Empirical optimization of divisor arithmetic on hyperelliptic curves over F_2m

Laurent Imbert, Michael Jacobson

▶ To cite this version:

Laurent Imbert, Michael Jacobson. Empirical optimization of divisor arithmetic on hyperelliptic curves over F_2m. RR-13008, 2012, pp.18. lirmm-00815484

HAL Id: lirmm-00815484
https://hal-lirmm.ccsd.cnrs.fr/lirmm-00815484v1

Submitted on 18 Apr 2013

Manuscript submitted to
AIMS' Journals
Volume **X**, Number **0X**, XX **200X**

Website: http://AIMsciences.org

pp. **X–XX**

EMPIRICAL OPTIMIZATION OF DIVISOR ARITHMETIC ON HYPERELLIPTIC CURVES OVER $\mathbb{F}_{2^m}$

Laurent Imbert

CNRS, LIRMM
Université Montpellier 2
161 rue Ada
F-34095 Montpellier, France

Michael J. Jacobson, Jr.

Department of Computer Science
University of Calgary
2500 University Drive NW
Calgary, Alberta, Canada T2N 1N4

(Communicated by *editor's name*)

Abstract. A significant amount of effort has been devoted to improving divisor arithmetic on low-genus hyperelliptic curves via explicit versions of generic algorithms. Moderate and high genus curves also arise in cryptographic applications, for example, via the Weil descent attack on the elliptic curve discrete logarithm problem, but for these curves, the generic algorithms are to date the most efficient available. Nagao [21] described how some of the techniques used in deriving efficient explicit formulas can be used to speed up divisor arithmetic using Cantor's algorithm on curves of arbitrary genus. In this paper, we describe how Nagao's methods, together with a sub-quadratic complexity partial extended Euclidean algorithm using the half-gcd algorithm can be applied to improve arithmetic in the degree zero divisor class group. We present numerical results showing which combination of techniques is more efficient for hyperelliptic curves over $\mathbb{F}_{2^n}$ of various genera.

1. Introduction.

Hyperelliptic curves defined over finite fields were first proposed for cryptographic use by Koblitz [18] in 1989, with the special case of genus one curves (elliptic curves) having been proposed earlier by Koblitz and Miller independently [19, 17]. The idea is to base protocols on arithmetic in the degree zero divisor class group (or Picard group) of the curve, a finite abelian group, basing security on the assumption that the discrete logarithm problem in that group is intractable. Further research has shown that the discrete logarithm problem becomes less intractable as the genus increases, leaving elliptic curves (genus one) and low-genus hyperelliptic curves (genus two and, perhaps, three) as the main settings of choice for cryptographic applications.

2000 *Mathematics Subject Classification.* Primary: 94A60, 14H45; Secondary: 14Q05.

Key words and phrases. Hyperelliptic curve, divisor addition, Cantor's algorithm, NUCOMP, half-gcd algorithm.

This research was partly supported by the first author's grants CNRS-PICS05886 and ANR-12-BS02-002-02. The second author is supported in part by NSERC of Canada.

In the case of elliptic curves, arithmetic in the class group is described geometrically using the fact that this group is isomorphic to the group of points on the curve. For hyperelliptic curves, the group law is described algebraically using an algorithm due to Cantor [2], and expressed in terms of polynomial arithmetic. Various improvements and extensions to Cantor's algorithm have been proposed, for example an adaptation of Shanks' NUCOMP algorithm for composing binary quadratic forms [13] and a tripling formula [11][1]. For curves of low genus, specifically four or less, optimized versions described in terms of arithmetic with field elements are used; see [4, Ch. 14] for a survey. The vast majority of work has focused on such formulas, due to the direct applicability of low-genus curves to discrete logarithm based cryptography.

However, mid- to high-genus curves also have cryptographic applications, most notably through the Weil descent attack initially suggested by Frey [8] and later refined by Gaudry, Hess, and Smart [9]. This attack, when successful, reduces the discrete logarithm problem on an elliptic curve defined over a composite degree extension of $\mathbb{F}_2$ to that on a higher-genus hyperelliptic curve over a smaller finite field, where the discrete logarithm problem may be easier to solve. One notable example is elliptic curves defined over $\mathbb{F}_{2^{155}}$. In the best case, these can be reduced to genus 31 curves over $\mathbb{F}_{2^5}$, where the discrete logarithm problem has been shown to be solvable in practice [24]. The Oakley key determination protocol, part of an IETF standard, required an elliptic curve defined over this field and another over $\mathbb{F}_{2^{185}}$, and these curves were recently showed by Musson [20] to reduce to genus 16 hyperelliptic curves over $\mathbb{F}_{2^{31}}$ or $\mathbb{F}_{2^{37}}$. Teske also describes a constructive application in which Weil descent is used to furnish a trap-door for an instance of the elliptic curve discrete logarithm problem. One recommended set of parameters results in a hyperelliptic curve of genus 7 or 8 defined over $\mathbb{F}_{2^{23}}$.

For all these cases, explicit formulas have not been developed for the group arithmetic, so Cantor's algorithm or its variants would have to be used. Compared to explicit formulas, relatively little work has been done on optimizing these directly. Nagao [21] described how some of the techniques for optimizing explicit formulas can be applied to generic formulation of Cantor's algorithm. These include eliminating inversions in extended greatest common divisor computation using pseudodivision, and partially computing products when a subsequent division is known to be exact. Nagao's experiments suggest that partial multiplication is the most useful, with some small improvements with pseudodivision.

In this paper, we extend Nagao's work by conducting a thorough empirical investigation of divisor class group arithmetic for hyperelliptic curves defined over binary fields. In addition to Cantor's algorithm, we apply similar optimizations to NUCOMP [13], the tripling formula introduced in [11], and the reduction algorithm proposed in [1]. Our primary motivation was to identify the fastest methods for divisor class group arithmetic in the cryptographically-interesting cases arising from the Weil descent mentioned above. Our second motivation was to optimize divisor class group arithmetic in general, by learning how the different algorithms and optimizations perform as functions of the genus and finite field sizes. The goal was to investigate the main algorithmic techniques available in an effort to identify the fastest overall approach to divisor arithmetic for various field size and genera. For example, the performance results for the reduction algorithm in [1] were somewhat

[1] The formula given in [11] is for cubing an ideal in an imaginary function field. The translation to the degree zero divisor class group is immediate.

inconclusive; through a careful implementation using all the known optimizations, we hoped to conclusively evaluate the performance of this algorithm with respect to that of Cantor.

We begin by numerically comparing algorithms for solving the extended GCD problem, a core component of divisor class group arithmetic, in order to find thresholds indicating which of Euclid's algorithm, Euclid's algorithm with pseudodivision, and the sub-quadratic half-gcd algorithm is optimal for various combinations of field size and polynomial degree. We do the same for the partial extended GCD computation required for NUCOMP and the reduction algorithm of [1], using a variation of the partial half-gcd algorithm suggested in the same context by Ding [6]. Improved threshold-based GCD algorithms are then applied to Cantor, NUCOMP, fast reduction, together with partial versions of polynomial multiplication using the basic quadratic-complexity algorithm. Extensive numerical results are reported, indicating which combination of methods is the more efficient for various combinations of genus and field size. The improvements obtained for the Weil descent examples are modest; however, our results give a clear picture of how the various algorithms and optimizations perform for different finite field and genus combinations. To the best of our knowledge, we have used the most relevant existing algorithmic techniques at our disposal; our conclusions suggest that further improvements are expected to require either significant new arithmetic algorithms or low-level code optimization.

The software used to test these algorithms is organized into a C++ library. The best collection of algorithms is selected adaptively based on the finite field size and curve genus, together with our experimentally-derived set of thresholds. The code is available upon request, and will be released publicly in the near future.

