PAPER Special Section on Cryptography and Information Security

Improvements of Addition Algorithm on Genus 3 Hyperelliptic Curves and Their Implementation

Masaki GONDA[†], *Nonmember*, **Kazuto MATSUO**[††a)], **Kazumaro AOKI**[†††], **Jinhui CHAO**[†], *Members*, *and* **Shigeo TSUJII**[††], *Fellow, Honorary Member*

SUMMARY Genus 3 hyperelliptic curve cryptosystems are capable of fast-encryption on a 64-bit CPU, because a 56-bit field is enough for their definition fields. Recently, Kuroki et al. proposed an extension of the Harley algorithm, which had been known as the fastest addition algorithm of divisor classes on genus 2 hyperelliptic curves, on genus 3 hyperelliptic curves and Pelzl et al. improved the algorithm. This paper shows an improvement of the Harley algorithm on genus 3 hyperelliptic curves using Toom's multiplication. The proposed algorithm takes only $I + 70M$ for an addition and $I + 71M$ for a doubling instead of $I + 76M$ and $I + 74M$ respectively, which are the best possible of the previous works, where I and M denote the required time for an inversion and a multiplication over the definition field respectively. This paper also shows 2 variations of the proposed algorithm in order to adapt the algorithm to various platforms. Moreover this paper discusses finite field arithmetic suitable for genus 3 hyperelliptic curve cryptosystems and shows implementation results of the proposed algorithms on a 64-bit CPU. The implementation results show a 160-bit scalar multiplication can be done within 172 μs on a 64-bit CPU Alpha EV68 1.25 GHz.

key words: *genus 3 hyperelliptic curves, hyperelliptic curve cryptosystems, addition algorithm, Cantor algorithm, Harley algorithm, Toom's multiplication*

1. Introduction

Hyperelliptic curve cryptosystems have been able to be used in practical systems since a fast addition algorithm of divisor classes on hyperelliptic curves (Cantor algorithm) was proposed by Cantor [7].

Recently, Harley [11], [12] proposed a new fast addition algorithm on genus 2 hyperelliptic curves (Harley algorithm). In the same fashion as the Cantor algorithm, the Harley algorithm uses Mumford's representation [23] for input and output divisor classes, but uses a method similar to chord-tangent law in an elliptic curve instead of the quadratic form computation for polynomials used in the Cantor algorithm. Moreover it uses modern polynomial computation techniques i.e. Chinese remainder theorem, Newton's iteration, and Karatsuba's multiplication. Consequently the Harley algorithm is faster than the Cantor algorithm in general. Besides [20] showed that the performance of hyperelliptic curve cryptosystems using the Harley algorithm is theoretically the same as elliptic curve cryptosystems and a lot of researches on the Harley algorithm has been proposed afterwords [3], [14]–[19], [21], [22], [25]–[27], [30]–[34]. Especially Kuroki et al. [15] proposed an extension of the Harley algorithm on genus 3 hyperelliptic curves over odd characteristic fields and Pelzl et al. [25] showed its improvement and generalization for arbitrary characteristic fields.

Genus 3 hyperelliptic curve cryptosystems can be constructed on 56-bit fields from the Hasse-Weil range [29], even if Thériault's improvement [35] of Gaudry's attack [10] is taken into account. Therefore the implementation on a 64-bit CPU does not need multi-precision arithmetic and genus 3 hyperelliptic curve cryptosystems are capable of fast-encryption.

This paper proposes improvements of the Harley algorithm on genus 3 hyperelliptic curves from [15] and [25] and shows implementation results of the proposed algorithms. Moreover this paper discusses finite field arithmetic suitable for genus 3 hyperelliptic curve cryptosystems to take the properties of the definition field and the platform chosen in this paper into consideration.

In this paper, A, I, M, denote the required time of computing $a + b$, $1/a$ and ab for the definition field elements a, b respectively. To take the properties of the definition field and the platform chosen in this paper into account, the required time for $-a$, $a - b$, $2a$, and $a/2$ are also denoted as A, moreover, the required time for a^2 is also denoted as M.

