

Rethinking Low Genus Hyperelliptic Jacobian Arithmetic over Binary Fields: Interplay of Field Arithmetic and Explicit Formulæ

R. Avanzi, N. Thériault, and Z. Wang

Communicated by

Abstract. In this paper, we present several improvements on the best known explicit formulæ for hyperelliptic curves of genus three and four in characteristic two, including the issue of reducing memory requirements.

To show the effectiveness of these improvements and to allow a fair comparison of the curves of different genera, we implement all formulæ using a highly optimized software library for arithmetic in binary fields. This library was designed to minimize the impact of a whole series of overheads which have a larger significance as the genus of the curves increases. The current state of the art in attacks against the discrete logarithm problem is taken into account for the choice of the field and group sizes. Performance tests are done on two personal computers with very different architectures.

Our results can be shortly summarized as follows: Curves of genus three provide performance similar, or better, to that of curves of genus two, and these two types of curves can perform faster than elliptic curves – indeed on some processors often twice as fast. Curves of genus four attain a performance level comparable to elliptic curves. A large choice of curves is therefore available for the deployment of curve-based cryptography, with curves of genus three and four providing their own advantages as larger cofactors can be allowed for the group order.

Keywords. Elliptic and hyperelliptic curves, cryptography, efficient implementation, explicit formulæ.

AMS classification. 14Q05, (11G20, 14H45).

1 Introduction

The *discrete logarithm problem* (DLP) is a quite popular primitive for the design of cryptosystems. It can be formulated as follows: *Given a group G generated by an element g , and a second element $h \in G$, find an integer t – called the discrete logarithm of h with respect to g – with $t \cdot g = h$.* The computation of scalar multiples of elements of a group (i.e. given an integer t and an element g , compute $t \cdot g$) is the fundamental operation of a DLP-based cryptosystem. The rational point groups of elliptic curves (EC) [35, 23] and hyperelliptic curves (HEC) [24] of higher genus have been suggested as groups for DLP-based cryptosystems.

After a slow start, the use of EC in cryptography has gained considerable momentum, their use is now regulated by standards, such as the NIST FIPS–2 document [12], and there are several books covering the subject (for example [6, 20, 7, 4]). HEC have been enjoying increasing attention in recent years. They have long been thought not to be competitive with EC because of construction and performance issues, but the situa-

tion has changed in the last few years. It is now possible to efficiently construct HEC whose Jacobian has cryptographically good order (for an overview see [4, Ch. 17]), and the performance has been considerably improved. For curves of genus two over binary fields, the fastest explicit formulæ are given in [27], and active research has also been done for other genera. A recent summary can be found in [4, Chs. 13 and 14], but of course research has not stopped: Recent improvements for genus three can be found in [11], and further results for genus three and four are the main subject of the present paper.

No subexponential algorithm is known for solving the DLP on elliptic and hyperelliptic curves of genus at most four (cfr. [4, Chs. 19 to 21] for an overview of the techniques involved). For curves of genus one and two, the index calculus method is not faster than Pollard’s methods, therefore the security level of these curves is assessed by the number of operations required to solve the DLP with Pollard’s Rho algorithm. For curves defined over a fixed field and increasing genus, the complexity of computing the discrete logarithm becomes subexponential in the group order [10] by using *index calculus* methods (see also [4, Ch. 20]), but for “small” fixed genus, the complexity of the methods remains exponential. Starting with genus three, one has to increase the number of bits of the group order by a constant factor.

On the other hand, the best algorithms for solving the factorization problem and the DLP in finite fields are subexponential. Therefore, to achieve a security increase equivalent to *doubling* the RSA key size, one needs to *add only a few bits* to an EC group. For example, [28] states that the security of 1323 bit RSA (or of a 137 bits subgroup of a 1024 bit finite field) is attained by an EC over a 157 bit field and with a group of prime order (of 157 bits). For that same level of security, the field would have 79 bits for a HEC of genus two, 59 bits for genus three and 53 bits for genus four (for all three, the curve must have a prime subgroup of order at least 157 bits). In comparison, the security of 2048 bits RSA is (roughly) achieved by 200-bits curve groups, and that of 3072 bits RSA by 240-bits curve groups [9]. NIST [39] suggests to use 224 and 256 in place of the bit sizes of 200 and 240. There are obvious bandwidth and performance advantages in using curve based systems, in particular when the security requirements increase. Curves of genus of up to three are now heavily investigated alternatives, but one of the purposes of this paper will be to show that systems based on curves of genus four may also prove interesting.

In this paper, we reconsider the issue of efficient implementation of low genus hyperelliptic Jacobians over fields of characteristic two. We briefly recall the state of the art of implementation of HEC arithmetic in low genus: This will motivate our investigations.

Explicit formulæ for curves of genus two have been studied extensively [21, 30, 25, 41, 27, 26], both in the odd and even characteristic cases. An overview of the different results can be found in [4, Ch. 14]. In particular, the fastest known algorithm for curves over binary fields makes use of the doubling formula by Lange and Stevens [27]. In that paper, the authors use Shoup’s NTL software library [44] for the field arithmetic and find that the best cases for curves of genus two perform about 10% better than elliptic curves.

A number of papers are also available on explicit formulæ for curves of genus three [37, 25, 19, 11] in both odd and even characteristic. A comparison with the performances of curves of genus one and two in odd characteristic can be found in [1], and some comparisons for characteristic two are found in [50].

The literature is much more restricted when it comes to curves of genus four, with only the formulæ of Pelzl, Wollinger and Paar for characteristic two [41], and no detailed comparison with the performance of other genera that includes a security analysis and a careful choice of fields.

This paper differs from previous literature because we simultaneously consider field arithmetic enhancements, the derivation of the explicit formulæ and the impact of recent attacks, as well as the interplay of all of these factors. The ensuing surprising implementation results can be listed in the following two groupings:

- (1) A thorough approach to finite field implementation can be used to deliver performance essentially independent of the granularity of the computing architecture employed. Special finite field functions can be used to speed up explicit formulæ by up to 20% and the impact of such functions increases with the genus. The implementation is described in [5].
- (2) The genus two formulæ by Lange and Stevens in even characteristic can improve on EC performance by as much as 50%, but curves of genus three deliver similar performance while using even smaller fields. Moreover, curves of genus four perform in fact quite well, often improving on EC performance.

Section 2 gives a detailed overview the general techniques used to derive explicit formulæ. In Section 3 we recall a few facts about our implementation [5] of the field arithmetic. Our improvements to the explicit formulæ are presented in Section 4. Security considerations are the subject of Section 5, where key and field size equivalence issues are addressed. A description of our experiments and the corresponding results are given in Section 6.1. Finally, we draw conclusions in Section 6.2.

2 From Cantor's algorithm to explicit formulæ

An excellent introduction to hyperelliptic curves is [33]. It includes elementary proofs of many facts used implicitly below. A more geometric presentation is given in [4, Chs. 4, 12 and 13].

Let \mathbb{F}_q be a finite field of characteristic two. Let us consider a hyperelliptic curve of genus g explicitly given by an equation of the form $y^2 + h(x)y = f(x)$ over the field \mathbb{F}_q , with $\deg(f) = 2g + 1$ and $\deg(h) \leq g$.

In general, the points on a hyperelliptic curve do *not* form a group (the notable exception being represented by the hyperelliptic curves of genus one, i.e. elliptic curves). Instead, the *divisor class group* of \mathcal{C} is used. The elements of this group are quotient classes of formal sums of points on the curve – these sums are called *divisors*. They can be represented as pairs of polynomials $[u(x), v(x)]$ such that (i) $\deg(u) \leq g$; (ii) $\deg(v) < \deg(u)$; and (iii) $u(x)$ divides $v(x)^2 + v(x)h(x) - f(x)$. This representation is usually attributed to Mumford [36]. Such a pair of polynomials is called a *reduced*

divisor. When the first condition (degree of u) is ignored, the divisor is called *semi-reduced*.

Algorithm 1. Group operation for hyperelliptic Jacobians in characteristic two

INPUT: Reduced divisors $D_1 = [u_1(x), v_1(x)]$ and $D_2 = [u_2(x), v_2(x)]$

OUTPUT: Reduced divisor $D_3 = [u_3(x), v_3(x)]$, $D_3 = D_1 + D_2$

1. **Composition:** $[u_C, v_C] = D_1 + D_2$ (semi-reduced)
 2. $d = \gcd(u_1, u_2, v_1 + v_2 + h) = r_1 u_1 + r_2 u_2 + r_3 (v_1 + v_2 + h)$
[Extended Euclidean Algorithm]
 3. $u_C \leftarrow u_1 u_2 / d^2$
 4. $v_C \leftarrow v_2 + \frac{u_2}{d} r_2 (v_1 + v_2) + r_3 \frac{v_2^2 + h v_2 + f}{d} \pmod{u_C}$
 5. **Reduction:** $D_3 = [u_3, v_3]$ (reduced)
 6. $\tilde{u}_0 \leftarrow u_C, \tilde{v}_0 \leftarrow v_C$
 7. **for** $i = 0$ **while** $\deg(\tilde{u}_i) > g$ **do**
 8. $\tilde{u}_{i+1} \leftarrow \text{Monic}\left(\frac{\tilde{v}_i^2 + h \tilde{v}_i + f}{\tilde{u}_i}\right)$
 9. $\tilde{v}_{i+1} \leftarrow \tilde{v}_i + h \pmod{\tilde{u}_{i+1}}$
 10. $i \leftarrow i + 1$
 11. $u_3 \leftarrow \tilde{u}_i, v_3 \leftarrow \tilde{v}_i$
-

For computational purposes, the group operation is based on Cantor's algorithm [8] that operates directly with elements in Mumford's representation. Cantor's original version worked only in odd characteristic and was later extended to work over all fields by Koblitz [24]. Algorithm 1 gives the algorithm restricted to curves of characteristic two. Note that step 4 is simply an interpolation to insure that $v_C(x)$ is congruent to $v_1(x)$ modulo $u_1(x)/d(x)$ and congruent to $v_2(x)$ modulo $u_2(x)/d(x)$.

The idea behind explicit formulæ is to replace the polynomial-based form of Cantor's algorithm by a coefficient-based approach [37]. These formulæ are case-specific, i.e. they depend on whether the divisors are distinct (*addition*) or equal (*doubling*), on the degrees of the polynomials involved, etc. (for a detailed case consideration in genus two, see [26] for example; for genus three see [19].) This approach has a number of advantages which result in a significant speed-up in the computations:

- Conditional statements are reduced to a minimum. Polynomial arithmetic is inherently dependent on conditional loops (mainly on the degree of the polynomial), which cannot be avoided in a general setting.
- Coefficients which have no impact on the final result are no longer computed. This is quite evident in step 8 where we do not compute the coefficients of x of degree less than $\deg(\tilde{u}_i)$ in $\tilde{v}_i^2 + h \tilde{v}_i + f$, since we know that the division $\frac{\tilde{v}_i^2 + h \tilde{v}_i + f}{\tilde{u}_i}$ is exact.

- In Cantor’s algorithm, some of the partial computations may be done twice, with only the variable names being different. These duplications are avoided in the explicit formulæ.
- Parts of the algorithm can be replaced by more efficient techniques that cannot be used in a general setting. Sections 4.1 and 4.3 are good examples of these situations.

We also take advantage of the following observations, which are used by nearly every author in the development of explicit formulæ, beginning with Harley [21]:

- For almost all reduced divisors $D = [u(x), v(x)]$, $u(x)$ has degree g .
- For almost all pairs of polynomials u and v with $u \mid v^2 + hv + f$, the degree of $v \bmod u$ is $\deg(u) - 1$.
- Almost all randomly chosen polynomials are relatively coprime.

In all three of these cases, “almost-all” can be interpreted as “all but a proportion of size $O(g/q)$ ”. This means that if we concentrate on developing additions formulæ which apply to the most general case then only a negligible proportion of all group operations requires a different implementation (e.g. the general Cantor algorithm) which has no noticeable impact on the computation of $[e]D$.

To reduce computational cost (mostly for the doublings), we restrict ourselves to curves of the form

$$y^2 + y = x^7 + f_3x^3 + f_1x + f_0 \quad \text{or} \quad y^2 + y = x^9 + f_7x^7 + f_3x^3 + f_1x + f_0 \quad (2.1)$$

for genus three¹ and four respectively. These forms are almost-reduced representatives of the isomorphism classes of curves with $h(x) = \text{constant}$ when $\gcd(\log_2 q, 6) = 1$. In the genus four case, we ask that $f_7 \neq 0$ (see Section 5 for the security aspects).

The form of the most common cases are very similar for genus three and four. The main difference is that $\frac{v_1^2 + hv_1 + f}{\tilde{u}_1}$ is already monic in the genus three formulæ, simplifying the computation of \tilde{u}_2 .

Since the formulæ are in terms of the coefficients instead of polynomials, we will denote p_i the coefficient of x^i in $p(x)$. As there could easily be confusions with the polynomials $u_1(x)$, $u_2(x)$ and $u_3(x)$, as well as $v_1(x)$, $v_2(x)$ and $v_3(x)$, we will denote their coefficients differently. We will denote the coefficients of $u_1(x)$ with a_i , those of $u_2(x)$ with b_i , those of $v_1(x)$ with c_i , those of $v_2(x)$ with d_i , those of $u_3(x)$ with e_i , and finally those of $v_3(x)$ with ϵ_i . We also replace $\tilde{u}_1(x)$ and $\tilde{v}_1(x)$ from steps 8 and 9 by $u_T(x)$ and $v_T(x)$.

2.1 Addition formula

For the addition, we want to compute $D_1 + D_2$ where the Mumford representations of the summands are $D_1 = [u_1(x), v_1(x)]$ and $D_2 = [u_2(x), v_2(x)]$, with $\deg(u_1) =$

¹We thank Peter Birkner for pointing out to us that using isomorphisms to have $f_5 = 0$ reduces the cost of the doubling and directing us to [11].

$\deg(u_2) = g$, $\deg(v_1) = \deg(v_2) = g - 1$ and $\gcd(u_1, u_2) = 1$. At step 2, we can take $r_3 = 0$ (since $\gcd(u_1, u_2)$ is already 1), and then $r_2 = u_2^{-1} \bmod u_1$, giving us $u_C = u_1 u_2$ and $v_C = v_1 + u_1(u_1^{-1} \bmod u_2)(v_1 + v_2) \bmod u_C$ at steps 3 and 4.