2. **Divisor Arithmetic on Hyperelliptic Curves.**

For background on hyperelliptic curves, see [4].

A hyperelliptic curve of genus g over a field $\mathbb{K}$ of characteristic 2 is an absolutely irreducible non-singular curve given by an equation of the form

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

where $h, f \in \mathbb{K}[x]$. In this paper, we restrict our attention to the imaginary model, so the equation can be normalized by taking f monic, $\deg(f) = 2g+1$ and $deg(h) \leq g$, where g is the genus of the curve.

Unless the genus of the curve is one (an elliptic curve), the points on the curve themselves do not form a group. Instead, one works with the degree zero divisor class group, the group of equivalence classes of degree zero divisors modulo principal divisors. It is well known that this group is isomorphic to the ideal class group of the function field $\mathbb{K}(\mathcal{C})$. Making this isomorphism explicit, each nontrivial divisor class over $\mathbb{K}$ can be represented uniquely as a pair of polynomials $(u(x), v(x)), u, v \in \mathbb{K}[x]$, where u is monic, $\deg(v) < \deg(u) \leq g$ and $u \mid v^2 + vh + f$. This is known as the Mumford representation of a reduced divisor.

Addition in the degree zero divisor class group is computed using an algorithm due to Cantor [2]. Given $D' = (u', v')$ and $D'' = (u'', v'')$ two reduced divisors in Mumford representation, Cantor's algorithm computes the reduced divisor $D = D' + D'' = (u, v)$. The formulation presented in Algorithm 1 is optimized for the frequently-occurring situation where $\gcd(u', u'') = 1$, based on a description due to Shanks of Gauss's composition formulas for binary quadratic forms. A more efficient doubling formula can be obtained by optimizing in the case that $D' = D''$, and an efficient tripling formula is given in [11, Algorithm 1]; the output of this algorithm requires subsequent reduction using the last part of Algorithm 1 (steps 17 to 28).

Algorithm 1 Addition of Divisor Classes (Cantor)

```
Input: $D' = (u', v')$, $w' = (f+hv'+(v')^2)/u'$, $D'' = (u'', v'')$, $w'' = (f+hv''+(v'')^2)$
Output: Reduced $D = D' + D'' = (u, v)$ and $w = (f + hv + v^2)/u$
 1: {Addition}
 2: $S = v_1u' + u_1u''$    {only compute $S$ and $v_1$}
 3: $t_1 = v' + v''$
 4: $K = v_1t_1 \bmod u'$
 5: if $S \neq 1$ then
 6:    $t_2 = t_1 + h$
 7:    $S = u_2S + v_2t_2$
 8:    $K = u_2K + v_2w''$
 9:    $u' = u'/S$,  $u'' = u''/S$,  $w'' = w''S$    {exact divisions}
10:    $K = K \bmod u'$
11: $T = u''K$
12: $u = u'u''$
13: $v = b'' + T$
14: $w = (Sw'' + K(h + T))/u'$    {exact division}
15: $w = w\,\mathrm{LeadCoeff}(u)$,  $u = u/\mathrm{LeadCoeff}(u)$
16: {Reduction}
17: if $\deg(u) \leq g$ then
18:    if $\deg(v) \geq \deg(u)$ then
19:       Compute $q, r$ such that $b = qu + r$
20:       $w = w - q(v + r + h)$
21:       $v = r$
22: else
23:    while $\deg(u) > g$ do
24:       $t_u = w$
25:       Compute $q, t_v$ such that $v + h = qt_u + t_v$
26:       $w = u + qt_v + v$
27:       $v = t_v$,  $u = t_u$
28:    $w = w\,\mathrm{LeadCoeff}(u)$, $u = u/\mathrm{LeadCoeff}(u)$
```

One drawback of all these algorithms is that they can have relatively large intermediate operands. Assuming that the input divisors are reduced (coefficients have degree $\leq g$), the degrees of u and v prior to reduction can be as large as $2g$ for addition and doubling, and $3g$ for tripling. The NUCOMP algorithm of Shanks, adapted to hyperelliptic curves [14, 13], seeks to reduce the sizes of the intermediate operands by carrying out a reduction on them before completing the addition step. In essence, the reduction step, which can be interpreted as computing the continued fraction expansion of an element of the function field [13], is replaced by a rational approximation of the continued fraction. In addition to reducing the sizes of intermediate operands, we can achieve further speed-ups by using fast extended GCD algorithms for this step.

In Algorithm 2, we give an optimized description of NUCOMP following the presentation in [15] for quadratic number fields. A doubling version (NUDUPL) can be obtained by setting $D' = D''$ and simplifying. A tripling version (NUTRIPL) can be immediately derived from the NUCUBE algorithm presented in [11, Algorithm 2]. In line 19 of Algorithm 2, the call to XGCD-PARTIAL(R_2, R_1, C_2, C_1, n) computes the decreasing Euclidean remainder sequence starting with R_2, R_1 (together with the Bezout's coefficients) until reaching a remainder of degree less then or equal to the given bound n.

Algorithm 2 NUCOMP

```
Input: $D' = (u', v')$, $w' = (f+hv'+(v')^2)/u'$, $D'' = (u'', v'')$, $w'' = (f+hv''+(v'')^2)$
Output: Reduced $D = D' + D'' = (u, v)$ and $w = (f + hv + v^2)/u$
 1: {Addition}
 2: $S = v_1 u' + u_1 u''$    {only compute $S$ and $v_1$}
 3: $t_1 = v' + v''$,  $t_2 = t_1 + h$
 4: $K = v_1 t_1 \bmod u'$
 5: if $S \neq 1$ then
 6:     $S = u_2 S + v_2 t_2$
 7:     $K = u_2 K + v_2 w''$
 8:     $u' = u'/S$,  $u'' = u''/S$,  $w'' = w'' S$    {exact divisions}
 9:     $K = K \bmod u'$
10: if $\deg(u') + \deg(u'') \leq g$ then
11:     {Sum will already be reduced}
12:     $T = u'' K$
13:     $u = u' u''$
14:     $v = b'' + T$
15:     $w = (S w'' + K(h + T))/u'$    {exact division}
16: else
17:     {Apply partial reduction before computing the sum}
18:     $R_2 = u'$, $R_1 = K$
19:     XGCD-PARTIAL$(R_2, R_1, C_2, C_1, (\deg(u') + \deg(u'') + g)/2)$
20:     $t_3 = u'' R_1$
21:     $M_1 = (t_3 + t_1 C_1)/u'$    {exact division}
22:     $M_2 = (m R_1 + w'' C_1)/u'$    {exact division}
23:     $u = M_1 R_1 + M_2 C_1$
24:     $v = v'' + h + (t_3 + u C_2)/C_1 \bmod u$    {exact division}
25:     $(v^2 + vh + f)/u$    {exact division}
26: $w = w\,\mathrm{LeadCoeff}(u)$, $u = u/\mathrm{LeadCoeff}(u)$
27: Compute $q, r$ such that $b = qu + r$
28: $w = w - q(v + r + h)$
29: $v = r$
```

The main idea of NUCOMP is that the element $(v + y)/u$ of the function field of $\mathcal{C}$ can be approximated by the rational u'/K. Cantor's algorithm first computes the non-reduced divisor $D = (u, v)$, and subsequently applies a reduction algorithm that can be expressed in terms of expanding the continued fraction of $(v + y)/u$. The rational approximation by u'/K allows us to compute the same sequence of partial quotients using the computationally simpler Euclidean algorithm (and various optimized version). It also allows us to avoid computing the intermediate non-reduced divisor D, and instead express the final reduced divisor in terms of formulas involving lower-degree operands.

