A Point Halving Algorithm for Hyperelliptic Curves

Izuru Kitamura[1], Masanobu Katagi[1], and Tsuyoshi Takagi[2]

[1] Sony Corporation, 6-7-35 Kitashinagawa Shinagawa-ku, Tokyo, 141-0001 Japan
{Izuru.Kitamura, Masanobu.Katagi}@jp.sony.com
[2] Technische Universität Darmstadt, Fachbereich Informatik,
Hochschulstr.10, D-64289 Darmstadt, Germany
takagi@informatik.tu-darmstadt.de

Abstract. We deal with a point halving algorithm on hyperelliptic curve cryptosystems (HECC), which can be used for scalar multiplication instead of doubling algorithm. In general, halving algorithm is slower than doubling algorithm due to the complicated addition formula of hyperelliptic curves. The efficiency of halving algorithm strongly depends on the choice of curve parameters. In this paper, we show some efficient classes for computing halving. For example, the halving algorithm for hyperelliptic curves with $h(x) = x^2 + x + 1$ is about 5% faster than the doubling algorithm using the normal basis. Moreover, the proposed algorithm was implemented in C code using the polynomial basis. We show a curve whose scalar multiplication can be implemented more efficiently using the proposed point halving algorithm.

Keywords. hyperelliptic curve cryptosystems, scalar multiplication, point halving, efficient computation

1 Introduction

Recent research has found that hyperelliptic curve cryptosystems (HECC) with small genus are competitive with elliptic curve cryptosystems (ECC) [Ava04,Lan02a-c, PWG+03]. With an eye to further improvement of HECC we utilize their abundant algebraic structure, so HECC can eventually achieve faster scalar multiplication than ECC. Recently, Lange and Duquesne independently showed that Montgomery scalar multiplication is applicable to HECC [Lan04,Duq04]. We expect that other fast algorithms used for ECC can also be efficiently implemented in HECC.

A point halving algorithm is one of these efficient algorithms for ECC, which tries to find point P such that $2P = Q$ for a given point Q. Knudsen proposed an efficient point halving algorithm for ECC [Knu99]. His point halving algorithm is efficient for elliptic curves over binary fields $\mathbb{F}_{2^n}$, because the computation of squaring, square root, and half trace is negligible, that is, virtually free for the normal basis representation. Moreover, the concern with the point halving algorithm has been growing, for instance, an efficient implementation [FHL+03], an application for Koblitz curve [ACF04], and an improvement for curves with cofactor 4 [KR04].

The explicit doubling formula of HECC is rather complicated comparing with that of ECC, and it is not obvious how Knudsen's algorithm can be extended to HECC.

In this paper, we investigate a point halving algorithm for HECC of genus 2 over binary fields. Let $D = (U, V)$ be a reduced divisor, where $U = x^2 + u_1 x + u_0$ and $V = v_1 x + v_0$. The doubled divisor $2D$ can be represented as the polynomial over $\mathbb{F}_{2^n}$ with coefficients u_1, u_0, v_1, v_0 and curve parameters $y^2 + h(x)y = f(x)$. We point out two crucial quadratic equations, which compute some candidates of the halved values. We also show a criteria and an efficient algorithm for selecting the correct divisor from two candidates. The correct divisor can be found if polynomial $h(x)$ is irreducible. The basic operations required for implementing the proposed algorithm are same as those to implement Knudsen's algorithm. The obtained point halving algorithm is usually slower than the doubling algorithm. However, some choices of the curve parameter make the proposed halving algorithm faster, for example $h(x) = x^2 + x + 1$. In this case, the proposed halving algorithm is about 5% faster than the doubling algorithm using the normal basis.

We implemented the proposed algorithm in software using the polynomial basis. In this case, the computation of squaring, square root, and half trace is not negligible. For example, our experiment shows $1S = 0.12M, 1SR = 0.57M$, and $1H = 0.58M$ over $\mathbb{F}_{2^{83}}$ defined by $\mathbb{F}_2[z]/(z^{83}+z^7+z^4+z^2+1)$, where S, SR, H, and M are the time of computing squaring, square root, half trace, and multiplication, respectively. The proposed algorithm with $h(x) = x^2 + x + 1$ requires $19M + 3S + 1I + 3SR + 2H + 2T$, where I and T are the time of computing inversion and trace, respectively. (*HarleyDBL* requires $20M + 5S + 1I$ [PWP03].) However, this estimation is the worst case of the proposed halving algorithm. In order to select the correct halved value, we do calculate as a test some multiplications, some squaring, and a square root. We found that number of M, S, and SR can be reduced if the correct halving value is first found. In our experiment, we have found a curve whose halving algorithm is faster than its doubling algorithm. The base field of this curve has a faster square root algorithm compared with that of other curves. We expect that the proposed algorithm can be optimized by carefully considering the basic operations.

This paper is organized as follows: in Section 2 we review the algorithms of a hyperelliptic curve. In Section 3 we present our proposed halving algorithm for HECC, and compare it with known doubling formulae. In Section 4 we present some results of the implementation of the proposed scheme in software. In Section 5 we state concluding remarks.

2 Hyperelliptic Curve

We review the hyperelliptic curve used in this work.

Let $\mathbb{F}_{2^n}$ be a binary finite field with 2^n elements. A hyperelliptic curve C of genus g over $\mathbb{F}_{2^n}$ is defined by $C : y^2 + h(x)y = f(x)$, where $f(x) \in \mathbb{F}_{2^n}[x]$ is a monic polynomial of degree $2g + 1$ and $h(x) \in \mathbb{F}_{2^n}[x]$ is a polynomial of degree at most g, and curve C has no singular point. Let $P_i = (x_i, y_i) \in \overline{\mathbb{F}_{2^n}} \times \overline{\mathbb{F}_{2^n}}$ be a point on

curve C and P_∞ be a point at infinity, where $\overline{\mathbb{F}_{2^n}}$ is the algebraic closure of $\mathbb{F}_{2^n}$. The inverse of $P_i = (x_i, y_i)$ is the point $-P_i = (x_i, y_i + h(x_i))$. P is called a ramification point if $P = -P$ holds. A divisor is a formal sum of points: $D = \sum m_i P_i, m_i \in \mathbb{Z}$. A semi-reduced divisor is represented by $D = \sum m_i P_i - (\sum m_i) P_\infty$, where $m_i \geq 0$ and $P_i \neq -P_j$ for $i \neq j$, and semi-reduced divisor D is called reduced if $\sum m_i \leq g$ holds.

