Efficient Arithmetic on Genus 2 Hyperelliptic Curves over Finite Fields via Explicit Formulae

Tanja Lange

Information-Security and Cryptography,
Ruhr-University of Bochum,
Universitätsstr. 150,
44780 Bochum, Germany,
lange@itsc.ruhr-uni-bochum.de,
http://www.itsc.ruhr-uni-bochum.de/tanja

December 15, 2003

Abstract

We extend the explicit formulae for arithmetic on genus two curves of [13, 21] to fields of even characteristic and to arbitrary equation of the curve. These formulae can be evaluated faster than the more general Cantor algorithm and allow to obtain faster arithmetic on a hyperelliptic genus 2 curve than on elliptic curves. We give timings for implementations using various libraries for the field arithmetic.

1 Introduction

In this note we first give a short introduction to the mathematical background of hyperelliptic curves and present the standard algorithms to do arithmetic in the ideal class group. This is the group used in the computations to have an efficient way of computing and storing the elements. It is isomorphic to the Jacobian of the curve. Then we present the explicit formulae which are faster than the usual algorithms, along with the analysis of the number of operations and a proof for their correctness.

So far, the fastest explicit formulae are given by Miyamoto, Doi, Matsuo, Chao, and Tsuji [13] and Takahashi [21], building upon the work of Harley [3], but applying to odd characteristic only. Here we give the generalization to even characteristic and to curves stated in the most general form, thus working for any finite field and any hyperelliptic curve given in Weierstraß representation. As to date the cited algorithms are only available in Japanese we provide a full description and explain the steps in detail.

Before giving an outlook, we present some timings obtained using these formula. It turns out that the comparison between elliptic curves and hyperelliptic curves depends heavily on the chosen library – respectively on the relation of inversion and multiplication and the overhead of function calls. We used GMP and NTL for prime fields and a program of Nöcker [15] for binary fields. The curves used for the reference implementations are appropriate for use in cryptography since the field size is of the correct order. As inversions are not too expensive in any arithmetic used, we choose affine representations for elliptic and hyperelliptic curves.

After the publication of the first version of this paper it was brought to the authors attention that there exist a further work on even characteristic by Sugizaki, Matsuo, Chao, and Tsujii [20].

2 Mathematical Background on Elliptic and Hyperelliptic Curves

In this section we briefly sketch what is needed in the remainder of this paper. As elliptic curves are hyperelliptic curves of genus 1 the results stated below apply to this case as well. The interested reader is referred to Menezes, Wu, and Zuccherato [12], Lorenzini [10], and Stichtenoth [19] for more details and proofs.

Let $\mathbb{F}_q$ be a finite field of characteristic $p, q = p^\nu$, and let $\bar{\mathbb{F}}_q$ denote the algebraic closure of $\mathbb{F}_q$.

Definition 2.1 *Let $\mathbb{F}_q(C)/\mathbb{F}_q$ be a quadratic function field defined via an equation*

$$C : y^2 + h(x)y = f(x) \text{ in } \mathbb{F}_q[x, y], \tag{1}$$

where $f(x) \in \mathbb{F}_q[x]$ is a monic polynomial of degree $2g+1$, $h(x) \in \mathbb{F}_q[x]$ is a polynomial of degree at most g, and there are no solutions $(a, b) \in \bar{\mathbb{F}}_q \times \bar{\mathbb{F}}_q$ which simultaneously satisfy the equation $b^2 + h(a)b = f(a)$ and the partial derivative equations $2b + h(a) = 0$ and $h'(a)b - f'(a) = 0$. The curve $C/\mathbb{F}_q$ associated to this function field is called a hyperelliptic curve of genus g defined over $\mathbb{F}_q$.

For odd characteristic it suffices to let $h(x) = 0$ and to have f squarefree to satisfy the last condition of the definition.

For our purposes it is enough to consider a point as a tuple $(a, b) \in \bar{\mathbb{F}}_q^2$ which satisfies $n^2 + h(a)b = f(a)$. Besides these tuples there is one point ∞ at infinity. The hyperelliptic involution ι maps (a, b) to $(a, -b - h(a))$ and leaves ∞ fixed.
A divisor D of C is an element of the free abelian group over the points of C, e. g. $D = \sum_{P \in C} n_P P$ with $n_P \in \mathbb{Z}$ and $n_P = 0$ for almost all points P. The degree of D is defined as $\deg(D) = \sum_{P \in C} n_P$. To every element F of the function field we can associate a divisor via the valuations at all points of the curve $\mathrm{div}(F) = \sum_{P \in C(\bar{\mathbb{F}}_q)} v_P(F)P$. These so called principal divisors are of degree zero and form a subgroup of the group of degree zero divisors. The quotient group is called the *divisor class group*. The function $(x - a)$ leads to a divisor $P + \iota P - 2\infty$. Hence, we can achieve that we represent a divisor class by a divisor $D = \sum_{i=1}^r P_i - r\infty$, where $P_i \neq \infty$ and $P_i \neq \iota P_j$ for $i \neq j$. Furthermore one finds a representative with $r \leq g$ for each class. Note that D defined over $\mathbb{F}_q$ does not imply that each P_i is defined over this field. If P_i is defined over $\mathbb{F}_{q^l}$ then all l conjugates of P_i must also occur in D. Therefore l is bounded by g.