2. Genus 3 Hyperelliptic Curves and Their Divisor Class Groups

Let n be a positive integer, $p \neq 2, 7$ be a prime number, and $q = p^n$. A genus 3 hyperelliptic curve C over $\mathbb{F}_q$ is defined as follows:

$$C : Y^2 = F(X)$$
$$F(X) = X^7 + f_5X^5 + f_4X^4 + \cdots + f_0 \in \mathbb{F}_q[X]$$

with $\mathrm{disc}(F) \neq 0$.

The divisor class group $\mathcal{J}_C(\mathbb{F}_q)$ of C forms a finite Abelian group and therefore cryptosystems based on discrete logarithm problems can be constructed on C. Any

Manuscript received March 22, 2004.
Manuscript revised June 17, 2004.
Final manuscript received August 30, 2004.

[†]The authors are with the Dept. of Information and System Engineering, Chuo University, Tokyo, 112-8851 Japan.
[††]The authors are with the Graduate School of Information Security, Institute of Information Security, Yokohama-shi, 221-0835 Japan, and also with the Research and Development Initiative of Chuo University, Tokyo, 112-8851 Japan.
[†††]The author is with NTT Information Sharing Platform Laboratories, NTT Corporation, Yokosuka-shi, 239-0847 Japan.
a) E-mail: matsuo@iisec.ac.jp

equivalent class $\mathcal{D}$ in $\mathcal{J}_C(\mathbb{F}_q)$ can be represented by Mumford's representation defined as follows [23].

$$\mathcal{D} = (U, V) \in \left(\mathbb{F}_q[X]\right)^2, \text{ where}$$
$$U = \prod (X - x_i)^{\mathrm{ord}_{P_i}(\mathcal{D})},$$
$$y_i = V(x_i)$$

for $P_i = (x_i, y_i) \in C$ with $\mathrm{ord}_{P_i}(\mathcal{D}) > 0$, and

$$\deg V < \deg U,\ F - V^2 \equiv 0 \bmod U.$$

The degree of U is called the weight of $\mathcal{D}$ [11] and $\mathcal{D}$ a reduced divisor, if its weight is less than or equal to 3. Any class in $\mathcal{J}_C(\mathbb{F}_q)$ is uniquely represented by a reduced divisor, i.e. each class includes a unique reduced divisor.

3. Harley Algorithm on Genus 3 Hyperelliptic Curves

This section shows a brief overview of the Harley algorithm on genus 3 hyperelliptic curves according to [15].

The Harley algorithm uses reduced divisors represented by Mumford's representation for input and output divisor classes. In the algorithm, first one executes classification of the input divisor classes by using their weights. Then one executes another classification of the divisor classes by testing $\gcd(U_1, U_2) = 1$ for addition $\mathcal{D}_3 = \mathcal{D}_1 + \mathcal{D}_2$, $\mathcal{D}_1 = (U_1, V_1)$, $\mathcal{D}_2 = (U_2, V_2)$ or by testing $\gcd(U_1, V_1) = 1$ for doubling $\mathcal{D}_2 = 2\mathcal{D}_1$, $\mathcal{D}_1 = (U_1, V_1)$. These gcd computations are carried out by a resultant computation in practice. The cases classified above, i.e. the case satisfied $\deg U_1 = \deg U_2 = 3$ and $\gcd(U_1, U_2) = 1$ for addition and the case satisfied $\deg U_1 = 3$ and $\gcd(U_1, V_1) = 1$ for doubling, are called the most frequent cases. See [15] for details of the classification.

For genus 2 hyperelliptic curves, more precisely classification is executed and a different procedure is used for each case [12], [20], [30]. However, for genus 3 hyperelliptic curves, there exist about 70 cases and therefore the classification needs large costs. On the other hand, the probability of occurring the cases except the most frequent case is $O(1/q)$ in both the addition and the doubling [24], so that these cases can be negligible and an efficient strategy for genus 3 hyperelliptic curves is to take the Harley algorithm only for the most frequent cases and the Cantor algorithm for the other cases. Therefore we will only consider the most frequent cases hereafter.

Algorithm 1 shows an overview of the Harley algorithm on genus 3 hyperelliptic curves for the most frequent case of addition $\mathcal{D}_3 = \mathcal{D}_1 + \mathcal{D}_2$.