Jacobian $\mathbf{J}$ is the set of reduced divisors and forms an additive group. The reduced and the semi-reduced divisors are expressed by a pair of polynomials (u, v), which satisfies the following conditions [Mum84]:

$$u(x) = \prod (x + x_i)^{m_i}, v(x_i) = y_i, \deg v < \deg u, v^2 + hv + f \equiv 0 \bmod u.$$

A semi-reduced divisor is defined over $\mathbb{F}_{2^n}$ when $u, v \in \mathbb{F}_{2^n}[x]$, and the set of such divisors is denoted by $\mathbf{J}(\mathbb{F}_{2^n})$. Note that even if $u, v \in \mathbb{F}_{2^n}[x]$, the coordinates x_i and y_i may be in extension field of $\mathbb{F}_{2^n}$.

To compute the additive group law of $\mathbf{J}(\mathbb{F}_{2^n})$, Cantor gave an addition algorithm for any hyperelliptic curve. However, the Cantor algorithm is relatively slow due to its generality. Harley then proposed an efficient addition and doubling algorithm for a hyperelliptic curve of genus 2 over $\mathbb{F}_p$ [GH00,Har00a,Har00b]. Sugizaki et al. expanded the Harley algorithm to over $\mathbb{F}_{2^n}$ [SMC+02], and around the same time Lange expanded the Harley algorithm to over general finite field [Lan02a]. The doubling algorithm (denoted by *HarleyDBL* in this paper) is as follows:

Algorithm 1 *HarleyDBL*

Input: $D_1 = (U_1, V_1)$
Output: $D_2 = (U_2, V_2)$
$U_i(x) = x^2 + u_{i1}x + u_{i0}, V_i = v_{i1}x + v_{i0}, \gcd(U_i, h) = 1$, where $i = 1, 2$

1. $U_1' \leftarrow U_1^2$
2. $S \leftarrow h^{-1}(f + hV_1 + V_1^2)/U_1 \bmod U_1$
3. $V_1' \leftarrow SU_1 + V_1$
4. $U_2' \leftarrow (f + hV_1' + {V_1'}^2)/U_1'$
5. $U_2 \leftarrow$ MakeMonic (U_2')
6. $V_2 \leftarrow V_1' + h \bmod U_2$
7. return (U_2, V_2)

The details of *HarleyDBL* refer to [Lan02a].

In a cryptographic application, we are only interested in a curve whose order of $\mathbf{J}(\mathbb{F}_{2^n})$ is $2 \times r$, where r is a large prime number. We assume that curves discussed in this paper satisfy this condition.

3 Proposed Point Halving Algorithm

In this section we propose a point halving algorithm on a hyperelliptic curve.

3.1 Main Idea

We follow the opposite path to *HarleyDBL*. From Step 6 of *HarleyDBL*, there is a unique polynomial $k = k_1 x + k_0$ such that $V_1' + h = (k_1 x + k_0)U_2 + V_2$. Substituting V_1' to equation $(f + hV_1' + V_1'^2)$ appeared in Step 4, the following relationship yields:

$$U_2' U_1' = f + h(kU_2 + V_2) + k^2 U_2^2 + V_2^2. \tag{1}$$

Because the doubled divisor (U_2, V_2) is known, we can obtain the relationship between k and U_1'. Note that $U_2' = k_1^2 U_2$ from the highest term of equation (1). Recall that $U_1' = U_1^2$ from Step 1, namely, we know

$$U_1' = x^4 + u_{11}^2 x^2 + u_{10}^2. \tag{2}$$

In other words, the coefficients of degree 3 and 1 are zero. From this observation, there are polynomials whose solutions includes k_0 and k_1. In our algorithm we try to find k_0 and k_1 by solving the polynomials. Once k_0 and k_1 are calculated, we can easily compute the halved divisor $D_1 = (U_1, V_1)$ from equation (1). We describe the sketch of the proposed algorithm (*HEC_HLV*) in the following.

Algorithm 2 *Sketch HEC_HLV*

Input: $D_2 = (U_2, V_2)$
Output: $D_1 = (U_1, V_1)$
$U_i(x) = x^2 + u_{i1}x + u_{i0}, V_i = v_{i1}x + v_{i0}$, $\gcd(U_i, h) = 1$, where $i = 1, 2$

1. reconstruct $k_0,\ k_1$
  1.1 $V_1' \leftarrow V_2 + h + kU_2,\ k(x) = k_1 x + k_0$
  1.2 $U_1' \leftarrow (f + hV_1' + V_1'^2)/(k_1^2 U_2)$
  1.3 derive $k_0,\ k_1$ from $\mathrm{coeff}(U_1', 3) = 0, \mathrm{coeff}(U_1', 1) = 0$
2. compute u_{11} by substituting k_0, k_1 in $\mathrm{coeff}(U_1', 2)$
3. compute u_{10} by substituting k_0, k_1 in $\mathrm{coeff}(U_1', 0)$
4. $U_1 \leftarrow x^2 + u_{11}x + u_{10}$
5. $V_1 \leftarrow V_2 + h + kU_2 \bmod U_1$
6. return (U_1, V_1)

In the following, we explain Algorithm 2 in detail. The coeff$(U,\ i)$ is the coefficient of x^i in polynomial U. In Step 1.2, we compute polynomial U_1' in equation (1):

$$\begin{aligned}
\mathrm{coeff}(U_1', 3) &= (k_1 h_2 + k_1^2 u_{21} + 1)/k_1^2 \\
\mathrm{coeff}(U_1', 2) &= (k_1 h_1 + k_0 h_2 + k_1^2 u_{20} + k_0^2 + c_2)/k_1^2 \\
\mathrm{coeff}(U_1', 1) &= (k_1 h_0 + k_0 h_1 + k_0^2 u_{21} + c_1)/k_1^2 \\
\mathrm{coeff}(U_1', 0) &= (k_0 h_0 + k_0^2 u_{20} + c_0)/k_1^2,
\end{aligned}$$

where

$$c_2 = f_4 + u_{21}, \quad c_1 = f_3 + h_2 v_{21} + u_{20} + c_2 u_{21},$$
$$c_0 = f_2 + h_2 v_{20} + h_1 v_{21} + v_{21}^2 + c_2 u_{20} + c_1 u_{21}.$$

Equation (2) yields the explicit relationship related to variables k_0, k_1, u_{11}, and u_{10}:

$$k_1 h_2 + k_1^2 u_{21} + 1 = 0 \tag{3}$$

$$k_1 h_0 + k_0 h_1 + k_0^2 u_{21} + c_1 = 0 \tag{4}$$

$$u_{11} = \sqrt{k_1 h_1 + k_0 h_2 + k_1^2 u_{20} + k_0^2 + c_2}/k_1 \tag{5}$$

$$u_{10} = \sqrt{k_0 h_0 + k_0^2 u_{20} + c_0}/k_1 \tag{6}$$

Then we prove the following lemma in order to uniquely find k_0, k_1. The proof of this lemma is in Appendix A.