The maximal ideals of $\mathbb{F}_q[x, y]/(y^2 + h(x)y - f(x))$ have a basis consisting of two polynomials and one can achieve that the first polynomial is in $\mathbb{F}_q[x]$, whereas the second one is of the form $y - v(x), v(x) \in \mathbb{F}_q[x]$, since we reduce modulo a polynomial of degree 2 in y. Now consider the ideal class group, i.e. the ideals modulo the principal ideals. In Mumford [14][page 3.17] the following representation is introduced which makes explicit the correspondence of ideal classes and divisor classes:

Theorem 2.2 (Mumford Representation)
Let the function field be given via the absolutely irreducible polynomial $y^2 + h(x)y = f(x)$, where $h, f \in \mathbb{F}_q[x]$, $\deg f = 2g + 1$, $\deg h \leq g$. Each nontrivial ideal class over $\mathbb{F}_q$ can be represented via a unique ideal generated by $u(x)$ and $y - v(x)$, $u, v \in \mathbb{F}_q[x]$, where

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

Let $D = \sum_{i=1}^r P_i - r\infty$, where $P_i \neq \infty, P_i \neq \iota P_j$ for $i \neq j$ and $r \leq g$. Put $P_i = (a_i, b_i)$. Then the corresponding ideal class is represented by $u = \prod_{i=1}^r (x - a_i)$ and if P_i occurs n_i times then $\left(\frac{d}{dt}\right)^j \left[v(x)^2 + v(x)h(x) - f(x)\right]_{|x=a_i} = 0,\ 0 \leq j \leq n_i - 1$.

The second part of the theorem means that for all points $P_i = (a_i, b_i)$ occurring in the support of D we have $u(a_i) = 0$ and the third condition guarantees that $v(a_i) = b_i$ with appropriate multiplicity.

For short we denote this ideal by $[u, v]$. The inverse of a class is represented by $[u, -h - v]$, where the second polynomial is understood modulo u if necessary. The ideal class group over $\mathbb{F}_q$ is denoted by $\mathrm{Cl}(C/\mathbb{F}_q)$. The zero element of of $\mathrm{Cl}(C/\mathbb{F}_q)$ is represented by $[1, 0]$.

3 Arithmetic using Cantor's Algorithm

In this section we consider the group operation. Here we still deal with general hyperelliptic curves, i.e. curves of arbitrary genus. Addition of divisor classes means multiplication of ideal classes, which consists in a composition of the ideals and a first reduction to a basis of two polynomials. The output of this algorithm is said to be semi-reduced. Then we need a second algorithm, which is usually called reduction, to find the unique representative in the class referred to above. Such an ideal is called *reduced*. Due to the work of Cantor [2] (for odd characteristic only) and Koblitz [4] one has an efficient algorithm to perform these operation, which uses only polynomial arithmetic over the finite field in which the ideal classes are defined.

Algorithm 3.1 (Composition)
`INPUT:` $D_1 = [u_1, v_1], D_2 = [u_2, v_2]$,
$C : y^2 + h(x)y = f(x)$.
`OUTPUT:` $D = [u, v]$ *semi-reduced with* $D \equiv D_1 D_2$.

1. *compute* $d_1 = \gcd(u_1, u_2) = e_1 u_1 + e_2 u_2$*;*
2. *compute* $d = \gcd(d_1, v_1 + v_2 + h) = c_1 d_1 + c_2(v_1 + v_2 + h)$*;*
3. *let* $s_1 = c_1 e_1,\ s_2 = c_1 e_2,\ s_3 = c_2$*;* /* *i.e.* $d = s_1 u_1 + s_2 u_2 + s_3(v_1 + v_2 + h)$ */
4. $u = \frac{u_1 u_2}{d^2}$*;*
   $v = \frac{s_1 u_1 v_2 + s_2 u_2 v_1 + s_3(v_1 v_2 + f)}{d} \bmod u$*.*

Algorithm 3.2 (Reduction)
`INPUT:` $D = [u, v]$ *semi-reduced.*
`OUTPUT:` $D' = [u', v']$ *reduced with* $D \equiv D'$.

1. *let* $u' = \frac{f - vh - v^2}{u}$, $v' = (-h - v) \bmod u'$*;*
2. *if* $\deg u' > g$ *put* $u = u', v = v'$*;*
   goto step 1;
3. *make* u' *monic.*

This algorithm provides a universal way of doing arithmetic in $\mathrm{Cl}(\mathbb{F}_q)$ which applies to any genus and characteristic. However, in a straightforward implementation several unneeded coefficients are computed. Therefore a careful study making the steps explicit is necessary. We deal with this in the following section.

4 Explicit formulae

The first attempt to avoid using Cantor's algorithm for faster arithmetic and deriving explicit formulae was made by Spallek [18] for genus two and odd characteristic. Harley [3] takes a slightly different approach to optimize the running time. This approach was generalized to even characteristic by Lange [7].
Building on the work of Matsuo, Chao, and Tsujii [11], recently Miyamoto, Doi, Matsuo, Chao, and Tsuji [13] and Takahashi [21] obtained an even larger speed-up using Montgomery's trick to reduce the number of inversions to 1. We generalize the setting in order to deal with even characteristic as well. To do so, we first give the case study of [7] investigating what can be the input of the combination algorithm and proceed in considering these different cases. We determine the exact number of operations needed to perform addition and doubling in the most common cases.
Unless stated otherwise the formulae hold independently of the characteristic, therefore we take care of the signs; in characteristic two, $2y$ is understood as zero.

4.1 Different Cases

Consider the composition step of Cantors Algorithm 3.1. The input are two classes represented by two polynomials $[u_i, v_i]$ each. As we consider curves of genus two the following holds by Theorem 2.2:

1. u is monic,
2. $\deg v < \deg u \leq 2$,
3. $u | v^2 + vh - f^2$.

Without loss of generality let $\deg u_1 \leq \deg u_2$.

1. u_1 is of degree zero, this is only possible in the case $[u_1, v_1] = [1, 0]$, i.e. for the zero element. The result of the combination and reduction is the second class $[u_2, v_2]$.
2. If u_1 is of degree one, then either u_2 is of degree one as well or it has full degree.
   (a) Assume $\deg u_2 = 1$, i.e. $u_i = x + u_{i0}$ and the v_i are constant. Then if $u_1 = u_2$ we obtain for $v_1 = -v_2 - h(-u_{10})$ the zero element $[1, 0]$ and for $v_1 = v_2$ we double the divisor to obtain

   $$\begin{aligned} u &= u_1^2, \\ v &= ((f'(-u_{10}) - v_1 h'(-u_{10}))x + (f'(-u_{10}) - v_1 h'(-u_{10}))u_{10})/(2v_1 + h(-u_{10})) + v_1. \end{aligned} \tag{2}$$

   Otherwise the composition leads to $u = u_1 u_2$ and
   $v = ((v_2 - v_1)x + v_2 u_{10} - v_1 u_{20})/(u_{10} - u_{20})$.
   In all cases the results are already reduced.