Step 1 in Algorithm 1 is called composition part and Step 2 to 3 reduction part. The composition part is accomplished by using Chinese remainder theorem for polynomials and, in doubling, Newton's iteration is used instead of Chinese remainder theorem. Unlike for genus 2 hyperelliptic curves, the reduction part needs 2 steps for genus 3 hyperelliptic curves. The reduction part of the doubling is the same as the addition. In practice, Steps 1 and 2 are computed simultaneously and all detailed algorithm is written down into arithmetic of the definition fields. Moreover the multiplication and the inversion costs are reduced by using Karatsuba's multiplication [13] and Montgomery's multiple inversion technique [8, Algorithm 10.3.4] respectively. In addition, [25] reduces the multiplication costs by applying Bézout's determinant to the resultant computations.

Input: Genus 3 HEC $C : Y^2 = F(X)$, weight 3 reduced divisors $\mathcal{D}_1 = (U_1, V_1)$, $\mathcal{D}_2 = (U_2, V_2)$ with $\gcd(U_1, U_2) = 1$
Output: The weight 3 reduced divisor $\mathcal{D}_3 = (U_3, V_3) = \mathcal{D}_1 + \mathcal{D}_2$
1: Compute $\mathcal{D}_s \sim -\mathcal{D}_3$ such that $U_s = U_1 U_2$ from $\mathcal{D}_1$ and $\mathcal{D}_2$
2: Compute $\mathcal{D}_t \sim \mathcal{D}_3$ such that $\deg U_t = 4$ from $\mathcal{D}_s$
3: Compute $\mathcal{D}_3$ from $\mathcal{D}_t$

Algorithm 1: Harley addition algorithm on genus 3 HEC for the most frequent case

4. Improvements of the Algorithm

This section shows improvements of the Harley algorithm on genus 3 hyperelliptic curves under the standard assumption $A \ll M \ll I$.

The essentials of the improvements are the following 3 points:

1. Using Toom's multiplication
2. Using virtual polynomial multiplication
3. Refining the details

The following discusses on using Toom's multiplication and virtual polynomial multiplication.

4.1 Toom's Multiplication

Toom's multiplication is known as an efficient multiplication algorithm for high-degree polynomials [5], [36]. This algorithm is generally inefficient for low-degree polynomials. However, in certain cases of low-degree polynomial multiplication, the cost of Toom's multiplication is smaller than Karatsuba's one. For example, a multiplication of a degree 2 polynomial and a degree 1 polynomial can be done within $4M$ by Toom's multiplication in stead of $5M$ by Karatsuba's one as the following procedure.

Input: $R = r_2X^2 + r_1X + r_0$, $S = s_1X + s_0$
Output: $T = t_3X^3 + t_2X^2 + t_1X + t_0 = RS$
1: $w_1 = (r_2 + r_1 + r_0)(s_1 + s_0)$
2: $w_2 = (r_2 - r_1 + r_0)(-s_1 + s_0)$
3: $t_0 = r_0 s_0$
4: $t_3 = r_2 s_1$
5: $t_1 = (-2t_3 + w_1 - w_2)/2$
6: $t_2 = (-2t_0 + w_1 + w_2)/2$

Toom's multiplication can be applied twice to reduce the number of multiplications in both the addition and the doubling procedure of the algorithm.

4.2 Virtual Polynomial Multiplication

There is a lot of "multiply-and-add" operations in the Harley algorithm on genus 3 hyperelliptic curves. For example, if there exist sequences such that

$$\cdots + s_1 z_4 + \cdots,$$
$$\cdots + s_1 z_3 + s_0 z_4 + \cdots,$$
$$\cdots + s_0 z_3 + \cdots,$$

then these can be regarded as the polynomial multiplication $(s_1X + s_0)(z_4X + z_3)$ and Karatsuba's multiplication is applicable. Therefore their computations can be done within $3M$ instead of $4M$.

This trick can be applied twice to reduce the number of multiplications in the addition procedure of the algorithm.

4.3 Results

Tables 1 and 2 show the addition procedure and the doubling procedure of the proposed algorithm respectively. In these tables, "(Toom)" and "(Karatsuba)" denote the step using Toom's and Karatsuba's multiplication respectively.