Lemma 1. *Let $h(x)$ be an irreducible polynomial. There is only one value k_1 which satisfies both equations (3) and (4). Equation (4) has a solution only for the correct k_1. There is only one value k_0 which yields the halved divisor D_1 in algorithm Sketch HEC_HLV. Equation $xh_2 + x^2 u_{11} + 1 = 0$ has a solution only for the correct k_0.*

After calculating k_0, k_1, we can easily compute u_{11}, u_{10}, v_{11}, and v_{10} via equations (5), (6), and $V_1 \leftarrow V_2 + h + (k_1 x + k_0)U_2 \bmod U_1$.

3.2 Proposed Algorithm

We present the proposed algorithm in Algorithm 3.

The proposed algorithm requires to solve quadratic equations. It is well known that equation $ax^2 + bx + c = 0$ has roots if and only if $\mathrm{Tr}(ac/b^2) = 0$. Let one root of $ax^2 + bx + c = 0$ be x_0, then the other root be $x_0 + b/a$. If this equation has roots, i.e. $\mathrm{Tr}(ac/b^2) = 0$, then we can solve this equation by using half trace, namely $x_0 = \mathrm{H}(ac/b^2), x_0' = x_0 + b/a$. On the other hand, if this equation has no root, i.e. $\mathrm{Tr}(ac/b^2) = 1$, then we cannot solve this equation.

We explain the proposed algorithm as follows. The correctness of this algorithm is shown in Lemma 1. In Step 1, we solve two solutions k_1 and k_1' of equation (3). In Step 2, the correct k_1 is selected by checking the trace of equation (4). Then we obtain two solutions k_0 and k_0' of equation (4). In Step 3, the correct k_0 is selected by checking trace of $xh_2 + x^2 u_{11} + 1 = 0$. In Steps 4 and 5 we compute the halved divisor.

Algorithm 3 *HEC_HLV*

Input: $D_2 = (U_2, V_2)$
Output: $D_1 = (U_1, V_1)$
$U_i(x) = x^2 + u_{i1}x + u_{i0}$, $V_i = v_{i1}x + v_{i0}$, $\gcd(U_i, h) = 1$

step	*procedure*	*cost*
1.	**Solve** $k_1h_2 + k_1^2u_{21} + 1 = 0$	$3M + 1S + 2I$
	$\alpha \leftarrow h_2/u_{21}, \gamma \leftarrow 1/(u_{21}\alpha^2)$ /* $1/(u_{21}\alpha^2) = u_{21}/h_2^2$ */	$+1H$
	$k_1 \leftarrow H(\gamma)\alpha$, $k_1' \leftarrow k_1 + \alpha$	
2.	**Select correct** k_1 **by solving** $k_1h_0 + k_0h_1 + k_0^2u_{21} + c_1 = 0$	$13M + 2S + 2I$
	$c_2 \leftarrow f_4 + u_{21}$, $c_1 \leftarrow f_3 + h_2v_{21} + u_{20} + c_2u_{21}$	$+1H + 1T$
	$c_0 \leftarrow f_2 + h_2v_{20} + h_1v_{21} + v_{21}^2 + c_2u_{20} + c_1u_{21}$	
	$\alpha \leftarrow h_1/u_{21}$, $\gamma \leftarrow (c_1 + k_1h_0)/(u_{21}\alpha^2)$	
	if $\mathrm{Tr}(\gamma) = 1$ *then*	
	$k_1 \leftarrow k_1'$, $\gamma \leftarrow (c_1 + k_1h_0)/(u_{21}\alpha^2)$	
	$k_0 \leftarrow H(\gamma)\alpha$, $k_0' \leftarrow k_0 + \alpha$	
3.	**Select correct** k_0 **by checking trace of** $xh_2 + x^2u_{11} + 1 = 0$	$7M + 1S + 2I$
	$u_{11} \leftarrow \sqrt{k_1h_1 + k_0h_2 + k_1^2u_{20} + k_0^2 + c_2}/k_1$, $\gamma \leftarrow u_{11}/h_2^2$	$+2SR + 1T$
	if $\mathrm{Tr}(\gamma) = 1$ *then*	
	$k_0 \leftarrow k_0'$, $u_{11} \leftarrow \sqrt{k_1h_1 + k_0h_2 + k_1^2u_{20} + k_0^2 + c_2}/k_1$	
4.	**Compute** U_1	$3M + 1S + 1SR$
	$u_{10} \leftarrow \sqrt{k_0h_0 + k_0^2u_{20} + c_0}/k_1$	
5.	**Compute** $V_1 = V_2 + h + kU_2 \bmod U_1$	$6M$
	$w \leftarrow h_2 + k_1u_{21} + k_0 + k_1u_{11}$	
	$v_{11} \leftarrow v_{21} + h_1 + k_1u_{20} + k_0u_{21} + u_{10}k_1 + u_{11}w$	
	$v_{10} \leftarrow v_{20} + h_0 + k_0u_{20} + u_{10}w$	
total	*worst case*	$32M + 5S + 6I$ $+3SR + 2H + 2T$

3.3 Efficiency of *HEC_HLV* and its Improvement

We estimate the efficiency of the proposed algorithm *HEC_HLV* and show its improvements.

When we get incorrect k_1 and k_0 (k_1' and k_0' are correct) in Steps 1 and 2, respectively, we have to replace $k_1 \leftarrow k_1'$, $k_0 \leftarrow k_0'$ and compute γ, u_{11} again in Steps 2 and 3, respectively. In the worst case, Algorithm 3 (*HEC_HLV*) requires $32M + 5S + 6I + 3SR + 2H + 2T$[3]. On the other hand, in the best case we get correct k_1 and k_0, $4M$ can be reduced, and we have another two cases: one is k_0 and k_1' are

[3] We evaluate *HEC_HLV*'s cost as follows: in Step 3, $\sqrt{k_1h_1 + k_0h_2 + k_1^2u_{20} + k_0^2 + c_2}/k_1 = (k_0 + \sqrt{k_1h_1 + k_0h_2 + k_1^2u_{20} + c_2})/k_1$ requires $4M + 1S + 1I + 1SR$ and $\sqrt{k_1h_1 + k_0'h_2 + k_1^2u_{20} + k_0'^2 + c_2}/k_1 = u_{11} + (\sqrt{\alpha h_2} + \alpha)/k_1$ requires $2M + 1SR$ because u_{11} and $1/k_1$ are computed, and in Step 5, $k_1u_{20} + u_{10}k_1 = k_1(u_{10} + u_{20})$ requires $1M$, etc.

correct and the other is k_0' and k_1 are correct. Our experimental observations found that these four cases occur with almost the same probability. We obtain the average of these four cases as the average case. Additionally in case of $h_2 = 1$, Algorithm 3 requires $25M + 5S + 4I + 3SR + 2H + 2T$, which is slower than *HarleyDBL*.