   (b) Now let the second polynomial be of degree two, $u_2 = x^2 + u_{21}x + u_{20}$. Then the corresponding divisors are given by $D_1 = P_1 - \infty$ and $D_2 = P_2 + P_3 - 2\infty$, $P_i \neq \infty$.
      i. If $u_2(-u_{10}) \neq 0$ then P_1 and $-P_1$ do not occur in D_2. This case will be dealt with below in Subsection 4.2.2.
      ii. Otherwise if $v_2(-u_{10}) = v_1 + h(-u_{10})$ then $-P_1$ occurs in D_2 and the resulting class is given by $u = x + u_{21} - u_{10}$ and $v = v_2(-u_{21} + u_{10})$ as $-u_{21}$ equals the sum of the x-coordinates of the points.
      Otherwise one first doubles $[u_1, v_1]$ by (2) and then adds $[x + u_{21} - u_{10}, v_2(-u_{21} + u_{10})]$, hence, reduces the problem to the case 2(b)i, unless $D_2 = 2P_1 - 2\infty$ where one first doubles D_2 as in 3(a)ii and then subtracts D_1 using 2(b)i.
3. Let $\deg u_1 = \deg u_2 = 2$.
   (a) Let first $u_1 = u_2$. This means that for an appropriate ordering $D_1 = P_1 + P_2 - 2\infty$, $D_2 = P_3 + P_4 - 2\infty$ the x-coordinates of P_i and P_{i+2} are equal.
      i. If $v_1 \equiv -v_2 - h \bmod u_1$ then the result is $[1, 0]$.
      ii. If $v_1 = v_2$ then we are in the case in which we double a class of order different from two and with first polynomial of full degree. Again we need to consider two sub-cases:
      If $D_1 = P_1 + P_2 - 2\infty$ where P_1 is equal to its opposite, then the result is $2P_2$ and can be computed like above. $P_1 = (x_{P_1}, y_{P_1})$ is equal to its opposite, iff $h(x_{P_1}) = -2y_{P_1}$. To check for this case we compute the resultant of $h + 2v_1$ and u_1.
         A. If $\text{res}(h + 2v_1, u_1) \neq 0$ then we are in the usual case where both points are not equal to their opposite. This will be considered in Subsection 4.2.3.
         B. Otherwise we compute the $\gcd(h + 2v_1, u_1) = (x - x_{P_1})$ to get the coordinate of P_1 and double $[x + u_{11} + x_{P_1}, v_1(-u_{11} - x_{P_1})]$.
      iii. Now we know that without loss of generality $P_1 = P_3$ and $P_2 \neq P_4$ is the opposite of P_4. Let $v_i = v_{i1}x + v_{i0}$, then the result $2P_1$ is obtained by doubling
      $[x - (v_{10} - v_{20})/(v_{21} - v_{11}), v_1((v_{10} - v_{20})/(v_{21} - v_{11}))]$ using (2).

(b) For the remaining case $u_1 \neq u_2$, we need to consider the following possibilities.

i. If $\operatorname{res}(u_1, u_2) \neq 0$ then no point of D_1 is equal to a point or its opposite in D_2. This is the most frequent case. We deal with it in Subsection 4.2.1.

ii. If the above resultant is zero then $\gcd(u_1, u_2) = x - x_{P_1}$ and we know that either $D_1 = P_1 + P_2 - 2\infty$, $D_2 = P_1 + P_3 - 2\infty$ or D_2 contains the opposite of P_1 instead. This can be checked by inserting x_{P_1} in both v_1 and v_2.

A. If the results are equal then we are in the first case and proceed by computing $D' = 2(P_1 - \infty)$, then $D'' = D' + P_2 - \infty$ and finally $D = D'' + P_3 - \infty$ by the formulae in 2. We extract the coordinates of P_2 and P_3 by $P_2 = (-u_{11} + x_{P_1}, v_1(-u_{11} + x_{P_1})), P_3 = (-u_{21} + x_{P_1}, v_2(-u_{21} + x_{P_1}))$.

B. In case $v_1(x_{P_1}) \neq v_2(x_{P_1})$ the result is $P_2 + P_3 - 2\infty$.

If one uses the resultant as recommended in 3(a)ii and 3(b)i then one needs to compute a greatest common divisor as well, to extract the coordinates of P_1 when needed. However, most frequently we are in the case of resultant nonzero and thus we save on average.

4.2 Addition and Doubling

We now present in detail the algorithms for the most common cases. Put $f(x) = x^5 + \sum_{i=0}^4 f_i x^i$, $h(x) = \sum_{i=0}^2 h_i x^i$. For the complexity estimates we always assume $h_2 \in \{0,1\}$. For nonzero h_2 this can be achieved by substituting $y = h_2^5 y', x = h_2^2 x'$ and dividing the resulting equation by h_2^{10}. If one does not want to transform some computations should be performed differently (like $s_0(s_0 + h_2)$ instead of $s_0^2 + s_0 h_2$). Similarly we assume $f_4 = 0$, as this can be derived for $p \neq 5$ by the substitution $x' = (x - f_4/5)$, and include this coefficient only for completeness. Furthermore we assume $h_1 \in \{0,1\}$. We would like to stress that the formulae remain correct for other values, one simply has to allow some more operations.

4.2.1 Addition in Most Common Case

In this case the two divisor classes to be combined consist of four points different from each other and from each other's negative. The results of the composition Algorithm 3.1 are $u_1 u_2$ and a polynomial v of degree ≤ 3 satisfying $u | v^2 + vh - f$ (see Theorem 2.2). As we started with $u_i | v_i^2 + v_i h - f$ we can obtain v using Chinese remaindering:

$$\begin{aligned} v &\equiv v_1 \bmod u_1, \\ v &\equiv v_2 \bmod u_2. \end{aligned} \tag{3}$$

Then we compute the resulting first polynomial u' by making $(f - vh - h^2)/(u_1 u_2)$ monic and taking $v' = (-h - v \bmod u')$.