The proposed algorithm takes $I + 70M + 113A$ for an addition and $I + 71M + 107A$ for a doubling respectively. Under the standard assumptions, the cost $I + 70M$ for an addition and $I + 71M$ for a doubling is the best possible as far as we know.

In practical usage of genus 3 hyperelliptic curve cryptosystems, the assumption used here, i.e. $A \ll M \ll I$, is not always satisfied. In that case, i.e. $A \not\ll M$ case, it may be more efficient that Karatsuba's one is used instead of Toom's multiplication or the classical one is used instead of both Toom's and Karatsuba's multiplication. These algorithms are easily obtained from Tables 1 and 2. Consequently we obtain 3 algorithms which can be adapt to various platforms. The costs for the proposed algorithms and the previous works appear in Table 3.

It depends on M/A which one is the fastest in the algorithms shown in Table 3 including the previous works. We show the fastest algorithm for the addition and the doubling

Table 1 Explicit formula of addition on genus 3 HEC (most frequent case).

In.	Genus 3 HEC $C : Y^2 = F(X)$, $F = X^7 + f_5X^5 + f_4X^4 + f_3X^3 + f_2X^2 + f_1X + f_0$; Reduced divisors $\mathcal{D}_1 = (U_1, V_1)$ and $\mathcal{D}_2 = (U_2, V_2)$, $U_1 = X^3 + u_{12}X^2 + u_{11}X + u_{10}$, $V_1 = v_{12}X^2 + v_{11}X + v_{10}$, $U_2 = X^3 + u_{22}X^2 + u_{21}X + u_{20}$, $V_2 = v_{22}X^2 + v_{21}X + v_{20}$;	
Out.	Reduced divisor $\mathcal{D}_3 = (U_3, V_3) = \mathcal{D}_1 + \mathcal{D}_2$, $U_3 = X^3 + u_{32}X^2 + u_{31}X + u_{30}$, $V_3 = v_{32}X^2 + v_{31}X + v_{30}$;	

