Formulae for Arithmetic on Genus 2 Hyperelliptic Curves

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

Abstract

The ideal class group of hyperelliptic curves can be used in cryptosystems based on the discrete logarithm problem. In this article we present explicit formulae to perform the group operations for genus 2 curves. The formulae are completely general but to achieve the lowest number of operations we treat odd and even characteristic separately. We present 3 different coordinate systems which are suitable for different environments, e. g. on a smart card we should avoid inversions while in software a limited number is acceptable. The presented formulae render genus two hyperelliptic curves very useful in practice.

The first system are affine coordinates where each group operation needs one inversion. Then we consider projective coordinates avoiding inversions on the cost of more multiplications and a further coordinate. Finally, we introduce a new system of coordinates and state algorithms showing that doublings are comparably cheap and no inversions are needed. A comparison between the systems concludes the paper.

Keywords: Public key cryptography, discrete logarithm, hyperelliptic curves, fast arithmetic, explicit formulae

1 Introduction

In the past years cryptosystems based on elliptic curves have received a lot of attention and nowadays they are ready for use in every day's applications and can be found on smart cards and cellular phones. Hyperelliptic curves are a generalization of elliptic curves and can likewise be used. Hyperelliptic curve cryptography is just beginning to receive more attention for use in practice to provide an alternative to the more established elliptic curve cryptography. The advantages of elliptic curve cryptography carry through. In this paper we deal with efficient arithmetic to achieve similar or even higher speed compared to elliptic curves. The presumably hard problem is the *discrete logarithm (DL)* problem which we state for arbitrary additively written groups: given a group element D and an element F of the cyclic group generated by D find the integer k such that $F = kD$. For curves of genus ≤ 3 no subexponential algorithms for solving the DL problem are known.

So far, the arithmetic for hyperelliptic curves was usually performed using Cantor's algorithm (see Cantor [6] and Koblitz [14] for a generalization to even characteristic). However, for fixed genus, one

can make the steps of the algorithm explicit and a more clever ordering results in faster formulae for addition and doubling of classes.
Here, we concentrate on genus 2 curves and provide explicit formulae for performing the arithmetic to close the performance gap between elliptic and hyperelliptic curves. This paper combines the contents of [19, 20, 21] by the same author. For several special cases we refer to the respective preprints. We give the algorithms and a theoretical comparison while two upcoming papers [1, 24] deal with implementation comparisons. First implementation results can be found in [19]. Formulae for genus 3 can be found in [12, 17, 32].
From a practical point of view it is worth studying larger genus curves as they can lead to faster computations of scalar multiples. Due to the smaller field sizes they might be better suited for some applications. It is interesting to notice that the explicit formulae offer a possibility to avoid simple side-channel attacks. In affine coordinates an addition needs 22 multiplications, 3 squarings and one inversion whereas a doubling needs two more squarings and the sequence of operations is very similar. Hence, the operations differ but it is not too hard to get equal characteristics by inserting some dummy field operations. This is far less than what is required for a complete group operation. Hence, it is easy to have countermeasures against simple power or timing analysis even in affine coordinates as observed in [23].

1.1 Situation for Elliptic Curves

For elliptic curves one can choose from several systems of coordinates (see Cohen, Miyaji, and Ono [7]) such that depending on the requirements of the computing environment one can use the optimal system for the respective purpose. Let $\mathbb{F}_q$ denote the finite field of $q = p^r$, p prime, elements. Put $\bar{\mathbb{F}}_q$ the algebraic closure of $\mathbb{F}_q$. Affine points are tuples $(x, y) \in \bar{\mathbb{F}}_q^2$, satisfying $y^2 + (a_1x + a_3)y = x^3 + a_2x^2 + a_4x + a_6, a_i \in \mathbb{F}_q$. The homogenized equations lead to projective points (X, Y, Z) with the correspondence $x = X/Z, y = Y/Z$. The addition formulae for points in projective coordinates avoid inversions. The idea to achieve formulae faster than the projective and still without using inversions is to allow *weighted projective* coordinates. In the elliptic setting (X, Y, Z) corresponds to $(x, y) = (X/Z^2, Y/Z^3)$. These coordinates are called *Jacobian* coordinates; in this system, additions are slightly more expensive while doublings get considerably cheaper than in projective coordinates. Some computations done in the process of adding or doubling can provide useful in the following operation. Thus, if the space is not too restricted, one can include them in the set of coordinates. The coordinates (X, Y, Z, Z^2, Z^3) are called *Chudnovsky Jacobian*; they are faster than projective coordinates with respect to both operations and allow faster additions and slower doublings compared to ordinary Jacobian coordinates. To obtain faster doublings it is useful to work with Cohen's modified Jacobian coordinates (X, Y, Z, aZ^4), where the elliptic curve is given by $y^2 = x^3 + ax + b$. This is very useful if one can store several precomputations and then use a binary (signed) window method to compute the scalar multiple as then there are much more doublings than additions. Furthermore, Cohen, Miyaji, and Ono investigate mixed coordinates, i. e. depending on the costs of inversions relative to multiplications they propose different sets of coordinates for the precomputations, the additions and the general doublings.

1.2 Situation for Genus 2 Curves

In this paper we try to mimic the approach to elliptic curves to generalize it to hyperelliptic curves. For genus 2 curves explicit formulae were first considered by Spallek [35] and by Krieger [16]. The first practical formulae were obtained by Harley [13], which were generalized to even characteristic by Lange [18]; an improvement of the former paper can be found in Matsuo, Chao, and Tsujii

[27]. A significant improvement was obtained independently by Takahashi [38] and Miyamoto, Doi, Matsuo, Chao, and Tsujii [30]; this was generalized to even characteristic in [37] and independently in [19]. The second reference allows more general curves and manages to trade more multiplications for squarings which is desirable for characteristic two implementations. All these formulae involve (at least) 1 inversion per addition or doubling respectively. In some environments inversions are extremely time or space critical. An example are smart cards, as usually multiplications are optimized there, whereas divisions are very slow – even with coprocessors .
So far there is only one article on other than affine coordinate systems for genus two curves by Miyamoto, Doi, Matsuo, Chao, and Tsujii [30]. In [20] we take a similar approach obtaining better running times and also allowing even characteristic for the finite field. To have a short term we call the usual representation 'affine' and the new one 'projective' due to the resemblance with the elliptic case. A milestone on the road towards hyperelliptic curve cryptography in real life is Kim Nguyen's implementation on a FameXE of Lange's projective formulae reported at *ECC 2002 - Workshop on elliptic curve cryptography, Essen.* This is the first implementation for an embedded system using inversion-free coordinates which shows hyperelliptic curves to be competitive with elliptic curves.
We introduce a further set of coordinates, which we call *new coordinates.* Like the projective coordinates they allow to avoid inversions in the group operations but the doublings are faster. For both these sets we only treat the main cases as the other generalize easily. Because they are needed with very low probability they could even be skipped for an actual implementation. This is possible as no division by zero can occur and a simple check at the end of the computation is enough to guarantee the correctness of the result.
For the new coordinates we need to treat odd and even characteristic separately as they need different ways of optimization. The consideration of even characteristic is done in the Appendix. We conclude each study by a comparison of the arithmetic in the different systems and also treat mixed coordinates, i.e. allow different sets of coordinates for the precomputations, main doublings and additions to achieve the lowest complexity. Certainly the actual choice for an implementation heavily depends on the cost of inversions relative to multiplications and on the number of (online) precomputations one is willing to store.

Since the submission of this paper some new results were obtained which we state here. For curves over prime fields Avanzi [1] did a complete comparison of the speed of scalar multiplication. He considers curves of genus one, two and three and compares the different coordinate systems available using different windowing methods. In [29] Mishra and Sakar obtained a parallel version of the arithmetic, Avanzi, Lange and Mishra [2, 23] consider resistance against side channel attacks and Duquesne and Lange [8, 22] generalized Montgomery arithmetic to genus two curves over fields of odd characteristic. The explicit formulae have now also been implemented on FPGAs [9, 3, 40, 39]. In even characteristic the doubling algorithm was improved for curves of special type, i. e. curves having at some coefficients equal to zero, in [5, 33, 25].

The remainder of this paper is organized as follows: 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. To make the steps explicit we need not consider all different inputs. For the usual set of coordinates 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. Then we present the respective algorithms for projective and new coordinates.

Acknowledgments

First of all I would like to thank Gerhard Frey and Henri Cohen for encouragement and interest. Furthermore, thanks go to Roberto Avanzi for proof reading and to Bertrand Byramjee, Sylvain Duquesne, Guilhem Castagnos, and Marc Stevens for spotting some typos in the formulae of the previous papers [19, 20, 21]. I acknowledge the help of MAGMA to check the consistency of the formulae.

2 Mathematical Background on Elliptic and Hyperelliptic Curves

In this section we briefly sketch what is needed in the remainder of this paper. The interested reader is referred to Menezes, Wu, and Zuccherato [28], Koblitz [15], Lorenzini [26], Stichtenoth [36], and Frey and Lange [10] 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*

$$C : y^2 + h(x)y = f(x), \tag{1}$$

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 our purposes it is enough to consider a point as an ordered pair $(a,b) \in \bar{\mathbb{F}}_q^2$ which satisfies $b^2 + h(a)b = f(a)$. 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 $C(\bar{\mathbb{F}}_q)$ is an element of the free abelian group over the points of $C(\bar{\mathbb{F}}_q)$, e.g. $D = \sum_{P \in C(\bar{\mathbb{F}}_q)} n_P P$ with $n_P \in \mathbb{Z}$ and $n_P = 0$ for almost all points P. The degree of D is defined as $\deg(D) = \sum_{P \in C(\bar{\mathbb{F}}_q)} n_P$. A divisor D is defined over $\mathbb{F}_q$ if $\sigma(D) = D$ for all $\sigma \in \mathrm{Gal}(\bar{\mathbb{F}}_q/\mathbb{F}_q)$. 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 $F_a = (x-a)$ leads to a divisor $\mathrm{div}(F_a) = P_a + \iota P_a - 2\infty$, where $P_a = (a,b) \in C(\bar{\mathbb{F}}_q)$. 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 has $\mathbb{F}_{q^l}$ as minimal field of definition 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 we consider the ideal class group, i.e. the ideals modulo the principal ideals. As the curve has only a single point at infinity the ideal class group and the divisor class group are isomorphic. Furthermore, they are isomorphic to the $\mathbb{F}_q$-rational points of the Jacobian of the curve, a g-dimensional abelian variety. In Mumford [31][page 3.17] the following representation is introduced which makes explicit the isomorphism between the ideal class group and divisor class group:

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* $\left(\frac{d}{dx}\right)^j \left[v(x)^2 + v(x)h(x) - f(x)\right]_{|x=a_i} = 0,\ 0 \leq j \leq n_i - 1$*.*

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 [6] (for odd characteristic only) and Koblitz [14] one has an efficient algorithm to perform these operations, which uses only polynomial arithmetic over the finite field in which the ideal classes are defined. Even though the composition in the ideal class group is multiplication, we write the group additively, as it has become common to speak about addition and doubling (as opposed to multiplication and squaring) of classes like for points on elliptic curves.