To optimize the computations we do not follow this literally. We now list the needed subexpressions and then show that in fact we obtain the desired result.

$$\begin{aligned} k &= (f - v_2 h - v_2^2)/u_2 \\ s &\equiv (v_1 - v_2)/u_2 \bmod u_1 \\ l &= s \cdot u_2 \\ u &= (k - s(l + h + 2v_2))/u_1 \\ u' &= u \text{ made monic} \\ v' &\equiv -h - (l + v_2) \bmod u' \end{aligned}$$

The divisions made to get k and u are exact divisions due to the definition of the polynomials. Let us first verify that $v = l + v_2 = s \cdot u_2 + v_2$ satisfies the system of equations (3). This is obvious for the second

equation. For the first one we consider $v \equiv s \cdot u_2 + v_2 \equiv ((v_1 - v_2)/u_2)u_2 + v_2 \equiv v_1 \bmod u_1$.
Now we check that $u = (f - vh - v^2)/(u_1 u_2)$ by expanding out

$$u_1 \cdot u_2 \cdot u = u_2(k - s(l + h + 2v_2)) = f - v_2 h - v_2^2 - l(l+h) - 2lv_2 = f - vh - v^2.$$

In the course of computing we do not need all coefficients of the polynomials defined above. As $f = x^5 + \sum_{i=0}^{4} f_i x^i$ is monic and of degree 5, u_2 is monic of degree 2, $\deg h \leq 2$, and $\deg v_2 = 1$ we have that $k = x^3 + (f_4 - u_{21})x^2 + cx + c'$, where $c,\ c'$ are some constants. In the computation of u we divide an expression involving k by a polynomial of degree 2, thus we only need the above known part of k. In the computation of a product of polynomials we use the following Karatsuba style formula to save one multiplication:

$$(ax + b)(cx + d) = acx^2 + ((a + b)(c + d) - ac - bd)x + bd.$$

To reduce a polynomial of degree 3 modulo a monic one of degree 2 we use

$$ax^3 + bx^2 + cx + d \equiv (c - (i + j)(a + (b - ia)) + ia + j(b - ia))x + d - j(b - ia) \bmod x^2 + ix + j$$

using only 3 multiplications instead of four. Furthermore we use an almost inverse in the computation of s and compute rs instead, where r is the resultant of u_1 and u_2, postponing and combining the inversion of r with that of s.
In the following Table 1 we list the intermediate steps together with the number of multiplications (M), squarings (S) and inversions (I) needed. As we assume $h_2, h_1, f_4 \in \{0, 1\}$ we do not count multiplications by these coefficients. In the case study we have already computed the resultant of u_1 and u_2 when we arrive at this algorithm. Hence, we can assume that $\tilde{u}_2 = u_2 \bmod u_1$ and $\mathrm{res}(\tilde{u}_2, u_1)$ are known. However, we include the costs in the table, as we use these expressions to compute $1/\tilde{u}_2 \bmod u_1$.

Remark: Note, that if we assume that our field is represented via a normal basis and work in characteristic two, the squarings are virtually for free and using a polynomial basis they still are by far cheaper than multiplications. Furthermore, for even characteristic one can save one multiplications as $(h_1 + 2v_1)w_4 = h_1 w_4$.

4.2.2 Addition in Case $\deg u_1 = 1,\ \deg u_2 = 2$

By the above considerations we can assume that for $u_1 = x + u_{10}$ we have that $u_2(-u_{10}) \neq 0$.
In principle we follow the same algorithm as stated in the previous subsection. But to obtain u we divide by a polynomial of degree one, therefore we need an additional coefficient of k and save a lot in the other operations. Table 2 shows that this case is much cheaper than the general one, however it is not too likely to happen like all special cases.

4.2.3 Doubling

The above case study left open how one computes the double of a class where the first polynomial has degree two and both points of the representing divisor are not equal to their opposite. Put $u = x^2 + u_1 x + u_0,\ v = v_1 x + v_0$. Combining $[u, v]$ with itself should result in a class $[u_{\mathrm{new}}, v_{\mathrm{new}}]$, where

$$\begin{aligned} u_{\mathrm{new}} &= u^2, \\ v_{\mathrm{new}} &\equiv v \bmod u, && (4) \\ u_{\mathrm{new}} &\mid v_{\mathrm{new}}^2 + v_{\mathrm{new}} h - f. && (5) \end{aligned}$$

Then this class is reduced to obtain $[u', v']$. We use the following subexpressions:

$$\begin{aligned} k &= (f - hv - v^2)/u \\ s &\equiv k/(h + 2v) \bmod u \\ l &= su \\ u_1 &= s^2 - ((h + 2v)s - k)/u \\ u' &= u_1 \text{ made monic} \\ v' &\equiv -h - (l + v) \bmod u' \end{aligned}$$

Table 1: **Addition,** $\deg u_1 = \deg u_2 = 2$

Input	$[u_1, v_1], [u_2, v_2], \deg u_1 = \deg u_2 = 2$ $u_i = x^2 + u_{i1}x + u_{i0}, v_i = v_{i1}x + v_{i0}$ $h = h_2x^2 + h_1x + h_0, f = x^5 + f_4x^4 + f_3x^3 + f_2x^2 + f_1x + f_0$
Output	$[u', v'] = [u_1, v_1] + [u_2, v_2]$