Algorithm 2. Group addition, most common case

INPUT: Reduced divisors $D_1 = [u_1(x), v_1(x)]$ and $D_2 = [u_2(x), v_2(x)]$

OUTPUT: Reduced divisor $D_3 = [u_3(x), v_3(x)]$, $D_3 = D_1 + D_2$

1. Almost inverse, $inv(x) = r \cdot u_1(x)^{-1} \bmod u_2(x)$
 2. $r = inv(x) \cdot u_1(x) \bmod u_2(x)$
 3. $s'(x) = r \cdot s(x)$, where $s(x) = u_1(x)^{-1} \cdot (v_2(x) + v_1(x)) \bmod u_2(x)$
 4. computation of inverses and $\tilde{s}(x) = Monic(s'(x))$
 5. $u_T(x) = \left\lfloor \frac{\tilde{s}(x)^2 u_1(x)}{u_2(x)} \right\rfloor + s_{g-1}^{-2} (x + a_{g-1} + b_{g-1})$
 6. $z(x) = \tilde{s}(x) u_1(x)$
 7. $v_T(x) = s_{g-1} z(x) + v_1(x) + 1 \bmod u_T(x)$
 8. $u_3(x) = Monic\left(\frac{f(x) + v_T(x) + v_T(x)^2}{u_T(x)}\right)$
 9. $v_3(x) = v_T(x) + 1 \bmod u_3(x)$
-

We then write $v_C(x)$ as $v_1(x) + s(x)u_1(x)$ where $s = (u_1^{-1} \bmod u_2)(v_1 + v_2) \bmod u_2$, and substitute the values of u_C and v_C (and $h(x) = 1$) in the first reduction step to simplify the equations:

$$\begin{aligned} u_T &= Monic\left(\frac{(v_1 + su_1)^2 + (v_1 + su_1) + f}{u_1 u_2}\right) \\ &= Monic\left(\frac{v_1^2 + hv_1 + f}{u_1 u_2} + \frac{s^2 u_1}{u_2} + \frac{s}{u_2}\right) \end{aligned} \quad (2.2)$$

and $v_T = v_1 + su_1 + 1 \bmod u_T$.

By construction of u_C and v_C , the division in step 8 is exact, so we can look at the quotient (denoted $\lfloor \cdot \rfloor$) and ignore the fractional parts of each of the terms in Equation 2.2.

- $\left\lfloor \frac{v_1^2 + hv_1 + f}{u_1 u_2} \right\rfloor$ is linear of the form $x + a_{g-1} + b_{g-1}$ (the sum of the coefficients of x^{g-1} in u_1 and u_2).
- $\left\lfloor \frac{s}{u_2} \right\rfloor = 0$ since $\deg(s) < \deg(u_2)$.
- The bulk of the computation is in $\left\lfloor \frac{s^2 u_1}{u_2} \right\rfloor$. Even though su_1 is required to compute v_T , it is more efficient to compute $s^2 u_1$ by first squaring s and then multiplying by u_1 .

Since we want $u_T(x)$ to be monic, it is more efficient to first make $s(x)$ monic (we will call this polynomial $\tilde{s}(x)$), compute $\left\lfloor \frac{\tilde{s}^2 u_1}{u_2} \right\rfloor$ and then multiply $\left\lfloor \frac{v_1^2 + h v_1 + f}{u_1 u_2} \right\rfloor$ by the corresponding factor. To make this idea more advantageous, we also replace $u_1^{-1} \bmod u_2$ (in the computation of s) by an *almost inverse* (the product of the inverse by a constant r). Many methods used to compute $u_1^{-1} \bmod u_2$ do in fact compute an almost inverse first and then multiply it by r^{-1} , so this is easily implemented. Given $inv(x)$, the almost inverse of $u_1(x)$ modulo $u_2(x)$, we define $s'(x) = inv(x)(v_1(x) + v_2(x)) \bmod u_2(x)$ and use it instead of $s(x)$.

The main advantage is that computing \tilde{s} from s or s' requires the same amount of work, so it is much more efficient to skip the computation of s . Once $s'(x)$ and r are known, it is relatively easy to compute \tilde{s} , s_{g-1} and s_{g-1}^{-2} , so u_T and v_T can be computed as

$$u_T = \left\lfloor \frac{\tilde{s}^2 u_1}{u_2} \right\rfloor + s_{g-1}^{-2} (x + a_{g-1} + b_{g-1}) \quad \text{and} \quad v_T = v_1 + s_{g-1} \tilde{s} u_1 + 1 \bmod u_T .$$

2.2 Doubling formula

The reason for choosing $h(x) = 1$ becomes apparent in the doubling formula. Here we want to compute $D_1 + D_1$ with $D_1 = [u_1(x), v_1(x)]$, $\deg(u_1) = g$ and $\deg(v_1) = g - 1$.

In step 2, we find $d = \gcd(u_1, u_1, 2u_1 + h) = \gcd(u_1, u_1, 1) = 1$, so we can take $r_1 = r_2 = 0$ and $r_3 = 1$. This situation is very favorable since we get $u_C = u_1^2$ (at step 3) and $v_C = v_1^2 + f \bmod u_C$ (at step 4). The computation of $u_C(x)$ and $v_C(x)$ can then be done in only $2g$ field squarings (to compute u_1^2 and v_1^2).

Algorithm 3. Group doubling, most common case

INPUT: Reduced divisors $D_1 = [u_1(x), v_1(x)]$

OUTPUT: Reduced divisor $D_3 = [u_3(x), v_3(x)]$, $D_3 = 2 \cdot D_1$

1. $u_C(x) = u_1(x)^2$
 2. $v_C(x) = v_1(x)^2 + f(x) \bmod u_C(x)$
 3. computation of inverses
 4. $u_T(x) = \text{Monic} \left(\frac{f(x) + v_C(x) + v_C(x)^2}{u_C(x)} \right)$
 5. $v_T(x) = v_C(x) + 1 \bmod u_T(x)$
 6. $u_3(x) = \text{Monic} \left(\frac{f(x) + v_T(x) + v_T(x)^2}{u_T(x)} \right)$
 7. $v_3(x) = v_T(x) + 1 \bmod u_3(x)$
-

Because of this, the composition step can be done much faster than for generic curves (with $\deg(h) = g$). Since u_C and v_C can be computed at very little cost, there is no need to combine this step with the first reduction. The two reduction steps are then done as in Cantor's algorithm, but with a number of coefficients known to be zero (which reduces the cost even further).

3 Field Arithmetic

To provide the basis for a fair comparison of curves of varying genera at the same level of security, a special finite field library has been developed, which is fully described in [5]. We now recall some of the problems that arise when considering finite field arithmetic for curve cryptography, and the solutions we adopted.

- (1) *To process (relatively small) operands using loops induces costs, including branch mispredictions, whose impact is heavier for shorter loops.*

Since the relative impact of these cost can be quite significant for smaller fields, we use field-specific routines rather than general ones.

- (2) *Inlining is used to reduce function call overheads.*

Since the code of binary field multiplication routines is larger than for multiprecision integer arithmetic, we do not inline these, or the inversion routines, in the main code. However, optimized modular reduction code is included in the multiplication routines and not called separately.

- (3) *Architecture granularity also induces irregular performance penalties that increase with the genus.*

Although little can be done to defeat granularity problems in the prime field case [1], most of the disadvantages due to fields sizes that do not exactly match the computer word size can be eliminated in even characteristic (cf. [5, § 3.1]).

- (4) *In various places of the formulæ, several different field elements are multiplied by the same field element.*

This situation is similar to the multiplication of a vector by a scalar, and it is possible to speed up these multiplications appreciably. The technique for doing this is described in [5, § 3.2].

Tables containing the detailed costs of field operations can be found in [5, § 4]. We summarize these results in Table 1 with the average costs, in terms of the field multiplication, for the fields of 47 to 101 bits (those we will use for curves of genus three and four). These operations are: squaring (Sqr), multiplication of 2 to 5 different field elements by a common one (columns from Mul2 to Mul5) and inversion (Inv).

Table 1. Average costs of field operations (relative to multiplications)

Processor	Timings relative to multiplication					
	Sqr	Mul2	Mul3	Mul4	Mul5	Inv
PowerPC G4, 1.5 GHz	.147	1.663	2.214	2.746	3.282	4.701
Core 2 Duo, 1.83 GHz	.189	1.647	2.270	2.880	3.512	16.531

In Section 4.6, we will use the costs of Mul2, Mul3, Mul4 and Mul5 to evaluate the impact of using “sequential” multiplications in the formulas (see Section 4.2), obtaining the “effective” cost of the multiplications. To compute these numbers, we will use simplified values of those in Table 1. Pairs of multiplications will be evaluated at 1.65 single multiplication, blocks of 3 at 2.25, blocks of 4 at 2.8, and blocks of 5 at 3.4.

4 Improvements in the formulæ

We now turn our attention to the algorithmic methods we used to improve on the previous state of the art on explicit formulæ, in particular on the works of Guyot, Kaveh and Patankar (for genus three, [19]) and of Pelzl, Wollinger and Paar (for genus four, [42, 50]). We follow three main approaches:

- (1) *Reduce the number of inversions.* As inversions are usually much more costly than other operations, it is often a good idea to combine as many of them together, even if this means increasing the number of multiplications. This is already common practice for genus two and three, but it can be pushed further for genus four as we will describe in Section 4.1.
- (2) *Reduce the number of multiplications.* It goes without saying that if the number of multiplications in the explicit formulæ can be reduced, the overall performance will improve. Most explicit formulæ do this by introducing Karatsuba multiplication to compute the product of polynomials. We obtain further improvements by selecting faster algorithms (Section 4.3), combining multiplications using Karatsuba-like tricks (Section 4.4) and keeping products in memory if they are used again later in the formula.
- (3) *Combine multiplications with a repeated operand.* The goal here is to make use of the sequential multiplications. Although this does not affect the total operation count, this approach reduces the effective cost of some steps by more than 30%.

Note that approaches 2 and 3 are not always compatible. In most cases, reducing the number of multiplications using Karatsuba-like tricks will hinder the use of sequential multiplications.

For example, the computation of $inv(x) \cdot (v_2(x) + v_1(x))$ in the genus four addition formula takes 16 multiplications using classical methods. With sequential multiplications, we still have the same number of multiplications, but the effective cost is close to that of 11.2 normal multiplications. With two layers of Karatsuba multiplications, we can do this in 9 multiplications, but then no operand is used in more than one product so we cannot use sequential multiplications to get any further savings.

In this example it is clear that a Karatsuba-like approach is a better choice. In many cases however, the choice is not always so clear as the savings obtained by giving precedence to Karatsuba-like tricks or to sequential multiplications may be very close and will vary depending on the processor and the field size. To avoid writing a different formula for each field size, the formulæ described in this paper assume the average case. For some field sizes, the formula may not be completely optimal, although the saving that could be obtained by using a field-specific formula would be marginal at best.

4.1 Inversions

As for curves of genus three, the common case of group addition for curves of genus four requires two reduction steps. However, $\frac{f+v_t+v_t^2}{u_T}$ in the second reduction step for

genus four (Step 8 of Algorithm 2) is not automatically monic since the leading term in $f + v_t + v_T^2$ is $v_{T,5}^2 x^{10}$. For these curves, we must therefore compute the inverses of s_3 (to make u_T monic), r (to get s_3 , for the computation of v_T), s'_3 (to make s' monic) and $v_{T,5}$ (to make u_3 monic).

The inverses of s_3 , r and s'_3 can be combined in the same way it is done for curves of genus three, but this still leaves us with two inversion to be performed. At the time the inverses of s'_3 and s_3 are obtained, only $inv(x)$, r and $s'(x)$ have been computed. To minimize the number of inversions, one has to express $v_{T,5}$ in terms of r and the coefficients $s'(x)$ and $u_2(x)$.

To improve readability, we use the coefficients of the polynomials $u_T(x)$, $z(x) = \tilde{s}(x)u_1(x)$, $s(x) = \frac{1}{r}s'(x)$ and $\tilde{s}(x) = \frac{1}{s'_3}s'(x)$, even though those polynomials are not computed before the inverses. Since $v_T(x) = s_3z(x) + v_1(x) + 1 \pmod{u_T(x)}$, we have

$$v_{T,5} = s_3(z_5 + u_{T,4} + u_{T,5}(z_6 + u_{T,5})) .$$

Furthermore, replacing $\tilde{s}(x)^2 u_1(x)$ by $\tilde{s}(x)z(x)$ in the equation for $u_T(x)$, we have

$$u_T(x) = \left\lfloor \frac{\tilde{s}(x)z(x)}{u_2(x)} \right\rfloor + s_3^{-2}(x + a_3 + b_3) ,$$

so $u_{T,5} = z_6 + \tilde{s}_2 + b_3$ and $u_{T,4} = z_5 + \tilde{s}_2 z_6 + \tilde{s}_1 + b_2 + u_{T,5} b_3$. Substituting back into the equation of $v_{T,5}$ and using the definitions of $s(x)$ and $\tilde{s}(x)$, we get

$$v_{T,5} = s_3(\tilde{s}_1 + b_2 + \tilde{s}_2^2 + b_3 \tilde{s}_2) = \frac{1}{rs'_3} \left(s_2'^2 + s'_3(s'_1 + b_3 s'_2 + b_2 s'_3) \right) .$$

Since $1/rs'_3$ is not yet known, we replace the computation of $v_{T,5}$ by the computation of

$$rs'_3 v_{T,5} = s_2'^2 + s'_3(s'_1 + b_3 s'_2 + b_2 s'_3) .$$

To obtain all the inverses required, we first compute rs'_3 and $rs'_3 v_{T,5}$, and let $t = ((rs'_3) \cdot (rs'_3 v_{T,5}))^{-1}$. Then, the inverses of $v_{T,5}$ and rs'_3 are obtained as

$$t \cdot (rs'_3)^2 = \frac{1}{v_{T,5}} \quad \text{and} \quad t \cdot (rs'_3 v_{T,5}) = \frac{1}{rs'_3} .$$

The inverses of s_3 , r and s'_3 are then obtained as before. In this manner, we combine the final inversion with the previous ones, at the cost of an extra five multiplications and two squarings. Note that the computation of $v_{T,5}$ (needed for the final step) can then be done in one multiplication instead of two and that we no longer need to compute z_5 (saving one more multiplication). This approach reduces computation time if an inversion costs more than three multiplications and two squarings.