Algorithm 3.1 (Composition)
`INPUT:` $D_1 = [u_1, v_1], D_2 = [u_2, v_2], \quad C : y^2 + h(x)y = f(x)$.
`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* $s_1 = c_1 e_1,\ s_2 = c_1 e_2,\ s_3 = c_2$*;*

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 $\mathrm{Cl}(C/\mathbb{F}_q)$ 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

From now on we restrict our attention to curves of genus two. To derive explicit formulae, we first give the case study of [18] investigating what can be the input of the composition algorithm and proceed in considering these different cases. We determine the exact number of operations needed to perform addition and doubling in the most frequent 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.
We fix the notation to refer to the coefficient of x^i in a polynomial $l(x)$ as l_i.

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_i is monic,
2. $\deg v_i < \deg u_i \le 2$,
3. $u_i | v_i^2 + v_i h - f$.

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

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

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

Otherwise the composition leads to $u = u_1u_2$ and
$v = ((v_2 - v_1)x + v_2u_{10} - v_1u_{20})/(u_{10} - u_{20})$.
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 4.3.2.

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 $\mathrm{res}(h + 2v_1, u_1) \neq 0$ then we are in the usual case where both points are not equal to their opposite. This will be considered in Subsection 4.3.3.

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 $\mathrm{res}(u_1, u_2) \neq 0$ 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 4.3.1.

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 nonzero resultant and thus we save on average.

4.2 Isomorphic Transformations

From the case study it is already obvious that the formulae depend on the equation of the curve. To ease the exposition we now introduce some transformations leading to isomorphic curves which might have an easier equation.
In odd characteristic upon application of the transformation $y \to y - h/2$ one can always assume $h(x) = 0$. To satisfy the last condition of the definition it then suffices to have f squarefree.
For $p \neq 5$ the substitution $x \to x - f_4/5$ provides that the coefficient of x^4 in f can be taken equal to zero.
In even characteristic h must be non-zero. For genus two we even have to guarantee that h is non-constant as otherwise the resulting curve is supersingular (see Galbraith [11]) and thus the DL problem is weaker than on an arbitrary curve. However, we note that recently supersingular curves found applications in identity based cryptosystems (see e. g. [4, 11, 34]). As the following formulae will depend heavily on the non-zero coefficients of the curve we consider transformations: Let first $\deg h = 2$. Then replacing

$$y \to h_2^5 y + f_3 h_2 x + \frac{f_3(f_3 + h_1 h_2 + f_4 h_2^2) + f_2 h_2^2}{h_2^3}, x \to h_2^2 x + f_4$$

and dividing the equation by h_2^{10} leads to $h_2 = 1$, and $f_4 = f_3 = f_2 = 0$.
If $\deg h = 1$ we obtain $f_3 = 0$ via $y \to y + f_3/h_1 x^2$. If additionally there exists a b such that $f_3 h_0 + b^2 h_1 + b h_1^2 = f_2 h_1$ has a solution one can even achieve $f_2 = 0$ by $y \to y + f_3/h_1 x^2 + bx$. Finally one has to ensure $f_4 = 0$ by $x \to x + f_4$ with the new f_4, which does not touch f_3 and f_2 as $p = 2$. Furthermore, with high probability one can achieve that h_1 is small by the transformation $x \to a^2 x, y \to a^5 y$, where $a \in \mathbb{F}_q$ is chosen such that h_1/a^3 is small.

Remark 4.1 *After the submission of this paper some more research revealed that for* $\deg h = 1$ *other changes of the curve equation lead to faster formulae, see [25].*

4.3 Addition and Doubling

We now present in detail the algorithms for the cases left out above. These are the most common cases. For the complexity estimates we always assume $h_2 \in \{0, 1\}$ and $f_4 = 0$. If the curve is not brought to this form some computations should be performed differently (e. g. $s_0(s_0 + h_2)$ instead of $s_0^2 + s_0 h_2$). We would like to stress that the formulae remain correct for other values of h_2 and f_4, only the operation count changes. Depending on the equation of the curve and the characteristic some further transformations can save operations. Note, however, that these improvements are rather immediate and therefore we do not list them separately. Finally, we mention that we only count multiplications, squarings, and inversions as additions and subtractions are comparably cheap.

4.3.1 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 $u = u_1 u_2$ 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 the Chinese Remainder Theorem:

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

Then we compute the resulting first polynomial u' by making $(f-vh-v^2)/(u_1u_2)$ 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.

$$\begin{array}{rcl} k &=& (f - v_2h - v_2^2)/u_2 \\ s &\equiv& (v_1 - v_2)/u_2 \bmod u_1 \\ l &=& su_2 \\ u &=& (k - s(l + h + 2v_2))/u_1 \\ u' &=& u \text{ made monic} \\ v' &\equiv& -h - (l + v_2) \bmod u' \end{array}$$

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 $v \equiv su_2 + v_2 \equiv ((v_1 - v_2)/u_2)u_2 + v_2 \equiv v_1 \bmod u_1$. Now we check that $u = (f - vh - v^2)/(u_1u_2)$ by expanding out

$$u_1u_2u = u_2(k - s(l + h + 2v_2)) = f - v_2h - v_2^2 - l(l + h) - 2lv_2 = f - vh - v^2.$$

In the course of computing we do not need all coefficients of the polynomials defined above. As $f = x^5 + \sum_{i=0}^{4} f_ix^i$ 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. This leads us to consider a further subcase, namely $\deg s = 1$.
In the following table we list the intermediate steps together with the number of multiplications (M), squarings (S) and inversions (I) needed. The names of the intermediate variables refer to the above computations. A dash $'$ indicates changes relative to there. For an actual implementation less variables are needed, however, we favored readability by humans in this exposition.

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 $\text{res}(\tilde{u}_2, u_1)$ are known. However, we include the costs in the table, as we use these expressions to compute $1/\tilde{u}_2 \bmod u_1$.

The following table presents the complete addition formula. We apply the trick introduced by Takahashi [38] to use a monic s''. Note that our algorithm needs the same number of operations (assuming M=S) as his but manages to trade one more multiplication for a squaring which might be advantageous for implementations.

For even characteristic the independent work [37] needs the same number of operations but considers only the case of $\deg h = 2$. Furthermore, in even characteristic squarings are much cheaper than multiplications and therefore our algorithm is faster. If $h_1 = 1$ our algorithm saves one multiplication in Step 6.

Addition, $\deg u_1 = \deg u_2 = 2$		
Input	$[u_1, v_1], [u_2, v_2], u_i = x^2 + u_{i1}x + u_{i0}, v_i = v_{i1}x + v_{i0}$	
Output	$[u', v'] = [u_1, v_1] + [u_2, v_2]$	
Step	Expression	Operations
1	compute resultant r of u_1, u_2: $z_1 = u_{11} - u_{21}$, $z_2 = u_{20} - u_{10}$, $z_3 = u_{11}z_1 + z_2$; $r = z_2 z_3 + z_1^2 u_{10}$;	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 $s' = rs \equiv (v_1 - v_2) inv \bmod u_1$: $w_0 = v_{10} - v_{20}$, $w_1 = v_{11} - v_{21}$, $w_2 = inv_0 w_0$, $w_3 = inv_1 w_1$; $s'_1 = (inv_0 + inv_1)(w_0 + w_1) - w_2 - w_3(1 + u_{11})$, $s'_0 = w_2 - u_{10}w_3$; if $s'_1 = 0$ see below	5M
4	compute $s'' = x + s_0/s_1 = x + s'_0/s'_1$ and s_1: $w_1 = (rs'_1)^{-1} (= 1/r^2 s_1)$, $w_2 = rw_1 (= 1/s'_1)$, $w_3 = s'^2_1 w_1 (= s_1)$; $w_4 = rw_2 (= 1/s_1)$, $w_5 = w_4^2$, $s''_0 = s'_0 w_2$;	I, 2S, 5M
5	compute $l' = s'' u_2 = x^3 + l'_2 x^2 + l'_1 x + l'_0$: $l'_2 = u_{21} + s''_0$, $l'_1 = u_{21}s''_0 + u_{20}$, $l'_0 = u_{20}s''_0$	2M
6	compute $u' = (s(l + h + 2v_2) - k)/u_1 = x^2 + u'_1 x + u'_0$: $u'_0 = (s''_0 - u_{11})(s''_0 - z_1 + h_2 w_4) - u_{10} + l'_1 + (h_1 + 2v_{21})w_4 + (2u_{21} + z_1 - f_4)w_5$; $u'_1 = 2s''_0 - z_1 + h_2 w_4 - w_5$;	3M
7	compute $v' \equiv -h - (l + v_2) \bmod u' = v'_1 x + v'_0$: $w_1 = l'_2 - u'_1$, $w_2 = u'_1 w_1 + u'_0 - l'_1$, $v'_1 = w_2 w_3 - v_{21} - h_1 + h_2 u'_1$; $w_2 = u'_0 w_1 - l'_0$, $v'_0 = w_2 w_3 - v_{20} - h_0 + h_2 u'_0$;	4M
total		I, 3S, 22M
Special case $s = s_0$		
$4'$	compute s: $inv = 1/r$, $s_0 = s'_0 inv$;	I, M
$5'$	compute $u' = (k - s(l + h + 2v_2))/u_1 = x + u'_0$: $u'_0 = f_4 - u_{21} - u_{11} - s_0^2 - s_0 h_2$;	S
$6'$	compute $v' \equiv -h - (l + v_2) \bmod u' = v'_0$: $w_1 = s_0(u_{21} + u'_0) + h_1 + v_{21} + h_2 u'_0$, $w_2 = s_0 + v_{20} + h_0$; $v'_0 = u'_0 w_1 - w_2$;	2M
total		I, 2S, 11M

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

We now treat the case of Section 4.1 in which 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. The next table shows that this case is much cheaper than the general one, however it is not too likely to happen like all special cases.

Addition, $\deg u_1 = 1, \deg u_2 = 2$		
Input	$[u_1, v_1], [u_2, v_2], u_1 = x + u_{10}, u_2 = x^2 + u_{21}x + u_{20}, v_1 = v_{10}, v_2 = v_{21}x + v_{20}$	
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 $l = su_2 = s_0x^2 + l_1x + l_0$: $l_1 = s_0u_{21}$, $l_0 = s_0u_{20}$;	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 $u' = (k - s(l + h + 2v_2))/u_1 = x^2 + u'_1x + u'_0$: $u'_1 = k_2 - s_0^2 - s_0h_2 - u_{10}$; $u'_0 = k_1 - s_0(l_1 + h_1 + 2v_{21}) - u_{10}u'_1$;	S, 2M
7	compute $v' \equiv -h - (l + v_2) \bmod u' = v'_1x + v'_0$: $v'_1 = (h_2 + s_0)u'_1 - (h_1 + l_1 + v_{21})$; $v'_0 = (h_2 + s_0)u'_0 - (h_0 + l_0 + v_{20})$;	2M
total		I, S, 10 M

4.3.3 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 opposites. Put $u = x^2 + u_1x + u_0,\ v = v_1x + v_0$. Composing $[u, v]$ with itself should result in a class $[u_{\text{new}}, v_{\text{new}}]$, where

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

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

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

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

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

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.