Step	Expression	Operations
1	compute resultant r of u_1, u_2: $z_1 = u_{11} - u_{21}$, $z_2 = u_{20} - u_{10}$, $z_3 = u_{11}z_1 + z_2$; $r = z_2z_3 + z_1^2u_{10}$;	1S, 3M
2	compute almost inverse of u_2 modulo u_1 ($inv = r/u_2 \bmod u_1$): $inv_1 = z_1$, $inv_0 = z_3$;	
3	compute $s' = rs \equiv (v_1 - v_2)inv \bmod u_1$: $w_0 = v_{10} - v_{20}$, $w_1 = v_{11} - v_{21}$, $w_2 = inv_0w_0$, $w_3 = inv_1w_1$; $s'_1 = (inv_0 + inv_1)(w_0 + w_1) - w_2 - w_3(1 + u_{11})$, $s'_0 = w_2 - u_{10}w_3$; If $s_1 = 0$ see below	5M
4	compute $s'' = x + s_0/s_1 = x + s'_0/s'_1$ and s_1: $w_1 = (rs'_1)^{-1}(= 1/r^2s_1)$, $w_2 = rw_1(= 1/s'_1)$, $w_3 = {s'}_1^2w_1(= s_1)$; $w_4 = rw_2(= 1/s_1)$, $w_5 = w_4^2$; $s''_0 = s'_0w_2$;	I, 2S, 5M
5	compute $l' = s''u_2 = x^3 + l'_2x^2 + l'_1x + l'_0$: $l'_2 = u_{21} + s''_0$, $l'_1 = u_{21}s''_0 + u_{20}$, $l'_0 = u_{20}s''_0$	2M
6	compute $u' = (s(l + h + 2v_2) - k)/u_1 = x^2 + u'_1x + u'_0$: $u'_0 = (s''_0 - u_{11})(s''_0 - z_1 + h_2w_4) - u_{10} + l'_1 + (h_1 + 2v_{21})w_4 + (2u_{21} + z_1 - f_4)w_5$; $u'_1 = 2s''_0 - z_1 + h_2w_4 - w_5$;	3M
7	compute $v' \equiv -h - (l + v_2) \bmod u' = v'_1x + v'_0$: $w_1 = l'_2 - u'_1$, $w_2 = u'_1w_1 + u'_0 - l'_1$; $v'_1 = w_2w_3 - v_{21} - h_1 + h_2u'_1$; $w_2 = u'_0w_1 - l'_0$; $v'_0 = w_2w_3 - v_{20} - h_0 + h_2u'_0$;	4M
total		I, 3S, 22M
	Special case $s = s_0$	
4’	compute s: $inv = 1/r$, $s_0 = s'_0inv$;	I, M
5’	compute $u' = (k - s(l + h + 2v_2))/u_1 = x + u'_0$: $u'_0 = f_4 - u_{21} - u_{11} - s_0^2 - s_0h_2$;	S
6’	compute $v' \equiv -h - (l + v_2) \bmod u' = v'_0$: $w_1 = s_0(u_{21} + u`_0) + h_1 + v_{21} + h_2u`_0$, $w_2 = s_0 + v_{20} + h_0$; $v'_0 = u'_0w_1 - w_2$;	2M
total		I, 2S, 11M

Table 2: **Addition,** $\deg u_1 = 1, \deg u_2 = 2$

Input	$[u_1, v_1], [u_2, v_2], \deg u_1 = 1, \deg u_2 = 2$ $u_1 = x + u_{10}, u_2 = x^2 + u_{21}x + u_{20}, v_1 = v_{10}, v_2 = v_{21}x + v_{20}$ $h = h_2x^2 + h_1x + h_0, f = x^5 + f_4x^4 + f_3x^3 + f_2x^2 + f_1x + f_0$
Output	$[u', v'] = [u_1, v_1] + [u_2, v_2]$

Step	Expression	Operations
1	compute $r \equiv u_2 \bmod u_1$: $r = u_{20} - (u_{21} - u_{10})u_{10}$;	M
2	compute inverse of u_2 modulo u_1: $inv = 1/r$	I
3	compute $s = (v_1 - v_2)inv \bmod u_1$: $s_0 = inv(v_{10} - v_{20} - v_{21}u_{10})$;	2M
4	compute $l = s \cdot u_2 = s_0x^2 + l_1x + l_0$: $l_1 = s_0u_{21}$, $l_0 = s_0u_{20}$;	2M
5	compute $k = (f - v_2h - v_2^2)/u_2 = x^3 + k_2x^2 + k_1x + k_0$: $k_2 = f_4 - u_{21}$, $k_1 = f_3 - (f_4 - u_{21})u_{21} - v_{21}h_2 - u_{20})$;	M
6	compute $u' = (k - s(l + h + 2v_2))/u_1 = x^2 + u_1'x + u_0'$: $u_1' = k_2 - s_0^2 - s_0h_2 - u_{10}$; $u_0' = k_1 - s_0(l_1 + h_1 + 2v_{21}) - u_{10}u_1'$;	S,2M
7	compute $v' \equiv -h - (l + v_2) \bmod u' = v_1'x + v_0'$: $v_1' = (h_2 + s_0)u_1' - (h_1 + l_1 + v_{21})$; $v_0' = (h_2 + s_0)u_0' - (h_0 + l_0 + v_{20})$;	2M
total		I, S, 10 M

Note that like above we do not compute the semi-reduced divisor explicitely, here $v_{\text{new}} = su + v$. Hence, we see that (4) holds. To prove (5) we consider

$$v_{\text{new}}^2 + v_{\text{new}}h - f \equiv l^2 + 2lv + v^2 + hl + hv - f = s^2u^2 + u(s(h + 2v) - k)$$

and

$$(h + 2v)s - k \equiv (h + 2v)k/(h + 2v) - k \equiv 0 \bmod u.$$

Finally, one finds by

$$(v_{\text{new}}^2 + v_{\text{new}}h - f)/u_{\text{new}} = (s^2u^2 + (h + 2v)su - ku)/u^2$$

that u_1 is in fact obtained as described in the reduction algorithm.
Table 3 lists the numbers of elementary operations needed in the following table. Unlike in the addition case we now need the exact polynomial k to compute d. We include the costs to compute $\operatorname{res}(\tilde{h}, u)$ and assume $h_2, h_1, f_4 \in \{0, 1\}$.