A similar observation can be used to speed the reduction algorithm itself. On input a non-reduced divisor $D = (u, v)$, the rational v/u is used to approximate $(v + y)/u$. This algorithm is a generalization of Sawilla's algorithm for quadratic number fields [12], and is described in [1]. Algorithm 3 describes the fast reduction algorithm using the same notation as the algorithms above. The algorithm XGCD-PARTIAL-REDUCE is the same as XGCD-PARTIAL, except that the algorithm takes one additional step if $\deg(R_1) = n$ and $\deg(u') - g$ is odd.

Algorithm 3 Fast Reduction

```
Input: Non-reduced divisor $D' = (u', v')$, $w' = (f + hv' + (v')^2)/u'$
Output: Equivalent reduced divisor $D = (u, v)$ and $w = (f + hv + v^2)/u$
$w = w \,\mathrm{LeadCoeff}(u')$, $u' = u'/\mathrm{LeadCoeff}(u')$
if $\deg(u') \leq g$ then
    if $\deg(v') \geq \deg(u')$ then
        Compute $q, r$ such that $b = qu + r$
        $w' = w' - q(v' + r + h)$
        $v' = r$
else
    $R_2 = v$,  $R_1 = u'$,  $t_u = u'$
    XGCD-PARTIAL-REDUCE($R_2, R_1, C_2, C_1, (\deg(u') + g + 1)/2$)
    $u = (R_1^2 + R_1 C_1 h + f C_1^2)/t_u$     {exact division}
    $v = h + (R_1 + u C_2)/C_1$     {exact division}
    $u = u/\mathrm{LeadCoeff}(u)$
    $v = v \bmod u$
    $w = (v^2 + vh + f)/u$     {exact division}
```

As can be seen, three different types of extended GCD computations are required for the above algorithms. For computing S, the divisor addition, doubling, and tripling algorithms all require an XGCD computation where only one of the two Bezout coefficients is computed. A full XGCD computation is required in the relatively rare case that the first GCD is not one. Finally, a partial XGCD computation, where the algorithm terminates once the remainders computed reach a given degree bound, is required for both NUCOMP and the analogous reduction algorithm. In the sequel, our main optimizations focus on these variants of the Euclidean algorithm. As pointed out in the comments of Algorithms 1–3, all divisions are exact. Hence, we also optimize these various exact divisions by computing only the required terms of the dividends.

3. **Extended GCD algorithms for polynomials.** As shown in the previous section, polynomial GCD computation is a pivotal component of arithmetic in the divisor class group. Our first level of optimization begins there. We consider the following three cases:

- XGCD: full extended GCD computation,
- XGCD-LEFT: extended GCD where only one of the Bezout coefficients is computed,
- XGCD-PARTIAL and XGCD-PARTIAL-REDUCE: partial extended GCD where the algorithm terminates once the operands reach a given degree bound.

Let A, B be two polynomials in $\mathbb{K}[X]$ with $B \neq 0$. We denote by Q and R the unique polynomials in $\mathbb{K}[X]$ given by the Euclidean division of A by B such that

$$A = BQ + R, \quad \text{with } \deg(R) < \deg(B). \tag{2}$$

The greatest common divisor (up to units) of A and B is the least remainder different from 0 in the Euclidean remainder sequence defined by $R_0 = A$, $R_1 = B$ and for $i > 1$, $R_{i+1} = R_{i-1} \bmod R_i$, which can be expressed in matrix form as

$$\begin{pmatrix} R_i \\ R_{i+1} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & -Q_i \end{pmatrix} \begin{pmatrix} R_{i-1} \\ R_i \end{pmatrix}.$$

Multiplying these Euclidean matrices $\left(\begin{smallmatrix} 0 & 1 \\ 1 & -Q_i \end{smallmatrix}\right)$ together (to the left), gives

$$\begin{pmatrix} R_n \\ 0 \end{pmatrix} = \begin{pmatrix} U_n & V_n \\ U_{n+1} & V_{n+1} \end{pmatrix} \begin{pmatrix} A \\ B \end{pmatrix}$$

where $R_n = \gcd(A, B) = AU_n + BV_n$.

For efficiency, it is recommended to make the polynomials monic in order to avoid field inversions in the course of the polynomial division. Note that this still requires one field inversion per Euclidean step, which might have a significant impact on the overall performance depending on the field size.

Alternatively, one can use Knuth's pseudo-division [16, 3], which requires no field inversion[2]. This is done by computing the pseudo-quotient Q' and pseudo-remainder R' such that

$$\ell(B)^{\deg(A)-\deg(B)+1} A = BQ' + R', \quad \text{with } \deg(R') < \deg(B), \tag{3}$$

where $\ell(B)$ denotes the leading coefficient of B. Using this pseudo-division, a greatest common divisor of A and B together with the Bezout's coefficients can be computed with no field inversion[3] at the extra cost of some field multiplications. Indeed, each Euclidean step now consists in computing

$$\begin{pmatrix} R_i \\ R_{i+1} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ \ell(R_i) & -Q_i \end{pmatrix} \begin{pmatrix} R_{i-1} \\ R_i \end{pmatrix}.$$

This is one of the optimizations suggested by Nagao [21] for improving Cantor's algorithm, and the numerical results suggest a modest improvement in general. However, whether or not pseudo-division is advantageous in the context of extended GCD and ideal arithmetic depends on several parameters, in particular the cost of a field inversion and the degrees of the input polynomials.

The algorithm XGCD-PARTIAL computes a given remainder in the remainder sequence. More specifically, given A, B with $\deg B < \deg A$, and $d_{red} < \deg A$, it computes a matrix M as above such that $M(A, B)^T = (R_{j-1}, R_j)^T$, where R_j is the first remainder of degree less than or equal to $\deg(A) - d_{red}$.

A faster method of subquadratic complexity, known as the half-gcd algorithm, aims at computing the matrix M without evaluating all the quotients Q_i. Assuming $\deg(B) < \deg(A) = n$, it does so using a divide-and-conquer strategy by computing a matrix M_h such that $M_h(A, B)^T = (R_{j-1}, R_j)$, where R_j is the first remainder of degree less than $n/2$. A recursive call with inputs of degree $\approx n/2$ then gives remainders of degree $\approx n/4$, then $\approx n/8$, *etc.* until one gets the greatest common

[2] Pseudo-division was originally proposed to deal with polynomials defined over a domain (UFD) that is not a field.

[3] Except one at the very end if one wants the greatest common divisor to be monic.

divisor. It can be proved that the complexity of this algorithm is essentially that of computing M_h, which can be done in subquadratic time using Algorithm 4.

Algorithm 4 HGCD

```
Input: $A, B \in \mathbb{K}[X]$ with $\deg B < \deg A = n$
Output: The matrix $M$ such that $M(A, B)^T = (A', B')^T$ with $\deg B' < n/2$ and
    $\deg A' \geq n/2$.
1: Let $m = \lceil n/2 \rceil$. If $\deg B < m$, return the identity matrix
2: Let $A_1 = A/X^m$, $B_1 = B/X^m$
        {$A_1, B_1$ have degree approximately $n/2$}
3: Recursively compute $M1 = HGCD(A_1, B_1)$ and $A', B'$ such that $M_1(A, B)^T =$
   $(A', B')^T$
4: If $\deg B' < m$ return $M_1$
5: Let $R' = A' \bmod B'$ and $M'$ the corresponding Euclidean matrix
        {After step 3, $\deg B'$ is at most $3n/4$ but $\deg A'$ might be greater.
        This division step allows to reduce both polynomials to degrees
        $\approx 3n/4$}
6: Let $\ell = 2m - \deg B'$. Compute $B'_1 = B'/X^\ell$ and $C'_1 = C'/X^\ell$
7: Recursively compute $M_2 = HGCD(B'_1, C'_1)$
8: Return $M_2 M' M_1$
```

The half-gcd algorithm requires $O(M(n) \log n)$ operations in $\mathbb{K}$. It can be adapted to XGCD-PARTIAL in the same complexity. As noted in [6], this subquadratic complexity naturally extends to the problem of computing a reduced sum of two divisors of a hyperelliptic curve.

