Inversion-Free 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

December 8, 2002

Abstract

We investigate formulae to double and add in the ideal class group of a hyperelliptic genus 2 curve avoiding inversions. To that aim we introduce a further coordinate in the representation of a class in which we collect the common denominator of the usual 4 coordinates. The analysis shows that this is practical and advantageous whenever inversions are expensive compared to multiplications like for example on smart cards.

1 Introduction

Hyperelliptic curve cryptography is now at the point of providing a real alternative to the established elliptic curve cryptography. Using explicit formulae instead of Cantor's algorithm the complexity of arithmetic in the ideal class group of hyperelliptic curves is comparable to that of elliptic curves (see [4, 5, 3] for genus 2). In the usual cases a general addition needs 1 inversion, 3 squarings and 22 multiplications whereas a doubling needs 2 more squarings. Compared to elliptic curves the finite field has only half of the bit-size. Thus whenever one would use elliptic curves in affine coordinates, i. e. allowing one inversion for each addition resp. doubling, the ideal class group of a hyperelliptic genus two curve can now take that role.

But there are environments in which 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 . For elliptic curves one can fall back on several inversion-free systems, like projective, Jacobian, Chudnovsky Jacobian, modified Jacobian and mixed coordinates (see [2]).

So far there is only one article on a generalization of such ideas to genus two curves by Miyamoto, Doi, Matsuo, Chao, and Tsuji [4]. Here, 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 obvious resemblance with the elliptic case.

In addition, we consider a special kind of mixed coordinates: In applications the usual scenario is that the device obtains a class in affine representation and computes a scalar multiple of it. As inversions are slow, we allow the intermediate results to have an additional coordinate containing the common denominator. In the course of computing, the doubling algorithm usually obtains such a projective class and outputs one, but in the addition step one input is in affine and the other in projective representation and the output is projective.

2 Background

In this note we use the same notation and approach as in Lange [3], thus we omit a long introduction here and refer the reader to that paper (obtainable online). Like there we study only the main cases in detail, as any special case occurs with very low probability and thus can be either omitted or done by a less efficient routine.
The group one works in is the ideal class group of a maximal order of a hyperelliptic function field. A hyperelliptic curve of genus two with an $\mathbb{F}_q$-rational Weierstraß point can be given by an equation of the form

$$C : y^2 + (h_2x^2 + h_1x + h_0)y = x^5 + f_4x^4 + f_3x^3 + f_2x^2 + f_1x + f_0, \quad h_i, f_i \in \mathbb{F}_q, q = p^r.$$

For short we write $y^2 + h(x)y = f(x)$. Each class can be represented by an ordered pair of polynomials $[u, v]$ with u monic, $\deg v < \deg u \leq 2$ and with $u|f - hv - v^2$. Most commonly we have $[u, v] = [x^2 + u_1x + u_0, v_1x + v_0]$ and the result $[u', v']$ of adding two classes or doubling one class has $\deg u' = 2$ as well. 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.
Like in the previous paper we state the formulae for arbitrary characteristic. If p is odd one usually has $h = 0$ and if $p \neq 5$ also $f_4 = 0$. In even characteristic h must be nonzero as otherwise the curve would be singular. Furthermore, for security purposes the curve should not be supersingular and this excludes the case of $h = 1$ which otherwise would be the most efficient. For nonzero h_2 one can always achieve $h_2 = 1$. Furthermore, $f_4 = 0$ can easily be achieved. In many cases one can also assume that $h_1, h_0 \in \{0, 1\}$. Therefore we do not count multiplications by these coefficients but include them in the formulae for completeness. If these assumptions do not hold some steps should be arranged differently to maintain or only slightly lower the efficiency.

3 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 (but see Avanzi [1] for an elegant and efficient way to avoid this) or when using precomputations to speed up and these are 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 following table lists the number of field operations needed to perform the respective steps. Numbers in brackets refer to the case, that the first input is affine, i. e. has $Z_1 = 1$. See the next section for an algorithm dedicated to this case.