Remarks:

1. Concerning the counting in Step 2 a remark is in order. Unless the characteristic is odd and $h \neq 0$, the computation of v_1^2, u_1^2 and $\tilde{v}_1^2$ needs only 2 squarings instead of the obvious 3. (In detail: if for odd characteristic $h = 0$ then $\tilde{v}_1^2 = 4v_1^2$. If $p = 2$ then $\tilde{v}_1^2 = h_2^2u_1^2 + h_1^2 = h_2^2w_1 + h_1^2$ and $h_2, h_1 \in \{0, 1\}$. These details can be fixed for an actual implementation.)

2. In characteristic 2 one can reduce the number of multiplications by 2 as in Step 1 $w_3 = h_2u_1^2 + h_1u_1 = h_2w_1 + h_1u_1$ and in the computation of $u_0' = {s_0''}^2 + w_4(h_2(s_0'' + u_1) + h_1) + w_5f_4$ the multiplication by f_4 need not be counted.
   If additionally $h_2 = 0$ the number of operations reduces even to $I, 5S, 17M$, as $r = h_1(w_0h_1 + h_0u_1)$ can be computed directly and for free, and $u_0' = {s_0''}^2 + w_4h_1 + w_5f_4$ needs only one squaring.

Table 3: **Doubling,** $\deg u = 2$

Input	$[u,v], \deg u = 2$ $u = x^2 + u_1x + u_0, v = v_1x + v_0$ $h = h_2x^2 + h_1x + h_0, f = x^5 + f_4x^4 + f_3x^3 + f_2x^2 + f_1x + f_0$	
Output	$[u', v'] = 2[u, v]$	
Step	Expression	Operations
1	compute $\tilde{v} \equiv (h + 2v) \bmod u = \tilde{v}_1x + \tilde{v}_0$: $\tilde{v}_1 = h_1 + 2v_1 - h_2u_1$, $\tilde{v}_0 = h_0 + 2v_0 - h_2u_0$;	
2	compute resultant $r = \mathrm{res}(\tilde{v}, u)$: $w_0 = v_1^2$, $w_1 = u_1^2$, $w_2 = \tilde{v}_1^2$, $w_3 = u_1\tilde{v}_1$; $r = u_0w_2 + \tilde{v}_0(\tilde{v}_0 - w_3)$;	2S, 3M (see below)
3	compute almost inverse $inv' = inv \cdot r$: $inv_1' = -\tilde{v}_1$, $inv_0' = \tilde{v}_0 - w_3$;	
4	compute $k' = (f - hv - v^2)/u \bmod u = k_1'x + k_0'$: $w_3 = f_3 + w_1$, $w_4 = 2u_0$; $k_1' = 2(w_1 - f_4u_1) + w_3 - w_4 - v_1h_2$; $k_0' = u_1(2w_4 - w_3 + f_4u_1 + v_1h_2) + f_2 - w_0 - 2f_4u_0 - v_1h_1 - v_0h_2$;	1M
5	compute $s' = k'inv' \bmod u$: $w_0 = k_0'inv_0'$, $w_1 = k_1'inv_1'$; $s_1' = (inv_0' + inv_1')(k_0' + k_1') - w_0 - w_1(1 + u1)$; $s_0' = w_0 - u_0w_1$; If $s_1 = 0$ see below	5M
6	compute $s'' = x + s_0/s_1$ and s_1: $w_1 = 1/(rs_1')(= 1/r^2s_1)$, $w_2 = rw_1(= 1/s_1')$, $w_3 = {s'}_1^2 w_1(= s_1)$; $w_4 = rw_2(= 1/s_1)$, $w_5 = w_4^2$; $s_0'' = s_0'w_2$;	I, 2S, 5M
7	compute $l' = s''u = x^3 + l_2'x^2 + l_1'x + l_0'$: $l_2' = u_1 + s_0''$, $l_1' = u_1s_0'' + u_0$, $l_0' = u_0s_0''$;	2M
8	compute $u' = s^2 + (h + 2v)s/u + (v^2 + hv - f)/u^2$: $u_0' = {s_0''}^2 + w_4(h_2(s_0'' - u_1) + 2v_1 + h_1) + w_5(2u_1 - f_4)$; $u_1' = 2s_0'' + w_4h_2 - w_5$;	S, 2M
9	compute $v' \equiv -h - (l + v) \bmod u' = v_1'x + v_0'$: $w_1 = l_2' - u_1'$, $w_2 = u_1'w_1 + u_0' - l_1'$; $v_1' = w_2w_3 - v_1 - h_1 + u_1'h_2$; $w_2 = u_0'w_1 - l_0'$; $v_0' = w_2w_3 - v_0 - h_0 + h_2u_0'$;	4M
total		I, 5S, 22 M
	Special case $s = s_0$	
6'	compute s and precomputations: $w_1 = 1/r$, $s_0 = s_0'w_1$; $w_1 = u_1s_0 + v_1 + h_1$, $w_2 = u_0s_0 + v_0 + h_0$;	I,3M
7'	compute $u' = (f - hv - v^2)/u^2 - (h + 2v)s/u - s^2$: $u_0' = f_4 - s_0^2 - s_0h_2 - 2u_1$;	S
8'	compute $v' \equiv -h - (su + v) \bmod u'$: $w_1 = w_1 - u_0'(s_0 + h_2)$, $v_0' = u_0'w_1 - w_2$;	2M
total		I, 3S, 14M

5 Timings

To compare the arithmetic on elliptic and hyperelliptic curves of genus 2 we implemented the above formulae in various environments. For elliptic curves we used the affine representation, i. e. the curve is given as in the introduction with $g = 1$. This is justified by the fact that for all libraries one inversion takes less than six multiplications. The times needed for multiplication, squaring and inversion by the different libraries are listed in the appendix. Addition formulae for elliptic curves can be found in almost any textbook on this subject, e. g. Blake, Seroussi, and Smart [1], Silverman [17] or Koblitz [5]. To add two distinct points one needs one inversion, one squaring and two multiplications whereas doubling takes one more squaring.