3.1. **Analysis and Profiling.** We compared the relative performances of the GCD algorithms presented above for polynomials of degree d ranging from 2 to 100 over binary fields $\mathbb{F}_{2^m}$ of sizes from 2^2 to 2^{2048}. Our algorithms are written in C++ (compiled using g++ version 4.4.5) and the experiments were performed on a workstation with 64 Intel Xeon cores, each of which is 64-bit and runs at 2.27 GHz. We used the NTL [22] computer algebra library for finite field and polynomial arithmetic, built on gf2x (http://gf2x.gforge.inria.fr/) for faster polynomial arithmetic over $\mathbb{F}_2$.

For the three gcd computations of interest mentioned above (XGCD, XGCD-LEFT, XGCD-PARTIAL) we ran many simulations and we compared the timings given by the following three approaches: subquadratic half-gcd, plain extended euclidean with pseudo-division, and plain extended euclidean algorithm without pseudo-division. For the later, we made polynomials monic at each step using a field inversion. In Table 1, we summarize our experimental results for XGCD-LEFT by giving the ratio between the time taken by the fastest method and the plain euclidean algorithm with division. The different gray areas clearly show the best strategy for a given set of parameters (input degree, field size). The cell tags should be self-explanatory.

Based on our experiments, pseudo-division is never interesting for word-size binary fields, *i.e.* for $m \leq 64$ and is only faster for larger fields ($m > 64$) for polynomials of degree at most 20. In the most favorable cases, the pseudo-division alternative

Table 1. Fastest XGCD-LEFT algorithms for polynomials of degree n defined over $\mathbb{F}_{2^m}$

n	m										
	2	4	8	16	32	64	128	256	512	1024	2048
2	1.00	1.00	1.00	1.00	1.00	1.00	0.62	0.52	0.49	0.48	0.47
3	1.00	1.00	1.00	1.00	1.00	1.00	0.60	0.50	0.48	0.45	0.45
4	1.00	1.00	1.00	1.00	1.00	1.00	0.61	0.53	0.50	0.49	0.47
5	1.00	1.00	1.00	1.00	1.00	1.00	0.65	0.54	0.53	0.52	0.51
10	1.00	1.00	1.00	1.00	1.00	1.00	0.83	0.73	0.71	0.72	0.71
15	1.00	1.00	1.00	1.00	1.00	1.00	0.97	0.88	0.87	0.89	0.87
20	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.99	1.00	1.00
30	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
40	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
50	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
60	1.00	0.97	0.97	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
70	0.99	0.91	0.92	0.96	0.99	1.00	1.00	1.00	1.00	1.00	1.00
80	0.94	0.82	0.82	0.87	0.93	0.99	1.00	1.00	1.00	1.00	1.00
90	0.88	0.76	0.76	0.80	0.85	0.92	1.00	1.00	1.00	1.00	1.00
100	0.82	0.71	0.71	0.76	0.80	0.88	1.00	1.00	1.00	1.00	1.00
150	0.58	0.55	0.55	0.59	0.65	0.70	1.00	1.00	1.00	1.00	1.00
200	0.53	0.45	0.45	0.49	0.53	0.59	1.00	1.00	1.00	1.00	1.00
250	0.45	0.38	0.39	0.42	0.46	0.51	0.90	0.85	0.85	0.85	0.82

Euclid
Pseudo
Half

can be twice as fast as the classical Euclid's algorithm. Not surprisingly, half-gcd is advantageous for polynomials of degree ≥ 30 when $m \leq 64$ and for polynomials of degree ≥ 60 for $m > 64$. Again, in the most favorable cases, it can be twice as fast as the plain euclidean algorithm.

Using a similar empirical strategy, we defined thresholds for plain XGCD (computing the two Bezout's coefficients) and XGCD-PARTIAL. For the latter, the thresholds also depend on the number of Euclidean steps that have to be performed to get the expected remainder. Those parameters are all taken into account by our library in order to select the best strategy according to the various parameters (genus, field size, input degrees).

4. **Partial Multiplication for Exact Divisions.** Our second level of optimization concerns the algorithms themselves. An important technique in generating explicit formulas is to make use of the fact that certain polynomial divisions are exact. Supposing the divisor has degree n, the n low degree terms of the dividend are not required, and can be omitted in the multiplication and squaring operations used to compute the dividend. As observed by Nagao [21], this is applicable to Cantor's algorithm, and numerical results suggest that this is an effective technique.

The problem is that it is only known how to achieve an efficient modification of the basic quadratic-complexity multiplication algorithm in this manner. Faster methods such as Karatsuba multiplication and FFT are not as amenable. Thus, we should not expect this method to improve divisor class arithmetic in all cases. Instead, we should expect a threshold below which the modified multiplication yields an improvement and after which the faster multiplication algorithms (aka sub-quadratic or quasi-linear) are superior.

We have implemented a partial multiply routine based on the `MulPlain` function in NTL's `GF2EX` class. This routine applies the basic school book multiplication algorithm, but only computes the coefficients of the product polynomial for terms of degree greater than a given bound. In Table 2, we provide a comparison of our partial multiply routine with the basic multiply routine in NTL.

Table 2. Fastest partial multiply algorithms for hyperelliptic curves defined over $\mathbb{F}_{2^{16}}$

d	$i =$ bit length of $d - n$						
	1	2	3	4	5	6	7
2	0.86						
3	0.57	1.00					
4	0.67	0.90					
5	0.48	0.58	0.96				
10	0.47	0.49	0.76	1.00			
20	0.34	0.35	0.51	0.90	1.00		
30	0.27	0.31	0.42	0.67	1.00		
40	0.23	0.27	0.35	0.54	1.00	1.00	
50	0.21	0.24	0.31	0.45	0.91	1.00	
60	0.20	0.21	0.27	0.41	0.80	1.00	
70	0.16	0.20	0.24	0.35	0.69	1.00	1.00
80	0.16	0.18	0.22	0.32	0.61	1.00	1.00
90	0.15	0.16	0.20	0.28	0.54	1.00	1.00
100	0.15	0.16	0.20	0.27	0.50	1.00	1.00

NTL

partial

For polynomials defined over $\mathbb{F}_{2^m}$, NTL uses a combination of a quadratic algorithm and Karatsuba, so we expect it to be superior asymptotically. The table gives the ratio of the average running time to compute a partial multiply over the time to compute the same product with NTL's `mul` method, and each cell is shaded according to which of the two algorithms was faster. The field d indicates the degree of the resulting product, and i denotes the bit length of $d - n$, *i.e.* the bit length of the number of terms in the product that must be computed, from 1 to $d/2$. This is the required range of operands for application to the divisor arithmetic algorithms described earlier, the typical case being $d/2$. The tables for different finite fields look similar.

The table shows that the partial multiplication algorithm is most efficient when the number of terms required in the product is small as compared to the degree of the output. For all but the smallest output degrees, NTL was faster when half the terms were required.

We used this data to set thresholds with which our ideal arithmetic programs can automatically switch between the partial multiplication and NTL multiplication routines. The threshold is a function of the finite field size and bit length of the difference between the output degree and bound on the terms to compute.

Notice that in the case of partial squaring, a partial algorithm will always be superior. This is because in characteristic two, the basic quadratic simplifies to a linear complexity algorithm, and this can be made partial as easily as the full algorithm.