For the doubling formula, the approach must be modified slightly since in this case $u_C(x)$ and $v_C(x)$ are computed directly. This time, we need the inverses of $v_{C,7}$ and $v_{T,5}$ and we have

$$v_{C,7} v_{T,5} = v_{C,6}^2 + v_{C,5} v_{C,7} + u_{C,6}(v_{C,7}^2) .$$

We then compute

$$t = \frac{1}{v_{C,7}(v_{C,7}v_{T,5})}$$

and the inverses of $v_{T,5}$ and $v_{C,7}$ are found as

$$t \cdot (v_{C,7}^2) = \frac{1}{v_{T,5}} \quad \text{and} \quad t \cdot (v_{C,7}v_{T,5}) = \frac{1}{v_{C,7}} .$$

The second inverse is then replaced by five multiplications and two squarings, of which one ($v_{C,6}^2$) is used again at a later step. Since two of these multiplications can be combined, the exchange is advantageous if one inversion cost more than (roughly) 4.65 multiplications and one squaring. This trade-off is slightly to our disadvantage on the PowerPC G4 (by about 0.1 multiplications on average for the fields used for genus four testing), but we still opted to implement it as it always produces significant savings in other processors.

4.2 Sequential multiplications

In the explicit formulæ for the curves of genus two to four, we can find several sets of multiplications with one common term. Since the fastest multiplication routines for binary fields are based on the method of López and Dahab [29], which uses precomputations from one of the operands, it seems natural to want to preserve the precomputations associated to the common operand and to re-use them in the following multiplications.

In order to implement this idea as cleanly as possible and simplify memory management, we use a new multiplication routine for sets of multiplications with a common operand. This routine does the precomputations for the common operand first, and then performs all the multiplications. We call this type of operation *sequential multiplication*. See Subsection 3.2 of [5] for more details. Because of the design of this routine, the precomputations never leave the (sequential) multiplication routine itself and are not available to the other routines. As a result, even if the common operand is used in another multiplication later on in the formula (where the second input depends on the outputs of the first set of multiplications), that new multiplication must be treated as completely independent.

For historical reasons, we need to mention that using static precomputations has already been done for multiplications by a constant parameter (coming from the curve or the field) – this is for example suggested in the context of square root extraction in [13]. However, we found no references on adapting the amount of precomputations to the number of multiplications by the same value.

4.3 Almost inverse

One of the most costly steps of explicit formulæ is the computation of the almost inverse. Most of the explicit formulæ published so far [21, 37, 30, 25, 41, 40, 42, 19, 27, 26] find the almost inverse via the computation of a resultant. However, this approach

is not necessarily optimal. For every genus bigger than two, the almost inverse can be computed with fewer field multiplications using Cramer's rule (for genus two, the cost are the same if both methods are implemented carefully). This approach has the added bonus of taking full advantage of sequential multiplications, making it even more efficient.

Cramer's rule computes a solution \mathbf{v} to the system $M\mathbf{v} = \mathbf{w}$ where M is an $n \times n$ matrix. The solution is

$$\mathbf{v} = \frac{1}{|M|} \begin{pmatrix} |Sub_0(M, \mathbf{w})| \\ |Sub_1(M, \mathbf{w})| \\ \vdots \\ |Sub_{n-1}(M, \mathbf{w})| \end{pmatrix}$$

where $Sub_i(M, \mathbf{w})$ is matrix M with the i^{th} column replaced by \mathbf{w} . However, since we want an almost inverse, we only compute $|M|$ and the column vector on the right hand side, which give us the inv_i 's. In this case, the i^{th} column of the matrix M is formed by the coefficients of $x^i u_1(x) \bmod u_2(x)$.

The regular structure of these computations makes them very convenient to do in combination with sequential multiplications. For example, in the genus four formulæ, we obtain 11 blocks of products (3 to compute M and 8 to compute the inv_i 's), leaving only the products in the computation of r as single multiplications and allowing us to perform the first 36 multiplications for the price of around 26.4 normal ones. Note that in the formulæ, the computation of r is done with some of the computations for $s'(x)$ in order to combine some of the multiplications into pairs (and use sequential multiplications for pairs of products).

4.4 Polynomial divisions

Although Karatsuba-like techniques are usually applied to polynomial multiplication, they can also be used when dealing with polynomial divisions (both for the quotient and the remainder). We consider three main cases: The reduction of $inv(x)(v_2(x) + v_1(x))$ modulo $u_2(x)$ (computation of $s'(x)$), the division on $(\tilde{s}(x)^2) \cdot u_1(x)$ by $u_2(x)$ (computation of $u_T(x)$), and the reduction of $v_T(x) + 1$ modulo $u_3(x)$ (computation of $v_3(x)$).

We give the details only for the computation of $u_T(x)$. For this computation, most of the work involves computing the quotient of $(\tilde{s}(x)^2 \cdot u_1(x))$ divided by $u_2(x)$. Since $u_2(x)$ has degree 4, the coefficients of x^i for $i < 4$ in $\tilde{s}(x)^2 \cdot u_1(x)$ do not have to be computed as they have no impact on the result. We assume that $p(x) = \tilde{s}(x)^2 \cdot u_1(x) = x^{10} + p_9 x^9 + p_8 x^8 + p_7 x^7 + p_6 x^6 + p_5 x^5 + p_4 x^4 + \dots$, with $p_9 = a_3$, is already computed, and we have to compute the coefficients of $u_T(x) = [p(x)/u_2(x)] + s_3^{-2}(x + a_3 + b_3)$. To compensate for the linear term which is missing in the division, we let $u_{T,1}^* = u_{T,1} + s_3^{-2}$ and $u_{T,0}^* = u_{T,0} + (a_3 + b_3)s_3^{-2}$. Also, the products of b_3, b_2, b_1 and b_0 by $u_{T,5} = (a_3 + b_3)$ are already known from the computation of the almost inverse (using Cramer's rule),

so we only need to compute products of the form

$$\begin{aligned}
u_{T,3} &= sum_3 + b_3u_{T,4} \\
u_{T,2} &= sum_2 + b_3u_{T,3} + b_2u_{T,4} \\
u_{T,1}^* &= sum_1 + b_3u_{T,2} + b_2u_{T,3} + b_1u_{T,4} \\
u_{T,0}^* &= sum_0 + b_3u_{T,1}^* + b_2u_{T,2} + b_1u_{T,3} + b_0u_{T,4} .
\end{aligned}$$

Even though we are looking at a polynomial division rather than a polynomial multiplication, many of the terms follow a structure which is similar, so it is natural to apply the idea of Karatsuba multiplication. Looking at the products, we find different combinations of Karatsuba-like pairs for which the two other products are also required: $b_3u_{T,3} + b_2u_{T,4}$, $b_3u_{T,2} + b_2u_{T,3}$ and $b_2u_{T,3} + b_1u_{T,4}$. It is easy to see that whenever we use Karatsuba multiplication to reduce one of these pairs, it requires us to compute (on it's own) one half of the terms of each of the other two pairs, so it is impossible to use Karatsuba on two of these pairs at the same time. Therefore, one might be tempted to conclude that the most we can save using Karatsuba-like tricks in this situation is one multiplication. This conclusion is too hasty however: it is possible to reduce the operation count by adding one new product. If one also computes $b_1u_{T,2}$, then it becomes possible to add two more combinations ($b_3u_{T,2} + b_1u_{T,4}$ and $b_2u_{T,2} + b_1u_{T,3}$) to $b_3u_{T,3} + b_2u_{T,4}$, in effect reducing the number of multiplication by two instead of one. It is slightly more efficient to use sequential multiplications after reducing the number of multiplications as much as possible than using them on the original equations.

For the reduction $inv(x)(v_2(x) + v_1(x))$ modulo $u_2(x)$, one can see that three multiplications can be saved using Karatsuba-like methods, but then we can still combine some operations using sequential multiplications. By doing this, the result is 6 products with no terms in common and 3 multiplications by b_0 (handled sequentially), for an effective cost of around 8.25 multiplications (down from 12). Note that in this case, if sequential multiplications are applied to the naive multiplication method the effective cost would have been around 8.45 multiplications.

For the final step of the the genus four formulæ (that step is essentially identical for both the addition and the doubling) we have to compute $v_3(x) = v_T(x) + 1 \bmod u_3(x)$. We using Karatsuba-like methods, but the order of operations is somewhat non-standard. We do $e_3v_{T,5}$ and $e_1v_{T,5}$ first, so we also get $t = v_{T,4} + e_3v_{T,5}$; then e_2t and e_0t ; and finally $(e_3 + e_2)(v_{T,5} + t)$ and $(e_1 + e_0)(v_{T,5} + t)$ to obtain the two sets of combined products $e_2v_{T,5} + e_3t$ and $e_0v_{T,5} + e_1t$. In this way, we can get the effective cost down to around 4.95 multiplications (from the original 8).

For the genus four addition, there are three remaining cases that we do not discuss where a Karatsuba-like approach is used:

- The computation of $z(x) = \tilde{s}(x) \cdot u_1(x)$. The pattern of multiplications encountered are the same as for the reduction of $inv(x)(v_2(x) + v_1(x))$ modulo $u_2(x)$, but viewed upside down since z_5 is not required (see Section 4.1).
- The computation of $(\tilde{s}(x)^2) \cdot u_1(x)$ (before the division by $u_2(x)$). Since only the coefficients of powers of x greater than 4 are needed and $\tilde{s}(x)^2$ only contains even powers of x , it is not possible to obtain any saving from this approach.

- The computation of $u_3(x)$. Here we have only one possible way of reducing multiplications, by combining the products in $e_3u_{T,4} + e_2u_{T,5}$.

4.5 Variables

A very serious issue with the genus four formulæ (and, to a lesser extent, the genus three formulæ as well), is the number of variables involved. If we did not worry about the memory requirements of our implementation of the genus four group addition, it would use 229 variables (in fact 239 when we count the copies required to handle the “sequential” form we use for the sequential multiplications). This number of variables is obviously too large for certain constrained environments, and even with two or three words per field elements (as is the case for the security levels considered in this paper), this would mean a storage in the order of one kilobyte for the variables alone. At this point, the memory allocation might affect the performance of the computations even for high-end processors: even if this memory is allocated statically, its use may still take too much of the level 1 cache of the processor.

The natural solution is to reallocate variables spaces which we tried to do without losing any (significant) efficiency in exchange. The result is a drastic reduction in memory usage which can then ease deployment in environments with limited RAM. To minimize memory allocations as much as possible, we chose to define an array of field elements, whose address is passed as part of the function calls for the addition and doubling and where all the intermediate results of the group operations are stored.

To improve readability, the formulæ in the appendix are given in terms of distinct variables and they are accompanied by an allocation schedule for the variable array. For example, in the genus three doubling formula, variable space 0 is used to store (in that order) variables $u_{C,4}$, t_{12} and t_{14} , which is done without risk since a new variable is allocated that space only if the previous one is no longer useful for the computation. In a few cases, a variable is copied into a different location in the array to obtain an adjacent sequence that can be used for the sequential multiplications (this is due to our choice for the function call). Some of the longer expressions (sums) also had to be broken up into partial sums so variables spaces could be reallocated and used for other computations. The final values are kept with the other variables until all the operations are done so the function’s output can safely replace one of its inputs if desired.

The resulting formulæ require 14 variables for the genus three addition and 13 for the doubling. For genus four, we need 19 variables the doubling and 23 for the addition. In all cases, these numbers are minimal with the present formulæ as there are “bottlenecks” where all the variables are either in use for the current operation or contain values that were computed earlier and will be used later. In all cases, the amount of memory required for the variables is only between two and three times that of a group element (6 field elements in genus three and 8 in genus four).

4.6 Operation counts

The addition and doubling formulæ for genus three, with the tables of variable allocation, are in Appendix A, and those for genus four are in Appendix B. Tables 2 and 3

compare the operation counts of our formulæ with previous works. For the genus four addition, we note that changing from $h = x$ to $h = 1$ does not affect the operation count (except for the number of field additions), and that the difference in operation counts comes from our improvements (obviously the same cannot be said in the case of the doubling, where the form of the curve is the dominant factor). We give two sets of numbers for our formulæ: one where sequential multiplications are handled as normal multiplication (the “classical” cost) and one where they are done sequentially as in Subsection 4.2 (the “effective” cost). For Cantor’s and Nagao’s algorithms, we use the odd characteristic numbers as an indicator of the cost in characteristic two (the numbers that would be obtained by adapting these algorithms to even characteristic should differ only slightly).

Table 2. *Field operation counts for genus three group operations*

Reference	Char.	Curve properties	Addition	Doubling
Cantor [8]	odd	$h = 0$	4I + 200 M/S	4I + 207 M/S
Nagao [37]	odd	$h = 0$	2I + 144 M/S	2I + 153 M/S
Pelzl et al. [40]	two	$h = 1, f_6 = 0$	1I + 65 M + 6 S	1I + 14 M + 11 S
Guyot et al. [19]	two	$h = 1, f_6 = 0$	1I + 58 M + 6 S	1I + 11 M + 11 S
Fan et al. [11]	two	$h = 1, f_6 = f_5 = 0$	–	1I + 10 M + 11 S
this work, classical	two	$h = 1, f_6 = f_5 = 0$	1I + 57 M + 6 S	1I + 10 M + 11 S
this work, effective	two	$h = 1, f_6 = f_5 = 0$	1I + 46.7 M + 6 S	1I + 8.55 M + 11 S

Table 3. *Field operation counts for genus four group operations*

Reference	Char.	Curve properties	Addition	Doubling
Cantor [8]	odd	$h = 0$	6I + 386 M/S	6I + 359 M/S
Nagao [37]	odd	$h = 0$	2I + 289 M/S	2I + 268 M/S
Pelzl et al. [42]	two	$h = x, f_8 = 0$	2I + 148 M + 6 S	2I + 75 M + 14 S
this work, classical	two	$h = 1, f_8 = 0, f_7 \neq 0$	1I + 119 M + 10 S	1I + 28 M + 16 S
this work, effective	two	$h = 1, f_8 = 0, f_7 \neq 0$	1I + 96.65 M + 10 S	1I + 23.25 M + 16 S

5 Security

In this section, we describe how the different security levels were selected for the performance comparisons, as well as the security of the form of curve used.

For curves of genus one and two, the fastest known attack is Pollard’s Rho algorithm which take $O(\sqrt{\text{group order}})$ group operations. Since the group order of a curve of genus g over a field of q elements is $q^g + O(gq^{g-1/2})$, this means $O(\sqrt{q_1})$ group operations for elliptic curves over the field \mathbb{F}_{q_1} and $O(q_2)$ group operations for curves of genus two over the field \mathbb{F}_{q_2} .