	ECC, $\mathbb{F}_{q_{ell}}$			**HEC**, $\mathbb{F}_{q_{hyp}}$		
	I	S	M	I	S	M
Addition	1	1	2	1	2	23
Doubling	1	2	2	1	5	22
m-fold $\log_2 m = \lambda$	6λ	10λ	12λ	6λ	24λ	134λ

For properly chosen curves (not supersingular, group order contains a large prime factor) the security mainly depends on the group size. By Weil's theorem we have

$$|\mathrm{Cl}(C/\mathbb{F}_{q^n})| \sim q^{ng}.$$

Hence, to achieve the same level of security the finite field for elliptic curves needs to be larger $q_{ell} \sim q_{hyp}^2$. As the arithmetic in $\mathbb{F}_q$ takes time $\tilde{O}(\log q)$, inversions are more expensive than multiplications, and squarings are even cheaper, one might expect that hyperelliptic curves could offer faster arithmetic than elliptic curves for the same level of security or at least achieve the same level of speed. The outcome of the experiments shows that this relation depends heavily on the underlying arithmetic. To be more precise, breaking down the operations to the smaller field $\mathbb{F}_{q_{hyp}}$ on an elliptic curve we need at least two times as many inversions but less multiplications than on a hyperelliptic curve.
For the implementations we used curves over prime fields $\mathbb{F}_p$ and curves over $\mathbb{F}_{2^n}$. The latter are Koblitz curves defined over $\mathbb{F}_2$, therefore one can achieve a much faster implementation using the Frobenius endomorphism (see [7]), however, here we were only interested in the effects of the usual arithmetic in the ideal class group. We worked with the following curves and fields.

		ECC
$\mathbb{F}_p$	1	$y^2 = x^3 + 2227264092164547360326443915353641005956041076603x +$ $+2705501697235753328656362326336626679936508382816$ $p_1 = 2923003274661805836407369665432566039311865086059$
$\mathbb{F}_p$	2	$y^2 = x^3 + 177373491766423492438941130272742535721285460756540682\,93x +$ $+1139812421165142041382549725831795824113175913476452\,6623$ $p_2 = 183899464903906663003001643258037102034248694662\,03226147$
$\mathbb{F}_2$	3	$y^2 + y = x^3 + 1$
$\mathbb{F}_2$	4	$y^2 + y = x^3 + x^2 + 1$
		HEC
$\mathbb{F}_p$	5	$y^2 = x^5 + 1538342954334616836340\,59x^3 + 1503542947764347319629935x^2 +$ $+1930714025804554453580068x + 7909928247998759052669\,69$ $p_3 = 1932005208863265003490787$
$\mathbb{F}_p$	6	$y^2 = x^5 + 2412164359980685576827425\,15x^3 + 5530115864651869801140364\,62x^2 +$ $+145662144625109198973105\,7514x + 3440013483680364963850133535$ $p_4 = 3713820117856140824697372689$
$\mathbb{F}_2$	7	$y^2 + xy = x^5 + x^2 + 1$
$\mathbb{F}_2$	8	$y^2 + (x^2 + x + 1)y = x^5 + x$

For the binary elliptic curves we considered the field extensions $n = 163$ and $n = 191$ and for hyperelliptic curves we took $n = 83$ and $n = 97$.

Note that in the genus 2 case we cannot determine the group order for the prime fields, but the arithmetic depends only on the field size, thus even if these particular curves should turn out to be weak the arithmetic for appropriate curves is equally fast.
In all the experiments the special cases never occurred, this goes along with the fact that the probability of occurrence is $\sim 1/p$.
All computations where performed on a Pentium IV, 1.5 GHz under linux.

5.1 Prime Fields

We did a C implementation using GMP as long integer package and a C++ implementation with NTL. For GMP the field elements were considered as integers and reduction took place only where necessary. With NTL we used the library to perform finite field arithmetic.
We carried out 10 000 scalar multiplications per curve using binary double-and-add, where the scalar is in the range of the group order. The table lists the average time needed to perform a scalar multiplication on the respective curve, where the scalar was of the order of the assumed group size. Time is given in ms.

GMP	ECC	HEC
$\sim$ 160 bits	4.577	8.232
$\sim$ 180 bits	5.668	9.121

NTL	ECC	HEC
$\sim$ 160 bits	10.785	11.326
$\sim$ 180 bits	14.303	16.324

Thus interestingly the results show that the relation between the cost of arithmetic for elliptic and hyperelliptic curves depends heavily on the chosen implementation of field arithmetic. In any case the complexity is of the same order (at most twice) for both genera. In GMP we have that an inversion is less expensive compared to multiplications than in NTL (see Appendix). This relation seems to be the reason why in GMP the arithmetic on elliptic curves is faster whereas in NTL the timings are balanced.

5.2 Binary Fields

The C++ program for curves over binary fields is based on an implementation of the arithmetic in $\mathbb{F}_{2^n}$ by Michael Nöcker [15], which allows to work with normal and polynomial bases and in the latter case accepts user defined irreducible polynomials. His program was built on bipolar, an implementation of polynomial arithmetic over $\mathbb{F}_2$. Here we use polynomial arithmetic for the computations in $\mathbb{F}_{2^n}$. See [9] for a detailed study of the appropriate implementation of Koblitz curves. As irreducible polynomials we took sparse polynomials. For $n = 163$ and $n = 83$ we used a pentanomial; for $n = 191$ and $n = 97$ irreducible trinomials exist. Again 10 000 scalar multiplications per curve and extension field were performed and the table lists the average time for one scalar multiplication.

binary field	ECC		HEC	
curve	3	4	7	8
$\sim$ 160 bits	26.208	26.504	18.875	21.143
$\sim$ 190 bits	31.958	32.240	25.215	27.188

First of all one notices that again with this library arithmetic on hyperelliptic curves is faster than on elliptic curves. In comparing curves 3 and 4 respectively 7 and 8 one realizes that the running-time also depends on the coefficients of the equation of the curve. The difference results from the distinct number of additions needed.

6 Outlook

The formulae for even characteristic have been implemented on a ARM processor by Pelzl [16]. He also generalizes the explicit formulae for genus three given by Kuroki, Gonda, Matsuo, Chao, and Tsuji [6] to even characteristic.