Unlike in the addition case we now need the exact polynomial k to compute D. For the doublings it is necessary to separately count the operations for odd and even characteristic. The formulae on the left are most general but for an actual implementation they should be modified according to the remarks below. For odd characteristic we assume $h = 0$.

Doubling, $\deg u = 2$			
Input	$[u,v], u = x^2 + u_1x + u_0, v = v_1x + v_0$		
Output	$[u',v'] = 2[u,v]$		
Step	Expression	odd	even
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 $r = \text{res}(\tilde{v}, u)$: $w_0 = v_1^2$, $w_1 = u_1^2$, $w_2 = \tilde{v}_1^2$, $w_3 = u_1\tilde{v}_1$, $r = u_0w_2 + \tilde{v}_0(\tilde{v}_0 - w_3)$;	2S, 3M $(w_2 = 4w_0)$	2S, 3M (see below)
3	compute almost inverse $inv' = invr$: $inv'_1 = -\tilde{v}_1$, $inv'_0 = \tilde{v}_0 - w_3$;		
4	compute $k' = (f - hv - v^2)/u \bmod u = k'_1x + k'_0$: $w_3 = f_3 + w_1$, $w_4 = 2u_0$, $k'_1 = 2(w_1 - f_4u_1) + w_3 - w_4 - h_2v_1$; $k'_0 = u_1(2w_4 - w_3 + f_4u_1 + h_2v_1) + f_2 - w_0 - 2f_4u_0 - h_1v_1 - h_2v_0$;	1M	2M (see below)
5	compute $s' = k'inv' \bmod u$: $w_0 = k'_0inv'_0$, $w_1 = k'_1inv'_1$; $s'_1 = (inv'_0 + inv'_1)(k'_0 + k'_1) - w_0 - w_1(1 + u_1)$, $s'_0 = w_0 - u_0w_1$; If $s_1 = 0$ see below	5M	5M
6	compute $s'' = x + s_0/s_1$ and s_1: $w_1 = 1/(rs'_1)(= 1/r^2s_1)$, $w_2 = rw_1(= 1/s'_1)$, $w_3 = s'^2_1w_1(= s_1)$; $w_4 = rw_2(= 1/s_1)$, $w_5 = w_4^2$, $s''_0 = s'_0w_2$;	I, 2S, 5M	I, 2S, 5M
7	compute $l' = s''u = x^3 + l'_2x^2 + l'_1x + l'_0$: $l'_2 = u_1 + s''_0$, $l'_1 = u_1s''_0 + u_0$, $l'_0 = u_0s''_0$;	2M	2M
8	compute $u' = s^2 + (h+2v)s/u + (v^2 + hv - f)/u^2$: $u'_0 = s''^2_0 + w_4(h_2(s''_0 - u_1) + 2v_1 + h_1) + w_5(2u_1 - f_4)$; $u'_1 = 2s''_0 + h_2w_4 - w_5$;	S, 2M	S, M
9	compute $v' \equiv -h - (l+v) \bmod u' = v'_1x + v'_0$: $w_1 = l'_2 - u'_1$, $w_2 = u'_1w_1 + u'_0 - l'_1$, $v'_1 = w_2w_3 - v_1 - h_1 + h_2u'_1$; $w_2 = u'_0w_1 - l'_0$, $v'_0 = w_2w_3 - v_0 - h_0 + h_2u'_0$;	4M	4M
total		I, 5S, 22 M	I, 5S, 22 M
Special case $s = s_0$			
$6'$	compute s and precomputations: $w_1 = 1/r$, $s_0 = s'_0w_1$, $w_2 = u_0s_0 + v_0 + h_0$;	I,2M	I,2M
$7'$	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	S
$8'$	compute $v' \equiv -h - (su + v) \bmod u'$: $w_1 = s_0(u_1 - u'_0) - h_2^2u'_0 + v_1 + h_1$, $v'_0 = u'_0w_1 - w_2$;	2M	2M
total		I, 3S, 13M	I, 3S, 14M

Remarks:

1. Compared to [38, 37] we have the same number of operations (assuming M=S) but our formulae trade 4 more multiplications for squarings which is a significant speed-up in even characteristic. Furthermore, we consider a more general defining equation.

2. In even characteristic we can skip the computation of w_0 in Step 2 and then compute $k_0' = u_1(w_3 + f_4 u_1 + h_2 v_1) + f_2 + v_1(v_1 + h_1) + h_2 v_0$. In u_0' we get $f_4 w_5$ for free.

3. If $h_2 = 0$ then $w_2 = h_1^2$. This squaring can be precomputed as it is fixed for the curve.

4. If one is willing to fix the curve and allow some special choices it is possible to reduce the number of operations significantly [25]. For binary Koblitz curves see also [24].

5 Projective Coordinates

So far, in the process of computation one inversion is required for each addition or doubling. Now, instead of following this line, we introduce a further coordinate called Z like for elliptic curves and let the quintuple $[U_1, U_0, V_1, V_0, Z]$ stand for $[x^2 + U_1/Z\, x + U_0/Z, V_1/Z\, x + V_0/Z]$. If the output of a scalar multiplication should be in the usual affine representation we need one inversion and four multiplications at the end of the computations. We now proceed in investigating the arithmetic in the main cases.
This idea was first proposed for genus two curves in [30] and then largely improved and generalized by the author in [20].

5.1 Addition

Here we consider the case that we add two classes both in projective representation. This is needed if the whole system avoids inversion and classes are transmitted using the quintuple representation, or if during the verification of a signature intermediate results should be added, or when using precomputations given in projective representation. Obviously this algorithm also works for affine inputs if one writes $[u_1, v_1]$ as $[u_{11}, u_{10}, v_{11}, v_{10}, 1]$. The number of operations in the following table refers to odd characteristic or to $h_i \in \{0, 1\}$ respectively. For even characteristic see the remark below. Numbers in brackets refer to the case, that the first input is affine, i. e. has $Z_1 = 1$. A more careful implementation of this case allows to save some further multiplications (see [20]), namely then one additions needs 3S, 40M in odd characteristic.

Addition		
Input	$[U_{11}, U_{10}, V_{11}, V_{10}, Z_1], [U_{21}, U_{20}, V_{21}, V_{20}, Z_2]$	
Output	$[U'_1, U'_0, V'_1, V'_0, Z'] = [U_{11}, U_{10}, V_{11}, V_{10}, Z_1] + [U_{21}, U_{20}, V_{21}, V_{20}, Z_2]$	
Step	Expression	Operations
1	precomputation: $Z = Z_1Z_2$, $\tilde{U}_{21} = Z_1U_{21}$, $\tilde{U}_{20} = Z_1U_{20}$, $\tilde{V}_{21} = Z_1V_{21}$, $\tilde{V}_{20} = Z_1V_{20}$;	5M (–)
2	compute resultant r of U_1, U_2: $z_1 = U_{11}Z_2 - \tilde{U}_{21}$, $z_2 = \tilde{U}_{20} - U_{10}Z_2$, $z_3 = U_{11}z_1 + z_2Z_1$; $r = z_2z_3 + z_1^2U_{10}$;	1S, 6M (1S, 5M)
3	compute almost inverse of u_2 modulo u_1 : $inv_1 = z_1$, $inv_0 = z_3$;	
4	compute s: $w_0 = V_{10}Z_2 - \tilde{V}_{20}$, $w_1 = V_{11}Z_2 - \tilde{V}_{21}$, $w_2 = inv_0w_0$, $w_3 = inv_1w_1$; $s_1 = (inv_0 + Z_1inv_1)(w_0 + w_1) - w_2 - w_3(Z_1 + U_{11})$; $s_0 = w_2 - U_{10}w_3$; If $s_1 = 0$ different case	8M (7M)
5	precomputations: $R = Zr$, $s_0 = s_0Z$, $s_3 = s_1Z$, $\tilde{R} = Rs_3$, $t = s_1(z_1 + \tilde{U}_{21})$; $S_3 = s_3^2$, $S = s_0s_1$, $\tilde{S} = s_3s_1$, $\tilde{\tilde{S}} = s_0s_3$, $\tilde{\tilde{R}} = \tilde{R}\tilde{S}$, $\tilde{h}_1 = h_1Z$;	1S, 9M
6	compute l: $l_2 = \tilde{S}\tilde{U}_{21}$, $l_0 = S\tilde{U}_{20}$, $l_1 = (\tilde{S} + S)(\tilde{U}_{21} + \tilde{U}_{20}) - l_2 - l_0$; $l_2 = l_2 + \tilde{\tilde{S}}$;	3M
7	compute U': $U'_0 = s_0^2 + s_1z_1(t - 2s_0) + z_2\tilde{S} +$ $R(h_2(s_0 - t) + s_1(\tilde{h}_1 + 2\tilde{V}_{21}) + r(z_1 + 2\tilde{U}_{21} - f_4Z))$; $U'_1 = 2\tilde{\tilde{S}} - \tilde{S}z_1 + h_2\tilde{R} - R^2$;	2S, 7M (see below)
8	precomputations: $l_2 = l_2 - U'_1$, $w_0 = U'_0l_2 - S_3l_0$, $w_1 = U'_1l_2 + S_3(U'_0 - l_1)$;	4M
9	adjust: $Z' = \tilde{R}S_3$, $U'_1 = \tilde{R}U'_1$, $U'_0 = \tilde{R}U'_0$;	3M
10	compute V': $V'_0 = w_0 + h_2U'_0 - \tilde{\tilde{R}}\tilde{V}_{20} - h_0Z'$; $V'_1 = w_1 + h_2U'_1 - \tilde{\tilde{R}}(\tilde{V}_{21} + \tilde{h}_1)$;	2M
total		4S, 47M (4S, 40M)

Remark: In characteristic 2 we need 2 more multiplications leading to 4S, 49M (4S, 39M). Note, that we present only the most general formulae – consideration of the special choices will improve the efficiency.

5.2 Doubling

For the doubling algorithm the input is almost always in projective representation. Like before we list the number of operations for odd and even characteristic separately. In the odd case $h = 0$ otherwise $h_2 \in \{0, 1\}, f_3 = 0$. Multiplications by f_4 are not counted in any case. The formulae might be performed differently for even characteristic (see remarks below).