For curves of genus three and four, the fastest known attack is the index calculus algorithm. Using the double large prime variations [18, 38], and (ignoring logarithmic terms), this attack requires $O(q^{2-2/g})$ group operations for a genus g over of field of q

elements. In terms of group operations, we get costs of $O(q_3^{4/3})$ for curves genus three over \mathbb{F}_{q_3} and $O(q_4^{3/2})$ for a curve of genus four over \mathbb{F}_{q_4} . To obtain an exact comparison with Pollard Rho, we should take into account any logarithmic term, as well as the underlying constants in both algorithms, however we decided to use only the $q^{2-2/g}$ term (assuming identical constants) to simplify the analysis. The constants are indeed expected to be of similar size [49], while the logarithmic terms are somewhat larger in the index calculus case (for the best proven running time [18]), so we are in fact disadvantaging a little the curves of genus three and four (with regards to currently known attacks).

For the discrete log to require the same amount on each curve, we need

$$\frac{1}{2} \log(q_1) \approx \log(q_2) \approx \frac{4}{3} \log(q_3) \approx \frac{3}{2} \log(q_4) ,$$

where q_g is the order of the field of definition for the curve of genus g . To compare with an EC over a field of n bits, we need a field of $n/2$ bits for genus two, $3n/8$ bits for genus three and $n/3$ bits for genus four. Note that the differences in performances in the group operations would force us to adjust the field size by a fraction of a bit to obtain really equivalent security, however this impact is minimal compared with the other issues dealt with in our security arguments, and are eliminated by the fact that the degree of the field extension must be an integer (and prime).

Since Pollard Rho can be adapted to take advantage of the existence of subgroups or knowledge of the key size, curves of genus one and two are assumed to groups of order twice a prime (the form of the curves forces the group order to be even) with keys of n bits. For genus three and four, the situation is different since the index calculus algorithm works on the algebraic group as a whole, so it cannot take advantage of the existence of subgroups or any information on the key (including the bit size). On the other hand, Pollard Rho could still be used if the subgroups were small enough, and Pollard Kangaroo can be used if the keys are known to be short enough. Therefore, to give an equivalent security level, the curves must have a prime-ordered subgroup of at least n bits and use keys at least n bits long.

The last remark is very important from an efficiency point of view, since it means that the same key (scalar) can be used for all four genera instead of having to increase the key length for genus three and four. The (sub)group sizes are also of interest, since finding curves whose order has a prime factor of at least n bits is much easier when the group order is larger in genus three and four ($9n/8$ and $4n/3$ bits respectively).

Another concern when choosing the fields is the Weil descent attack, which is a risk for some field extensions (see [17] for EC, [48] for HEC). Although these attacks may not weaken all the curves over a given field extension, recent developments [22, 32] show that for some extension degrees a large proportion of them are at risk. Gaudry [16] also showed that small factors in the extension degree can expose all curves defined over that field to a Weil descent-like attack. However, no known variation exists for prime extensions, so we ask that all fields be of the form \mathbb{F}_{2^p} , where p is a prime.

The only security aspect that remains to be discussed is the choice of form for the equations defining the curves. For genus one and two, curves of the form $y^2 + y = f(x)$

are supersingular and are exposed to the MOV [31] or Frey-Rück [14] attack and their hyperelliptic variant [15] so they should be avoided for designing DLP-based systems. We therefore selected curves of the form $y^2 + xy = f(x)$ for security and efficiency. Since we use curves of the form $y^2 + y = f(x)$ for genus three and four, it is natural to ask whether they are supersingular or not. Using results of Scholten and Zhu [43], we know that none of the curves of genus three over binary fields are supersingular, while the only supersingular curves of genus four over binary fields are of the (simplified) form $y^2 + y = x^9 + f_5x^5 + f_3x^3 + f_1x + f_0$. We can then safely use the special form for curves of genus three, and for genus four we note that none of the curve isomorphisms allow to map a curve of the form 2.1 with $f_7 \neq 0$ into a curve with $f_7 = 0$, so those curves cannot be supersingular.

To complete the security discussion, a short remark on a recent paper by Smith [45]. This paper shows how, in some hyperelliptic curves of genus three over fields of characteristic bigger than three, it may be possible to map the discrete logarithm to a non-hyperelliptic curve over the same field, where it can be computed more easily. Although this result reduces some of the gap in security between hyperelliptic and non-hyperelliptic curves, it currently has no impact on curves over binary fields. First of all, part of the curve structure used in this method is not present in curves over binary fields, so an extension to binary fields would require a new approach. Secondly, even for those fields where the attack may be applied, it requires the polynomial $f(x)$ of the curve equation to have a certain type of factorization or it will fail completely, leaving close to half of the curves completely secure (not to mention those curves where the attack fails because the field of definition of the new curve is in fact larger). Because of this, we did not take the method of Smith into account for our choice of field sizes.

6 Performance results and Conclusions

6.1 Timings

We now show the timings of our implementation of EC and of HEC jacobians of genus of up to four. Since our goal is to make a comparison between the performance of curves of different genera offering the same security level, we looked for quadruplets of degrees of field extensions (p_1, p_2, p_3, p_4) where $p_2 \approx p_1/2$, $p_3 \approx 3p_1/8$, and $p_4 \approx p_1/3$ (see Section 5), and used randomly chosen curves of genus i over $\mathbb{F}_{2^{p_i}}$ for $i = 1, 2, 3$, and 4. We admitted tolerances of at most 2% (in bits) of security level between the “most” and the “least” secure curves in each quadruplet. Despite the irregular distribution of primes, we can still find 8 good sets of matches for security levels ranging from roughly 160 bits (low-cost security) to 270 bits (high security), and in four of these some curves are missing (we included them to offer a broader range of cases).

For EC, we do not necessarily use the NIST fields, as they do not produce significant speedups compared to other binary fields (unlike NIST prime fields). Increasing the field size to have a NIST field for EC would in fact have disadvantaged them.

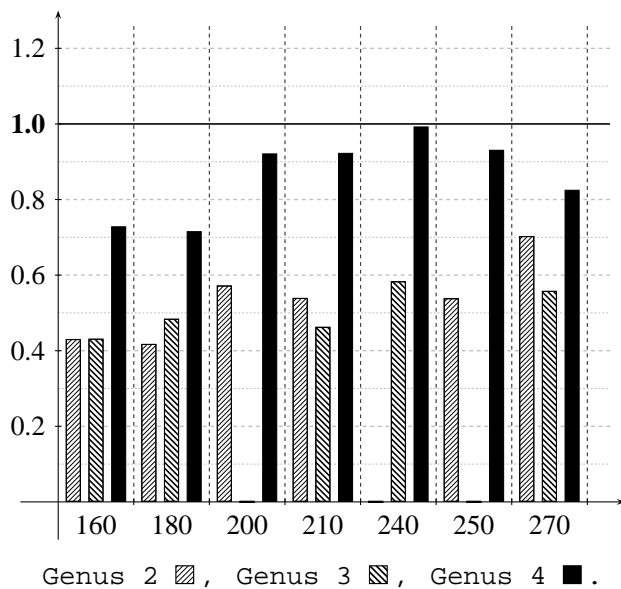
In Table 4, we report, for each curve and security level, the timings on a 1.5 GHz Powerpc G4 for the doubling of a group element (DBL) and the addition of two different group elements (ADD), then scalar multiplication timings using the non-adjacent

form (NAF) and a signed windowing method based on the NAF_w . The NAF_w can be defined as a binary-like representation of the integer $n = \sum_{j=0}^{\ell} n_j 2^j$ where the coefficients n_j are either 0 or odd of absolute value less than 2^{w-1} , and such that there are at least $w - 1$ zeros between any two non-zero coefficients. The NAF_w was introduced independently by Miyaji, Ohno, and Cohen [34] and by Solinas [46, 47] to speed up scalar multiplications, due to its property of having the smallest number of non-zero terms gives its coefficient sets (for variants and properties of the NAF_w see also [2] and references therein).

We also compare the timings of our formulæ with and without using sequential multiplications (i.e. replacing the routines described in Section 4.2 by separate multiplications).

We did not devise and implement projective coordinates for curves of genus greater than one. We either looked at the formulæ currently available in literature, or estimated the number of multiplications that such formulæ would require, and verified that in our situation (with relatively low inversion-to-multiplication ratios) projective coordinates would be more expensive. For EC the situation is slightly different, and the best option turned out to be to use mixed affine/López-Dahab coordinates. Figure 1 shows the timings of curves of genus at least two normalized with respect to EC performance.

Figure 1: *Scalar multiplication performance for curves of genera two, three and four at various security levels, relative to EC performance (normalized to 1.0) on a 1.5 Ghz Powerpc G4*



Other methods may be available to perform the scalar multiplication efficiently on some of the curves we considered. For instance, point halving methods ([13] and references therein) can improve scalar multiplication performance by about 25% in half

the possible elliptic curves. Cost estimates in [3] based on some recent improvements in field arithmetic implementation essentially confirm the findings in [13]. Note that the gains of halving-based methods are bound to decrease as the field size increases since that scalar-multiplication method is not inversion-free (unless a simple NAF is used). Even using point halving on these elliptic curves would not be enough to close the gap observed with curves of genus 2 and 3. Research on divisor halving for genus 2 and above is ongoing.

Table 4. *Scalar multiplication timings in μsec (1.5 GHz Powerpc G4)*

Sec. lvl.	Curve	Using sequential mults				No sequential mults			
		DBL	ADD	NAF	NAF _w	DBL	ADD	NAF	NAF _w
160	ec (157)	6.09	10.20	1517	1284	6.09	10.60	1539	1295
	g2 (79)	2.39	5.17	645	546	2.39	5.41	656	554
	g3 (59)	2.03	6.92	680	547	2.24	8.42	792	730
	g4 (53)	3.44	11.59	1146	924	4.04	14.36	1386	1112
180	ec (179)	7.33	12.27	2082	1756	7.33	12.47	2094	1762
	g2 (89)	2.78	6.42	880	734	2.78	6.67	895	743
	g3 (67)	2.80	9.49	1068	851	3.18	11.98	1283	1010
	g4 (59)	4.14	14.04	1579	1258	4.84	17.38	1904	1507
200	ec (199)	7.62	12.69	2410	2066	7.62	13.20	2443	2083
	g2 (101)	4.14	8.94	1417	1183	4.14	9.41	1448	1202
	g3 (no curve)	–	–	–	–	–	–	–	–
	g4 (67)	5.59	19.76	2422	1905	6.73	25.10	3004	2347
210	ec (211)	8.36	14.13	2827	2407	8.36	14.67	2865	2426
	g2 (107)	4.32	9.16	1557	1299	4.32	9.65	1592	1320
	g3 (79)	3.11	10.88	1422	1115	3.44	13.12	1648	1279
	g4 (71)	6.09	22.24	2849	2223	6.99	26.85	3363	2607
220	ec (no curve)	–	–	–	–	–	–	–	–
	g2 (109)	4.41	9.56	1673	1389	4.41	10.05	1708	1410
	g3 (83)	3.48	12.20	1659	1297	3.81	14.38	1893	1466
	g4 (73)	6.42	22.99	3098	2416	7.29	27.29	3606	2796
240	ec (239)	7.87	13.87	3058	2600	7.87	14.16	3081	2612
	g2 (no curve)	–	–	–	–	–	–	–	–
	g3 (89)	3.72	13.32	1958	1518	4.17	16.69	2335	1784
	g4 (79)	6.32	22.71	3332	2583	7.35	27.44	3958	3052
250	ec (251)	9.38	15.78	3758	3193	9.38	16.37	3808	3218
	g2 (127)	4.67	11.36	2115	1721	4.67	11.81	2152	1743
	g3 (no curve)	–	–	–	–	–	–	–	–
	g4 (83)	7.06	24.89	3838	2975	8.08	30.16	4534	3488
270	ec (269)	11.19	21.31	5021	4249	11.19	21.89	5079	4276
	g2 (137)	7.53	17.14	3576	2925	7.53	17.51	3610	2945
	g3 (101)	5.18	18.37	3052	2354	5.97	23.28	3707	2822
	g4 (89)	7.60	27.51	4528	3482	9.13	34.70	5588	4323

In order to put our results in a wider perspective, we also provide running times on an Intel XEON (Prescott Architecture, sSpec number SL7Z5) processor. The finite

field implementation is the same and has been documented in [5]. Timings are given in Table 5, whereas the graph of EC-relative performance is given in Figure 2.

Table 5. *Scalar multiplication timings in μsec (3.6 GHz Intel XEON)*

Sec. lvl.	Curve	Using sequential mults			Sec. lvl.	Curve	Using sequential mults		
		DBL	ADD	NAF _w			DBL	ADD	NAF _w
160	ec	2.21	3.92	500	220	ec	–	–	–
	g2	2.19	4.02	477		g2	3.27	6.01	981
	g3	1.72	4.20	410		g3	2.94	7.52	975
	g4	2.57	7.02	636		g4	4.35	12.66	1509
180	ec	3.18	5.65	808	240	ec	5.19	8.97	1714
	g2	2.41	4.40	594		g2	–	–	–
	g3	2.46	6.21	669		g3	2.93	7.49	1052
	g4	2.77	7.58	775		g4	4.29	12.38	1605
200	ec	3.26	5.85	933	250	ec	6.03	10.33	2073
	g2	3.17	5.81	863		g2	3.88	7.88	1357
	g3	–	–	–		g3	–	–	–
	g4	2.77	7.59	855		g4	4.85	13.89	1895
210	ec	3.90	6.89	1165	270	ec	5.85	10.24	2180
	g2	3.26	5.92	938		g2	4.77	9.34	1768
	g3	2.64	6.75	843		g3	3.93	10.33	1593
	g4	4.10	11.90	1366		g4	4.79	13.77	2001

As explained in [5], the Intel architecture has a smaller number of registers than the Powerpc architecture, making relative inversion performance significantly worse, especially for smaller fields. For instance, the inversion-to-multiplication ratio goes from 5.224 to 13.546 in the 53 bits field.

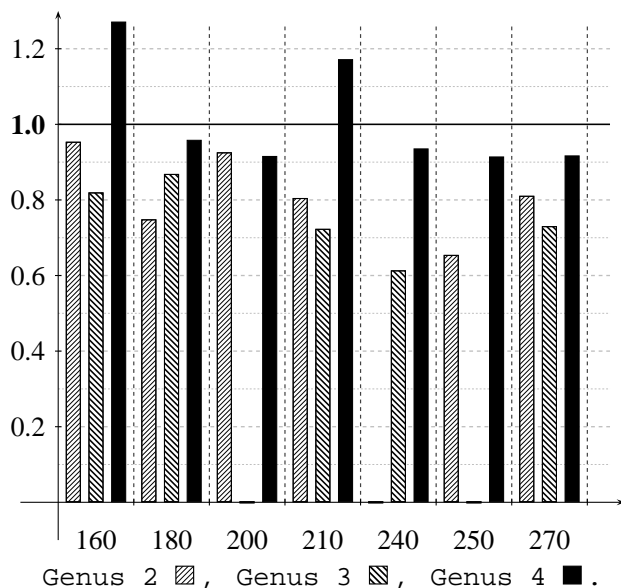
Therefore, the various types of curves show performance much closer to each other on the XEON than on a Powerpc, with genus two and three curves being comparable to each other in performance, and still faster than elliptic curves, but by a smaller factor. Genus four performs quite well, in many cases it is faster than elliptic curves, and in one case (at the 200 bit security level) it even offers the fastest choice. One of the reasons for this efficiency (compared to genus two and three) is due to the sequential multiplications: the average costs for blocks of 2, 3, 4, and 5 multiplications are lower on the Xeon than on other processors, and this gives a bigger improvement to genus four curves than to lower genus ones.