Choices of the curve parameter. Now we consider how to reduce field operations. To do this, we use $h(x) = x^2 + x + h_0$ (recall we assume $h(x)$ to be an irreducible polynomial) and $f_4 = 0$ which is proposed by Pelzl [PWP03], because *HEC_HLV* requires $13M$ in the worst case, and $3S$, and $3I$ with the coefficients of $h(x)$. In Appendix B, we show this method as Algorithm 5 ($h_2 = h_1 = 1, f_4 = 0$) which is allowed to use $1/u_{21}$ as input and to calculate $1/u_{11}$ for the next halving in Step 5 via the Montgomery trick. This can be applicable in right-to-left scalar multiplication by adding $[\frac{1}{2^r}]D$, not left-to-right. This improved method requires $19.5M + 3S + 1I + 3SR + 2H + 2T$ with $h_0 \neq 0, 1$ and $18.5M + 3S + 1I + 3SR + 2H + 2T$ with $h_0 = 1$ in the average case. On the other hand, *HarleyDBL* with $h_2 = h_1 = 1, f_4 = 0$ requires $20M + 5S + 1I$.

We consider another special curve $y^2 + xy = x^5 + f_1 x + f_0$ which is proposed by Pelzl [PWP03]. Pelzl's doubling algorithm requires only $9M + 6S + I$. We adapt *HEC_HLV* to this special curve, and show it as Algorithm 6 in Appendix B, which requires $11.5M + 2S + 1I + 4.5SR + 1H + 1T$ in the average case.

Fixed base point. In the case of scalar multiplication with fixed base point, we propose another improved method via table-lookup. Recall we use right-to-left scalar multiplication by adding $[\frac{1}{2^r}]D$ in each halving and whether $k_1(k_0)$ is correct or not depends on D. In Algorithm 3, if we know the correct k_1 and k_0 in advance, $2M$ and $2M + 1SR$ can be reduced in Steps 2 and 3, respectively. So if D is fixed, we save finite field operations via the precomputing table which shows whether $k_1(k_0)$ or $k_1'(k_0')$ is the correct value in each halving. This table requires only the same bit length of order D for each k_1 and k_0, because in each halving we only use one bit — zero means $k_1(k_0)$ is correct and one means $k_1'(k_0')$ is correct.

Comparison of doubling and halving. We use this improved method in Algorithm 5 and 6. Table 1 provides a comparison of *HarleyDBL* and the above halving algorithm in the average case.

By using the normal basis, we can neglect the computation time of a squaring, a square root, a half trace and a trace compared to that of a field multiplication or an inversion [Knu99]. Therefore, we can say that *HEC_HLV* ($h_2 = h_1 = 1, f_4 = 0$) is faster than *HarleyDBL*, because *HEC_HLV* and *HarleyDBL* require $19.5M + 1I$ ($18.5M + 1I, h_0 = 1$) and $20M + 1I$, respectively, by using the normal basis i.e. counting only field multiplication and inversion. Menezes [Men93] showed that an inversion operation requires $\lfloor \log_2(n-1) \rfloor + w(n-1) - 1$ multiplications, where $w(n-1)$ is the number of 1's in the binary representation of $n - 1$. For example, an inversion over $\mathbb{F}_{2^{83}}$ requires $\lfloor \log_2(82) \rfloor + w((1010010)_2) - 1 = 8$ multiplications. If we assume $1I = 8M$, the proposed halving algorithm with $h(x) = x^2 + x + 1$ is about 5% faster than doubling algorithm.

Table 1. Comparison of Doubling and Halving

Scheme	*HEC_HLV*	*HarleyDBL*
$h_i, f_j \in \mathbb{F}_{2^n} \setminus \{0,1\}$	$30M + 5S + 6I + 2.5SR + 2H + 2T$	—
$h_2 = 1$	$24M + 5S + 4I + 2.5SR + 2H + 2T$	$22M + 5S + 1I$ [Lan02a]
$h_2 = h_1 = 1, f_4 = 0$	$19.5M + 3S + 1I + 3SR + 2H + 2T$	$20M + 5S + 1I$ [PWP03]
with table-lookup	$17M + 3S + +1I + 2SR + 2H$	—
$h_2 = h_1 = h_0 = 1, f_4 = 0$	$18.5M + 3S + 1I + 3SR + 2H + 2T$	$20M + 5S + 1I$ [PWP03]
with table-lookup	$16M + 3S + 1I + 2SR + 2H$	—
$y^2 + xy = x^5 + f_1x + f_0$	$11.5M + 2S + 1I + 4.5SR + 1H + 1T$	$9M + 6S + 1I$ [PWP03]
with table-lookup	$9M + 2S + 1I + 3SR + 1H$	—

On the other hand, by using polynomial basis, we cannot ignore the computation time of a squaring, a square root, a half trace. Assuming that $1S = 0.1M$, $1SR = 0.5M$, $1H = 0.5M$, and $1I = 8M$, *HarleyDBL* is about 1% faster than *HEC_HLV* ($h_2 = h_1 = 1, f_4 = 0$). By selecting polynomial basis, however, we can compute these arithmetic faster than half the time of multiplication. We must look more carefully into the estimation.

4 Implementation of Point Halving

In this section we show implementation results for some hyperelliptic curves.

Our implementation is mainly based on Knudsen [Knu99] and Fong et al [FHL+03]. Focusing on the polynomial basis, we first consider finite field arithmetic — squaring, square root, half trace, and trace, then we implement scalar multiplications for some hyperelliptic curves using these arithmetic.

4.1 Some Basic Algorithms

We show some basic algorithms used in this paper, namely computing trace, half trace, square root, and scalar multiplication.

Choices of Field Polynomial. We deal with hyperelliptic curves for genus 2 that are suitable for cryptographic purposes, so that the extension degree n of the base field is larger than 80. In addition, we have to choose n as the odd prime to avoid the Weil descent attack. In our implementation, we choose three types of field polynomial, $n = 83$, 89, and 113, to study the efficiency of finite field arithmetic. We applied the field polynomial $F(z)$ as low-weight binary polynomial.

The choice of irreducible polynomials brings about an efficient arithmetic. The performance of finite field arithmetic depends on the hamming weight of the irreducible polynomial. In particular, the 2nd-highest degree t of trinomial $(z^n + z^t + 1)$ is important for square root.