Doubling			
Input Output	$[U_1, U_0, V_1, V_0, Z]$ $[U'_1, U'_0, V'_1, V'_0, Z'] = 2[U_1, U_0, V_1, V_0, Z]$		
Step	Expression	odd	even
1	compute resultant and precomputations: $\tilde{h}_1 = h_1 Z$, $\tilde{h}_0 = h_0 Z$, $Z_2 = Z^2$, $\tilde{V}_1 = \tilde{h}_1 + 2V_1 - h_2 U_1$; $\tilde{V}_0 = \tilde{h}_0 + 2V_0 - h_2 U_0$, $w_0 = V_1^2$, $w_1 = U_1^2$, $w_2 = \tilde{V}_1^2$; $w_3 = \tilde{V}_0 Z - U_1 \tilde{V}_1$, $r = \tilde{V}_0 w_3 + w_2 U_0$;	3S, 4M $(w_2 = 4w_0)$	4S, 6M
2	compute almost inverse: $inv_1 = -\tilde{V}_1$, $inv_0 = w_3$;		
3	compute k: $w_3 = f_3 Z_2 + w_1$, $w_4 = 2U_0$; $k_1 = 2w_1 + w_3 - Z(w_4 + 2f_4 U_1 + h_2 V_1)$; $k_0 = U_1(Z(2w_4 + f_4 U_1 + h_2 V_1) - w_3)$ $+Z(Z(f_2 Z - V_1 h_1 - V_0 h_2 - 2f_4 U_0) - w_0)$	5M (see below)	5M (see below)
4	compute $s = kinv \bmod u$: $w_0 = k_0 inv_0$, $w_1 = k_1 inv_1$; $s_3 = (inv_0 + inv_1)(k_0 + k_1) - w_0 - (1 + U_1) w_1$ $s_1 = s_3 Z$, $s_0 = w_0 - Z U_0 w_1$; If $s_1 = 0$ different case	7M	7M
5	precomputations: $R = Z_2 r$, $\tilde{R} = R s_1$, $S_1 = s_1^2$, $S_0 = s_0^2$, $t = h_2 s_0$; $s_1 = s_1 s_3$, $s_0 = s_0 s_3$, $S = s_0 Z$, $\tilde{\tilde{R}} = \tilde{R} s_1$;	2S, 6M	2S, 6M
6	compute l: $l_2 = U_1 s_1$, $l_0 = U_0 s_0$, $l_1 = (s_1 + s_0)(U_1 + U_0) - l_2 - l_0$;	3M	3M
7	compute U': $U'_0 = S_0 + R(s_3(2V_1 - h_2 U_1 + \tilde{h}_1) + t + Zr(2U_1 - f_4 Z))$; $U'_1 = 2S + h_2 \tilde{R} - R^2$;	1S, 4M	1S, 2M
8	precomputations: $l_2 = l_2 + S - U'_1$, $w_0 = U'_0 l_2 - S_1 l_0$, $w_1 = U'_1 l_2 + S_1(U'_0 - l_1)$;	4M	4M
9	adjust: $Z' = S_1 \tilde{R}$, $U'_1 = \tilde{R} U'_1$, $U'_0 = \tilde{R} U'_0$;	3M	3M
10	compute V': $V'_0 = w_0 + h_2 U'_0 - \tilde{\tilde{R}}(V_0 - \tilde{h}_0)$; $V'_1 = w_1 + h_2 U'_1 - \tilde{\tilde{R}}(V_1 - \tilde{h}_1)$;	2M	2M
total		6S, 38M	7S, $\leq$38M

Remarks:

1. First of all one notices that doublings are much faster than general additions; this is especially interesting as doubling occur much more frequently than additions in any algorithm to compute scalar multiples, most striking in windowing methods.

2. In odd characteristic and for $f_4 = 0$, Step 3 is computed as $k_0 = U_1(2Zw_4 - w_3) + Z(f_2 Z_2 - w_0)$ as Zw_4 is already obtained during the computation of k_1.

3. For $p = 2$ the following alternatives allow to obtain the stated complexity (we mention those

which are not completely obvious) leading to ≤ 38M.

(a) In Step 3 as $f_3 = 0$ one needs not compute $Z_2 f_3$. We get $k_1 = w_3 + h_2 Z V_1$. If $\deg h = 2$ we have $f_2 = 0$ and get $k_0 = U_1 k_1 + Z_2(h_2 V_0 + h_1 V_1) + Z w_0$, otherwise $k_0 = U_1 k_1 + Z(f_2 Z_2 + V_1 \tilde{h}_1 + w_0)$.

(b) Furthermore, U_0' can be computed with only two multiplications as $U_0' = S_0 + R(s_3(h_2 U_1 + \tilde{h}_1) + t)$.

(c) If $h_2 = 0$ and the number of multiplications can even be reduced to 36M: In Step 3, the multiplication to obtain k_1 vanishes and $U_0' = S_0 + h_1 \tilde{R}$ and $U_1' = R^2$ using only 1S, 1M.

4. Using binary Koblitz curves one has $h, f \in \mathbb{F}_2[x]$, thus multiplications by the coefficients need not be counted at all.
   Combining with what was said above, the number of operations for a doubling in this case is 6S, 34M for $h_2 = 1$ and with $h_2 = 0$ even 6S, 31M.

It is interesting to note that the value of the additional coordinate Z' was not kept minimal. One could have avoided (at least) a factor of $Z_1^2 Z_2$ of the denominator in the addition and of Z in the doubling. However, as we tried to minimize the number of operations, we allowed the larger value of $Z^4 r s_1^3$ in both cases as this proved to be more efficient. Besides one sees that U_1' and U_0' have to be adjusted to have the same (larger) denominator Z' as V_1', V_0'.

6 New Coordinates

As we just noticed the denominator of the V_i's differs from that of the U_i's. This leads us to consider a further set of coordinates. Here, we suggest to let $[U_1, U_0, V_1, V_0, Z_1, Z_2]$ correspond to the affine point $[x^2 + U_1/Z_1^2\, x + U_0/Z_1^2, V_1/(Z_1^3 Z_2)\, x + V_0/(Z_1^3 Z_2)]$. This means that now a point corresponds to a sextuple, thus one needs one more entry than for projective coordinates. Such coordinate systems are called *weighted coordinates*. This is the first proposal of such a system for hyperelliptic curves. For elliptic curves this corresponds to Jacobian coordinates. Compared to the case of elliptic curves, for equal security the entries for $g = 2$ are of only half the size, thus the space requirements are similar.
Here, we treat only the case of odd characteristic. The considerations for $p = 2$ can be found in the appendix. As usual we assume $h(x) \equiv 0$ and $f_4 = 0$; this time we do not include these coefficients in the formulae. To increase the performance we enlarge the set of coordinates to $[U_1, U_0, V_1, V_0, Z_1, Z_2, z_1, z_2]$, where $z_1 = Z_1^2, z_2 = Z_2^2$. These additional entries are computed anyway during each addition or doubling and keeping them saves in the following operation. Both addition and doubling profit from z_1 whereas z_2 is only used for the doublings. As additions occur asymptotically at most half as often as doublings we do not include $Z_1 Z_2$ which would be useful for additions, because it is not useful in doublings and is not automatically computed.
If space is more restricted such that we can only use the sextuple of coordinates, we need two extra squarings in the first step of the doubling or addition. We first list the algorithm to add two points in these coordinates and then consider doubling. Finally, we put together these algorithms to compute scalar multiples, also paying attention to other systems of coordinates.

These *new coordinates* were first proposed by the author; they generalize the concepts of Jacobian, Chudnovsky Jacobian and modified Jacobian coordinates from elliptic to hyperelliptic curves. The respective counterparts can be seen if one varies the additional coordinates – the original Jacobian coordinates correspond to allowing only Z_1, Z_2. In the following we state the most efficient algorithms, the additional coordinates become more and more useful with the increase of the window size in scalar multiplications and hence they form the counterpart to modified Jacobian coordinates. The topic of which coordinates to choose in which system is treated further in Section 7.

6.1 Addition

If one computes a scalar multiple of a point given in affine coordinates and has the intermediate results not-normalized, then in the addition the intermediate result enters in the new weighted coordinates whereas the other class enters always as $[U_{11}, U_{10}, V_{11}, V_{10}, 1, 1, 1, 1]$. The number in brackets refer to this (cheaper) case. For an algorithm devoted to this case see [21]. It needs only 5S and 36M. To save space we skip the comments and do not list the in- and output. They are equal to those in the previous tables.

Addition, odd characteristic		
Step	Expression	Operations
1	$z_{13} = Z_{11}Z_{12}$, $z_{23} = Z_{21}Z_{22}$, $z_{12} = z_{11}z_{13}$, $z_{22} = z_{21}z_{23}$; $\tilde{U}_{21} = U_{21}z_{11}$, $\tilde{U}_{20} = U_{20}z_{11}$, $\tilde{V}_{21} = V_{21}z_{12}$, $\tilde{V}_{20} = V_{20}z_{12}$;	8M (2M)
2	$y_1 = U_{11}z_{21} - \tilde{U}_{21}$, $y_2 = \tilde{U}_{20} - U_{10}z_{21}$, $y_3 = U_{11}y_1 + y_2z_{11}$; $r = y_2y_3 + y_1^2U_{10}$; $Z_2' = Z_{11}Z_{21}$, $\tilde{Z}_2 = Z_{12}Z_{22}$, $Z_1 = Z_2'^2$, $\tilde{Z}_2 = \tilde{Z}_2Z_1$, $\tilde{Z}_2 = \tilde{Z}_2r$; $Z_2' = Z_2'\tilde{Z}_2$, $\tilde{Z}_2 = \tilde{Z}_2^2$, $z_2' = Z_2'^2$;	4S, 11M (3S, 8M)
3	$inv_1 = y_1$, $inv_0 = y_3$;	
4	$w_0 = V_{10}z_{22} - \tilde{V}_{20}$, $w_1 = V_{11}z_{22} - \tilde{V}_{21}$, $w_2 = inv_0w_0$, $w_3 = inv_1w_1$; $s_1 = (inv_0 + z_{11}inv_1)(w_0 + w_1) - w_2 - w_3(z_{11} + U_{11})$; $s_0 = w_2 - U_{10}w_3$;	8M (7M)
5	$S_1 = s_1^2$, $S_0 = s_0Z_1$, $Z_1' = s_1Z_1$, $S = Z_1'S_0$, $S_0 = S_0^2$; $R = rZ_1'$, $s_0 = s_0Z_1'$, $s_1 = s_1Z_1'$, $z_1' = Z_1'^2$;	3S, 6M
6	$l_2 = s_1\tilde{U}_{21}$, $l_0 = s_0\tilde{U}_{20}$, $l_1 = (s_0 + s_1)(\tilde{U}_{20} + \tilde{U}_{21}) - l_0 - l_2$; $l_2 = l_2 + S$;	3M
7	$V_1' = R\tilde{V}_{21}$; $U_0' = S_0 + y_1(S_1(y_1 + \tilde{U}_{21}) - 2s_0) + y_2s_1 + 2V_1' + (2\tilde{U}_{21} + y_1)\tilde{Z}_2$; $U_1' = 2S - y_1s_1 - z_2'$;	6M
8	$l_2 = l_2 - U_1'$, $w_0 = l_2U_0'$, $w_1 = l_2U_1'$;	2M
9	$V_1' = w_1 - z_1'(l_1 + V_1' - U_0')$, $V_0' = w_0 - z_1'(l_0 + R\tilde{V}_{20})$;	3M
total		7S, 47M (6S, 37M)

6.2 Doubling

The formulae for doubling make obvious why we include $z_2 = Z_2^2$ as well. If space is very limited such that one cannot apply windowing methods at all, this is the first coordinate to drop if one needs to restrict the algorithm. Still any binary method is fastest when including z_2.