As a matter of fact, the large relative cost of inversion on the Intel CPU could make projective coordinates interesting in this situation. For example, the single inversion in the genus three doubling becomes once again the dominant cost of that group operation. However, developing these new sets of formulæ is beyond the scope of this work and is left open for future research.

We also note that we have used a 32-bit code base for all architectures, including the Core2 and the specific Prescott CPU type we use in our test, where 64-bit extensions

are available. Among other things, code explicitly written and compiled for the 64-bit ISA has access to more registers (16 instead of 8). Rewriting the code for 64-bit machines and using more registers would not only increase performance (by roughly a factor of two), but most likely also make the Intel performance behave in a more similar way to the PowerPC, i.e. give a stronger advantage to genus 2 and 3 curves. Granularity problems would still be partially solved as in the 32-bit case, as the finite field performance tables in [5] show. Only curves on fields smaller than 64 bits may behave in a noticeably different way because no multi-precision operands are used. In particular, genus 3 curves over $\mathbb{F}_{2^{59}}$ and genus 4 curves over $\mathbb{F}_{2^{53}}$ and $\mathbb{F}_{2^{59}}$ would be much faster (and making curves of genera 3 and 4 even more interesting for moderate security applications).

Figure 2: *Scalar multiplication performance for curves of genera two, three and four at various security levels, relative to EC performance (normalized to 1.0) on a 3.6 GHz Intel XEON*



6.2 Conclusions

In this paper we reconsidered the issue of implementing low genus hyperelliptic Jacobian arithmetic over fields of even characteristic. Instead of independently addressing field arithmetic, the derivation of explicit formulæ, and the impact of recent security research, we studied the interplay of these issues.

When explicit formulæ are designed, it is customary to speed-up polynomial operations by means of Karatsuba-like tricks. Combining these with the use of sequential multiplication routines [5] brings substantial performance gains, especially as the genus increases. This is reflected in the real world results reported in Section 6.1. The

gains due to sequential multiplications alone are often close to 15% and 20% for genus three and four respectively. To a much smaller extent, gains are obtained even for López-Dahab coordinates for elliptic curves as well as for affine genus two formulæ. Our genus four formulæ are significantly better than the best ones previously published (for example, addition costs decrease from $2I+148M+6S$ in [42] to $1I+119M+10S$ in this paper).

Moreover, we restricted the computations to specific subgroups or subsets of the considered algebraic groups to speed up scalar multiplication (Section 5). This can be done without losing security since the index calculus methods attack the DLP in the whole algebraic group and do not consider subgroups or key sizes - whereas square root methods (such as Pollard's methods) can be restricted to search in subgroups or among keys of a certain size.

Some interesting results and observations stemming from the benchmarks are:

- Curves of genus three provide similar performance to curves of genus two, and perform even better in some circumstances.
- Cryptographically secure curves of genus three and four may be easier to find since we can allow larger cofactors than in the genus one and two cases.
- Genus four, used wisely, is still interesting, as its performance compares to that of elliptic curves.

Acknowledgments. Research for this paper was partially supported by FONDECYT (Chile) grants 1070242 (regular) and 7070216 (international cooperation) and by the Programa Reticulados y Ecuaciones of the Universidad de Talca.

References

- [1] R. Avanzi, *Aspects of hyperelliptic curves over large prime fields in software implementations*. Cryptographic Hardware and Embedded Systems – CHES 2004, Lecture Notes in Computer Science 3156, pp. 148–162. Springer-Verlag, 2004.
- [2] ———, *A Note on the Signed Sliding Window Integer Recoding and its Left-to-Right Analogue*. Selected Areas in Cryptography – SAC 2004, Lecture Notes in Computer Science 3357, pp. 130–143. Springer-Verlag, 2005.
- [3] ———, *Another Look at Square Roots (and Other Less Common Operations) in Fields of Even Characteristic*. Selected Areas in Cryptography – SAC 2007, Lecture Notes in Computer Science 4876, pp. 138–154. Springer-Verlag, 2007.
- [4] R. Avanzi, H. Cohen, C. Doche, G. Frey, T. Lange, K. Nguyen, and F. Vercauteren, *The Handbook of Elliptic and Hyperelliptic Curve Cryptography*. CRC Press, 2006.
- [5] R. Avanzi and N. Thériault, *Effects of Optimizations for Software Implementations of Small Binary Field Arithmetic*. WAIFI 2007, Lecture Notes in Computer Science 4547, pp. 69–84. Springer-Verlag, 2007.
- [6] I. F. Blake, G. Seroussi, and N. P. Smart, *Elliptic curves in cryptography*, London Mathematical Society Lecture Note Series 265. Cambridge University Press, 1999.
- [7] ———, *Advances in Elliptic Curve Cryptography*, London Mathematical Society Lecture Note Series 317. Cambridge University Press, 2005.

-
- [8] D. G. Cantor, *Computing in the Jacobian of hyperelliptic curves*, Mathematics of Computation 48 (1987), pp. 95–101.
- [9] ECRYPT, *Encrypt yearly report on algorithms and key sizes*, ECRYPT, Report, 2005.
- [10] A. Enge, *Computing discrete logarithms in high-genus hyperelliptic Jacobians in provably subexponential time*, Mathematics of Computation 71 (2002), pp. 729–742.
- [11] X. Fan, T. Wollinger, and Y. Wang, *Efficient Doubling on Genus 3 Curves over Binary Fields*. Topics in Cryptology – CT-RSA 2006, Lecture Notes in Computer Science 3860, pp. 64–81. Springer-Verlag, 2006.
- [12] FIPS 186–2, Digital Signature Standard (DSS), *Federal Information Processing Standards Publication 186–2*, 2000.
- [13] K. Fong, D. Hankerson, J. López, and A. Menezes, *Field Inversion and Point Halving Revisited*, IEEE Transactions on Computers 53 (2004), pp. 1047–1059.
- [14] G. Frey and H. Rück, *A remark concerning m -divisibility and the discrete logarithm problem in the divisor class group of curves*, Mathematics of Computation 62 (1994), pp. 865–874.
- [15] S. D. Galbraith, *Supersingular Curves in Cryptography*. Advances in Cryptology – Asiacrypt 2001, Lecture Notes in Computer Science 2248, pp. 49–513. Springer-Verlag, 2001.
- [16] P. Gaudry, *Index calculus for abelian varieties and the elliptic curve discrete logarithm problem*, to appear, Journal of Symbolic Computation, 2007.
- [17] P. Gaudry, F. Hess, and N. P. Smart, *Constructive and destructive facets of Weil descent on elliptic curves*, Journal of Cryptology 15 (2002), pp. 19–46.
- [18] P. Gaudry, E. Thomé, N. Thériault, and C. Diem, *A double large prime variation for small genus hyperelliptic index calculus*, Mathematics of Computation 76 (2007), pp. 475–492.
- [19] C. Guyot, K. Kaveh, and V. M. Patankar, *Explicit algorithm for the arithmetic on the hyperelliptic Jacobians of genus 3*, Journal of the Ramanujan Mathematical Society 19 (2004), pp. 75–115.
- [20] D. Hankerson, A. J. Menezes, and S. A. Vanstone, *Guide to elliptic curve cryptography*. Springer-Verlag, 2003.
- [21] R. Harley, *Fast arithmetic on genus two curves*, preprint, 2000.
- [22] F. Hess, *The GHS Attack Revisited*. Advances in Cryptology – Eurocrypt 2003, Lecture Notes in Computer Science 2656, pp. 374–387. Springer-Verlag, 2003.
- [23] N. Koblitz, *Elliptic curve cryptosystems*, Mathematics of Computation 48 (1987), pp. 203–209.
- [24] ———, *Hyperelliptic cryptosystems*, Journal of Cryptology 1 (1989), pp. 139–150.
- [25] J. Kuroki, M. Gonda, K. Matsuo, J. Chao, and S. Tsujii, *Fast genus three hyperelliptic curve cryptosystems*. The 2002 Symposium on Cryptography and Information Security, Japan – SCIS 2002, 29 Jan. – 1 Feb., 2002.
- [26] T. Lange, *Formulae for arithmetic on genus 2 hyperelliptic curves*, Appl. Algebra Engrg. Comm. Comput. 15 (2005), pp. 295–328.
- [27] T. Lange and M. Stevens, *Efficient doubling for genus two curves over binary fields*. Selected Areas in Cryptography – SAC 2004, Lecture Notes in Computer Science 3357, pp. 170–181. Springer-Verlag, 2005.
- [28] A. K. Lenstra and E. R. Verheul, *Selecting Cryptographic Key Sizes*, Journal of Cryptology 14 (2001), pp. 255–293.
- [29] J. López and R. Dahab, *High-speed software multiplication in \mathbb{F}_{2^m}* . Progress in Cryptology – INDOCRYPT 2000, Lecture Notes in Computer Science 1977, pp. 203–212. Springer-Verlag, 2000.

-
- [30] K. Matsuo, J. Chao, and S. Tsujii, *Fast genus two hyperelliptic curve cryptosystems*. ISEC2001-31. IEICE, 2001.
- [31] A. Menezes, T. Okamoto, and S. Vanstone, *Reducing elliptic curve logarithms in a finite field*, IEEE Transactions on Information Theory IT-39 (1993), pp. 1639–1646.
- [32] A. Menezes and E. Teske, *Cryptographic Implications of Hess' Generalized GHS Attack*, Appl. Algebra Engrg. Comm. Comput. 16 (2006), pp. 439–460.
- [33] A. Menezes, Y.-H. Wu, and R. Zuccherato, *An Elementary Introduction to Hyperelliptic Curves*, Algebraic aspects of cryptography (N. Koblitz, ed.), Springer-Verlag, 1998, pp. 155–178.
- [34] A. Miyajiri, H. Cohen, and T. Ono, *Efficient elliptic curve exponentiation*. Information and Communication Security – ICICS 1997, Lecture Notes in Computer Science 1334, pp. 282–290. Springer-Verlag, 1997.
- [35] V. S. Miller, *Use of elliptic curves in cryptography*. Advances in Cryptology – Crypto 1985, Lecture Notes in Computer Science 218, pp. 417–426. Springer-Verlag, 1986.
- [36] D. Mumford, *Tata Lectures on Theta II*. Birkhäuser, 1984.
- [37] K. Nagao, *Improving group law algorithms for Jacobians of hyperelliptic curves*. Algorithmic Number Theory – ANTS-IV, Lecture Notes in Computer Science 1838, pp. 439–448. Springer-Verlag, 2000.
- [38] ———, *Improvement of Thériault Algorithm of Index Calculus for Jacobian of Hyperelliptic Curves of Small Genus*, preprint, 2004.
- [39] NIST, *Key Management Guideline - Workshop Document*, Draft, October 2001.
- [40] J. Pelzl, T. Wollinger, J. Guajardo, and C. Paar, *Hyperelliptic curve cryptosystems: closing the performance gap to elliptic curves*. Cryptography Hardware and Embedded Systems – CHES 2003, Lecture Notes in Computer Science 2779, pp. 351–365. Springer-Verlag, 2003.
- [41] J. Pelzl, T. Wollinger, and C. Paar, *High Performance Arithmetic for Hyperelliptic Curve Cryptosystems of Genus Two*. Information Technology: Coding and Computing – ITCC 2004, 2, pp. 513–517, 2004.
- [42] ———, *Low cost Security: Explicit Formulae for Genus 4 Hyperelliptic Curves*. Selected Areas in Cryptography – SAC 2003, Lecture Notes in Computer Science 3006, pp. 1–16. Springer-Verlag, 2004.
- [43] J. Scholten and H. J. Zhu, *Hyperelliptic Curves in Characteristic 2*, Inter. Math. Research Notices 17 (2002), pp. 905–917.
- [44] V. Shoup, *NTL: A Library for doing number theory*.
- [45] B. Smith, *Isogenies and the Discrete Logarithm Problem on Jacobians of Genus 3 Hyperelliptic Curves*. Advances in Cryptology – EUROCRYPT 2008, Lecture Notes in Computer Science 4965, pp. 163–180. Springer-Verlag, 2008.
- [46] J.A. Solinas, *An improved algorithm for arithmetic on a family of elliptic curves*. Advances in Cryptology – CRYPTO '97, Lecture Notes in Computer Science 1294, pp. 357–371. Springer-Verlag, 1997.
- [47] ———, *Improved Algorithms for Arithmetic on Anomalous Binary Curves*, University of Waterloo, Combinatorics and Optimization Research Report no. CORR 99-46, 1999. updated and corrected version of [46].
- [48] N. Thériault, *Weil Descent for Artin-Schreier Curves*, preprint, 2003.
- [49] N. Thériault (with R. Avanzi), *Towards an Exact Cost Analysis of Index Calculus Algorithms*, Presented at ECC 2006.
- [50] T. Wollinger, J. Pelzl, and C. Paar, *Cantor versus Harley: Optimization and Analysis of Explicit Formulae for Hyperelliptic Curve Cryptosystems*, IEEE Transactions on Computers 54 (2005), pp. 861–872.

