x)$, where $G(x) := g_3x^3 + g_2x^2 + g_1x + g_0$, and $u_T = x^4 + t_3x^3 + t_2x^2 + t_1x + t_0$. Then, we get

$$\begin{aligned}\mu &= -t_3 + \tilde{s}_1 + a_2 \\ g_3 &= -(t_3\mu + t_2) + \tilde{s}_0 + a_1 + a_2\tilde{s}_1 \\ g_2 &= -(t_2\mu + t_1) + a_0 + \tilde{s}_0a_2 + \tilde{s}_1a_1 \\ g_1 &= -(t_1\mu + t_0) + (\tilde{s}_1a_0 + \tilde{s}_0a_1) \\ &= -(t_1\mu + t_0) + (\tilde{s}_0 + \tilde{s}_1)(a_0 + a_1) - (\tilde{s}_1a_1 + \tilde{s}_0a_0) \\ g_0 &= -t_0\mu + \tilde{s}_0a_0\end{aligned}$$

The computation of G is **FREE**, since we already have all the data.

13. Compute

$$\begin{aligned}v_T &= -v_A - Su_A \bmod u_T \\ \tau_3x^3 + \tau_2x^2 + \tau_1x + \tau_0 &= -v_A - s_2(\tilde{S}u_A \bmod u_T)\end{aligned}$$

This takes 4M.

14. To compute: $u_E = x^3 + e_2x^2 + e_1x + e_0 := \frac{f-v_T^2}{u_T}$. Let $J := x^7 + j_6x^6 + j_5x^5 + j_4x^4 + j_3x^3 + j_2x^2 + j_1x + j_0 := v_T^2 - f = u_Tu_E$. Since this is an exact division the e_i 's can be computed as follows:

$$\begin{aligned}j_6 &= -\tau_3^2 & e_2 &= j_6 - t_3 \\ j_5 &= f_5 - 2\tau_2\tau_3 & e_1 &= j_5 - t_2 - \mathbf{t}_3\mathbf{e}_2 \\ &= f_5 - ((\tau_2 + \tau_3)^2 - (\tau_2^2 + \tau_3^2)) & e_0 &= j_4 - (t_1 + \mathbf{t}_2\mathbf{e}_2 + \mathbf{t}_3\mathbf{e}_1) \\ j_4 &= f_4 - (\tau_2^2 + 2\tau_1\tau_3)\end{aligned}$$

This takes 4M+3S.

15. To compute $v_E := \epsilon_2x^2 + \epsilon_1x + \epsilon_0 = -v_T \bmod u_E$.

$$\begin{aligned}-v_T &= -\tau_3x^3 - \tau_2x^2 - \tau_1x - \tau_0 \\ &= -\tau_3u_E + v_E \\ v_E &= (\tau_3\mathbf{e}_2 - \tau_2)x^2 + (\tau_3\mathbf{e}_1 - \tau_1)x + (\tau_3\mathbf{e}_0 - \tau_0)\end{aligned}$$

This takes 3M.

References

- [1] I. F. Blake, G. Seroussi, and N.P. Smart, *Elliptic curves in cryptography*, London Mathematical Society Lecture Notes Series 265, Cambridge University Press, 1999.
- [2] D. G. Cantor, *Computing in the Jacobian of hyperelliptic curve*, Mathematics of Computation, 48 (177) (1987), 95-101.
- [3] W. Diffie, M. E. Hellman, *New directions in cryptography*, IEEE Transactions on Information Theory, IT-22 (1976), 644-654.
- [4] J. Denef and F. Vercauteren, *An extension of Kedlaya's algorithm to Artin-Schreier curves in characteristic 2*, in C. Fieker and D. R. Kohel (editors), ANTS-V, Lecture Notes in Computer Science 2369, Springer-Verlag, 2002, 308-323.
- [5] J. Denef and F. Vercauteren, *An extension of Kedlaya's algorithm to hyperelliptic curves in characteristic 2*, to appear in Journal of Cryptology.
- [6] J. Denef and F. Vercauteren, *Computing zeta functions of C_{ab} curves using Monsky-Washnitzer cohomology*, preprint (2003).
- [7] R. Dahab and J. Lopez, *Improved algorithms for elliptic curve arithmetic in $GF(2^n)$* , Selected Areas in Cryptography- SAC 98, Lecture Notes in Computer Science No. 1556, (1999), 201-212.
- [8] S. D. Galbraith, *Supersingular curves in cryptography*, in Advances in Cryptology-ASIACRYPT 2001, Lecture Notes in Computer Science 2248, (2001), 495-517
- [9] P. Gaudry, F. Hess and N. P. Smart, *Constructive and destructive facets of Weil descent*, Journal of Cryptology 15(1), (2002), 19-46.
- [10] P. Gaudry, R. Harley, *Counting points on hyperelliptic curves over finite fields*, In W. Bosma, editor, The forth Algorithmic Number Theory Symposiums - ANTS IV, Lecture Notes in Computer Science Volume 18, Springer-Verlag, Berlin, (2000), 297-312.
- [11] D. Hankerson, J. L. Hernandez, A. Menezes, *Software implementation of elliptic curve cryptography over binary fields*, In C. Koc and C. Paar, editors, Workshop on Cryptographic Hardware and Embedded Systems -CHES 2000, Lecture Notes in Computer Science, Springer-Verlag, Berlin, (2000).
- [12] R. Harley, *Fast arithmetic on genus two curves*, at <http://cristal.inria.fr/~harley/hyper/>, (2000).
- [13] K. Kedlaya, *Counting points on hyperelliptic curves using Monsky-Washnitzer cohomology*, Journal of the Ramanujan Mathematical Society Volume 16, No 4 (2001), 323-338.