5. **Empirical Optimization of Divisor Arithmetic.** We have implemented addition, doubling, and tripling algorithms using both Cantor's algorithm (Algorithm 1) and NUCOMP (Algorithm 2), using the specializations for doubling and tripling mentioned in Section 2. The performance of each algorithm was benchmarked, in order to set thresholds based on the genus and finite field determining which algorithm is more efficient for particular hyperelliptic curves.

We have compared the practical performance of two variations of each algorithm. The first one uses none of the optimizations described above. For the various XGCD algorithms, we simply used the implementations in NTL [22] modified to compute one Bezout coefficient or to terminate once the remainders reach a given degree bound. For polynomials with coefficients in a characteristic two finite field, this meant that the standard extended Euclidean algorithm was used. The second variation makes use of the XGCD and partial multiplication optimizations described in the previous two sections. Mechanisms were implemented by which the best variation of a particular algorithm was automatically used, based on the thresholds presented above.

5.1. **Divisor Addition.** In Table 3, we provide a comparison of the four resulting variations of divisor addition: Cantor's algorithm and NUCOMP without using our improved XGCD and partial multiplication, and both algorithms with our improvements. Using Cantor's algorithm without our improvements as a baseline, we list for each combination of genus and field size the ratio of the fastest of the four algorithms with Cantor's. Shading is used to indicate which of the four was the fastest.

Table 3. Fastest algorithms for divisor addition in hyperelliptic curves defined over binary fields $\mathbb{F}_{2^m}$

g	m										
	2	4	8	16	32	64	128	256	512	1024	2048
5	░ 1.00	░ 1.00	░ 1.00	░ 1.00	░ 1.00	▓ 0.98	▓ 0.89	▓ 0.84	▓ 0.82	▓ 0.81	▓ 0.82
6	█ 1.00	░ 1.00	░ 1.00	░ 1.00	▓ 0.99	█ 0.96	█ 0.87	▓ 0.85	▓ 0.83	█ 0.83	█ 0.82
7	▓ 0.94	░ 1.00	░ 1.00	▓ 0.99	█ 0.95	█ 0.94	█ 0.87	█ 0.86	█ 0.85	█ 0.84	█ 0.82
8	█ 1.00	▒ 0.99	░ 1.00	█ 0.94	█ 0.93	█ 0.93	█ 0.89	█ 0.84	█ 0.84	█ 0.82	█ 0.83
9	█ 1.00	▒ 0.96	▒ 0.97	▒ 0.96	█ 0.95	█ 0.90	█ 0.87	█ 0.85	█ 0.85	█ 0.84	█ 0.81
10	█ 1.00	▒ 0.91	▒ 0.93	▒ 0.93	█ 0.94	█ 0.88	█ 0.86	█ 0.83	█ 0.83	█ 0.83	█ 0.82
20	▒ 0.75	▒ 0.81	▒ 0.82	▒ 0.83	█ 0.84	█ 0.79	█ 0.85	█ 0.86	█ 0.87	█ 0.86	█ 0.86
30	█ 0.67	▒ 0.76	▒ 0.70	▒ 0.78	▒ 0.77	█ 0.75	▒ 0.82	▒ 0.83	▒ 0.84	▒ 0.83	▒ 0.85
40	▒ 0.75	▒ 0.73	▒ 0.73	▒ 0.74	▒ 0.75	█ 0.73	▒ 0.82	▒ 0.82	▒ 0.83	█ 0.83	▒ 0.83
50	▒ 0.71	▒ 0.71	▒ 0.71	▒ 0.72	█ 0.72	█ 0.72	▒ 0.79	▒ 0.79	▒ 0.80	▒ 0.81	▒ 0.81
60	▒ 0.71	▒ 0.70	▒ 0.70	▒ 0.70	▒ 0.70	▒ 0.72	▒ 0.77	▒ 0.77	▒ 0.78	▒ 0.79	▒ 0.78
70	▒ 0.67	▒ 0.69	▒ 0.69	▒ 0.69	█ 0.69	▒ 0.71	▒ 0.77	▒ 0.77	▒ 0.79	▒ 0.79	▒ 0.79
80	▒ 0.67	▒ 0.68	█ 0.66	█ 0.65	█ 0.66	█ 0.67	▒ 0.75	▒ 0.77	▒ 0.78	▒ 0.79	▒ 0.79
90	▒ 0.68	█ 0.67	█ 0.62	█ 0.62	█ 0.61	█ 0.63	▒ 0.75	▒ 0.76	▒ 0.77	▒ 0.77	▒ 0.77
100	▒ 0.69	█ 0.67	█ 0.62	█ 0.61	█ 0.59	█ 0.60	▒ 0.75	▒ 0.75	▒ 0.77	▒ 0.77	▒ 0.77

░ Cantor

▒ NUCOMP

▓ Cantor enhanced

█ NUCOMP enhanced

From the data, we see that the NUCOMP algorithm is the fastest except for a few of the smaller genera. The threshold seems to be around genus 6 for all field sizes as opposed to 7 or 8 as reported in [13], indicating that our careful implementation and the improved formulation in Algorithm 2 increase the algorithm's range of applicability. We also see that our optimized XGCD and partial multiplications do have an effect. For the parameter ranges where pseudodivision and half-gcd are effective, our enhanced algorithms are the fastest.

In Table 4, we list average running times for the four divisor addition algorithms when applied to some cryptographically relevant examples arising from Weil descent. The first two of these, genus 7 or 8 over $\mathbb{F}_{2^{23}}$, comes from Teske's trapdoor cryptosystem [23]. The third and fourth, genus 16 over $\mathbb{F}_{2^{31}}$ and $\mathbb{F}_{2^{37}}$, come from Musson's application of Weil descent to the elliptic curves defined over $\mathbb{F}_{2^{155}}$ and $\mathbb{F}_{2^{185}}$ in the Oakley key determination protocol [20]. The last, genus 31 over $\mathbb{F}_{2^5}$ is the Weil descent example considered by Velichka et. al. [24] from an elliptic curve defined over $\mathbb{F}_{2^5}$. For each example, the fastest of the four algorithms is shaded, and the ratio of the runtime over that of Cantor's algorithm without our improved XGCD and partial multiplication is given.

Table 4. Timings (in ms) for divisor addition in hyperelliptic curves over $\mathbb{F}_{2^m}$ of cryptographic relevance

	Genus and Finite Field				
Algorithm	7, $\mathbb{F}_{2^{23}}$	8, $\mathbb{F}_{2^{23}}$	16, $\mathbb{F}_{2^{31}}$	16, $\mathbb{F}_{2^{37}}$	31, $\mathbb{F}_{2^5}$
Cantor	0.135	0.147	0.508	0.605	0.789
NUCOMP	0.139	0.148	0.453	0.536	0.602 (0.76)
CANTOR E	0.135	0.148	0.517	0.615	0.858
NUCOMP E	0.130 (0.97)	0.139 (0.95)	0.438 (0.86)	0.517 (0.86)	0.628

The data in Table 4 shows that our improved version of NUCOMP does indeed offer some modest improvements, except for the last example. In the last case, we observe that for genus 31 and $\mathbb{F}_{2^5}$, none of the pseudodivision XGCD variations and half-gcd variations offer improvements over the basic Euclidean algorithm. The partial multiplication algorithm also does not yield a big improvement for the operand sizes arising. Thus, we expect the basic version without our enhancements to be as fast or better than the version with them.

To get an idea of how much time in the overall discrete logarithm computation would be saved using our improved arithmetic, we make use of the following estimates. Teske [23] estimates that the algorithm of Enge and Gaudry [7] would require approximately 2^{34} Jacobian additions to compute a discrete logarithm for the first curve of Table 4, and 2^{37} for the second. Musson [20, p.123] estimates 2^{59} operations using the double-large prime algorithm of Gaudry et. al. [10] for the third curve, and 2^{70} for the fourth. Velichka et. al. [24] estimate 2^{37} group operations for the last curve using an empirically-optimized version of Enge and Gaudry's algorithm. Using these estimates, we can calculate how much CPU time on our reference architecture would be saved using our optimized arithmetic as compared to an implementation using an unoptimized version of Cantor's algorithm — this data is listed in Table 5. Even though our improvements will likely not affect a significant percentage of the entire discrete logarithm computation (for example, previous estimates indicated that group arithmetic accounted for roughly 20% of the relation generation time for the fourth example in the table, and 33% for the fifth), they do result in a non-negligible savings in terms of total CPU time.