Table 2. Irreducible Polynomial

Degree	Type	$F(z)$
83	pentanomial	$z^{83}+z^7+z^4+z^2+1$
89	trinomial, t is even	$z^{89}+z^{38}+1$
113	trinomial, t is odd	$z^{113}+z^9+1$

Computing Square roots, Trace, and Half Trace. The point halving algorithm on HECC, as well as ECC, requires the solution of two quadratic equations over $\mathbb{F}_{2^n}$ in terms of k_1 and k_0. As seen in equations 5 and 6, the halving formulae must recover $u_{11}(u_{10})$ from $u_{11}^2(u_{10}^2)$. Therefore trace, half trace, and square roots are crucial arithmetic for *HEC_HLV*.

The square root of element c is given by $\sqrt{c} = c^{2^{n-1}}$.
This can be written as

$$\sqrt{c} = (\sum_{i=0}^{n-1} c_i z^i)^{2^n-1} = \sum_{i \text{ even}} c_i z^{\frac{i}{2}} + \sqrt{z} \sum_{i \text{ odd}} c_i z^{\frac{i-1}{2}}.$$

This algorithm requires about half the time of a field multiplication. According to [FHL+03], a trinomial with odd t is the most efficient. For example $\mathbb{F}_{2^{113}}$ has $\sqrt{z} = z^{56} + z^5$. This requires about one-tenth the time of field multiplication.

The trace of element $c = \sum_{i=0}^{n-1} c_i z^i \in \mathbb{F}_{2^n}$ is given by

$$\mathrm{Tr}(c) = \mathrm{Tr}(\sum_{i=0}^{n-1} c_i z^i) = \sum_{i=0}^{n-1} c_i \mathrm{Tr}(z^i). \tag{7}$$

If $\mathrm{Tr}(z^i)$ is precomputed, the time required to compute the trace of an element is negligible, because $\mathrm{Tr}(z^i) = 0$ for most i.

On the other hand, the time required to compute the half trace is not negligible. The half trace of element $c = \sum_{i=0}^{n-1} c_i z^i \in \mathbb{F}_{2^n}$ is given by

$$\mathrm{H}(c) = \sum_{i=0}^{(n-1)/2} c^{2^{2i}}. \tag{8}$$

We used an efficient method based on Algorithm 4.7 in [FHL+03].

Scalar Multiplication using Halving. Knudsen gave applications for scalar multiplication for elliptic curve [Knu99]. Let r be the order of the underlying divisor D. For a given integer d we can represent $d = \sum_{i=0}^{m} \hat{d}_i 2^{i-m}$, where $\hat{d}_i \in \{0, 1\}$. This representation $\hat{d}_i$ is used for the computation of scalar multiplication. Algorithm 4 is the scalar multiplication using the point halving algorithm for HECC. As discussed in Section 3, we require a modification of the original algorithm [Knu99] because we have to employ Algorithm 5.

Algorithm 4 *Scalar Multiplication via Point Halving.*

Input: $d \in \mathbb{Z}, D \in \mathbf{J}(\mathbb{F}_{2^n}),\ r$: order of $D,\ m = \lfloor \log_2 r \rfloor$
Output: scalar multiplication dD

step	*procedure*
1.	$2^m d \pmod r = \sum_{i=0}^{m} \hat{d}_i 2^i, \quad \hat{d}_i \in \{0,1\}$
2.	$\sum_{i=0}^{m} \frac{d_i}{2^i} \leftarrow \sum_{i=0}^{m} \hat{d}_i 2^{i-m}, \quad d_i \in \{0,1\}$
3.	$Q \leftarrow O, R \leftarrow D, invu \leftarrow 1/u_1$
4.	*For* i *from* 0 *to* m *do:*
	If $d_i = 1$ *then* $Q \leftarrow Q + R$.
5.	$(R, invu) \leftarrow HEC\_HLV\,(R, invu)$.
6.	*Return* Q.

4.2 Timing Result

We implemented these algorithms on an Intel Pentium II Processor 300MHz using operation system RedHat Linux 7.3 and gcc 2.96. In order to demonstrate the improvement of our halving algorithm, we compared *HEC_HLV* (Algorithm 5) with doubling ([PWP03]) on a special curve with $h(x) = x^2 + x + h_0$. As we noted before, the degree of extension is 83, 89, and 113. *HEC_HLV* has 1 multiplication with h_0, whose value is 1 or a random value over $\mathbb{F}_{2^n}$. We investigated 6 types of curve of a combination of the extension degree and h_0. The chosen curves C_1 to C_6 are listed in Appendix B.1.

We programmed the finite field arithmetic for the purpose of implementation halving and doubling of divisors. The two divisor operations shared the finite field multiplication(M) and squaring(S), and inversion(I). We applied faster algorithms for the multiplication and squaring, which are based on Algorithm 4, 6, and 7 in [HHM00]. The inversion algorithm is the binary extended GCD method [Sha01]. The trace and half trace, and the square root are dedicated functions for halving. We have measured the average timings of the basic arithmetic. Table 3 shows the results with 10 million random samples. As noted before, the timing of trace is negligible since the $\mathrm{Tr}(z^i)$ is precomputed. On the other hand, the timing of half trace H requires about $0.5M$. Thus, the half trace affects the performance of halving formulae. Moreover, the timing of SR brings about a difference of performance in terms of the definition field. SR over $\mathbb{F}_{2^n}$, where $F(z)$ is a trinomial with odd t, provides a good performance.

Table 3. Timing of the inversion, multiplication, squaring, square root, and half trace (in μs)

	mul	squaring	inversion	square root	half trace	S/M	I/M	SR/M	H/M
$\mathbb{F}_{2^{83}}$	20.63	2.38	163.75	11.73	12.03	0.12	7.96	0.57	0.58
$\mathbb{F}_{2^{89}}$	16.18	0.88	141.37	2.27	9.94	0.05	8.74	0.14	0.61
$\mathbb{F}_{2^{113}}$	31.61	1.82	270.67	3.18	15.77	0.06	8.56	0.10	0.50

The timing of the divisor operations and scalar multiplication are shown in Table 4 and Table 5, respectively. The addition of divisors is commonly used both with scalar multiplication with doubling [PWP03] and with scalar multiplication with halving in Algorithm 5. Doubling formulae is faster than halving formulae for most curves. Note that halving formulae is faster than doubling in C_6. Recall that the definition field of C_6 matches the faster condition in computing square roots. This result suggest that halving formulae using the polynomial basis is faster than doubling formulae such that an irreducible polynomial is a trinomial with odd t, e.g. $n = 103$, 113, and 127.