A Genus three formulæ

Doubling formula $D_3 = [2]D_1$

Inputs: $D_1 = [u_1(x), v_1(x)]$, $u_1(x) = a_0 + a_1x + a_2x^2 + x^3$ and $v_1(x) = c_0 + c_1x + c_2x^2$ $C : y^2 + h(x)y = f(x)$ with $h(x) = 1$ and $f(x) = f_0 + f_1x + f_3x^3 + x^7$		
Outputs: $D_3 = [u_3(x), v_3(x)]$, $u_3(x) = e_0 + e_1x + e_2x^2 + x^3$ and $v_3(x) = \epsilon_0 + \epsilon_1x + \epsilon_2x^2$		
Step	Operations	Cost (Effective)
1	$u_C(x) = u_1(x)^2$, $u_C(x) = u_{C,0} + u_{C,2}x^2 + u_{C,4}x^4 + x^6$ $u_{C,0} = (a_0)^2$; $u_{C,2} = (a_1)^2$; $u_{C,4} = (a_2)^2$; If $u_{C,4} (= v_{C,5})$ is 0 use Cantor's algorithm	3 S (3 S)
2	$v_C(x) = v_1(x)^2 + f(x) \bmod u_C(x)$, and $v_C(x) = v_{C,0} + v_{C,1}x + v_{C,2}x^2 + v_{C,3}x^3 + v_{C,4}x^4 + v_{C,5}x^5$ $t_0 = (c_0)^2$; $v_{C,2} = (c_1)^2$; $v_{C,4} = (c_2)^2$; $v_{C,0} = t_0 + f_0$; $v_{C,1} = u_{C,0} + f_1$; $v_{C,3} = f_3 + u_{C,2}$;	3 S (3 S)
3	computation of inverse $T_1 = 1/u_{C,4}$;	1 I (1 I)
4	$u_T(x) = \text{Monic}\left(\frac{f(x)+v_C(x)+v_C(x)^2}{u_C(x)}\right)$, $u_T(x) = u_{T,0} + u_{T,1}x + u_{T,2}x^2 + x^4$ $u_{T,1} = (T_1)^2$; $t_2 = (v_{C,4})^2$; $t_3 = (v_{C,3})^2$; $(t_4, t_5) = u_{T,1} \cdot (t_2, t_3)$; $e_1 = t_4 + u_{C,4}$; $t_6 = u_{C,2} + t_5$; $(T_7, T_8) = e_1 \cdot (u_{C,4}, v_{C,4})$; $u_{T,0} = t_6 + T_7$;	4 M + 3 S (3.3 M + 3 S)
5	$v_T(x) = v_C(x) + 1 \bmod u_T(x)$, $v_T(x) = v_{T,0} + v_{T,1}x + v_{T,2}x^2 + v_{T,3}x^3$ $t_9 = v_{C,4} \cdot u_{T,0}$; $t_{10} = v_{C,4} + u_{C,4}$; $t_{11} = u_{T,0} + u_{T,1}$; $t_{12} = t_{10} \cdot t_{11}$; $T_{13} = v_{C,0} + t_9$; $v_{T,1} = v_{C,1} + t_{12} + t_9 + T_1$; $v_{T,2} = v_{C,2} + T_8 + T_1$; $v_{T,3} = v_{C,3} + T_7$;	2 M (2 M)
6	$u_3(x) = \text{Monic}\left(\frac{f(x)+v_T(x)+v_T(x)^2}{u_T(x)}\right)$, $u_3(x) = e_0 + e_1x + e_2x^2 + x^3$ $e_2 = (v_{T,3})^2$; $t_{14} = (v_{T,2})^2$; $t_{15} = e_2 \cdot e_1$; $e_0 = u_{T,1} + t_{14} + t_{15}$;	1 M + 2 S (1 M + 2 S)
7	$v_3(x) = v_T(x) + 1 \bmod u_3(x)$, $v_3(x) = \epsilon_0 + \epsilon_1x + \epsilon_2x^2$ $(t_{16}, t_{17}, t_{18}) = v_{T,3} \cdot (e_0, e_1, e_2)$; $\epsilon_0 = t_{16} + T_{13}$; $\epsilon_1 = t_{17} + v_{T,1}$; $\epsilon_2 = t_{18} + v_{T,2}$;	3 M (2.25 M)
Total: 1 I + 10 M + 11 S (1 I + 8.55 M + 11 S)		

Variable schedule for the genus 3 doubling formula

0	1	2	3	4	5	6	7	8	9	10	11	12
$u_{C,4}$	$v_{C,4}$	$v_{C,3}$	$v_{C,2}$	$u_{C,0}$	t_0	$u_{T,1}$	t_4	$u_{C,2}$	t_2	t_5	T_1	t_3
t_{12}	t_{10}	$v_{T,3}$	$v_{T,2}$	$v_{C,1}$	$v_{C,0}$	e_0	e_1	t_6	T_7	T_8	t_{17}	t_9
t_{14}	t_{15}			$v_{T,1}$	T_{13}			$u_{T,0}$		t_{16}	ϵ_1	t_{18}
								t_{11}		ϵ_0		ϵ_2
								e_2				

Addition formula $D_3 = D_1 \oplus D_2$

Step	Operations	Cost (Effective)
Inputs: $D_1 = [u_1(x), v_1(x)]$, $u_1(x) = a_0 + a_1x + a_2x^2 + x^3$ and $v_1(x) = c_0 + c_1x + c_2x^2$ $D_2 = [u_2(x), v_2(x)]$, $u_2(x) = b_0 + b_1x + b_2x^2 + x^3$ and $v_2(x) = d_0 + d_1x + d_2x^2$ $C : y^2 + h(x)y = f(x)$ with $h(x) = 1$ and $f(x) = f_0 + f_1x + f_3x^3 + x^7$		
Outputs: $D_3 = [u_3(x), v_3(x)]$, $u_3(x) = e_0 + e_1x + e_2x^2 + x^3$ and $v_3(x) = \epsilon_0 + \epsilon_1x + \epsilon_2x^2$		
1	$inv(x) = r \cdot u_1(x)^{-1} \bmod u_2(x)$, via Cramer's rule, $inv(x) = inv_0 + inv_1x + inv_2x^2$ $M_{0,0} = b_0 + a_0$; $M_{1,0} = b_1 + a_1$; $M_{2,0} = b_2 + a_2$; $(M_{0,1}, T_0, T_1) = M_{2,0} \cdot (b_0, b_1, b_2)$; $M_{1,1} = T_0 + M_{0,0}$; $M_{2,1} = T_1 + M_{1,0}$; $(M_{0,2}, t_2, t_3) = M_{2,1} \cdot (b_0, b_1, b_2)$; $M_{1,2} = t_2 + M_{0,1}$; $M_{2,2} = t_3 + M_{1,1}$; $(t_4, t_5) = M_{1,0} \cdot (M_{2,2}, M_{2,1})$; $(t_6, t_7) = M_{1,1} \cdot (M_{2,2}, M_{2,0})$; $(t_8, t_9) = M_{1,2} \cdot (M_{2,0}, M_{2,1})$; $inv_0 = t_6 + t_9$; $inv_1 = t_4 + t_8$; $inv_2 = t_5 + t_7$;	12 M (9.45 M)
2	$r = inv(x) \cdot u_1(x) \bmod u_2(x)$ $q_0 = d_0 + c_0$; $q_1 = d_1 + c_1$; $q_2 = d_2 + c_2$; $(t_{10}, T_{11}) = inv_0 \cdot (M_{0,0}, q_0)$; $(t_{12}, \lambda_1) = inv_2 \cdot (M_{0,2}, q_2)$; $t_{13} = t_{12} + t_{10}$; $(t_{14}, T_{15}) = inv_1 \cdot (M_{0,1}, q_1)$; $r = t_{13} + t_{14}$; If r is 0 use Cantor's algorithm	6 M (4.95 M)
3	$s'(x) = r \cdot s(x)$, $s(x) = u_1(x)^{-1} \cdot (v_2(x) + v_1(x)) \bmod u_2(x)$, $s'(x) = s'_0 + s'_1x + s'_2x^2$ $t_{16} = inv_0 + inv_1$; $t_{17} = inv_0 + inv_2$; $t_{18} = inv_1 + inv_2$; $t_{19} = q_1 + q_2$; $t_{20} = q_1 + q_0$; $t_{21} = q_0 + q_2$; $t_{22} = t_{17} \cdot t_{21}$; $t_{23} = t_{18} \cdot t_{19}$; $(t_{24}, t_{25}) = \lambda_1 \cdot (b_1, b_2)$; $\lambda_0 = t_{25} + T_{15} + \lambda_1 + t_{23}$; $(t_{26}, t_{27}) = \lambda_0 \cdot (b_0, b_2)$; $s'_2 = t_{22} + T_{11} + \lambda_1 + T_{15} + t_{24} + t_{27}$; $s'_0 = t_{26} + T_{11}$; $t_{28} = t_{20} \cdot t_{16}$; $t_{29} = \lambda_0 + \lambda_1$; $t_{30} = b_0 + b_1$; $t_{31} = t_{29} \cdot t_{30}$; $s'_1 = t_{31} + s'_0 + T_{15} + t_{24} + t_{28}$;	8 M (7.3 M)
4	computation of inverses and $\tilde{s}(x)$ ($s'(x)$ made monic), $\tilde{s}(x) = \tilde{s}_0 + \tilde{s}_1x + x^2$ $t_{32} = r \cdot s'_2$; If t_{32} is 0 use Cantor's algorithm $t_{33} = 1/t_{32}$; $t_{34} = (s'_2)^2$; $(t_{35}, s_2) = t_{33} \cdot (r, t_{34})$; $(T_{36}, \tilde{s}_0, \tilde{s}_1) = t_{35} \cdot (r, s'_0, s'_1)$;	1 I + 6 M + 1 S (1 I + 4.9 M + 1 S)
5	$u_T(x) = \left\lfloor \frac{\tilde{s}(x)u_1(x)}{u_2(x)} \right\rfloor + s_2^{-2}(x + a_2 + b_2)$, $u_T(x) = u_{T,0} + u_{T,1}x + u_{T,2}x^2 + u_{T,3}x^3 + x^4$ $t_{37} = (\tilde{s}_0)^2$; $t_{38} = (\tilde{s}_1)^2$; $(t_{39}, t_{40}) = t_{38} \cdot (a_1, a_2)$; $u_{T,3} = a_2 + b_2$; $u_{T,2} = t_{38} + a_1 + b_1 + T_1$; $(t_{41}, t_{42}) = u_{T,2} \cdot (b_1, b_2)$; $l_1 = t_{42} + a_0 + b_0 + T_0 + t_{40}$; $t_{43} = l_1 \cdot b_2$; $l_0 = t_{43} + t_{37} + M_{0,1} + t_{39} + t_{41}$; $t_{44} = (T_{36})^2$; $t_{45} = t_{44} \cdot u_{T,3}$; $u_{T,1} = t_{44} + l_1$; $u_{T,0} = t_{45} + l_0$;	6 M + 3 S (5.3 M + 3 S)
6	$z(x) = \tilde{s}(x)u_1(x)$, $z(x) = z_0 + z_1x + z_2x^2 + z_3x^3 + z_4x^4 + x^5$ $(t_{46}, t_{47}) = \tilde{s}_1 \cdot (a_1, a_2)$; $(z_0, t_{48}) = \tilde{s}_0 \cdot (a_0, a_2)$; $t_{49} = \tilde{s}_0 + \tilde{s}_1$; $t_{50} = a_0 + a_1$; $z_3 = t_{47} + a_1 + \tilde{s}_0$; $t_{51} = t_{49} \cdot t_{50}$; $z_2 = a_0 + t_{46} + t_{48}$; $z_1 = t_{51} + z_0 + t_{46}$;	5 M (4.3 M)
7	$v_T(x) = s_2z(x) + v_1(x) + 1 \bmod u_T(x)$, $v_T(x) = v_{T,0} + v_{T,1}x + v_{T,2}x^2 + v_{T,3}x^3$ $t_{52} = \tilde{s}_1 + u_{T,3} + a_2$; $(t_{53}, t_{54}, t_{55}, t_{56}) = t_{52} \cdot (u_{T,0}, u_{T,1}, u_{T,2}, u_{T,3})$; $t_{57} = t_{53} + z_0$; $t_{58} = t_{54} + u_{T,0} + z_1$; $t_{59} = t_{55} + u_{T,1} + z_2$; $t_{60} = t_{56} + u_{T,2} + z_3$; $(t_{61}, t_{62}, t_{63}, v_{T,3}) = s_2 \cdot (t_{57}, t_{58}, t_{59}, t_{60})$; $T_{64} = t_{61} + c_0$; $v_{T,1} = t_{62} + c_1$; $v_{T,2} = t_{63} + c_2$;	8 M (5.6 M)
8	$u_3(x) = \text{Monic} \left(\frac{f(x) + v_T(x) + v_T(x)^2}{u_T(x)} \right)$, $u_3(x) = e_0 + e_1x + e_2x^2 + x^3$ $t_{65} = (v_{T,3})^2$; $t_{66} = (v_{T,2})^2$; $e_2 = t_{65} + u_{T,3}$; $(t_{67}, t_{68}) = e_2 \cdot (u_{T,2}, u_{T,3})$ $e_1 = t_{68} + u_{T,2}$; $t_{69} = u_{T,3} \cdot e_1$; $e_0 = t_{66} + u_{T,1} + t_{67} + t_{69}$;	3 M + 2 S (2.65 M + 2 S)
9	$v_3(x) = v_T(x) + 1 \bmod u_3(x)$, $v_3(x) = \epsilon_0 + \epsilon_1x + \epsilon_2x^2$ $(t_{70}, t_{71}, t_{72}) = v_{T,3} \cdot (e_0, e_1, e_2)$; $\epsilon_2 = v_{T,2} + t_{72}$; $\epsilon_1 = v_{T,1} + t_{71}$; $\epsilon_0 = T_{64} + t_{70}$;	3 M 2.25 M
Total: 11 + 57 M + 6 S (1 I + 46.7 M + 6 S)		

Variable schedule for the genus 3 addition formula

0	1	2	3	4	5	6	7	8	9	10	11	12	13
$M_{0,0}$	$M_{1,0}$	$M_{2,0}$	$M_{1,1}$	$M_{2,1}$	$M_{0,2}$	t_2	t_3	t_6	t_4	t_5	$M_{0,1}$	T_0	T_1
t_{12}	t_7	q_1	t_8	λ_1	t_{14}	$M_{1,2}$	$M_{2,2}$	inv_0	inv_1	inv_2	t_{49}	t_{43}	t_{42}
t_{13}	q_0	t_{20}	q_2	t_{32}	t_{16}	t_{10}	t_9	t_{17}	t_{18}	t_{19}	t_{54}	l_0	l_1
r	t_{21}	t_{31}	t_{22}	t_{34}	t_{29}	T_{15}	T_{11}	t_{27}	t_{25}	t_{24}	t_{58}	t_{50}	t_{48}
t_{39}	t_{23}	s'_1	s'_2	t_{37}	t_{33}	T_{36}	t_{28}	t_{30}	λ_0	t_{41}	t_{66}	t_{55}	t_{56}
t_{45}	t_{26}	t_{38}	t_{35}	t_{47}	z_0	t_{51}	\tilde{s}_0	\tilde{s}_1	s_2	t_{46}	e_0	t_{59}	t_{60}
$u_{T,0}$	s'_0	$u_{T,2}$	$u_{T,3}$	z_3	t_{61}	z_1	z_2	t_{52}	t_{69}	t_{53}		t_{68}	t_{65}
	t_{40}	t_{71}	t_{72}		T_{64}	t_{62}	t_{63}	$v_{T,3}$		t_{57}		e_1	e_2
	t_{44}				ϵ_0	$v_{T,1}$	$v_{T,2}$			t_{67}			
	$u_{T,1}$					ϵ_1	ϵ_2						
	t_{70}												