Doubling, odd characteristic		
Step	Expression	Operations
1	$\tilde{U}_0 = U_0 z_1$, $w_0 = V_1^2$, $w_1 = U_1^2$, $w_3 = V_0 z_1 - U_1 V_1$; $r = w_0 U_0 + V_0 w_3$, $\tilde{Z}_2 = Z_2 r$, $\tilde{Z}_2 = \tilde{Z}_2 z_1$, $Z_2' = 2\tilde{Z}_2 Z_1$, $\tilde{Z}_2 = \tilde{Z}_2^2$;	3S, 8M
2	$inv_1 = -V_1$, $inv_0 = w_3$;	
3	$z_3 = z_1^2$, $w_3 = f_3 z_3 + w_1$, $k_1 = z_2(2(w_1 - \tilde{U}_0) + w_3)$; $z_3 = z_3 z_1$, $k_0 = z_2(U_1(4\tilde{U}_0 - w_3) + z_3 f_2) - w_0$;	1S, 6M
4	$w_0 = k_0 inv_0$, $w_1 = k_1 inv_1$; $s_1 = (inv_0 + inv_1)(k_0 + k_1) - w_0 - w_1(1 + U_1)$, $s_0 = w_0 - w_1 \tilde{U}_0$; If $s_1 = 0$ different case	5M
5	$S_0 = s_0^2$, $Z_1' = s_1 z_1$, $z_1' = {Z_1'}^2$, $S = s_0 Z_1'$; $R = r Z_1'$, $z_2' = Z_2'^2$, $s_0 = s_0 s_1$, $s_1 = Z_1' s_1$;	3S, 5M
6	$l_2 = s_1 U_1$, $l_0 = s_0 U_0$, $l_1 = (s_0 + s_1)(U_0 + U_1) - l_0 - l_2$; $l_2 = l_2 + S$;	3M
7	$V_1' = R V_1$, $U_0' = S_0 + 4(V_1' + 2\tilde{Z}_2 U_1)$, $U_1' = 2S - z_2'$;	2M
8	$l_2 = l_2 - U_1'$, $w_0 = l_2 U_0'$, $w_1 = l_2 U_1'$;	2M
9	$V_1' = w_1 - z_1'(l_1 + 2V_1' - U_0')$, $V_0' = w_0 - z_1'(l_0 + 2RV_0)$;	3M
total		7S, 34M

7 Different Sets of Coordinates in Odd Characteristic

So far we have given algorithms to perform the computations within one system and briefly mentioned additions involving one input in affine coordinates. Now we are concerned with mixes of coordinate systems. To have suitable abbreviations, we denote by $\mathcal{C}_1 + \mathcal{C}_2 = \mathcal{C}_3$ the computation of an addition, where the first input is in coordinate system $\mathcal{C}_1$, the second in $\mathcal{C}_2$ and the output is in $\mathcal{C}_3$. Similarly, $2\mathcal{C}_1 = \mathcal{C}_2$ denotes a doubling with input in system $\mathcal{C}_1$ and output in $\mathcal{C}_2$. We denote the affine system by $\mathcal{A}$, the projective by $\mathcal{P}$ and the new by $\mathcal{N}$. In the following we estimate the costs of computing scalar multiples using various systems of coordinates. To have the figures in mind, the following Table 1 lists the costs for the most useful additions and doublings.

7.1 Scalar Multiples in Odd Characteristic

In this section we concentrate on the computation of k-folds kD, where k is an integer and D is an ideal class. For references how to compute the respective expansions of k see [7] and the references given therein.

Let $\ell = \lfloor \log_2 k \rfloor$, i. e. $k = \sum_{i=0}^{\ell} k_i 2^i$. The direct approach to compute kD for a given class D is to use binary double-and-add starting with the most significant bit of k. For every $k_i = 1$ we need to perform an addition as well as a doubling, for a 0 one only doubles. The density, i. e. the number of ones in the expansion divided by the length, is asymptotically $1/2$.

Here we deal with the ideal class group of hyperelliptic curves and the negative of a class is obtained by negating the coordinates V_i or v_i respectively. Hence, signed binary expansions are useful. They have the lower density of $1/3$ if one uses a NAF (non-adjacent form) of the multiplier, and are approximately of the same length.

If we can afford some precomputations, windowing methods provide better performance; we consider signed expansions here. Let the window be of width w. The expansions we consider are of the form

$$k = 2^{k_0}(2^{k_1}(\cdots 2^{k_{v-1}}(2^{k_v} W[v] + W[v-1]) \cdots) + W[0]),$$

Table 1: Addition and Doubling in Different Systems, Odd Characteristic

Doubling		Addition	
operation	costs	operation	costs
$2\mathcal{N} = \mathcal{P}$	7S, 38M	$\mathcal{N} + \mathcal{N} = \mathcal{P}$	7S, 51M
$2\mathcal{P} = \mathcal{P}$	6S, 38M	$\mathcal{N} + \mathcal{P} = \mathcal{P}$	4S, 51M
$2\mathcal{N} = \mathcal{N}$	7S, 34M	$\mathcal{N} + \mathcal{N} = \mathcal{N}$	7S, 47M
$2\mathcal{P} = \mathcal{N}$	6S, 34M	$\mathcal{N} + \mathcal{P} = \mathcal{N}$	4S, 48M
		$\mathcal{P} + \mathcal{P} = \mathcal{P}$	4S, 47M
		$\mathcal{P} + \mathcal{P} = \mathcal{N}$	4S, 44M
		$\mathcal{A} + \mathcal{N} = \mathcal{P}$	5S, 40M
		$\mathcal{A} + \mathcal{P} = \mathcal{P}$	3S, 40M
		$\mathcal{A} + \mathcal{N} = \mathcal{N}$	5S, 36M
		$\mathcal{A} + \mathcal{P} = \mathcal{N}$	3S, 37M
$2\mathcal{A} = \mathcal{A}$	1I, 5S, 22M	$\mathcal{A} + \mathcal{A} = \mathcal{A}$	1I, 3S, 22M

Table 2: Without Precomputations, Odd Characteristic

Systems	Cost
$2\mathcal{A} = \mathcal{A}$, $\mathcal{A} + \mathcal{A} = \mathcal{A}$	$\ell/3$(4I, 18S, 88M)
$2\mathcal{N} = \mathcal{N}$, $\mathcal{A} + \mathcal{N} = \mathcal{N}$	$\ell/3$(26S, 138M)
$2\mathcal{N} = \mathcal{N}$, $\mathcal{N} + \mathcal{P} = \mathcal{N}$	$\ell/3$(25S, 149M)

where $W[i]$ is an odd integer in the range $-2^w + 1 \leq W[i] \leq 2^w - 1$ for all i, $W[v] > 0, k_0 \geq 0$ and $k_i \geq w + 1$ for $i \geq 1$.
We first consider systems without precomputations and then investigate good matches of coordinate systems for windowing methods. The reason for treating these cases separately is that for precomputations the addition will involve the set of coordinates which is advantageous for the precomputations, whereas in the system without precomputations this choice depends on the efficiency of the mixed addition only.

No Precomputations In this approach we perform approximately ℓ doublings and $\ell/3$ additions per scalar multiple of length ℓ. Table 2 lists the number of operations depending on the coordinate system, details are given below. We assume ℓ to be large and therefore leave out the costs for the initial moving from one system to the other as they occur only once. However, note that except for the first line, where inversions are assumed to be cheap, this conversion involves no divisions.
If inversions are relatively cheap, affine coordinates provide the best performance, thus, if the class is given in a non-normalized system we first normalize it. This takes 1I, 4M for $\mathcal{P} \rightarrow \mathcal{A}$ and 1I, 7M for $\mathcal{N} \rightarrow \mathcal{A}$. Then we double ℓ times and add on average $\ell/3$ times using $\ell/3$(4I, 18S, 88M).
Otherwise, for affine input the new system is best, as the most common operation (doubling) is cheaper than in any other fixed system and the mixed addition is also fast.
If the input is in $\mathcal{P}$ and inversions are very expensive, we need to find two systems $\mathcal{C}_1$ and $\mathcal{C}_2$ such

Table 3: Precomputations, Odd Characteristic

System	I	S	M
$\mathcal{A}$	$2^{w-1}+w-2$	$3\cdot 2^{w-1}+5w-8$	$22(2^{w-1}+w-2)$
$\mathcal{A}$	w	$3\cdot 2^{w-1}+5w-8$	$25\cdot 2^{w-1}+22w-50$
$\mathcal{A}$	1	$5\cdot 2^{w-1}+6w-11$	$48\cdot 2^{w-1}+38w-89$
$\mathcal{P}$		$5\cdot 2^{w-1}+6w-11$	$45\cdot 2^{w-1}+38w-83$

that $2\mathcal{C}_1=\mathcal{C}_1$, $2\mathcal{C}_1=\mathcal{C}_2$ and $\mathcal{C}_2+\mathcal{P}=\mathcal{C}_1$ are as cheap as possible, the first being the most frequent operation. By Table 1 we choose $\mathcal{C}_1=\mathcal{N}$. For $\mathcal{C}_2$ it is equal to choose $\mathcal{N}$ or $\mathcal{P}$, therefore we use $\mathcal{N}$ to save some bookkeeping. Thus the first doubling is done as $2\mathcal{P}=\mathcal{N}$ and all further as $2\mathcal{N}=\mathcal{N}$. There are approximately ℓ doublings $2\mathcal{N}=\mathcal{N}$ and $\ell/3$ additions $\mathcal{N}+\mathcal{P}=\mathcal{N}$ leading to $\ell/3$(25S, 149M).
If the input is in new coordinates we do the same except that the first doubling is $2\mathcal{N}=\mathcal{N}$ needing 1 more S and we use 4M for $\mathcal{N}\to\mathcal{P}$ of the initial point.
To compare, using only projective coordinates we would need $\ell/3$(23S, 156M) and only new coordinates results in $\ell/3$(28S, 149M), thus mixing the coordinate systems is advantageous.

Windowing Methods To obtain the table of precomputed values, i.e. all multiples $W[i]D$ for $1\le W[i]\le 2^w-1$, $W[i]$ odd, we need $w-1$ doublings and $2^{w-1}-1$ additions.
Like before we distinguish cases depending on the relative cost of inversions. If inversions are not too expensive, the precomputations are performed in affine coordinates. To still trade off some inversions for multiplications, we make use of Montgomery's trick of simultaneous inversions. Like in [7] we first compute $2D$, then $(3D,4D)$, then $(5D,7D,8D)$, $\ldots$,$((2^{w-2}+1)D,\ldots,(2^{w-1}-1)D,2^{w-1}D)$, and finally $((2^{w-1}+1)D,\ldots,(2^w-1)D)$, where each sequence involves only 1I. Computing m inversions simultaneously is done by 1I, $3(m-1)$M. Thus we need

$$w\text{I},\ w-1 \text{ class-doublings},\ 2^{w-1}-1 \text{ class-additions, and } 3(2^{w-1}-2) \text{ extra M}.$$

As most of the operations for the precomputations are additions, we choose projective coordinates $\mathcal{P}$ if we want to perform the precomputations avoiding inversions. Table 1 shows that additions involving at least one point in affine coordinates and leading to inversion free coordinates are much faster than those involving two non-affine points. Therefore, it can be useful to allow some more multiplications and 1 inversion to transform the precomputed points to affine coordinates. The costs for these three approaches leading to $\mathcal{A}$ and also for precomputations in $\mathcal{P}$ are listed in Table 3.
If inversions are cheap we stick to the affine system to compute the scalar multiplication. If we can afford the w inversions (or one more for non-affine input) to do the precomputations in affine, we use the new system for doublings, and perform the additions using the new mixed system $\mathcal{A}+\mathcal{N}=\mathcal{N}$. Finally, if inversions are very expensive, the best match is obtained if one uses projective coordinates for the precomputations, and the doublings are performed as $2\mathcal{N}=\mathcal{N}$. Then the addition is done as $\mathcal{N}+\mathcal{P}=\mathcal{N}$. This approach is equal to that of the previous subsection with non-affine input. Again, here we did not need a second system $\mathcal{C}_2$ for doublings.
In the main loop we perform $K=\sum_{i=0}^{v}k_i$ doublings and v additions. To ease the formulae, we assume $K=\ell-w/2+\theta$, $v=(\ell-w/2-\theta)/(w+2)$, where $\theta=1/2-1/(w+2)$. Let $n=\ell-w/2$.