Table 4. Timing of the addition, doubling, and halving (in μs)

	C_1	C_2	C_3	C_4	C_5	C_6
addition	609.41	609.07	487.04	487.14	956.30	956.30
doubling	593.09	592.99	471.77	471.83	926.48	926.47
halving	638.58	617.00	489.00	473.39	952.00	918.64

Table 5. Timing of the scalar multiplication (in ms)

	C_1	C_2	C_3	C_4	C_5	C_6
double-and-add	149.80	149.37	128.16	127.62	317.85	319.22
halve-and-add	158.54	155.07	131.18	128.59	323.13	316.70

5 Conclusion

In this paper, we presented a point halving algorithm for hyperelliptic curve cryptosystems (HECC). The proposed formula is an extension of the halving formula for elliptic curves reported by Knudsen [Knu99], in which the halved divisors are computed by solving some special equations that represent the doubled divisor. Because the doubling formula for HECC is relatively complicated, the underlying halving algorithm is in general less efficient than that for elliptic curves. However, we can achieve a faster halving algorithm for some curves using the normal basis. For example, the proposed halving algorithm for a curve with $h(x) = x^2 + x + 1$ is about 5% faster than the doubling algorithm. The proposed algorithm using the polynomial basis was implemented in software. We have found a curve whose halving formula can be computed faster. Note that the computation of the square root for this curve is faster than the standard one. We expect that this formula can be further improved by choosing other combinations of basic field operations suitable for the curve.

References

[Ava04] R. Avanzi, "Aspects of Hyperelliptic Curves over Large Prime Fields in Software Implementations," CHES 2004, LNCS 3156, pp.148-162, 2004.

[ACF04] R. Avanzi, M. Ciet, and F. Sica, "Faster Scalar Multiplication on Koblitz Curves Combining Point Halving with the Frobenius Endomorphism," PKC 2004, LNCS 2947, pp.28-40, 2004.

[Can87] D. Cantor, "Computing in the Jacobian of a Hyperelliptic Curve," Mathematics of Computation, 48, 177, pp.95-101, 1987.

[Duq04] S. Duquesne, "Montgomery Scalar Multiplication for Genus 2 Curves," ANTS 2004, LNCS 3076, pp.153-168, 2004.

[FHL+03] K. Fong, D. Hankerson, J. López, and A. Menezes, "Field inversion and point halving revised," Technical Report CORR2003-18, http://www.cacr.math.uwaterloo.ca/techreports/2003/corr2003-18.pdf

[GH00] P. Gaudry and R. Harley, "Counting Points on Hyperelliptic Curves over Finite Fields," ANTS 2000, LNCS 1838, pp.313-332, 2000.

[HHM00] D. Hankerson, J. Hernandez, A. Menezes, "Software Implementation of Elliptic Curve Cryptography over Binary Fields," CHES 2000, LNCS 1965, pp.1-24, 2000.

[Har00a] R. Harley, "Adding.txt," 2000. http://cristal.inria.fr/~harley/hyper/

[Har00b] R. Harley, "Doubling.c," 2000. http://cristal.inria.fr/~harley/hyper/

[KR04] B. King and B. Rubin, "Improvements to the Point Halving Algorithm," ACISP 2004, LNCS 3108, pp.262-276, 2004.

[Kob89] N. Koblitz, "Hyperelliptic Cryptosystems," Journal of Cryptology, Vol.1, pp.139-150, 1989.

[Knu99] E. Knudsen, "Elliptic Scalar Multiplication Using Point Halving," ASIACRYPT '99, LNCS 1716, pp.135-149, 1999.

[Lan02a] T. Lange, "Efficient Arithmetic on Genus 2 Hyperelliptic Curves over Finite Fields via Explicit Formulae," Cryptology ePrint Archive, 2002/121, IACR, 2002.

[Lan04] T. Lange, "Montgomery Addition for Genus Two Curves," ANTS 2004, LNCS 3076, pp.309-317, 2004.

[Lan02b] T. Lange, "Inversion-Free Arithmetic on Genus 2 Hyperelliptic Curves," Cryptology ePrint Archive, 2002/147, IACR, 2002.

[Lan02c] T. Lange, "Weighed Coordinate on Genus 2 Hyperelliptic Curve," Cryptology ePrint Archive, 2002/153, IACR, 2002.

[MAG] MAGMA: The Magma Computational Algebra System for Algebra, Number Theory and Geometry http://magma.maths.usyd.edu.au/magma/

[Men93] A. Menezes, *Elliptic Curve Public Key Cryptosystems*, Kluwer Academic Publishers, 1993.

[Mum84] D. Mumford, *Tata Lectures on Theta II*, Progress in Mathematics 43, Birkhäuser, 1984.

[MCT01] K. Matsuo, J. Chao and S. Tsuji, "Fast Genus Two Hyperelliptic Curve Cryptosystems," Technical Report ISEC2001-31, IEICE Japan, pp.89-96, 2001.

[PWP03] J. Pelzl, T. Wollinger, and C. Paar, "High Performance Arithmetic for Hyperelliptic Curve Cryptosystems of Genus Two," Cryptology ePrint Archive, 2003/212, IACR, 2003.

[PWG+03] J. Pelzl, T. Wollinger, J. Guajardo and C. Paar, "Hyperelliptic Curve Cryptosystems: Closing the Performance Gap to Elliptic Curves," CHES 2003, LNCS 2779, pp.351-365, 2003.

[Sha01] S. Shantz, "From Euclid's GCD to Montgomery Multiplication to the Great Divide," TR-2001-95, Sun Microsystems, Inc., 2001.

[SMC+02] T. Sugizaki, K. Matsuo, J. Chao, and S. Tsujii, "An Extension of Harley Addition Algorithm for Hyperelliptic Curves over Finite Fields of Characteristic Two," Technical Report ISEC2002-9, IEICE Japan, pp.49-56, 2002.

A Proof of Lemma

Lemma 1. *Let $h(x)$ be an irreducible polynomial. There is only one value k_1 which satisfies both equations (3) and (4). Equation (4) has a solution only for the correct k_1. There is only one value k_0 which yields the halved divisor D_1 in algorithm Sketch HEC_HLV. Equation $xh_2 + x^2u_{11} + 1 = 0$ has a solution only for the correct k_0.*

Proof. In this paper we assume that the order of $\mathbf{J}(\mathbb{F}_{2^n})$ is $2 \times r$, where r is a large prime number. For a given divisor D_2, there are two points whose doubled reduced divisor is equal to D_2. The halved divisor equals either $[\frac{1}{2}]D_2$ or $[\frac{1}{2}]D_2 + T_2$, where T_2 is an element in the kernel of the multiplication-by-two in $\mathbf{J}(\mathbb{F}_{2^n})$. We call $[\frac{1}{2}]D_2$ the proper halved divisor.