TABLE 5. Estimated total CPU time (days) saved in discrete logarithm computation

	Genus and Finite Field				
	7, $\mathbb{F}_{2^{23}}$	8, $\mathbb{F}_{2^{23}}$	16, $\mathbb{F}_{2^{31}}$	16, $\mathbb{F}_{2^{37}}$	31, $\mathbb{F}_{2^5}$
Exp. additions	2^{34}	2^{37}	2^{59}	2^{70}	2^{37}
Est. Time Saved (days)	1	12	467039961	1202454428508	297

It has been shown for the genus 31 example [24] that sieving results in further computational improvements in such a way that divisor arithmetic plays a negligible role. In that case, our optimized arithmetic is unlikely to have much effect. However, it is not clear whether sieving will out-perform the random walk based algorithms mentioned above for the lower genus examples; in those cases, the current state-of-the-art is that our optimized arithmetic algorithms will yield some savings as indicated in the table.

5.2. **Divisor Doubling.** In Table 6, we provide a comparison of the four variations of divisor doubling. Using Cantor's algorithm without our improvements as a baseline as above, we list for each combination of genus and field size the ratio of the fastest of the four algorithms with Cantor's. Shading is used to indicate which of the four was the fastest.

TABLE 6. Fastest algorithms for divisor doubling in hyperelliptic curves defined over binary fields $\mathbb{F}_{2^m}$

g	m										
	2	4	8	16	32	64	128	256	512	1024	2048
5	1.00 ░	0.96 █	1.00 ░	1.00 ░	0.96 █	0.99 █	0.87 ▓	0.82 ▓	0.81 ▓	0.81 ▓	0.79 ▓
6	1.00 ░	1.00 ░	0.99 ▓	0.97 █	0.92 █	0.95 █	0.87 █	0.84 ▓	0.83 ▓	0.83 ▓	0.83 ▓
7	0.97 ▒	0.97 ▒	0.97 ▒	0.98 █	0.93 █	0.92 █	0.85 █	0.86 ▓	0.85 ▓	0.86 ▓	0.85 ▓
8	1.00 █	0.96 ▒	0.93 █	0.95 █	0.92 █	0.91 █	0.88 █	0.86 █	0.86 █	0.87 █	0.86 █
9	0.95 ▒	0.95 ▒	0.94 ▒	0.94 █	1.00 ▒	0.97 █	0.81 █	0.86 █	0.87 █	0.87 █	0.84 █
10	0.96 █	0.89 █	0.92 ▒	0.91 █	0.84 █	0.87 █	0.87 █	0.88 █	0.86 █	0.87 █	0.86 █
20	0.75 █	0.79 ▒	0.79 ▒	0.82 ▒	0.82 ▒	0.79 █	0.87 █	0.89 █	0.91 █	0.90 █	0.92 █
30	0.80 ▒	0.74 ▒	0.75 ▒	0.76 ▒	0.76 ▒	0.75 █	0.84 ▒	0.85 ▒	0.93 ▒	0.90 ▒	0.93 ▒
40	0.86 █	0.72 ▒	0.73 ▒	0.73 ▒	0.74 ▒	0.73 █	0.83 ▒	0.84 ▒	0.87 ▒	0.87 ▒	0.88 ▒
50	0.73 ▒	0.70 ▒	0.71 ▒	0.71 ▒	0.72 ▒	0.72 █	0.81 ▒	0.82 ▒	0.84 ▒	0.85 ▒	0.86 ▒
60	0.72 ▒	0.69 ▒	0.70 ▒	0.70 ▒	0.70 ▒	0.72 ▒	0.78 ▒	0.80 ▒	0.82 ▒	0.83 ▒	0.84 ▒
70	0.67 ▒	0.68 ▒	0.69 ▒	0.69 ▒	0.70 ▒	0.70 ▒	0.79 ▒	0.80 ▒	0.83 ▒	0.84 ▒	0.84 ▒
80	0.69 ▒	0.68 ▒	0.69 ▒	0.68 █	0.67 █	0.67 █	0.78 ▒	0.80 ▒	0.82 ▒	0.79 ▒	0.79 ▒
90	0.69 ▒	0.67 ▒	0.66 █	0.64 █	0.63 █	0.63 █	0.77 ▒	0.79 ▒	0.81 ▒	0.77 ▒	0.78 ▒
100	0.68 ▒	0.65 █	0.62 █	0.61 █	0.59 █	0.61 █	0.77 ▒	0.78 ▒	0.77 ▒	0.78 ▒	0.78 ▒

░	Cantor
▒	NUDUPL
▓	Cantor enhanced
█	NUDUPL enhanced

From the data, we see similar behavior to that of NUCOMP. The NUDUPL algorithm is the fastest except for a few of the smaller genera. The threshold seems to be around genus 5 or 6 for all field sizes as opposed to 7 or 8 as reported in [13], indicating again that our careful implementation and the improved formulation in Algorithm 2 increase the algorithm's range of applicability. We also see that our optimized XGCD and partial multiplications do offer an improvement when the parameter ranges are suitable.

5.3. **Divisor Tripling.** In Table 7, we provide a comparison of six variations of divisor tripling. As above, we use the basic tripling/reduction and NUTRIPL algorithms from [11], with and without our improved XGCD and partial multiplication methods. In addition, we compare these cubing algorithms with simply doubling and adding the input. For the variation of this method that does not use our enhancements, we use the thresholds described above to automatically select the fastest unenhanced addition and doubling algorithms (Cantor or NUCOMP/NUDUPL). We do the same for the enhanced version. As above, we list for each combination of genus and field size the ratio of the fastest of the four algorithms with Cantor's. Shading is used to indicate which of the six was the fastest.

Table 7. Fastest algorithms for divisor tripling in hyperelliptic curves defined over binary fields $\mathbb{F}_{2^m}$

g	m										
	2	4	8	16	32	64	128	256	512	1024	2048
5	0.96	1.00	1.00	1.00	0.96	1.00	0.90	1.00	0.86	0.91	0.90
6	1.00	0.99	1.00	1.00	0.95	0.96	0.80	0.80	0.80	0.93	0.90
7	1.00	0.98	0.96	1.00	0.86	0.83	1.00	1.00	0.86	0.94	0.85
8	0.92	1.00	0.88	0.93	1.00	0.80	0.60	0.25	0.56	0.85	0.86
9	0.94	0.93	0.95	0.85	0.56	0.60	1.00	0.60	0.80	0.86	0.84
10	0.94	0.89	0.95	0.94	0.91	0.76	0.88	0.80	0.83	0.85	0.82
20	0.75	0.75	0.80	0.77	0.79	0.70	0.75	0.80	0.85	0.79	0.80
30	0.68	0.70	0.71	0.73	0.72	0.76	0.65	0.74	0.72	0.74	0.75
40	0.65	0.68	0.64	0.71	0.75	0.70	0.73	0.72	0.71	0.68	0.70
50	0.66	0.65	0.63	0.68	0.68	0.68	0.66	0.69	0.66	0.65	0.66
60	0.73	0.72	0.69	0.66	0.63	0.67	0.64	0.66	0.66	0.66	0.68
70	0.72	0.71	0.65	0.62	0.63	0.63	0.65	0.67	0.66	0.65	0.67
80	0.73	0.70	0.61	0.62	0.59	0.58	0.67	0.65	0.66	0.67	0.69
90	0.70	0.67	0.63	0.59	0.53	0.57	0.61	0.66	0.65	0.64	0.65
100	0.65	0.60	0.57	0.54	0.51	0.51	0.63	0.60	0.63	0.63	0.64

Cantor
NUTRIPLE
Double/Add
Cantor enhanced
NUTRIPLE enhanced
Double/Add enhanced

Here, is is somewhat harder to determine the trends in the data, but some observations are apparent. As with addition and doubling, our enhancements to XGCD and partial multiplication have an effect for the corresponding parameter ranges. We also see that the tripling formulas from [11] in many cases do not offer an improvement over doubling and adding. Without our enhancements, the tripling algorithm is only better until about genus 12. With our enhancements, this threshold is about 10, NUTRIPL does again become faster for parameters where the half-gcd algorithm is effective.