Table 4: Windowing Method, Odd Characteristic

Systems	I	S	M
$2\mathcal{A}=\mathcal{A}$, $\mathcal{A}+\mathcal{A}=\mathcal{A}$	$n+\theta+\frac{n-\theta}{w+2}$	$5(n+\theta)+3\frac{n-\theta}{w+2}$	$22(n+\theta+\frac{n-\theta}{w+2})$
$2\mathcal{N}=\mathcal{N}$, $\mathcal{A}+\mathcal{N}=\mathcal{N}$		$7(n+\theta)+5\frac{n-\theta}{w+2}$	$34(n+\theta)+36\frac{n-\theta}{w+2}$
$2\mathcal{N}=\mathcal{N}$, $\mathcal{N}+\mathcal{P}=\mathcal{N}$		$7(n+\theta)+4\frac{n-\theta}{w+2}$	$34(n+\theta)+47\frac{n-\theta}{w+2}$

The costs are listed in Table 4. Some more computations can be saved using the tricks of [7]. Again we leave out the initial costs for conversions. For the precomputations see Table 3.

8 Conclusion and Outlook

As mentioned before one can save some constant number of operations in considering the first and last additions separately. This is worthwhile for an implementation where the system of coordinates of the input and output are fixed.

Since for hyperelliptic curve cryptography the finite field is of only half the bit size of that for elliptic curve cryptography, the field operations are 3 to 4 times faster. Comparing the results with the similar ones for elliptic curves one can assume that the complexity of scalar multiplications is similar. [19] compares timings for genus 1 and 2 using standard long integer libraries. However, as the integers considered here are rather small the comparison is biased. Avanzi [1] shows that genus 2 curves are competitive with elliptic curves over prime fields, both using affine and inversion free coordinate systems. His implementation uses a tailored long integer library suited for these comparably small finite fields. For fields of even characteristic the thesis of Wollinger [39] gives an overview of implementations also taking into account FPGA implementations. We also refer to his thesis for a list of what has been done so far for curves of larger genus.

With the new coordinates we tried to find a balance between the number of coordinates needed to represent a class and the speed-up obtainable. We took the case of elliptic curves as a guideline and use the same number of additional variables as the modified Jacobian coordinates. Note that one can apply them with fewer additional coordinates loosing not too much efficiency.

A further possibility to choose weighted coordinates is to allow even more entries to have a finer distinction. One notices that U_1' and V_1' are divisible by Z_1. Perhaps this can lead to further savings.

References

[1] R. M. Avanzi. Aspects of Hyperelliptic Curves over Large Prime Fields in Software Implementations. Cryptology ePrint Archive, Report 2003/253, 2003. to appear in CHES 2004.

[2] R. M. Avanzi. Countermeasures Against Differential Power Analysis for Hyperelliptic Curve Cryptosystems. In *Proceedings of CHES 2003*, volume 2779 of *LNCS*, pages 366–381, 2004.

[3] G. Bertoni, L. Breveglieri, T. Wollinger, and C. Paar. *Embedded Cryptographic Hardware: Design and Security*, chapter Hyperelliptic Curve Cryptosystem: What is the Best Parallel Hardware Architecture. Nova Science Publishers, 2004. to appear.

[4] D. Boneh and M. Franklin. Identity based encryption from the Weil pairing. In *Advances in cryptology – Crypto '2001*, volume 2139 of *Lect. Notes Comput. Sci.*, pages 213–229. Springer, 2001.

[5] B. Byramjee and S. Duqesne. Classification of genus 2 curves over $\mathbb{F}_{2^n}$ and optimization of their arithmetic. Cryptology ePrint Archive, Report 2004/107, 2004. `http://eprint.iacr.org/`.

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

[7] H. Cohen, A. Miyaji, and T. Ono. Efficient elliptic curve exponentiation using mixed coordinates. In *Proceedings of Asiacrypt'98*, volume 1514 of *Lecture Notes in Comput. Sci.*, pages 51–65. Springer, 1998.

[8] S. Duquesne. Montgomery scalar multiplication for genus 2 curves. In *Algorithmic Number Theory Seminar ANTS-VI*, 2004. to appear.

[9] G. Elias, A. Miri, , and T. Yeap. High-performance,FPGA-based hyperelliptic curve cryptosystems. In *The Proceedings of the 22nd Biennial Symposium on Communications – to appear*, Queens University, Kingston, Ontario, Canada, 2004.

[10] G. Frey and T. Lange. Mathematical Background of Public Key Cryptography. Technical Report 10, IEM Essen, 2003.

[11] S. D. Galbraith. Supersingular Curves in Cryptography. In *Advances in Cryptology – Asiacrypt 2001*, volume 2248 of *Lect. Notes Comput. Sci.*, pages 495–513. Springer, 2001.

[12] M. Gonda, K. Matsuo, K. Aoki, J. Chao, and S. Tsuji. Improvements of addition algorithm on genus 3 hyperelliptic curves and their implementations. In *Proc. of SCIS2004, IEICE Japan*, pages 995–1000, 2004.

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

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

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

[16] U. Krieger. Anwendung hyperelliptischer Kurven in der Kryptographie. Master's thesis, University Essen, 1997.

[17] J. Kuroki, M. Gonda, K. Matsuo, J. Chao, and S. Tsuji. Fast Genus Three Hyperelliptic Curve Cryptosystems. In *Proc. of SCIS2002, IEICE Japan*, pages 503–507, 2002.

[18] T. Lange. *Efficient Arithmetic on Hyperelliptic Curves.* PhD thesis, University Essen, 2001.

[19] T. Lange. Efficient Arithmetic on Genus 2 Hyperelliptic Curves over Finite Fields via Explicit Formulae. Cryptology ePrint Archive, Report 2002/121, 2002.

[20] 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`.

[21] T. Lange. Weighted Coordinates on Genus 2 Hyperelliptic Curves. Cryptology ePrint Archive, Report 2002/153, 2002.

[22] T. Lange. Montgomery Addition for Genus Two Curves. In *Algorithmic Number Theory Seminar ANTS-VI*, 2004. to appear.

[23] T. Lange and P. K. Mishra. SCA resistant Parallel Explicit Formula for Addition and Doubling of Divisors in the Jacobian of Hyperelliptic Curves of Genus 2. submitted, 2004.

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

[25] T. Lange and M. Stevens. Efficient Doubling for Genus Two Curves over Binary Fields. submitted, 2004.

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

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

[28] A. Menezes, Y.-H. Wu, and R. Zuccherato. An Elementary Introduction to Hyperelliptic Curves. In *Algebraic Aspects of Cryptography* [15].

[29] P. K. Mishra and P. Sakar. Parallelizing Explicit Formula in the Jacobian of Hyperelliptic Curves. In *Proceedings of Asiacrypt 2003*, volume 2894 of *LNCS*, pages 93–110, 2003.

[30] 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.

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

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

[33] J. Pelzl, T. Wollinger, and C. Paar. Special Hyperelliptic Curve Cryptosystems of Genus Two: Efficient Arithmetic and Fast Implementation. In *Embedded Cryptographic Hardware: Design and Security*, 2004. to appear.

[34] A. Silverberg and K. Rubin. Supersingular abelian varieties in cryptology. In *Advances in Cryptology - Crypto 2002*, volume 2442 of *Lect. Notes Comput. Sci.*, pages 336–353. Springer, 2002.

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

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

[37] 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.

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

[39] T. Wollinger. *Software and Hardware Implementation of Hyperelliptic Curve Cryptosystems.* PhD thesis, Ruhr-University of Bochum, 2004.

[40] T. Wollinger and C. Paar. *New Algorithms, Architectures, and Applications for Reconfigurable Computing*, chapter Security aspects of FPGAs in cryptographic applications. Kluwer, 2004.

Appendix 1: Case of Even Characteristic with $h_2 \neq 0$

In this section we first state the algorithms to compute with the new coordinates in even characteristic. Here, we assume $h_2 \neq 0$, however, the formulae hold universally. The more special case $h_2 = 0$ is considered in the following section. Like before it is interesting to include some precomputations in the coordinates which are updated during each iteration. In this approach we let:

$$z_1 = Z_1^2, z_2 = Z_2^2, z_3 = Z_1Z_2.$$

This turns out to be useful for both additions and doublings. While the costs remain unchanged for doublings if one additionally puts $z_4 = z_1z_3$ one saves 1M in the addition. Thus, here $\mathcal{N}$ denotes $[U_1, U_0, V_1, V_0, Z_1, Z_2, z_1, z_2, z_3, z_4]$. However, the formulae show that Z_1 and Z_2 are not used separately. Therefore they can be left out leading again to 6 coordinates only.
As $p = 2, h_2 \neq 0$ we assume $f_3 = f_2 = 0, h_2 = 1$ and include them in the table (but not in the counting) only for the sake of completeness, f_4 is left out completely. As shown in Subsection 4.2 this form can always be achieved.
For the addition we assume that both classes are in $\mathcal{N}$. If one is in $\mathcal{A}$ the costs are given in brackets. A dedicated algorithm for $\mathcal{N} + \mathcal{A} = N$ needs 5S, 37M (see [21]).