For restricted devices inversions are very expensive. A first attempt to avoid divisions on the cost of more multiplications and a further coordinate is given by Miyamoto, Doi, Matsuo, Chao, and Tsuji [13]. An optimized version that also covers even characteristic and considers the case of scalar multiplication is to appear [8].

References

[1] I.F. Blake, G. Seroussi, and N.P. Smart. *Elliptic curves in cryptography.* London Mathematical Society Lecture Note Series. 265. Cambridge University Press, 1999.

[2] D.G. Cantor. Computing in the Jacobian of a hyperelliptic curve. *Math. Comp.*, 48:95–101, 1987.

[3] R. Harley. Fast arithmetic on genus 2 curves. available at `http://cristal.inria.fr/~harley/hyper`, 2000.

[4] N. Koblitz. Hyperelliptic cryptosystems. *J. of Cryptology*, 1:139–150, 1989.

[5] N. Koblitz. *Algebraic aspects of cryptography.* Springer, 1998.

[6] J. Kuroki, M. Gonda, K. Matsuo, J. Chao, and S. Tsuji. Fast Genus Three Hyperelliptic Curve Cryptosystems. In *Proc. of SCIS2002, IEICE Japan*, 2002.

[7] T. Lange. *Efficient Arithmetic on Hyperelliptic Curves.* PhD thesis, Universität Gesamthochschule Essen, 2001.

[8] T. Lange. Inversion-Free Arithmetic on Genus 2 Hyperelliptic Curves. Cryptology ePrint Archive, Report 2002/147, 2002. `http://eprint.iacr.org/` or `http://www.itsc.ruhr-uni-bochum.de/tanja`.

[9] T. Lange and M. Nöcker. Optimal implementation of hyperelliptic Koblitz curves over $\mathbb{F}_{2^n}$. in preparation.

[10] D. Lorenzini. *An Invitation to Arithmetic Geometry*, volume 9 of *Graduate studies in mathematics.* AMS, 1996.

[11] K. Matsuo, J. Chao, and S. Tsujii. Fast genus two hyperellptic curve cryptosysytems. Technical Report ISEC2001-23, IEICE, 2001. pages 89-96.

[12] A. Menezes, Y.-H. Wu, and R. Zuccherato. An Elementary Introduction to Hyperelliptic Curves. In *Algebraic Aspects of Cryptography* [5].

[13] Y. Miyamoto, H. Doi, K. Matsuo, J. Chao, and S. Tsuji. A fast addition algorithm of genus two hyperelliptic curve. In *Proc. of SCIS2002, IEICE Japan*, pages 497–502, 2002. in Japanese.

[14] D. Mumford. *Tata Lectures on Theta II.* Birkhäuser, 1984.

[15] M. Nöcker. *Data structures for parallel exponentiation.* PhD thesis, Universität Paderborn, 2001.

[16] J. Pelzl. Fast Hyperelliptic Curve Cryptosystems for Embedded Processors. Master's thesis, Ruhr-University of Bochum, 2002.

[17] J. H. Silverman. *The arithmetic of elliptic curves*, volume 106 of *Graduate texts in mathematics.* Springer, 1986.

[18] A.M. Spallek. *Kurven vom Geschlecht 2 und ihre Anwendung in Public-Key-Kryptosystemen.* PhD thesis, Universität Gesamthochschule Essen, 1994.

[19] H. Stichtenoth. *Algebraic Function Fields and Codes.* Springer, 1993.

[20] H. Sugizaki, K. Matsuo, J. Chao, and S. Tsujii. An Exntension of Harley algorithm addition algorithm for hyperelliptic curves over finite fields of characteritic two. Technical Report ISEC2002-9(2002-5), IEICE, 2002. pages 49-56.

[21] M. Takahashi. Improving Harley Algorithms for Jacobians of genus 2 Hyperelliptic Curves. In *Proc. of SCIS2002, IEICE Japan*, 2002. in Japanese.

Appendix

Here we provide the timings for the field operations using the respective libraries. The timings were obtained on a Pentium IV, 1.5 GHz under linux. We carried out 1 000 000 times each of the operations, times are given in μs.
As using GMP we carried out modular reductions only when it lead to an increase of speed, the table lists both the costs for multiplication and squaring of integers of the respective size without further reduction and including a modular reduction. Accordingly, in the implementation the times needed on average for these two operations is slightly larger if we disregard the reduction and smaller otherwise. Inversion is carried out as modular inversion in any case. Using NTL or working with binary fields we automatically have modular reduction. Finally we state the quotients of inversion and squaring over multiplications in all cases.

	prime/fieldsize	I	S	M	S_{red}	M_{red}	$\frac{I}{M}$	$\frac{S}{M}$	$\frac{I}{M_{red}}$	$\frac{S_{red}}{M_{red}}$
GMP 4.0.1	$p_1 \sim 161$ bits	16.22	3.04	5.93	3.99	6.84	2.73	0.51	2.37	0.58
	$p_2 \sim 183$ bits	17.23	2.31	4.42	3.18	5.37	3.89	0.52	3.20	0.59
	$p_3 \sim 81$ bits	7.64	1.52	2.89	2.14	3.49	2.64	0.52	2.18	0.61
	$p_4 \sim 91$ bits	8.39	1.63	3.01	2.14	3.55	2.78	0.54	2.36	0.60
NTL 5.2	$p_1 \sim 161$ bits	35.36	–	–	4.95	7.74	–	–	4.56	0.63
	$p_2 \sim 183$ bits	40.62	–	–	5.66	8.88	–	–	4.57	0.63
	$p_3 \sim 81$ bits	18.24	–	–	3.19	5.53	–	–	3.29	0.57
	$p_4 \sim 91$ bits	20.55	–	–	3.06	4.98	–	–	4.12	0.61
$\mathbb{F}_{2^n}$	$n = 163$	49.81	–	–	6.81	9.16	–	–	5.43	0.74
	$n = 191$	58.79	–	–	7.19	9.60	–	–	6.12	0.74
	$n = 83$	25.68	–	–	5.33	5.64	–	–	4.55	0.94
	$n = 97$	28.79	–	–	5.45	5.96	–	–	4.83	0.91