Addition

Input	$[U_{11}, U_{10}, V_{11}, V_{10}, Z_1], [U_{21}, U_{20}, V_{21}, V_{20}, Z_2]$ $h = h_2x^2 + h_1x + h_0, f = x^5 + f_4x^4 + f_3x^3 + f_2x^2 + f_1x + f_0$
Output	$[U_1', U_0', V_1', V_0', Z'] = [U_{11}, U_{10}, V_{11}, V_{10}, Z_1] + [U_{21}, U_{20}, V_{21}, V_{20}, Z_2]$

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: $R = Zr$, $s_0 = s_0Z$, $s_3 = s_1Z$, $\tilde{R} = Rs_3$, $t = s_1(z_1 + \tilde{U}_{21})$; $S_3 = s_3^2$, $S = s_0s_1$, $\tilde{S} = s_3s_1$, $\tilde{\tilde{S}} = s_0s_3$, $\tilde{\tilde{R}} = \tilde{R}\tilde{S}$;	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': $U_0' = s_0^2 + s_1z_1(t - 2s_0) + z_2\tilde{S} +$ $+R(h_2(s_0 - t) + s_1(h_1Z + 2\tilde{V}_{21}) + r(z_1 + 2\tilde{U}_{21} - f_4Z))$; $U_1' = 2\tilde{\tilde{S}} - \tilde{S}z_1 + h_2\tilde{R} - R^2$;	2S, 7M
8	precomputations: $l_2 = l_2 - U_1'$, $w_0 = U_0'l_2 - S_3l_0$, $w_1 = U_1'l_2 + S_3(U_0' - l_1)$;	4M
9	adjust: $Z' = \tilde{R}S_3$, $U_1' = \tilde{R}U_1'$, $U_0' = \tilde{R}U_0'$;	3M
10	compute V': $V_0' = w_0 + h_2U_0' - \tilde{\tilde{R}}\tilde{V}_{20} - h_0Z'$; $V_1' = w_1 + h_2U_1' - \tilde{\tilde{R}}\tilde{V}_{21} - h_1Z'$;	2M
total		4S, 47M (4S, 40M)

Remark: If in characteristic 2 one additionally has $f_4 = 0$, one need not precompute t in Step 5 and therefore saves 1 multiplication as

$$U_0' = s_0^2 + S_1z_1(z_1 + \tilde{U}_{21}) + z_3\tilde{S} + R(h_2s_0 + z_1(r + h_2s_1) + s_1(h_1Z + h_2\tilde{U}_{21})).$$

4 Mixed Addition

Assuming that one computes a scalar multiple of a class in affine representation using (signed) double-and-add, one enters the addition algorithm with the fixed class in affine representation and the intermediate result in projective. In the previous algorithm we treated this as a less expensive case of the general addition. Now we show how a special algorithm for this case can even do better.

Mixed Addition

Input	$[U_{11}, U_{10}, V_{11}, V_{10}], [U_{21}, U_{20}, V_{21}, V_{20}, Z_2]$ $h = h_2x^2 + h_1x + h_0, f = x^5 + f_4x^4 + f_3x^3 + f_2x^2 + f_1x + f_0$
Output	$[U_1', U_0', V_1', V_0', Z'] = [U_{11}, U_{10}, V_{11}, V_{10}] + [U_{21}, U_{20}, V_{21}, V_{20}, Z_2]$

Step	Expression	Operations
1	compute resultant r of U_1, U_2: $z_1 = U_{11}Z_2 - U_{21}$, $z_2 = U_{20} - U_{10}Z_2$, $z_3 = U_{11}z_1 + z_2$; $r = z_2z_3 + z_1^2U_{10}$;	1S, 5M
2	compute almost inverse of u_2 modulo u_1 : $inv_1 = z_1$, $inv_0 = z_3$;	
3	compute s: $w_0 = V_{10}Z_2 - V_{20}$, $w_1 = V_{11}Z_2 - V_{21}$, $w_2 = inv_0w_0$, $w_3 = inv_1w_1$; $s_1 = (inv_0 + inv_1)(w_0 + w_1) - w_2 - w_3(1 + U_{11})$; $s_0 = w_2 - U_{10}w_3$; If $s_1 = 0$ different case	7M
4	precomputations: $R = s_1r$, $R_2 = r^2$, $S_1 = s_1^2$, $S = s_1s_0$, $\tilde{S} = S_1Z_2$; $\tilde{\tilde{S}} = SZ_2$, $\tilde{R} = RZ_2$, $\tilde{\tilde{R}} = \tilde{R}S_1$;	2S, 6M
5	compute l: $l_2 = S_1U_{21}$, $l_0 = SU_{20}$, $l_1 = (S_1 + S)(U_{21} + U_{20}) - l_2 - l_0$; $l_2 = l_2 + \tilde{\tilde{S}}$;	3M
6	compute U': $U_0' = (s_0 - U_{11}s_1)(Z_2(s_0 + h_2r) - z_1s_1) + z_2S_1 + U_{21}S +$ $+R(h_1Z_2 + 2V_{21}) + R_2(z_1 + 2U_{21} - f_4Z_2)$; $U_1' = 2\tilde{\tilde{S}} - z_1S_1 + h_2\tilde{R} - Z_2R_2$;	10M
7	precomputations: $l_2 = l_2 - U_1'$, $w_0 = U_0'l_2 - \tilde{S}l_0$, $w_1 = U_1'l_2 + \tilde{S}(U_0' - l_1)$;	4M
8	adjust: $Z' = \tilde{\tilde{R}}\tilde{S}$, $U_1' = \tilde{R}U_1'$, $U_0' = \tilde{R}U_0'$;	3M
9	compute V': $V_0' = w_0 + h_2U_0' - \tilde{\tilde{R}}V_{20} - h_0Z'$; $V_1' = w_1 + h_2U_1' - \tilde{\tilde{R}}V_{21} - h_1Z'$;	2M
total		3S, 40M