Addition, even characteristic, $h_2 \neq 0$		
Step	Expression	Operations
1	$\tilde{U}_{21} = U_{21}z_{11}$, $\tilde{U}_{20} = U_{20}z_{11}$, $\tilde{V}_{21} = V_{21}z_{14}$, $\tilde{V}_{20} = V_{20}z_{14}$; $Z_1 = z_{11}z_{21}$, $Z_3 = z_{13}z_{23}$;	6M (–)
2	$y_1 = U_{11}z_{21} + \tilde{U}_{21}$, $y_2 = U_{10}z_{21} + \tilde{U}_{20}$, $y_3 = U_{11}y_1 + y_2z_{11}$; $r = y_2y_3 + y_1^2U_{10}$, $\tilde{Z}_2 = rZ_3$, $Z_2' = \tilde{Z}_2Z_1$;	1S, 8M (1S, 7M)
3	$inv_1 = y_1$, $inv_0 = y_3$;	
4	$w_0 = V_{10}z_{24} + \tilde{V}_{20}$, $w_1 = V_{11}z_{24} + \tilde{V}_{21}$, $w_2 = inv_0w_0$, $w_3 = inv_1w_1$; $s_1 = (inv_0 + inv_1z_{11})(w_0 + w_1) + w_2 + w_3(z_{11} + U_{11})$; $s_0 = w_2 + w_3U_{10}$; If $s_1 = 0$ different case	8M (7M)
5	$\tilde{s}_0 = s_0Z_1$, $S_0 = \tilde{s}_0^2$, $Z_1' = s_1Z_1$, $R = rZ_1'$; $t = s_1(y_1 + \tilde{U}_{21})$, $U_1' = y_1s_1$, $s_1 = s_1Z_1'$, $s_0 = s_0Z_1'$; $z_1' = Z_1'^2$, $z_2' = Z_2'^2$, $z_3' = Z_1'Z_2'$, $z_4' = z_1'z_3'$, $\tilde{h}_1 = h_1z_3'$;	3S, 10M
6	$l_2 = s_1\tilde{U}_{21}$, $l_0 = s_0\tilde{U}_{20}$, $l_1 = (s_0 + s_1)(\tilde{U}_{20} + \tilde{U}_{21}) + l_0 + l_2$;	3M
7	$U_0' = S_0 + tU_1' + y_2s_1 + Z_2'(h_2(\tilde{s}_0 + t) + y_1\tilde{Z}_2) + \tilde{h}_1$; $U_1' = U_1'Z_1' + h_2z_3' + z_2'$;	5M
8	$l_2 = l_2 + Z_1'\tilde{s}_0 + h_2z_3' + U_1'$, $w_0 = l_2U_0'$, $w_1 = l_2U_1'$;	3M
9	$V_1' = w_1 + z_1'(l_1 + R\tilde{V}_{21} + U_0' + \tilde{h}_1)$; $V_0' = w_0 + z_1'(l_0 + R\tilde{V}_{20}) + h_0z_4'$;	5M
total		4S, 48M (4S, 40M)

Now finally we consider doublings.

Doubling, even characteristic, $h_2 \neq 0$		
Step	Expression	Operations
1	$\tilde{h}_1 = z_1 h_1$, $\tilde{h}_0 = z_1 h_0$, $\tilde{V}_1 = \tilde{h}_1 + h_2 U_1$, $\tilde{V}_0 = \tilde{h}_0 + h_2 U_0$; $w_0 = V_1^2$, $w_1 = U_1^2$, $w_2 = (\tilde{h}_1)^2 + h_2^2 w_1$; $w_3 = z_1(h_1 U_1 + h_2 U_0 + \tilde{h}_0) + h_2 w_1$; $r = w_2 U_0 + \tilde{V}_0 w_3$, $\tilde{Z}_2 = z_3 r$, $Z_2' = \tilde{Z}_2 z_4$;	3S, 8M
2	$inv_1 = \tilde{V}_1$, $inv_0 = w_3$;	
3	$w_3 = f_3 z_1^2 + w_1$, $k_1 = w_3 z_2 + V_1 h_2 z_3$; $k_0 = U_1 k_1 + w_0 + z_4(V_1 h_1 + V_0 h_2 + f_2 z_4)$;	5M
4	$w_0 = k_0 inv_0$, $w_1 = k_1 inv_1$; $s_1 = (inv_0 + inv_1)(k_0 + k_1) + w_0 + w_1(1 + U_1)$; $s_0 = w_0 + U_0 w_1 z_1$; If $s_1 = 0$ different case	6M
5	$t = h_2 s_0 + s_1(h_2 U_1 + \tilde{h}_1)$, $Z_1' = s_1 z_1$, $S_0 = s_0^2$, $z_1' = {Z_1'}^2$, $S = s_0 Z_1'$; $R = \tilde{Z}_2 Z_1'$, $s_0 = s_0 s_1$, $s_1 = Z_1' s_1$, $z_2' = {Z_2'}^2$, $z_3' = Z_1' Z_2'$, $z_4' = z_1' z_3'$;	3S, 8M
6	$l_2 = s_1 U_1$, $l_0 = s_0 U_0$, $l_1 = (s_1 + s_0)(U_1 + U_0) - l_0 - l_2$; $l_2 = l_2 + S + h_2 z_3'$;	3M
7	$U_0' = S_0 + Z_2' t$; $U_1' = z_2' + h_2 z_3'$;	1M
8	$l_2 = l_2 + U_1'$, $w_0 = l_2 U_0'$, $w_1 = l_2 U_1'$;	2M
9	$V_1' = w_1 + z_1'(l_1 + RV_1 + U_0') + z_4' h_1$; $V_0' = w_0 + z_1'(l_0 + RV_0) + z_4' h_0$;	6M
total		6S, 39M

Different Sets of Coordinates

Using the same abbreviations as in odd characteristic, we state the costs for the operations in different coordinate systems in Table 5. Note, that contrary to the odd characteristic case the advantage of using the new coordinates is smaller.

Computation of Scalar Multiples

We follow the same lines as in the odd characteristic and distinguish between precomputations and no precomputations.

No Precomputations For cheap inversions one again uses the affine system alone. If one wants to avoid inversions and has an affine input (or can allow 1I to achieve this) we perform the doublings as $2\mathcal{N} = \mathcal{N}$ and the addition as $\mathcal{A} + \mathcal{N} = \mathcal{N}$. For non-normalized input we use the new coordinates for doublings and as non-normalized input system if necessary.

Windowing Methods To obtain the table of precomputed values we need $w - 1$ doublings and $2^{w-1} - 1$ additions. Here it is advantageous to choose either $\mathcal{C}_3 = \mathcal{A}$ or $\mathcal{C}_3 = \mathcal{N}$
The costs of computing scalar multiples are listed in Table 8 for the most useful matches of sets of coordinates. We use the same abbreviations as in the odd characteristic case. Again we leave out

Table 5: Different Systems, Even Characteristic, $h_2 \neq 0$

Doubling		**Addition**	
operation	costs	operation	costs
$2\mathcal{N} = \mathcal{P}$	6S, 39M	$\mathcal{N} + \mathcal{P} = \mathcal{P}$	4S, 51M
$2\mathcal{P} = \mathcal{P}$	7S, 38M	$\mathcal{N} + \mathcal{N} = \mathcal{P}$	4S, 50M
$2\mathcal{N} = \mathcal{N}$	6S, 37M	$\mathcal{N} + \mathcal{P} = \mathcal{N}$	4S, 50M
$2\mathcal{P} = \mathcal{N}$	7S, 36M	$\mathcal{P} + \mathcal{P} = \mathcal{P}$	4S, 49M
		$\mathcal{N} + \mathcal{N} = \mathcal{N}$	4S, 48M
		$\mathcal{P} + \mathcal{P} = \mathcal{N}$	4S, 48M
		$\mathcal{A} + \mathcal{N} = \mathcal{P}$	5S, 39M
		$\mathcal{A} + \mathcal{P} = \mathcal{P}$	4S, 39M
		$\mathcal{A} + \mathcal{P} = \mathcal{N}$	4S, 38M
		$\mathcal{A} + \mathcal{N} = \mathcal{N}$	5S, 37M
$2\mathcal{A} = \mathcal{A}$	1I, 5S, 22M	$\mathcal{A} + \mathcal{A} = \mathcal{A}$	1I, 3S, 22M

Table 6: Without Precomputations, Even Characteristic, $h_2 \neq 0$

Systems	Cost
$2\mathcal{A} = \mathcal{A}$, $\mathcal{A} + \mathcal{A} = \mathcal{A}$	$\ell/3$(4I, 18S, 88M)
$2\mathcal{N} = \mathcal{N}$, $\mathcal{A} + \mathcal{N} = \mathcal{N}$	$\ell/3$(23S, 148M)
$2\mathcal{N} = \mathcal{N}$, $\mathcal{N} + \mathcal{N} = \mathcal{N}$	$\ell/3$(22S, 159M)

Table 7: Precomputations, Even Characteristic, $h_2 \neq 0$

System	I	S	M
$\mathcal{A}$	$2^{w-1} + w - 2$	$3 \cdot 2^{w-1} + 5w - 8$	$22(2^{w-1} + w - 2)$
$\mathcal{A}$	w	$3 \cdot 2^{w-1} + 5w - 8$	$25 \cdot 2^{w-1} + 22w - 50$
$\mathcal{A}$	1	$2^{w+1} + 6w - 10$	$51 \cdot 2^{w-1} + 37w - 91$
$\mathcal{P}$		$2^{w+1} + 6w - 10$	$48 \cdot 2^{w-1} + 37w - 85$

Table 8: Windowing Method, Even Characteristic, $h_2 \neq 0$

Systems	I	S	M
$2\mathcal{A}=\mathcal{A}$, $\mathcal{A}+\mathcal{A}=\mathcal{A}$	$n+\theta+\frac{n-\theta}{w+2}$	$5(n+\theta)+3\frac{n-\theta}{w+2}$	$22((n+\theta)+\frac{n-\theta}{w+2}))$
$2\mathcal{N}=\mathcal{N}$, $\mathcal{A}+\mathcal{N}=\mathcal{N}$		$6(n+\theta)+5\frac{n-\theta}{w+2}$	$37((n+\theta)+\frac{n-\theta}{w+2})$
$2\mathcal{N}=\mathcal{N}$, $\mathcal{N}+\mathcal{N}=\mathcal{N}$		$6(n+\theta)+4\frac{n-\theta}{w+2}$	$37(n+\theta)+48\frac{n-\theta}{w+2}$

the costs for the initial conversions and mention that some constant number of operations can be saved if one considers in more detail the first doubling and the final addition/doubling like in [7]. Compared to the results in odd characteristic this case is a bit more expensive. On the other hand the arithmetic in binary fields is easier to implement and usually faster.

Appendix 2: Case of Even Characteristic with $h_2 = 0$

Obviously this case can be considered as a special case of Appendix 1 where f_2 cannot assumed to be 0. However, if one restricts the algorithms to this still very frequent case one can save some operations. Like before we suggest to enlarge the set of coordinates by $z_1 = Z_1^2, z_2 = Z_2^2, z_3 = Z_1Z_2, z_4 = Z_1^3Z_2$. To save space we can again discard Z_1, Z_2. In this case we can assume $h_2 = f_4 = f_3 = 0$. In the following tables the multiplications by h_1 are still counted even though h_1 is small with large probability.