From the condition of coeff$(U', 3) = 0$, equation (3) is always solvable. For each solution of equation (3), there exist two solutions of equation (4) due to coeff$(U', 1) = 0$. From these values we obtain four different divisors using equations (5),(6), and one of them is the proper halved value $[\frac{1}{2}]D$. In the following, we discuss how to select the proper divisor.

At first we prove that if $h(x)$ is an irreducible polynomial, equation (4) is solvable only for one solution of equation (3). It is well known that equation $ax^2 + bx + c = 0$ has roots if and only if $\mathrm{Tr}(ac/b^2) = 0$. Let one root of $ax^2 + bx + c = 0$ be x_0 and the other be $x_0 + b/a$. Let the two roots of equation (3) be k_1 and $k_1' = k_1 + h_2/u_{21}$. Equation (4) with k_1' substituted is as follows:

$$(k_1 + h_2/u_{21})h_0 + k_0h_1 + k_0^2u_{21} + c_1 = 0. \tag{9}$$

Now we compute $\mathrm{Tr}(ac/b^2)$ of the equation (9):

$$\mathrm{Tr}((k_1 + h_2/u_{21})h_0 + c_1)u_{21}/h_1^2) = \mathrm{Tr}((k_1h_0 + c_1)u_{21}/h_1^2) + \mathrm{Tr}((h_0h_2)/h_1^2) \tag{10}$$

Because $\mathrm{Tr}(ac/b^2)$ of the equation (4) is $\mathrm{Tr}((k_1h_0 + c_1)u_{21}/h_1^2)$, if $\mathrm{Tr}((h_0h_2)/h_1^2) = 1$ i.e. $h(x) = h_2x^2 + h_1x + h_0$ is irreducible, one equation has two roots and the other equation has no roots. This leads to the uniqueness of k_1. Therefore, we can select a proper value from $\{k_1, k_1'\}$ by checking the trace of equation (4).

Next we show how to choose the proper k_0. Equation (4) has two roots k_0 and k_0' for the proper k_1 described above. Two different halved divisor $[\frac{1}{2}]D_2$ and $[\frac{1}{2}]D_2 + T_2$ can be obtained by k_0 and k_0'. We will distinguish the proper divisor by applying the above halving algorithm again. Let $D_1 = [\frac{1}{2}]D_2$. The halving algorithm for D_1 yields two divisors $[\frac{1}{2}]D_1 \in \mathbf{J}(\mathbb{F}_{2^n})$ and $[\frac{1}{2}]D_1 + T_2 \in \mathbf{J}(\mathbb{F}_{2^n})$. On the other hand, for $D_1' = [\frac{1}{2}]D_2 + T_2$, there are two halved divisors: $[\frac{1}{2}]D_1' + T_4$ and $[\frac{1}{2}]D_1' + [3]T_4$, where T_4

and $[3]T_4$ are two divisors of order four in $\mathbf{J}$ not in $\mathbf{J}(\mathbb{F}_{2^n})$, namely $[\frac{1}{2}]D_1' + T_4 \notin \mathbf{J}(\mathbb{F}_{2^n})$ and $[\frac{1}{2}]D_1' + [3]T_4 \notin \mathbf{J}(\mathbb{F}_{2^n})$. Therefore, the proper k_0 should satisfy halved D_1 in $\mathbf{J}(\mathbb{F}_{2^n})$. In the other words, if and only if k_0 (or k_0') is proper, equation $xh_2 + x^2 u_{11} + 1 = 0$ has two roots over $\mathbb{F}_{2^n}$, where u_{11} is computed from k_0 (or k_0') using equation (5). Consequently, we can select the proper k_0 by checking the trace of equation $xh_2 + x^2 u_{11} + 1 = 0$. □

B Improved Algorithms

Algorithm 5 *HEC_HLV* ($h_2 = h_1 = 1, f_4 = 0$)

Input: $D_2 = (U_2, V_2),\ invu = 1/u_{21}$
Output: $D_1 = (U_1, V_1),\ invu = 1/u_{11}$
$U_i(x) = x^2 + u_{i1}x + u_{i0},\ V_i = v_{i1}x + v_{i0},\ \gcd(U_i, h) = 1$

step	*procedure*	*cost*
1.	**Solve** $k_1 + k_1^2 u_{21} + 1 = 0$	$1M + 1H$
	$\alpha \leftarrow invu,\ k_1 \leftarrow \mathrm{H}(u_{21})\alpha,\ k_1' \leftarrow k_1 + \alpha$	
2.	**Select correct** k_1 **by solving** $k_1 h_0 + k_0 + k_0^2 u_{21} + c_1 = 0$	$4M + 2S + 1H + 1T$
	$c_1 \leftarrow f_3 + v_{21} + u_{20} + u_{21}^2$	$(h_0 = 1,\ 3M)$
	$c_0 \leftarrow f_2 + v_{20} + v_{21} + v_{21}^2 + u_{21}(u_{20} + c_1)$	
	$\gamma \leftarrow (c_1 + k_1 h_0) u_{21}$	
	if $\mathrm{Tr}(\gamma) = 1$ *then* $k_1 \leftarrow k_1',\ \gamma \leftarrow \gamma + h_0$	
	$k_0 \leftarrow \mathrm{H}(\gamma)\alpha,\ k_0' \leftarrow k_0 + \alpha$	
3.	**Select correct** k_0 **by solving** $x + x^2 u_{11} + 1 = 0$	$6M + 1S + 1I + 2SR + 1T$
	$w_0 \leftarrow k_1^2,\ w_1 \leftarrow w_0 u_{20} + k_1 + u_{21}$	
	$w_2 \leftarrow k_0 + \sqrt{w_1 + k_0},\ w_3 \leftarrow k_0' + \sqrt{w_1 + k_0'},\ w_4 \leftarrow w_2 w_3$	
	$w_1 \leftarrow 1/(w_4 k_1),\ w_4 \leftarrow w_1 w_4,\ u_{11} \leftarrow w_2 w_4$	
	if $\mathrm{Tr}(u_{11}) = 1$ *then*	
	$k_0 \leftarrow k_0',\ u_{11} \leftarrow w_3 w_4,\ w_2 \leftrightarrow w_3$	
4.	**Compute** U_1	$6M + 1SR$
	$w_0 \leftarrow w_0 w_1,\ w_1 \leftarrow k_0 u_{20},\ w_5 \leftarrow k_1 u_{21}$	
	$w_6 \leftarrow (k_0 + k_1)(u_{20} + u_{21})$	
	$u_{10} \leftarrow w_4 \sqrt{k_0(w_1 + h_0) + c_0}$	
5.	**Compute** $V_1 = V_2 + h + kU_2 \bmod U_1$	$3M$
	$w_4 \leftarrow w_5 + k_0 + 1,\ w_5 \leftarrow w_1 + w_5 + w_6 + v_{21} + 1$	
	$w_6 \leftarrow w_1 + v_{20} + h_0$	
	$invu \leftarrow w_0 w_3$	
	$w_0 \leftarrow w_2 + w_4,\ w_1 \leftarrow w_0 u_{10},\ w_3 \leftarrow (k_0 + k_1)(u_{10} + u_{11})$	
	$v_{11} \leftarrow w_1 + w_2 + w_3 + w_5$	
	$v_{10} \leftarrow w_1 + w_6$	
total	*worst case* ($h_0 \neq 0, 1$)	$20M + 3S + 1I$ $+3SR + 2H + 2T$
	worst case ($h_0 = 1$)	$19M + 3S + 1I$ $+3SR + 2H + 2T$