Remarks:

1. Using the above algorithm one saves 1 squaring more than is obvious form the general addition algorithm. Thus it is worthwhile to implement this special algorithm in the setting that the algorithm is to compute scalar multiples, classes given in affine representation. If on the other hand one wants to avoid inversions at all and also usually gets inputs in projective representation, the loss is not too large if the general algorithm is implemented and is occasionally fed with an affine class.

2. In characteristic 2 one saves an additional multiplication in Step 6 as $R(h_1Z_2 + 2V_{21}) = \tilde{R}h_1$ and h_1 is assumed to be small.

5 Doubling

For the doubling algorithm the input is almost always in projective representation.

Doubling

Input	$[U_1, U_0, V_1, V_0, Z]$	
	$h = h_2x^2 + h_1x + h_0, f = x^5 + f_4x^4 + f_3x^3 + f_2x^2 + f_1x + f_0$	
Output	$[U_1', U_0', V_1', V_0', Z'] = 2[U_1, U_0, V_1, V_0, Z]$	
Step	Expression	Operations
1	compute resultant and precomputations: $Z_2 = Z^2$, $\tilde{V}_1 = h_1Z + 2V_1 - h_2U_1$, $\tilde{V}_0 = h_0Z + 2V_0 - h_2U_0$; $w_0 = V_1^2$, $w_1 = U_1^2$, $w_2 = \tilde{V}_1^2$, $w_3 = \tilde{V}_0Z - U_1\tilde{V}_1$; $r = \tilde{V}_0w_3 + w_2U_0$;	3S, 4M (see below)
2	compute almost inverse: $inv_1 = -\tilde{V}_1$, $inv_0 = w_3$;	
3	compute k: $w_3 = f_3Z_2 + w_1$, $w_4 = 2U_0$; $k_1 = 2w_1 + w_3 - Z(w_4 + 2f_4U_1 + h_2V_1)$; $k_0 = U_1(Z(2w_4 + f_4U_1 + h_2V_1) - w_3) +$ $+Z(Z(f_2Z - V_1h_1 - V_0h_2 - 2f_4U_0) - w_0)$;	7M
4	compute $s = kinv \bmod u$: $w_0 = k_0inv_0$, $w_1 = k_1inv_1$; $s_3 = (inv_0 + inv_1)(k_0 + k_1) - w_0 - (1 + U_1)w_1$; $s_1 = s_3Z$, $s_0 = w_0 - ZU_0w_1$; If $s_1 = 0$ different case	7M
5	precomputations: $R = Z_2r$, $\tilde{R} = Rs_1$, $S_1 = s_1^2$, $S_0 = s_0^2$, $t = h_2s_0$; $s_1 = s_1s_3$, $s_0 = s_0s_3$, $S = s_0Z$, $\tilde{\tilde{R}} = \tilde{R}s_1$;	2S, 6M
6	compute l: $l_2 = U_1s_1$, $l_0 = U_0s_0$, $l_1 = (s_1 + s_0)(U_1 + U_0) - l_2 - l_0$;	3M
7	compute U': $U_0' = S_0 + R(s_3(2V_1 - h_2U_1 + h_1Z) + t + Zr(2U_1 - f_4Z))$; $U_1' = 2S + h_2\tilde{R} - R^2$;	1S, 4M
8	precomputations: $l_2 = l_2 + S - U_1'$, $w_0 = U_0'l_2 - S_1l_0$, $w_1 = U_1'l_2 + S_1(U_0' - l_1)$;	4M
9	adjust: $Z' = S_1\tilde{R}$, $U_1' = \tilde{R}U_1'$, $U_0' = \tilde{R}U_0'$;	3M
10	compute V': $V_0' = w_0 + h_2U_0' - \tilde{\tilde{R}}V_0 - h_0Z'$; $V_1' = w_1 + h_2U_1' - \tilde{\tilde{R}}V_1 - h_1Z'$;	2M
total		6S, 40M

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.