Addition, even characteristic, $h_2 = 0$		
Step	Expression	Operations
1	$\tilde{U}_{21} = U_{21}z_{11}$, $\tilde{U}_{20} = U_{20}z_{11}$, $\tilde{V}_{21} = V_{21}z_{14}$, $\tilde{V}_{20} = V_{20}z_{14}$; $Z_1 = z_{11}z_{21}$, $Z_3 = z_{13}z_{23}$;	6M (–)
2	$y_1 = U_{11}z_{21} + \tilde{U}_{21}$, $y_2 = U_{10}z_{21} + \tilde{U}_{20}$, $y_3 = U_{11}y_1 + y_2z_{11}$; $r = y_2y_3 + y_1^2U_{10}$, $\tilde{Z}_2 = rZ_3$, $Z_2' = \tilde{Z}_2Z_1$, $\tilde{Z}_2 = \tilde{Z}_2^2$, $\tilde{Z}_2 = \tilde{Z}_2Z_1$;	2S, 9M (2S, 8M)
3	$inv_1 = y_1$, $inv_0 = y_3$;	
4	$w_0 = V_{10}z_{24} + \tilde{V}_{20}$, $w_1 = V_{11}z_{24} + \tilde{V}_{21}$, $w_2 = inv_0w_0$, $w_3 = inv_1w_1$; $s_1 = (inv_0 + inv_1z_{11})(w_0 + w_1) + w_2 + w_3(z_{11} + U_{11})$; $s_0 = w_2 + w_3U_{10}$; If $s_1 = 0$ different case	8M (7M)
5	$S_1 = s_1^2$, $Z_1' = s_1Z_1$, $R = rZ_1'$, $S_0 = s_0Z_1$, $S = S_0Z_1'$, $S_0 = S_0^2$; $z_1' = Z_1'^2$, $z_2' = Z_2'^2$, $z_3' = Z_1'Z_2'$, $z_4' = z_1'z_3'$, $\tilde{h}_1 = h_1z_3'$, $s_1 = s_1Z_1'$, $s_0 = s_0Z_1'$;	4S, 9M
6	$l_2 = s_1\tilde{U}_{21}$, $l_0 = s_0\tilde{U}_{20}$, $l_1 = (s_0 + s_1)(\tilde{U}_{20} + \tilde{U}_{21}) + l_2 + l_0$; $l_2 = l_2 + S$;	3M
7	$U_0' = S_0 + y_1(S_1(y_1 + \tilde{U}_{21}) + \tilde{Z}_2) + y_2s_1 + \tilde{h}_1$; $U_1' = y_1s_1 + z_2'$;	4M
8	$l_2 = l_2 + U_1'$, $w_0 = l_2U_0'$, $w_1 = l_2U_1'$;	2M
9	$V_1' = w_1 + z_1'(l_1 + R\tilde{V}_{21} + U_0' + \tilde{h}_1)$; $V_0' = w_0 + z_1'(l_0 + R\tilde{V}_{20}) + z_4'h_0$;	5M
total		6S, 46M (6S, 38M)

The numbers in brackets apply if one of the input variables is affine. An algorithm dedicated to that case needs only 6S, 36M.

Table 9: Different Systems, Even Characteristic, $h_2 = 0$

Doubling		**Addition**	
operation	costs	operation	costs
$2\mathcal{N} = \mathcal{P}$	6S, 37M	$\mathcal{N} + \mathcal{P} = \mathcal{P}$	4S, 51M
$2\mathcal{P} = \mathcal{P}$	7S, 36M	$\mathcal{N} + \mathcal{N} = \mathcal{P}$	6S, 48M
$2\mathcal{N} = \mathcal{N}$	6S, 35M	$\mathcal{N} + \mathcal{P} = \mathcal{N}$	4S, 49M
$2\mathcal{P} = \mathcal{N}$	7S, 34M	$\mathcal{P} + \mathcal{P} = \mathcal{P}$	4S, 49M
		$\mathcal{N} + \mathcal{N} = \mathcal{N}$	6S, 46M
		$\mathcal{P} + \mathcal{P} = \mathcal{N}$	4S, 47M
		$\mathcal{A} + \mathcal{P} = \mathcal{P}$	4S, 39M
		$\mathcal{A} + \mathcal{N} = \mathcal{P}$	6S, 38M
		$\mathcal{A} + \mathcal{P} = \mathcal{N}$	4S, 38M
		$\mathcal{A} + \mathcal{N} = \mathcal{N}$	6S, 36M
$2\mathcal{A} = \mathcal{A}$	1I, 5S, 22M	$\mathcal{A} + \mathcal{A} = \mathcal{A}$	1I, 3S, 22M

Doubling, even characteristic, $h_2 = 0$		
Step	Expression	Operations
1	$w_1 = h_1U_1 + h_0z_1,\ r = h_0w_1 + h_1^2U_0,\ \tilde{Z}_2 = rz_4,\ Z_2' = \tilde{Z}_2z_4;$	1S, 6M
2	$inv_1 = h_1,\ inv_0 = w_1;$	
3	$w_0 = V_1^2,\ w_1 = U_1^2,\ k_1 = z_2(f_3z_1^2 + w_1);$ $k_0 = U_1k_1 + w_0 + z_4(f_2z_4 + V_1h_1);$	2S, 5M
4	$w_0 = k_0inv_0,\ w_1 = k_1inv_1;$ $s_1 = (inv_0 + inv_1)(k_0 + k_1) + w_0 + w_1(1 + U_1);$ $s_0 = w_0 + U_0w_1z_1;$ If $s_1 = 0$ different case	6M
5	$Z_1' = s_1z_1,\ S_0 = s_0^2,\ S = s_0Z_1',\ R = \tilde{Z}_2Z_1';$ $z_1' = Z_1'^2,\ z_2' = Z_2'^2,\ z_3' = Z_1'Z_2',\ z_4' = z_1'z_3';$ $s_0 = s_0s_1,\ s_1 = Z_1's_1,\ \tilde{h}_1 = h_1z_3';$	3S, 8M
6	$l_2 = s_1U_1,\ l_0 = s_0U_0,\ l_1 = (s_1 + s_0)(U_1 + U_0) + l_0 + l_2;$ $l_2 = l_2 + S;$	3M
7	$U_0' = S_0 + \tilde{h}_1,\ U_1' = z_2';$	
8	$l_2 = l_2 + U_1',\ w_0 = l_2U_0',\ w_1 = l_2U_1';$	2M
9	$V_1' = w_1 + z_1'(l_1 + RV_1 + U_0' + \tilde{h}_1);$ $V_0' = w_0 + z_1'(l_0 + RV_0 + z_3'h_0);$	5M
total		6S, 35M

Different Sets of Coordinates

Finally, we state the number of operations in the case of even characteristic and with $h_2 = 0$ in Table 9.

Computation of Scalar Multiples

Table 9 reveals that for this case additions involving $\mathcal{N}$ are less expensive than those involving $\mathcal{P}$.

Table 10: Without Precomputations, Even Characteristic, $h_2 = 0$

Systems	Cost
$2\mathcal{A} = \mathcal{A}$, $\mathcal{A} + \mathcal{A} = \mathcal{A}$	$\ell/3$(4I, 18S, 88M)
$2\mathcal{N} = \mathcal{N}$, $\mathcal{A} + \mathcal{N} = \mathcal{N}$	$\ell/3$(24S, 141M)
$2\mathcal{N} = \mathcal{N}$, $\mathcal{N} + \mathcal{N} = \mathcal{N}$	$\ell/3$(24S, 151M)

Table 11: Precomputations, Even Characteristic, $h_2 = 0$

System	I	S	M
$\mathcal{A}$	$2^{w-1} + w - 2$	$3 \cdot 2^{w-1} + 5w - 8$	$22(2^{w-1} + w - 2)$
$\mathcal{A}$	w	$3 \cdot 2^{w-1} + 5w - 8$	$25 \cdot 2^{w-1} + 22w - 50$
$\mathcal{A}$	1	$6(2^{w-1} + w - 2)$	$49 \cdot 2^{w-1} + 35w - 87$
$\mathcal{N}$		$6(2^{w-1} + w - 2)$	$46 \cdot 2^{w-1} + 35w - 81$

No Precomputations For cheap inversions one again uses the affine system alone. If one wants to avoid inversions and has an affine input (or can allow 1I to achieve this) we do the same as in the general case and perform the doublings as $2\mathcal{N} = \mathcal{N}$ and the addition as $\mathcal{A} + \mathcal{N} = \mathcal{N}$. For non-normalized input we here suggest to use $\mathcal{C}_1 = \mathcal{C}_2 = \mathcal{C}_3 = \mathcal{N}$.

Windowing Methods To obtain the table of precomputed values we need $w - 1$ doublings and $2^{w-1} - 1$ additions. Here we choose either $\mathcal{C}_3 = \mathcal{A}$ or $\mathcal{C}_3 = \mathcal{N}$
Table 12 states the number of operations for the most useful matches of sets of coordinates.

Appendix 3: Koblitz curves

Koblitz curves, also called subfield curves, are curves defined over a comparably small field which are then considered over a large extension field. To compute scalar multiples one can make use of the Frobenius endomorphism (see [18]). Furthermore, as the coefficients of the curve come from the small field, the individual additions and doublings get cheaper as multiplications by h_i and f_i are for free. While for odd characteristic the savings are obvious and not too significant we have the following Table 13 for even characteristic.

Table 12: Windowing Method, Even Characteristic, $h_2 = 0$

Systems	I	S	M
$2\mathcal{A} = \mathcal{A}$, $\mathcal{A} + \mathcal{A} = \mathcal{A}$	$n + \theta + \frac{n-\theta}{w+2}$	$5(n + \theta) + 3\frac{n-\theta}{w+2}$	$22((n + \theta) + \frac{n-\theta}{w+2})$
$2\mathcal{N} = \mathcal{N}$, $\mathcal{A} + \mathcal{N} = \mathcal{N}$		$6((n + \theta) + \frac{n-\theta}{w+2})$	$35(n + \theta) + 36\frac{n-\theta}{w+2}$
$2\mathcal{N} = \mathcal{N}$, $\mathcal{N} + \mathcal{N} = \mathcal{N}$		$6((n + \theta) + \frac{n-\theta}{w+2})$	$35(n + \theta) + 46\frac{n-\theta}{w+2}$

Table 13: Precomputations, Even Characteristic, $h_2 = 0$

Doubling			**Addition**		
operation	costs		operation	costs	
	$h_2 \neq 0$	$h_2 = 0$		$h_2 \neq 0$	$h_2 = 0$
$2\mathcal{N} = \mathcal{P}$	6S, 35M	5S, 29M	$\mathcal{N} + \mathcal{P} = \mathcal{P}$	4S, 48M	4S, 48M
$2\mathcal{P} = \mathcal{P}$	6S, 35M	6S, 32M	$\mathcal{N} + \mathcal{N} = \mathcal{P}$	4S, 48M	4S, 46M
$2\mathcal{N} = \mathcal{N}$	5S, 34M	5S, 27M	$\mathcal{P} + \mathcal{P} = \mathcal{P}$	4S, 46M	4S, 46M
$2\mathcal{P} = \mathcal{N}$	6S, 33M	6S, 30M	$\mathcal{N} + \mathcal{P} = \mathcal{N}$	4S, 46M	4S, 46M
			$\mathcal{N} + \mathcal{N} = \mathcal{N}$	4S, 46M	6S, 44M
			$\mathcal{P} + \mathcal{P} = \mathcal{N}$	4S, 44M	6S, 44M
			$\mathcal{A} + \mathcal{N} = \mathcal{P}$	5S, 37M	4S, 36M
			$\mathcal{A} + \mathcal{P} = \mathcal{P}$	4S, 36M	4S, 36M
			$\mathcal{A} + \mathcal{N} = \mathcal{N}$	5S, 35M	4S, 34M
			$\mathcal{A} + \mathcal{P} = \mathcal{N}$	4S, 35M	4S, 34M
$2\mathcal{A} = \mathcal{A}$	1I, 5S, 20M	1I, 5S, 17M	$\mathcal{A} + \mathcal{A} = \mathcal{A}$	1I, 3S, 21M	1I, 3S, 21M