5.4. **Scalar Multiplication.** We apply our optimized arithmetic operations to the problem of scalar multiplication. We compare the non-adjacent form exponentiation method with a double-base right-to-left method using bases 2 and 3, using both divisor arithmetic with and without our enhancements. In each case, our implementation adaptively selects the fastest operation (Cantor/NUCOMP, Cantor/NUDUPL, Triple/NUTRIPL/SqrMult) using the thresholds presented above. In Table 8, we list for each combination of genus and field size the ratio of the fastest of the four algorithms with unenhanced non-adjacent form. Shading is used to indicate which of the six was the fastest.

Table 8. Fastest algorithms for scalar multiplication in function fields defined over binary fields $\mathbb{F}_{2^m}$

g	m										
	2	4	8	16	32	64	128	256	512	1024	2048
5	█ 0.92	█ 0.96	█ 0.97	█ 1.00	▒ 0.91	▓ 0.96	▓ 0.84	█ 0.84	▓ 0.81	█ 0.79	█ 0.68
6	▒ 0.74	▓ 1.00	█ 0.91	█ 0.92	▒ 0.98	▓ 0.95	█ 0.87	▓ 0.82	▓ 0.84	▓ 0.78	█ 0.86
7	█ 0.93	▒ 0.91	█ 0.94	▓ 0.89	█ 0.94	█ 0.95	▓ 0.84	█ 0.81	█ 0.86	▓ 0.87	█ 0.85
8	▓ 0.94	▒ 0.88	█ 0.92	█ 1.00	▓ 0.88	▓ 0.91	▓ 0.88	█ 0.83	█ 0.87	▓ 0.88	█ 0.85
9	█ 0.94	▒ 0.97	░ 1.00	▒ 0.98	▓ 0.94	█ 0.95	▓ 0.87	▓ 0.86	▓ 0.88	▓ 0.86	▓ 0.83
10	▓ 0.97	░ 1.00	▓ 0.98	█ 1.00	▓ 0.98	▓ 0.91	▓ 0.86	▓ 0.86	▓ 0.86	▓ 0.87	▓ 0.85
15	█ 0.94	░ 1.00	▒ 1.00	▒ 0.96	▓ 1.00	▓ 0.94	▓ 0.90	░ 1.00	▓ 0.97	▓ 0.99	▓ 0.95
20	░ 1.00	░ 1.00	░ 1.00	▓ 1.00	▓ 1.00	▓ 0.94	▓ 0.96	▓ 0.98	▓ 0.97	▒ 0.90	▓ 0.84
25	░ 1.00	░ 1.00	░ 1.00	█ 1.00	░ 1.00	█ 0.89	░ 1.00	░ 1.00	░ 1.00	░ 1.00	▓ 0.94
30	░ 1.00	░ 1.00	▒ 1.00	░ 1.00	▒ 1.00	▓ 0.96	░ 1.00	░ 1.00	▒ 0.99	░ 1.00	░ 1.00
35	░ 1.00	░ 1.00	▒ 1.00	░ 1.00	░ 1.00	▓ 0.96	▓ 0.99	▓ 1.00	░ 1.00	░ 1.00	░ 1.00
40	░ 1.00	░ 1.00	▒ 1.00	▒ 1.00	░ 1.00	▓ 0.97	░ 1.00	░ 1.00	▒ 0.99	▓ 0.99	░ 1.00
45	░ 1.00	░ 1.00	░ 1.00	░ 1.00	█ 1.00	▓ 0.98	░ 1.00	░ 1.00	░ 1.00	▒ 1.00	▒ 0.99
50	░ 1.00	░ 1.00	░ 1.00	░ 1.00	▒ 1.00	▓ 0.98	░ 1.00	▒ 1.00	▒ 1.00	▒ 0.99	░ 1.00
60	▒ 0.99	░ 1.00	░ 1.00	▒ 0.96	░ 1.00	░ 1.00	░ 1.00	░ 1.00	▒ 0.99	░ 1.00	░ 1.00
70	▒ 1.00	░ 1.00	░ 1.00	█ 0.98	█ 0.98	░ 1.00	▒ 0.99	░ 1.00	░ 1.00	▒ 1.00	▓ 0.92
80	▒ 1.00	░ 1.00	█ 0.95	█ 0.97	█ 0.95	█ 0.95	░ 1.00	▒ 0.99	░ 1.00	░ 1.00	░ 1.00
90	▒ 0.99	█ 0.96	█ 0.92	█ 0.91	█ 0.89	█ 0.91	▒ 0.98	░ 1.00	░ 1.00	░ 1.00	░ 1.00
100	░ 1.00	▓ 0.96	█ 0.89	█ 0.87	█ 0.86	█ 0.88	▒ 1.00	▒ 0.99	░ 1.00	░ 1.00	▒ 0.98

░ NAF

▒ DBNS

▓ NAF enhanced

█ DBNS enhanced

We see from the data that, as before, our enhanced versions of the algorithms offer improvements for the usual parameter ranges. We also note that double-base scalar multiplication is faster than NAF in the cases where the cubing algorithms are faster than the double/add method. Our enhance versions are most applicable for smaller genera and sufficiently large genera that the half-gcd algorithm is used.

5.5. **Fast Reduction.** We apply our optimized XGCD and partial multiplication algorithms to divisor reduction. We compare Cantor's reduction algorithm (the last part of Algorithm 1), the reduction algorithm from [1] (Algorithm 3), and the latter using our enhanced XGCD and partial multiplication methods. Note that Cantor's reduction algorithm uses neither XGCD nor exact divisions, so our enhancements are not applicable. For the enhanced version of the algorithm from [1], the thresholds from above are used to adaptively select the best XGCD variation of partial multiplication for the given input sizes.

Table 9 lists data comparing the performance of the three algorithms for reducing divisors in hyperelliptic curves of various genera defined over $\mathbb{F}_{2^{16}}$, the same field presented in [1]. The divisors reduced were such that the coefficient u has degree mg for various integers m, simulating a non-reduced mth power of a typical reduced divisor. In Table 9, we list for each combination of genus and field size the ratio of the fastest of the three algorithms with Cantor's reduction (last part of Algorithm 1). Shading is used to indicate which of the three was the fastest.