B Genus four formulæ

Doubling formula $D_3 = [2]D_1$

Inputs: $D_1 = [u_1(x), v_1(x)]$, $u_1(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + x^4$ and $v_1(x) = c_0 + c_1x + c_2x^2 + c_3x^3$ $C : y^2 + h(x)y = f(x)$ with $h(x) = 1$ and $f(x) = f_0 + f_1x + f_3x^3 + f_7x^7 + x^9$		
Outputs: $D_3 = [u_3(x), v_3(x)]$, $u_3(x) = e_0 + e_1x + e_2x^2 + e_3x^3 + x^4$ and $v_3(x) = \epsilon_0 + \epsilon_1x + \epsilon_2x^2 + \epsilon_3x^3$		
Step	Operations	Cost (Effective)
1	$u_C(x) = u_1(x)^2$, $u_C(x) = u_{C,0} + u_{C,2}x^2 + u_{C,4}x^4 + u_{C,6}x^6 + x^8$ $u_{C,0} = (a_0)^2$; $u_{C,2} = (a_1)^2$; $u_{C,4} = (a_2)^2$; $u_{C,6} = (a_3)^2$;	4 S (4 S)
2	$v_C(x) = v_1(x)^2 + f(x) \bmod u_C(x)$, $v_C(x) = v_{C,0} + v_{C,1}x + v_{C,2}x^2 + v_{C,3}x^3 + v_{C,4}x^4 + v_{C,5}x^5 + v_{C,6}x^6 + v_{C,7}x^7$ $t_0 = (c_0)^2$; $v_{C,2} = (c_1)^2$; $v_{C,4} = (c_2)^2$; $v_{C,6} = (c_3)^2$; $v_{C,0} = f_0 + t_0$; $v_{C,1} = f_1 + u_{C,0}$; $v_{C,3} = f_3 + u_{C,2}$; $v_{C,7} = f_7 + u_{C,6}$; (Note: $v_{C,5} = u_{C,4}$)	4 S (4 S)
3	computation of inverses $t_1 = v_{C,7} \cdot u_{C,4}$; $T_2 = (v_{C,6})^2$; $t_3 = (v_{C,7})^2$; $t_4 = t_3 \cdot u_{C,6}$; $t_5 = T_2 + t_1 + t_4$; $t_6 = v_{C,7} \cdot t_5$; If t_6 is 0 use Cantor's algorithm; $t_7 = 1/t_6$; $(t_8, t_9) = t_7 \cdot (t_3, t_5)$; $e_3 = (t_8)^2$; $u_{T,1} = (t_9)^2$;	1 I + 5 M + 4 S (1 I + 4.65 M + 4 S)
4	$u_T(x) = \text{Monic} \left(\frac{f(x) + v_C(x) + v_C(x)^2}{u_C(x)} \right)$, $u_T(x) = u_{T,0} + u_{T,1}x + u_{T,2}x^2 + u_{T,4}x^4 + x^6$ $t_{10} = (u_{C,4})^2$; $t_{11} = (v_{C,4})^2$; $(t_{12}, t_{13}, t_{14}) = u_{T,1} \cdot (T_2, t_{10}, t_{11})$; $u_{T,4} = t_{12} + u_{C,6}$; $(t_{15}, t_{16}) = u_{T,4} \cdot (u_{C,6}, u_{C,4})$; $u_{T,2} = t_{15} + t_{13} + u_{C,4}$; $t_{17} = u_{C,6} \cdot u_{T,2}$; $u_{T,0} = t_{16} + t_{14} + u_{C,2} + t_{17}$;	6 M + 2 S (4.9 M + 2 S)
5	$v_T(x) = v_C(x) + 1 \bmod u_T(x)$, $v_T(x) = v_{T,0} + v_{T,1}x + v_{T,2}x^2 + v_{T,3}x^3 + v_{T,4}x^4 + v_{T,5}x^5$ $(t_{18}, t_{19}, t_{20}) = v_{C,6} \cdot (u_{T,0}, u_{T,4}, u_{T,1})$; $(t_{21}, t_{22}, t_{23}) = v_{C,7} \cdot (u_{T,2}, u_{T,0}, u_{T,4})$; $t_{24} = v_{C,6} + v_{C,7}$; $t_{25} = u_{T,1} + u_{T,2}$; $t_{26} = t_{24} \cdot t_{25}$; $T_{27} = v_{C,0} + t_{18}$; $v_{T,1} = v_{C,1} + t_{20} + t_{22}$; $v_{T,2} = v_{C,2} + t_{26} + t_{20} + t_{21}$; $v_{T,3} = v_{C,3} + t_{21}$; $v_{T,4} = v_{C,4} + t_{19}$; $v_{T,5} = u_{C,4} + t_{23}$;	7 M (5.5 M)
6	$u_3(x) = \text{Monic} \left(\frac{f(x) + v_T(x) + v_T(x)^2}{u_T(x)} \right)$, $u_3(x) = e_0 + e_1x + e_2x^2 + e_3x^3 + x^4$ $t_{28} = (v_{T,3})^2$; $t_{29} = (v_{T,4})^2$; $t_{30} = f_7 + u_{T,4}$; $(t_{31}, e_1, t_{32}) = e_3 \cdot (t_{28}, t_{30}, t_{29})$; $e_2 = t_{32} + u_{T,4}$; $t_{33} = e_2 \cdot u_{T,4}$; $e_0 = t_{31} + u_{T,2} + t_{33}$;	4 M + 2 S (3.25 M + 2 S)
7	$v_3(x) = v_T(x) + 1 \bmod u_3(x)$, $v_3(x) = \epsilon_0 + \epsilon_1x + \epsilon_2x^2 + \epsilon_3x^3$ $(t_{34}, t_{35}) = v_{T,5} \cdot (e_1, e_3)$; $t_{36} = v_{T,4} + t_{35}$; $t_{37} = t_{36} + v_{T,5}$; $t_{38} = e_0 + e_1$; $t_{39} = e_2 + e_3$; $(t_{40}, t_{41}) = t_{36} \cdot (e_0, e_2)$; $(t_{42}, t_{43}) = t_{37} \cdot (t_{38}, t_{39})$; $\epsilon_0 = T_{27} + t_{40}$; $\epsilon_1 = v_{T,1} + t_{42} + t_{34} + t_{40}$; $\epsilon_2 = v_{T,2} + t_{34} + t_{41}$; $\epsilon_3 = v_{T,3} + t_{43} + t_{35} + t_{41}$;	6 M (4.95 M)
Total: 1 I + 28 M + 16 S (1 I + 23.25 M + 16 S)		