2. Concerning the counting in Step 1 a remark is in order. Unless the characteristic is odd and $h \neq 0$, the computation of Z^2, V_1^2, U_1^2 and $\tilde{V}_1^2$ needs only 3 squarings instead of the obvious 4. (In detail: if for odd characteristic $h = 0$ then $\tilde{V}_1^2 = 4V_1^2$. If $p = 2$ then $\tilde{V}_1^2 = h_2^2 U_1^2 + h_1^2 Z^2$ and $h_2, h_1 \in \{0, 1\}$. These details can be fixed for an actual implementation.)

3. The numbers and formulae in the algorithm present the best average; for any specific choice of p, h and f some operations can be left out or be replaced by cheaper ones. We mention the following.

   (a) If $f_3 = 0$ one saves 1S and 1M as $Z_2 = Z^2$ and $Z_2 f_3$ need not be computed. R is computed via $R = (rZ)Z$, as rZ is needed anyway, unless $p = 2$ and $f_4 = 0$

   (b) If in odd characteristic $h = 0$ and $f_4 = 0$ one saves 2M in the computation of $k_0 = U_1(2Zw_4 - w_3) + Z(f_2 Z_2 - w_0)$ as Zw_4 is already obtained during the computation of k_1.

   (c) In characteristic 2 we can do even better. In the first step, $\tilde{V}_1 = h_1 Z + h_2 U_1$ and $\tilde{V}_0 = h_0 Z + h_2 U_0$. Therefore we have $w_3 = h_0 Z_2 + Z(h_2 U_0 + h_1 U_1) + h_2 w_1$ which can be computed directly, hence, saving 1 multiplication. In Step 3 we replace $w_4 = ZV_1$ and use it in k_1 and k_0 leading to $6M$ in this step. Furthermore, U_0' can be computed with only two multiplications as $U_0' = S_0 + R(s_3(h_2 U_1 + h_1 Z) + t + R f_4)$ reducing the number of multiplications by $2M$. In total this is -4M.
   If additionally $h_2 = 0$ the number of multiplications can even be reduced to 33M: In Step 1 the computation of w_2 is not needed, w_3 can be done with 1 multiplication like above and $r = Z_2(h_0^2 Z + h_0 h_1 U_1 + h_1^2 U_0)$. In Step 3, $w_4 = 0$ and the multiplication to obtain k_1 vanishes. Including the computation of $R_2 = R^2$ in the precomputation Step 5 allows to reduce the number of multiplications by two as $U_0' = S_0 + h_1 \tilde{R} + f_4 R_2$ and $U_1' = 2S + R_2$.

   (d) Using binary Koblitz curves one has $h, f \in \mathbb{F}_2[x]$, thus multiplications by f_2 and f_3 need not be counted, there are 2 such occurrences.
   Combining with what was said above, the minimal number (up to my present knowledge ...) of operations for a doubling in this case with $h_2 = 0$ is

   6S, 31M.

6 General Remarks and Outlook

We gave algorithms to perform inversion free arithmetic on hyperelliptic genus two curves that are faster than any previous and also dealt with mixed addition. The practical implementation on smart cards is investigated, first results will be presented at *ECC 2002 - The 6th Workshop on Elliptic Curve Cryptography.*
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 $Z_1^2 Z_2$ 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'.
The author is currently investigating a generalization of weighted projective coordinates to genus two curves not at least to avoid these extra multiplications in the adjustment step. Additionally she looks for optimal matches to use mixed coordinates (this time the term 'mixed coordinates' in the meaning of [2]).

References

[1] R.M. Avanzi. On multi-exponentiation in cryptography. Preprint.

[2] 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, New–York, 1998.

[3] T. Lange. Efficient Arithmetic on Genus 2 Hyperelliptic Curves over Finite Fields via Explicit Formulae. Cryptology ePrint Archive, Report 2002/121, 2002. `http://eprint.iacr.org/` or `http://www.itsc.ruhr-uni-bochum.de/tanja`.

[4] 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.

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