- [14] N. Koblitz, *Elliptic curve cryptosystems*, Mathematics of Computation, 48 (1987), 203-209.
- [15] N. Koblitz, *A Family of Jacobians suitable for discrete log cryptosystems*, Advances in Cryptology-Crypto 88 (Editor Shafi Goldwasser), Lecture Notes in Computer Science No. 403, Springer-Verlag, (1988), 94-99.
- [16] N. Koblitz, *Hyperelliptic cryptosystems*, Journal of Cryptology, 150, (1989), 139-150.
- [17] C. K. Koc and E. Savas, *Architectures for unified field inversion with applications in elliptic curve cryptography*, 9th IEEE International Conference on Electronics, Circuits and Systems, ICECS 2002, Dubrovnik, Croatia, September 15-18 2002, Volume 3, (2002), 1155-1158.
- [18] J. Kuroki, M. Gonda, K. Matsuo, Jinhui Chao, and Shigeo Tsujii, *Fast genus three hyperelliptic curve cryptosystems*, The 2002 Symposium on Cryptography and Information Security, Japan - SCIS 2002, 29 Jan.-1 Feb. 2002.
- [19] T. Lange, *Efficient arithmetic on genus 2 hyperelliptic curves over finite fields via explicit formulae*, Cryptology ePrint Archive, Report 2002/121, 2002. <http://eprint.iacr.org>.
- [20] T. Lange, *Inversion-free arithmetic on genus 2 hyperelliptic curves*, Cryptology ePrint Archive, Report 2002/147, 2002. <http://eprint.iacr.org/>.
- [21] T. Lange, *Weighted coordinates on genus 2 hyperelliptic curves*, Cryptology ePrint Archive, Report 2002/153, 2002. <http://eprint.iacr.org/>.
- [22] K. Matsuo, J. Chao, and S. Tsujii, *Fast genus two hyperelliptic curve cryptosystems* In ISEC2001-31, IEICE, 2001.
- [23] V. Miller, *Use of elliptic curves in cryptography*, in H. C. Williams, editor, Advances in Cryptology –CRYPTO’ 85, Lecture notes in Computer Science volume 218, Springer-Verlag, Berlin (1986), 417-426.
- [24] D. Mumford, *Tata lectures on theta II*, Progress in Mathematics, Number 43, Birkhauser (1984).
- [25] V. K. Murty, *The addition law on hyperelliptic Jacobians*, in S. Adhikari, S. A. Katre, B. Ramakrishnan, editors, Current Trends in Number Theory, Hindustan Book Agency, New Delhi (2002), 101-110.
- [26] K. Nagao, *Improving group law algorithms for Jacobians of hyperelliptic curves*, ANTS IV, Lecture Notes in Computer Science Volume 1838, Editor W. Bosma, Springer-Verlag, Berlin, (2000), 439-448.
- [27] Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology, <http://csrc.nist.gov/>

- [28] Jan Pelzl, Thomas Wollinger, Jorge Guajardo, and Christof Paar, *Hyperelliptic curve cryptosystems: closing the performance gap to elliptic curves (Update)*, *Cryptology ePrint Archive: Report 2003/026*, 2003.
- [29] Y. Sakai and K. Sakurai, *Design of hyperelliptic cryptosystems in small characteristic and a software implementation over F_{2^n}* , Advances in Cryptology, ASIACRYPT 98, Lecture Notes in Computer Science 1514, Springer Verlag, Berlin, (1998), 80-94.
- [30] Y. Sakai and K. Sakurai, *On the practical performance of hyperelliptic curve cryptosystems in software implementation*, IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, Volume E83-A No.4, IEICE Transactions, (April 2000), 692-703.
- [31] Y. Sakai, K. Sakurai, and H. Ishizuka, *Secure hyperelliptic cryptosystems and their performance*, IN Public Key Cryptography, Lecture Notes in Computer Science 1481, Springer-Verlag, Berlin, (1998), 164-181.
- [32] R. Schoof, *Elliptic curves over finite fields and the computation of square roots mod p* , Mathematics of Computation 44 (1985), 483-494.
- [33] R. Schoof, *Counting points on elliptic curves over finite fields*, Journal de Theorie des Nombres de Bordeaux 7 (1995), 219-254.
- [34] A. Stein, *Sharp upper bounds for arithmetic in hyperelliptic function fields*, Journal of the Ramanujan Mathematical Society, Volume 16, No. 2 (2001), 119-203.
- [35] T. Satoh, *The canonical lift of an ordinary elliptic curve over a finite field and its point counting*, Journal of Ramanujan Mathematical Society 15, (200), 247-270.
- [36] N. Thériault, *Index calculus attack for hyperelliptic curves of small genus*, Advances in Cryptology, Asiacrypt 2003, Lecture Notes in Computer Science 2894, Springer-Verlag (2003).
- [37] A. Weng, *A class of hyperelliptic CM-curves of genus 3 of genus three*, Journal of the Ramanujan Mathematical Society 16, No. 4 (2001), 339-372