Table 9. Fastest algorithms for ideal reduction in function fields defined over binary fields $\mathbb{F}_{2^{16}}$,

g	m							
	2	4	8	16	32	64	128	256
5	1.00	1.00	1.00	0.80	0.54	0.34	0.21	0.12
6	1.00	1.00	0.92	0.66	0.45	0.28	0.17	0.10
7	1.00	1.00	0.88	0.62	0.41	0.26	0.15	0.09
8	1.00	1.00	0.82	0.57	0.37	0.23	0.14	0.08
9	1.00	1.00	0.80	0.55	0.35	0.22	0.13	0.07
10	1.00	1.00	0.78	0.52	0.33	0.20	0.12	0.07
15	1.00	0.88	0.61	0.40	0.25	0.15	0.09	0.05
20	1.00	0.76	0.51	0.33	0.20	0.12	0.07	0.04
25	1.00	0.71	0.44	0.28	0.17	0.10	0.06	0.04
30	1.00	0.60	0.38	0.25	0.15	0.09	0.05	0.03
40	0.91	0.51	0.33	0.20	0.12	0.07	0.04	
50	0.85	0.46	0.28	0.17	0.10	0.06	0.03	
60	0.79	0.40	0.24	0.15	0.09	0.05		
70	0.76	0.37	0.22	0.13	0.08	0.05		
80	0.68	0.34	0.20	0.12	0.07	0.04		
90	0.65	0.32	0.19	0.11	0.06	0.04		
100	0.61	0.30	0.17	0.10	0.06	0.03		

Cantor
FAST
FAST/new

Unlike the data in [1], our results here indicate the expected behavior, namely that the new reduction algorithm should become increasingly better than Cantor's as both the genus and degree of the non-reduced input divisor increase. The performance becomes significantly better as the half-gcd threshold for partial XGCD are surpassed. Notice that pseudodivision-base XGCD does not offer any improvements for this field size.

6. **Conclusions.** We have attempted to optimize divisor arithmetic for hyperelliptic curves of genus larger than 5, using as many of the known methods as possible. Our results show that the methods used by Nagao, in particular partial multiplication and computing extended GCDs using pseudodivision are only effective for certain parameter ranges. As expected, the half-gcd algorithm is effective for divisor arithmetic as soon as the input degrees are sufficiently large, including its application to the computation of partial XGCDs. NUCOMP and its variants are superior

after a smaller genus threshold than previously thought, except for NUTRIPL, which is often slower than an optimized double and add method except when half-gcd applies. Our adaptive implementation of exponentiation, using thresholds to select between XGCD variations, partial multiplication variations, and basic divisor arithmetic variations, offers improvements over the non-adaptive version using enhanced algorithms. It shows that the double-base right-to-left method is indeed faster than NAF as long as the tripling algorithm is faster than doubling and adding. Finally, using our enhanced methods confirms that the reduction algorithm of [1] does in fact out-perform Cantor's.

One conclusion to be drawn is that there are very few performance gains left to be realized with the known algorithms. It is possible that some improvements could be obtained by using the geometric approach to the group law [5]. Our thresholds used could also be sampled at finer granularity. However, neither approach is expected to yield significant improvements; new ideas, or more extensive low-level optimizations, are required to go further.

The fact that NUCOMP and its variations is faster for smaller genera than previously believed suggests that some improvements to small genus explicit formulas could be obtained based on this algorithm. The NUTRIPL algorithm also appears practical for somewhat small genera, and an explicit version of that combined with double-base scalar multiplication may also yield improvements for small genera.

It would be of some interest to perform the same optimizations for the odd characteristic case. All the same techniques would apply, but different thresholds would almost certainly occur. For example, as inversion in odd characteristic finite fields is more expensive than the even characteristic case, we would expect a larger impact from pseudodivision as applied to the various XGCD methods.

REFERENCES

[1] R. Avanzi, M. J. Jacobson, Jr., and R. Scheidler. Efficient divisor reduction on hyperelliptic curves. *Advances in Mathematics of Communications*, 4(2):261–279, 2010.

[2] D. G. Cantor. Computing in the Jacobian of a hyperelliptic curve. *Math. Comp.*, 48(177):95–101, 1987.

[3] H. Cohen. *A Course in Computational Algebraic Number Theory*. Springer, 1996.

[4] H. Cohen, G. Frey, R. Avanzi, C. Doche, T. Lange, K. Nguyen, and F. Vercouteren. *Handbook of Elliptic and Hyperelliptic Curve Cryptography*. Chapman & Hall/CRC, Boca Raton, 2006.

[5] Craig Costello and Kristin Lauter. Group law computations on jacobians of hyperelliptic curves. In *Selected Areas in Cryptography*, pages 92–117, 2011.

[6] X. Ding. Acceleration of algorithm for the reduced sum of two divisors of a hyperelliptic curve. In *Information Computing and Applications*, volume 105 of *Communications in Computer and Information Science*, pages 177–184. Springer, 2011.

[7] A. Enge and P. Gaudry. A general framework for subexponential discrete logarithm algorithms. *Acta Arithmetica*, 102:83–103, 2002.

[8] G. Frey. Applications of arithmetical geometry to cryptographic constructions. In *Proceedings of the Fifth International Conference on Finite Fields and Applications*, pages 128–161. Springer, 2001.

[9] P. Gaudry, F. Hess, and N. Smart. Constructive and destructive facets of weil descent on elliptic curves. *Journal of Cryptology*, 15:19–46, 2002.

[10] P. Gaudry, E. Thomé, N. Thériault, and C. Diem. A double large prime variation for small genus hyperelliptic index calculus. *Math. Comp.*, 76(257):475–492, 2007.

[11] L. Imbert, M. J. Jacobson, Jr., and A. Schmidt. Fast ideal cubing in imaginary quadratic number and function fields. *Advances in Mathematics of Communications*, 4(2):237–260, 2010.

[12] M. J. Jacobson, Jr., R. E. Sawilla, and H. C. Williams. Efficient ideal reduction in quadratic fields. *International Journal of Mathematics and Computer Science*, 1:83–116, 2006.

[13] M. J. Jacobson, Jr., R. Scheidler, and A. Stein. Fast arithmetic on hyperelliptic curves via continued fraction expansions. In *Advances in Coding Theory and Cryptology*, volume 3 of *Series on Coding Theory and Cryptology*, pages 201–244. World Scientific Publishing, 2007. Invited Paper.

[14] M. J. Jacobson, Jr. and A. J. van der Poorten. Computational aspects of NUCOMP. In *Algorithmic Number Theory - ANTS-V*, volume 2369 of *Lecture Notes in Computer Science*, pages 120–133, Sydney, Australia, 2002. Springer-Verlag, Berlin.

[15] M. J. Jacobson, Jr. and H. C. Williams. *Solving the Pell Equation*. CMS Books in Mathematics. Springer-Verlag, 2009. ISBN 978-0-387-84922-5.

[16] D. E. Knuth. *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms*. Addison-Wesley, Reading, MA, third edition, 1997.

[17] N. Koblitz. Elliptic curve cryptosystems. *Math. Comp.*, 48:203–209, 1987.

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

[19] V. Miller. Use of elliptic curves in cryptography. In *Advances in Cryptology — CRYPTO '85*, volume 218 of *Lecture Notes in Computer Science*, pages 417–426, 1986.

[20] M. Musson. *Another Look at the Gaudry, Hess and Smart Attack on the Elliptic Curve Discrete Logarithm Problem*. PhD thesis, Dept. of Mathematics and Statistics, University of Calgary, Calgary, Alberta, 2011.

[21] K. Nagao. Improving group law algorithms for Jacobians of hyperelliptic curves. In *Algorithmic number theory (Leiden, 2000)*, volume 1838 of *Lecture Notes in Computer Science*, pages 439–447. Springer, Berlin, 2000.

[22] V. Shoup. NTL: A Library for doing Number Theory. Software, 2010. http://www.shoup.net/ntl.

[23] E. Teske. An elliptic curve trapdoor system. *Journal of Cryptology*, 19:115–133, 2006.

[24] M. D. Velichka, M. J. Jacobson, Jr., and A. Stein. Computing discrete logarithms on high-genus hyperelliptic curves over even characteristic finite fields. to appear in Mathematics of Computation, 2013.

E-mail address: Laurent.Imbert@lirmm.fr

E-mail address: jacobs@cpsc.ucalgary.ca