Algorithm 6 *HEC_HLV* ($y^2 + xy = x^5 + f_1x + f_0$)

Input: $D_2 = (U_2, V_2)$

Output: $D_1 = (U_1, V_1)$, $U_i(x) = x^2 + u_{i1}x + u_{i0}$, $V_i = v_{i1}x + v_{i0}$, $\gcd(U_i, h) = 1$

step	*procedure*	*cost*
1.	**Solve** $k_1^2 u_{21} + 1 = 0$	$1I + 1SR$
	$w_0 \leftarrow 1/u_{21},\ k_1 \leftarrow \sqrt{w_0}$	
2.	**Solve** $k_0 + k_0^2 u_{21} + c_1 = 0$	$3M + 2S + 1SR + 1H$
	$c_1 \leftarrow u_{20} + u_{21}^2,\ w_1 \leftarrow c_1 u_{21},\ c_0 \leftarrow v_{21} + v_{21}^2 + u_{21}u_{20} + w_1$	
	$invk_1 \leftarrow \sqrt{u_{21}},\ w_2 \leftarrow H(w_1),\ w_3 \leftarrow w_2 + 1$	
	$k_0 \leftarrow w_0 w_2,\ k_0' \leftarrow k_0 + w_0$	
3.	**Compute** U_1	$4M + 3SR + 1T$
	$u_{11} \leftarrow \sqrt{invk_1 + k_0}, u_{10} \leftarrow \sqrt{(k_0 + c_1)u_{20} + c_0 u_{21}}$	
	if $Tr(u_{11}(u_{10} + invk_1 + k_0)) = 1$ *then*	
	$k_0 \leftarrow k_0',\ w_2 \leftarrow w_3,\ u_{11} \leftarrow u_{11} + k_1,\ u_{10} \leftarrow u_{10} + \sqrt{w_0 u_{20}}$	
5.	**Compute** $V_1 = V_2 + h + kU_2 \bmod U_1$	$5M$
	$w_1 \leftarrow k_1(u_{21} + u_{11}) + k_0$	
	$v_{11} \leftarrow k_1(u_{20} + u_{10}) + w_2 + v_{21} + 1 + u_{11}w_1$	
	$v_{10} \leftarrow k_0 u_{20} + v_{20} + u_{10}w_1$	
total	*worst case*	$12M + 2S + 1I$ $+5SR + 1H + 1T$

B.1 Implemented Curves

We implement scalar multiplication via a point halving algorithm for some hyperelliptic curves. Let C_1 and C_2 be the genus 2 hyperelliptic curve over $\mathbb{F}_{2^{83}}$, C_3 and C_4 be over $\mathbb{F}_{2^{89}}$, C_5 and C_6 be over $\mathbb{F}_{2^{113}}$. The curves are generated by MAGMA2.11-2 [MAG]. These finite fields are defined as follows: $\mathbb{F}_{2^{83}}$ defined by $\mathbb{F}_2[z]/(z^{83}+z^7+z^4+z^2+1)$, $\mathbb{F}_{2^{89}}$ defined by $\mathbb{F}_2[z]/(z^{89}+z^{38}+1)$, and $\mathbb{F}_{2^{113}}$ defined by $\mathbb{F}_2[z]/(z^{113}+z^9+1)$.

C_1	$h(x) = x^2 + x + 26c76058679188f41f7bc,$
	$f(x) = x^5 + 3510a16ab31776237482dx^3 + 2905946b804eae26b529dx^2$
	$+12e53b7f0fb5f060de6f0x + 15000c12e0c816fa3c90d,$
	$\#\mathbf{J} = 2 \times 4676805239460357292759365238347420157849190892559 9.$
C_2	$h(x) = x^2 + x + 1,$
	$f(x) = x^5 + 6aeccc919ba7b17905576x^3 + 674ad22c4ae3a624f2662x^2$
	$+64ed5727767c5bd20a4d2x + 12d5d21396915b5770a14,$
	$\#\mathbf{J} = 2 \times 4676805239460630644707862135743424067480738494865 7.$
C_3	$h(x) = x^2 + x + 10b4e25d9f772a17f079633,$
	$f(x) = x^5 + 1cdcfccf952029e11a6d591x^3 + 57a45459240e1bff1e9c51x^2$
	$+1233884427d30a923afef30x + 5232b0a2cc9c4205f0a77e,$
	$\#\mathbf{J} = 2 \times 1915619426082477278898178152171702688497048250258700 87.$
C_4	$h(x) = x^2 + x + 1,$
	$f(x) = x^5 + 12c8551e2a5509652647b44x^3 + 10eed3aaec17720e2588277x^2$
	$+1e27989afc4eb5c4288db79x + 16cc764d307ce3295795cbd,$
	$\#\mathbf{J} = 2 \times 1915619426082384539288415427652557953002822708092835 83.$
C_5	$h(x) = x^2 + x + 18950df491b5d1a9a6e87a774fad5,$
	$f(x) = x^5 + 709a4534d9cf5dd288afe6599b38x^3 + c3fcbbe4bfa11c829415ea195110x^2$
	$+8c519e519c221e675f5070bb301cx + 18950df491b5d1a9a6e87a774fad5,$
	$\#\mathbf{J} = 2 \times 5391989333430127925729172349231469275041057667978090967549080059 4763.$
C_6	$h(x) = x^2 + x + 1,$
	$f(x) = x^5 + 9b682c368ea87846032596a2dcd9x^3 + 6b2e43411662ce9d27825f186566x^2$
	$+636a408c351d9d674bffa8b62409x + 60e0f7a6be308f37cc499646fa58,$
	$\#\mathbf{J} = 2 \times 5391989333430127930832531717139226944890835727648206378821370888 6977.$