Addition formula $D_3 = D_1 \oplus D_2$

Step	Operations	Cost (Effective)
Inputs: $D_1 = [u_1(x), v_1(x)]$, $u_1(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + x^4$ and $v_1(x) = c_0 + c_1x + c_2x^2 + c_3x^3$; $D_2 = [u_2(x), v_2(x)]$, $u_2(x) = b_0 + b_1x + b_2x^2 + b_3x^3 + x^4$ and $v_2(x) = d_0 + d_1x + d_2x^2 + d_3x^3$; $C : y^2 + h(x)y = f(x)$ with $h(x) = 1$ and $f(x) = f_0 + f_1x + f_2x^2 + f_3x^3 + f_4x^4 + f_5x^5$		
Outputs: $D_3 = [u_3(x), v_3(x)]$, $u_3(x) = e_0 + e_1x + e_2x^2 + e_3x^3 + x^4$ and $v_3(x) = e_0 + e_1x + e_2x^2 + e_3x^3$		
1	Almost inverse, $inv(x) = r \cdot u_1(x)^{-1} \bmod u_2(x)$, via Cramer's rule, $inv(x) = inv_0 + inv_1x + inv_2x^2 + inv_3x^3$ $M_{0,0} = a_0 + b_0$; $M_{1,0} = a_1 + b_1$; $M_{2,0} = a_2 + b_2$; $M_{3,0} = a_3 + b_3$; $(M_{0,1}, t_0, t_1, t_2) = M_{3,0} \cdot (b_0, b_1, b_2, b_3)$; $M_{1,1} = t_0 + M_{0,0}$; $M_{2,1} = t_1 + M_{1,0}$; $M_{3,1} = M_{2,0} + t_2$; $(M_{0,2}, t_3, t_4, t_5) = M_{3,1} \cdot (b_0, b_1, b_2, b_3)$; $M_{1,2} = M_{0,1} + t_3$; $M_{2,2} = M_{1,1} + t_4$; $M_{3,2} = M_{2,1} + t_5$; $(M_{0,3}, t_6, t_7, t_8) = M_{3,2} \cdot (b_0, b_1, b_2, b_3)$; $M_{1,3} = M_{0,2} + t_6$; $M_{2,3} = M_{1,2} + t_7$; $M_{3,3} = t_8 + M_{2,2}$; $(t_9, t_{10}, t_{11}) = M_{2,2} \cdot (M_{3,3}, M_{3,0}, M_{3,1})$; copy ($M_{3,3}$); copy ($M_{3,2}$); copy ($M_{3,1}$); $(t_{12}, t_{13}, t_{14}) = M_{2,3} \cdot (M_{3,0}, M_{3,1}, M_{3,2})$; copy ($M_{3,0}$); $(t_{15}, t_{16}, t_{17}) = M_{2,0} \cdot (M_{3,1}, M_{3,2}, M_{3,3})$; $\alpha_{0,2} = t_{16} + t_{10}$; $\alpha_{0,3} = t_{12} + t_{17}$; $\alpha_{2,3} = t_{14} + t_9$; $(t_{18}, t_{19}, t_{20}) = M_{2,1} \cdot (M_{3,2}, M_{3,3}, M_{3,0})$; $\alpha_{0,1} = t_{20} + t_{15}$; $\alpha_{1,2} = t_{18} + t_{11}$; $\alpha_{1,3} = t_{13} + t_{19}$; $T_{21} = M_{2,1} + M_{1,0}$; $T_{22} = M_{3,1} + M_{2,0}$; copy ($\alpha_{0,3}$); copy ($\alpha_{1,3}$); copy ($\alpha_{1,2}$); $(t_{23}, t_{24}, t_{25}) = M_{1,2} \cdot (\alpha_{0,3}, \alpha_{0,1}, \alpha_{1,3})$; copy ($\alpha_{0,1}$); $(t_{26}, t_{27}, t_{28}) = M_{1,0} \cdot (\alpha_{1,3}, \alpha_{2,3}, \alpha_{1,2})$; $(t_{29}, t_{30}, t_{31}) = M_{1,3} \cdot (\alpha_{1,2}, \alpha_{0,2}, \alpha_{0,1})$; copy ($\alpha_{0,2}$); $(t_{32}, t_{33}, t_{34}) = M_{1,1} \cdot (\alpha_{0,3}, \alpha_{0,2}, \alpha_{2,3})$; $inv_0 = t_{29} + t_{25} + t_{34}$; $inv_1 = t_{30} + t_{23} + t_{27}$; $inv_2 = t_{31} + t_{32} + t_{26}$; $inv_3 = t_{28} + t_{33} + t_{24}$; $T_{35} = M_{1,1} + M_{0,0}$;	36 M (26.4 M)
2	$r = inv(x) \cdot u_1(x) \bmod u_2(x)$ $q_0 = c_0 + d_0$; $q_2 = c_2 + d_2$; $(t_{36}, T_{37}) = inv_0 \cdot (M_{0,0}, q_0)$; $(t_{38}, T_{39}) = inv_2 \cdot (M_{0,2}, q_2)$; $q_1 = c_1 + d_1$; $q_3 = c_3 + d_3$; $(t_{40}, T_{41}) = inv_1 \cdot (M_{0,1}, q_1)$; $(t_{42}, \lambda_2) = inv_3 \cdot (M_{0,3}, q_3)$; $r = t_{36} + t_{40} + t_{38} + t_{42}$; If r is 0 use Cantor's algorithm	8 M (6.6 M)
3	$s'(x) = r \cdot s(x)$, $s(x) = u_1(x)^{-1} \cdot (v_2(x) + v_1(x)) \bmod u_2(x)$, $s'(x) = s'_0 + s'_1x + s'_2x^2 + s'_3x^3$ $t_{43} = q_0 + q_1$; $t_{44} = q_2 + q_3$; $t_{45} = q_0 + q_2$; $t_{46} = q_1 + q_3$; $t_{47} = inv_0 + inv_1$; $t_{48} = inv_2 + inv_3$; $t_{49} = inv_2 + inv_0$; $t_{50} = inv_3 + inv_1$; $t_{51} = t_{43} \cdot t_{47}$; $t_{52} = t_{44} \cdot t_{48}$; $t_{53} = t_{45} \cdot t_{49}$; $t_{54} = t_{46} \cdot t_{50}$; $t_{55} = t_{43} + t_{44}$; $t_{56} = t_{48} + t_{47}$; $t_{57} = t_{55} \cdot t_{56}$; $t_{58} = \lambda_2 \cdot b_3$; $\lambda_1 = t_{52} + T_{39} + \lambda_2 + t_{58}$; $t_{59} = \lambda_1 \cdot b_2$; $t_{60} = \lambda_1 + \lambda_2$; $t_{61} = b_2 + b_3$; $t_{62} = t_{60} \cdot t_{61}$; $\lambda_0 = t_{54} + T_{41} + \lambda_2 + T_{39} + t_{62} + t_{58} + t_{59}$; $t_{63} = \lambda_0 \cdot b_1$; $t_{64} = \lambda_0 + \lambda_1$; $t_{65} = \lambda_0 + \lambda_2$; $t_{66} = b_1 + b_2$; $t_{67} = b_1 + b_3$; $t_{68} = t_{58} + t_{54} + t_{57} + t_{52}$; $t_{69} = t_{64} \cdot t_{66}$; $t_{70} = t_{65} \cdot t_{67}$; $s'_2 = t_{68} + t_{51} + t_{53} + T_{37} + T_{41} + T_{39} + \lambda_2 + t_{70} + t_{59} + t_{63}$; $(t_{71}, t_{72}, t_{73}) = b_0 \cdot (\lambda_0, \lambda_1, \lambda_2)$; $s'_2 = t_{59} + t_{53} + T_{37} + T_{39} + T_{41} + t_{69} + t_{63} + t_{73}$; $s'_1 = t_{51} + T_{37} + T_{41} + t_{63} + t_{72}$; $s'_0 = T_{37} + t_{71}$;	14 M (13.25 M)
4	computation of inverses and $\tilde{s}(x)$ ($s'(x)$ made monic), $\tilde{s}(x) = \tilde{s}_0 + \tilde{s}_1x + \tilde{s}_2x^2 + x^3$ $t_{74} = b_3 \cdot s'_1$; $(t_{75}, t_{76}) = s'_2 \cdot (b_2, r)$; $t_{77} = (s'_2)^2$; $t_{78} = t_{74} + t_{75} + s'_1$; $t_{79} = t_{78} \cdot s'_2$; $t_{80} = (s'_2)^2$; $t_{81} = (t_{76})^2$; $t_{82} = t_{79} + t_{80}$; $t_{83} = t_{76} \cdot t_{82}$; If t_{83} is 0 use Cantor's algorithm; $t_{84} = 1/t_{83}$; $(t_{85}, t_{86}) = t_{84} \cdot (t_{82}, t_{81})$; $(s_3, v_{T,5}, t_{87}) = t_{85} \cdot (t_{77}, t_{82}, r)$; $(t_{88}, \tilde{s}_0, \tilde{s}_1, \tilde{s}_2) = t_{87} \cdot (r, s'_0, s'_1, s'_2)$; $T_{89} = (t_{86})^2$; $T_{90} = (t_{88})^2$;	11 + 14 M + 5 S (11 + 11.35 M + 5 S)
5	$u_T(x) = [\tilde{s}(x)^2 u_1(x) / u_2(x)] + s_3^{-2} (x + a_3 + b_3)$, $u_T(x) = u_{T,0} + u_{T,1}x + u_{T,2}x^2 + u_{T,3}x^3 + u_{T,4}x^4 + u_{T,5}x^5 + x^6$ $t_{91} = (\tilde{s}_0)^2$; $t_{92} = (\tilde{s}_1)^2$; $t_{93} = (\tilde{s}_2)^2$; $(t_{94}, t_{95}, t_{96}, t_{97}) = t_{93} \cdot (a_0, a_1, a_2, a_3)$; $(t_{98}, t_{99}) = t_{92} \cdot (a_2, a_3)$; $u_{T,5} = a_3 + b_3$; $u_{T,4} = T_{22} + a_2 + b_2 + t_{93}$; $(t_{100}, t_{101}) = u_{T,4} \cdot (b_3, b_0)$; $u_{T,3} = T_{21} + a_1 + b_1 + t_{97} + t_{100}$; $t_{102} = t_{94} + t_{101}$; $t_{103} = u_{T,3} + u_{T,4}$; $t_{104} = b_2 + b_3$; $t_{105} = t_{103} \cdot t_{104}$; $t_{106} = u_{T,3} \cdot b_2$; $u_{T,2} = T_{35} + a_0 + b_0 + t_{96} + t_{92} + t_{105} + t_{106} + t_{100}$; $t_{107} = u_{T,2} + u_{T,4}$; $t_{108} = b_1 + b_3$; $t_{109} = u_{T,2} \cdot b_1$; $t_{110} = t_{107} \cdot t_{108}$; $l_1 = M_{0,1} + t_{95} + t_{99} + t_{106} + t_{110} + t_{100} + t_{109}$; $t_{111} = u_{T,2} + u_{T,3}$; $t_{112} = b_1 + b_2$; $t_{113} = l_1 \cdot b_3$; $t_{114} = t_{111} \cdot t_{112}$; $l_0 = t_{102} + t_{98} + t_{91} + t_{113} + t_{114} + t_{106} + t_{109}$; $t_{115} = T_{90} \cdot u_{T,5}$; $u_{T,1} = l_1 + T_{90}$; $u_{T,0} = l_0 + t_{115}$;	15 M + 3 S (13.1 M + 3 S)
6	$z(x) = \tilde{s}(x)u_1(x)$, $z(x) = z_0 + z_1x + z_2x^2 + z_3x^3 + z_4x^4 + z_5x^5 + z_6x^6 + x^7$ $(t_{116}, z_0) = \tilde{s}_0 \cdot (a_3, a_0)$; $(t_{117}, t_{118}) = \tilde{s}_1 \cdot (a_3, a_1)$; $t_{119} = a_2 \cdot \tilde{s}_2$; $t_{120} = a_0 + a_1$; $t_{121} = a_0 + a_2$; $t_{122} = a_1 + a_2$; $t_{123} = \tilde{s}_0 + \tilde{s}_1$; $t_{124} = \tilde{s}_0 + \tilde{s}_2$; $t_{125} = \tilde{s}_1 + \tilde{s}_2$; $t_{126} = t_{120} \cdot t_{123}$; $t_{127} = t_{121} \cdot t_{124}$; $t_{128} = t_{122} \cdot t_{125}$; $z_4 = a_1 + \tilde{s}_0 + t_{117} + t_{119}$; $z_3 = a_0 + t_{116} + t_{128} + t_{119} + t_{118}$; $z_2 = t_{118} + t_{127} + t_{119} + z_0$; $z_1 = t_{126} + t_{118} + z_0$; $z_0 = a_3 + \tilde{s}_2$;	8 M (7.3 M)
7	$v_T(x) = s_3 z(x) + v_1(x) + 1 \bmod u_T(x)$, $v_T(x) = v_{T,0} + v_{T,1}x + v_{T,2}x^2 + v_{T,3}x^3 + v_{T,4}x^4 + v_{T,5}x^5$ $t_{129} = u_{T,5} + z_6$; $(t_{130}, t_{131}, t_{132}, t_{133}, t_{134}) = t_{129} \cdot (u_{T,0}, u_{T,1}, u_{T,2}, u_{T,3}, u_{T,4})$; $t_{135} = z_0 + t_{130}$; $t_{136} = z_1 + t_{131} + u_{T,0}$; $t_{137} = z_2 + t_{132} + u_{T,1}$; $t_{138} = z_3 + t_{133} + u_{T,2}$; $t_{139} = z_4 + t_{134} + u_{T,3}$; $(v_{T,4}, t_{140}, t_{141}, t_{142}, t_{143}) = s_3 \cdot (t_{139}, t_{138}, t_{137}, t_{136}, t_{135})$; $T_{144} = t_{143} + c_0$; $v_{T,1} = t_{142} + c_1$; $v_{T,2} = t_{141} + c_2$; $v_{T,3} = t_{140} + c_3$;	10 M (6.8 M)
8	$u_3(x) = \text{Monic}((f(x) + v_T(x) + v_T(x)^2) / u_T(x))$, $u_3(x) = e_0 + e_1x + e_2x^2 + e_3x^3 + x^4$ $t_{145} = (v_{T,3})^2$; $t_{146} = (v_{T,4})^2$; copy (f_7); $e_3 = T_{89} + v_{T,5}$; $(t_{147}, t_{148}, t_{149}) = T_{89} \cdot (t_{146}, t_{145}, f_7)$; $(t_{150}, t_{151}) = e_3 \cdot (u_{T,5}, v_{T,3})$; $e_2 = t_{147} + t_{150} + v_{T,4}$; $t_{152} = e_2 + e_3$; $t_{153} = u_{T,4} + v_{T,5}$; $t_{154} = e_2 \cdot u_{T,4}$; $t_{155} = t_{152} \cdot t_{153}$; $e_1 = t_{149} + t_{155} + t_{154} + t_{150} + v_{T,3}$; $t_{156} = e_1 \cdot v_{T,5}$; $e_0 = t_{151} + t_{148} + t_{154} + t_{156} + v_{T,2}$;	8 M + 2 S (6.9 M + 2 S)
9	$v_3(x) = v_T(x) + 1 \bmod u_3(x)$, $v_3(x) = e_0 + e_1x + e_2x^2 + e_3x^3$ $(t_{157}, t_{158}) = v_{T,5} \cdot (e_1, e_3)$; $t_{159} = v_{T,4} + t_{158}$; $t_{160} = t_{159} + v_{T,5}$; $t_{161} = e_0 + e_1$; $t_{162} = e_2 + e_3$; $(t_{163}, t_{164}) = t_{159} \cdot (e_0, e_2)$; $(t_{165}, t_{166}) = t_{160} \cdot (t_{161}, t_{162})$; $e_0 = t_{163} + T_{144}$; $e_1 = t_{165} + t_{157} + t_{163} + v_{T,1}$; $e_2 = t_{157} + t_{164} + v_{T,2}$; $e_3 = t_{166} + t_{158} + t_{164} + v_{T,3}$;	6 M (4.95 M)
Total: 11 + 119 M + 10 S (11 + 96.65 M + 10 S)		

Variable schedule for the genus 4 doubling formula

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$u_{C,2}$	t_{22}	$u_{C,6}$	t_0	t_3	t_4	t_5	$v_{C,0}$	$v_{C,1}$	$v_{C,2}$	$v_{C,3}$	$v_{C,4}$	$u_{C,4}$	$v_{C,6}$	$v_{C,7}$	$u_{C,0}$	t_9	t_{14}	e_3
t_{21}	t_{30}	t_{23}	T_2	t_{10}	t_7	$u_{T,1}$	T_{27}	$v_{T,1}$	$v_{T,2}$	$v_{T,3}$	$v_{T,4}$	$v_{T,5}$	t_{24}	t_{35}	t_1	t_{13}	t_{20}	
t_{28}	t_{38}	t_{29}	t_{15}	t_{16}	t_{11}	t_{26}	ϵ_0	ϵ_1	ϵ_2	ϵ_3	t_{36}		t_{34}		t_6	t_{19}	t_{32}	
t_{37}		t_{33}	$u_{T,2}$	$u_{T,0}$	$u_{T,4}$	t_{43}									t_8	e_1	e_2	
		t_{39}	t_{40}	t_{25}	t_{42}										t_{12}			
				t_{41}											t_{17}			
															t_{18}			
															t_{31}			
															e_0			

Variable schedule for the genus four addition formula (copies indicated by *)

0	$M_{0,0}$	q_1	t_{47}	t_{57}	t_{69}	t_{74}	t_{79}	s_3	t_{154}	t_{160}								
1	$M_{0,2}$	q_3	t_{48}	t_{56}	t_{63}	t_{75}	t_{80}	t_{84}	$v_{T,5}$									
2	t_3	$M_{0,3}$	t_{52}	t_{70}	t_{76}	t_{87}	T_{89}	t_{155}	t_{162}									
3	t_4	t_6	t_{12}	$\alpha_{0,3}$	t_{36}	t_{64}	t_{71}	t_{88}	t_{91}	t_{119}	t_{130}	$v_{T,4}$	t_{159}					
4	t_5	t_7	t_{13}	$\alpha_{1,3}$	$\alpha_{0,2}^*$	t_{40}	t_{43}	t_{55}	t_{60}	t_{65}	t_{72}	t_{78}	t_{83}	t_{85}	\tilde{s}_0	t_{131}	t_{140}	$v_{T,3}$
5	t_8	$M_{3,3}$	t_{14}	$\alpha_{2,3}$	t_{38}	t_{44}	t_{61}	t_{66}	t_{73}	t_{81}	\tilde{s}_1	t_{126}	t_{132}	t_{141}	$v_{T,2}$			
6	$M_{3,0}$	t_{15}	$\alpha_{1,2}^*$	t_{32}	t_{42}	t_{45}	t_{62}	t_{67}	\tilde{s}_2	t_{133}	t_{142}	$v_{T,1}$						
7	$M_{3,1}$	t_{16}	$\alpha_{0,2}$	t_{33}	t_{46}	λ_0	T_{90}	t_{116}	t_{134}	t_{143}	T_{144}							
8	$M_{3,2}$	t_{17}	t_{18}	$\alpha_{1,2}$	$\alpha_{0,1}^*$	t_{34}	q_0	λ_1	t_{92}	t_{107}	t_{117}	t_{129}	t_{151}	t_{157}				
9	t_9	t_{19}	$\alpha_{0,3}^*$	t_{26}	λ_2	t_{86}	t_{93}	t_{101}	t_{103}	t_{106}	t_{118}	t_{153}	t_{156}	t_{158}				
10	t_{10}	t_{20}	$\alpha_{0,1}$	t_{27}	T_{37}	$u_{T,5}$	t_{163}											
11	t_{11}	$\alpha_{1,3}^*$	t_{28}	T_{41}	t_{104}	t_{108}	t_{111}	l_0	$u_{T,0}$	t_{150}	t_{164}							
12	$M_{0,1}$	l_1	$u_{T,1}$	t_{152}	t_{161}													
13	t_0	$M_{1,1}$	T_{35}	$u_{T,2}$														
14	t_1	$M_{2,1}$	T_{21}	$u_{T,3}$														
15	t_2	$M_{3,1}^*$	T_{22}	$u_{T,4}$	ϵ_0													
16	$M_{3,2}^*$	t_{23}	T_{39}	t_{77}	t_{100}	t_{120}	t_{127}	t_{147}	t_{165}	ϵ_1								
17	$M_{2,2}$	$M_{3,3}^*$	t_{24}	q_2	t_{54}	t_{82}	t_{94}	t_{102}	t_{123}	t_{128}	z_6	t_{148}	ϵ_2					
18	$M_{2,3}$	$M_{3,0}^*$	t_{25}	r	t_{95}	t_{113}	t_{122}	z_4	t_{139}	t_{149}	t_{166}	ϵ_3						
19	$M_{2,0}$	t_{29}	inv_0	t_{53}	s'_0	t_{96}	t_{109}	t_{121}	z_3	t_{138}	t_{146}	ϵ_0						
20	$M_{1,0}$	t_{30}	inv_1	t_{51}	s'_1	t_{97}	t_{105}	t_{110}	t_{114}	t_{124}	z_2	t_{137}	t_{145}	e_1				
21	$M_{1,2}$	t_{31}	inv_2	t_{49}	t_{59}	s'_2	t_{98}	t_{125}	z_1	t_{136}	f_7^*	e_2						
22	$M_{1,3}$	inv_3	t_{50}	t_{58}	t_{68}	s'_3	t_{99}	t_{112}	t_{115}	z_0	t_{135}	ϵ_3						

Received

Author information

R. Avanzi, Faculty of Mathematics and Horst Görtz Institute for IT-Security, Ruhr University Bochum, Germany.

Email: Roberto.Avanzi@ruhr-uni-bochum.de

N. Thériault, Instituto de Matemática y Física, Universidad de Talca, Talca, Chile.

Email: ntheriau@inst-mat.utalca.cl

Z. Wang, University of Waterloo, Department of Combinatorics and Optimization, Canada.

Email: --