Step	Procedure	Cost
1	**Compute the resultant r of U_1 and U_2** $t_1 = u_{11}u_{20} - u_{10}u_{21}$; $t_2 = u_{12}u_{20} - u_{10}u_{22}$; $t_3 = u_{20} - u_{10}$; $t_4 = u_{21} - u_{11}$; $t_5 = u_{22} - u_{12}$; $t_6 = t_4^2$; $t_7 = t_3t_4$; $t_8 = u_{12}u_{21} - u_{11}u_{22} + t_3$; $t_9 = t_3^2 - t_1t_5$; $t_{10} = t_2t_5 - t_7$; $r = t_8t_9 + t_2(t_{10} - t_7) + t_1t_6$;	$14M + 12A$
2	**If $r = 0$ then call the Cantor algorithm**	–
3	**Compute the pseudo-inverse $I = i_2X^2 + i_1X + i_0 \equiv r/U_1 \bmod U_2$** $i_2 = t_5t_8 - t_6$; $i_1 = u_{22}i_2 - t_{10}$; $i_0 = u_{21}i_2 - (u_{22}t_{10} + t_9)$;	$4M + 4A$
4	**Compute $S' = s'_2X^2 + s'_1X + s'_0 = rS \equiv (V_2 - V_1)I \bmod U_2$ (Karatsuba, Toom)** $t_1 = v_{10} - v_{20}$; $t_2 = v_{11} - v_{21}$; $t_3 = v_{12} - v_{22}$; $t_4 = t_2i_1$; $t_5 = t_1i_0$; $t_6 = t_3i_2$; $t_7 = u_{22}t_6$; $t_8 = t_4 + t_6 + t_7 - (t_2 + t_3)(i_1 + i_2)$; $t_9 = u_{20} + u_{22}$; $t_{10} = (t_9 + u_{21})(t_8 - t_6)$; $t_9 = (t_9 - u_{21})(t_8 + t_6)$; $s'_0 = -(u_{20}t_8 + t_5)$; $s'_2 = t_6 - (s'_0 + t_4 + (t_1 + t_3)(i_0 + i_2) + (t_{10} + t_9)/2)$; $s'_1 = t_4 + t_5 + (t_9 - t_{10})/2 - (t_7 + (t_1 + t_2)(i_0 + i_1))$;	$10M + 31A$
5	**If $s'_2 = 0$ then call the Cantor algorithm**	–
6	**Compute S, w and $w_i = 1/w$ s.t. $wS = S'/r$ and S is monic** $t_1 = (rs'_2)^{-1}$; $t_2 = rt_1$; $w = t_1s'^2_2$; $w_i = rt_2$; $s_0 = t_2s'_0$; $s_1 = t_2s'_1$;	$I + 7M$
7	**Compute $Z = X^5 + z_4X^4 + z_3X^3 + z_2X^2 + z_1X + z_0 = SU_1$ (Toom)** $t_6 = s_0 + s_1$; $t_1 = u_{10} + u_{12}$; $t_2 = t_6(t_1 + u_{11})$; $t_3 = (t_1 - u_{11})(s_0 - s_1)$; $t_4 = u_{12}s_1$; $z_0 = u_{10}s_0$; $z_1 = (t_2 - t_3)/2 - t_4$; $z_2 = (t_2 + t_3)/2 - z_0 + u_{10}$; $z_3 = u_{11} + s_0 + t_4$; $z_4 = u_{12} + s_1$;	$4M + 15A$
8	**Compute $U_t = X^4 + u_{t3}X^3 + u_{t2}X^2 + u_{t1}X + u_{t0} =$** **$(S(Z + 2w_iV_1) - w_i^2((F - V_1^2)/U_1))/U_2$ (Karatsuba)** $t_1 = s_0z_3$; $t_2 = (u_{22} + u_{21})(u_{t3} + u_{t2})$; $t_3 = u_{21}u_{t2}$; $t_4 = t_1 - t_3$; $u_{t3} = z_4 + s_1 - u_{22}$; $t_5 = s_1z_4 - u_{22}u_{t3}$; $u_{t2} = z_3 + s_0 + t_5 - u_{21}$; $u_{t1} = z_2 + t_6(z_4 + z_3) + w_i(2v_{12} - w_i) - (t_5 + t_2 + t_4 + u_{20})$; $u_{t0} = z_1 + t_4 + s_1z_2 + w_i(2(v_{11} + s_1v_{12}) + w_iu_{12}) - (u_{22}u_{t1} + u_{20}u_{t3})$;	$13M + 26A$
9	**Compute $V_t = v_{t2}X^2 + v_{t1}X + v_{t0} \equiv wZ + V_1 \bmod U_t$** $t_1 = u_{t3} - z_4$; $v_{t0} = w(t_1u_{t0} + z_0) + v_{10}$; $v_{t1} = w(t_1u_{t1} + z_1 - u_{t0}) + v_{11}$; $v_{t2} = w(t_1u_{t2} + z_2 - u_{t1}) + v_{12}$; $v_{t3} = w(t_1u_{t3} + z_3 - u_{t2})$;	$8M + 11A$
10	**Compute $U_3 = X^3 + u_{32}X^2 + u_{31}X + u_{30} = (F - V_t^2)/U_t$** $t_1 = 2v_{t3}$; $u_{32} = -(u_{t3} + v_{t3}^2)$; $u_{31} = f_5 - (u_{t2} + u_{32}u_{t3} + t_1v_{t2})$; $u_{30} = f_4 - (u_{t1} + v_{t2}^2 + u_{32}u_{t2} + u_{31}u_{t3} + t_1v_{t1})$;	$7M + 11A$
11	**Compute $V_3 = v_{32}X^2 + v_{31}X + v_{30} \equiv V_t \bmod U_3$** $v_{32} = v_{t2} - u_{32}v_{t3}$; $v_{31} = v_{t1} - u_{31}v_{t3}$; $v_{30} = v_{t0} - u_{30}v_{t3}$;	$3M + 3A$
Total		$I + 70M + 113A$

Table 2 Explicit formula of doubling on genus 3 HEC (most frequent case).

In.	Genus 3 HEC $C : Y^2 = F(X)$, $F = X^7 + f_5X^5 + f_4X^4 + f_3X^3 + f_2X^2 + f_1X + f_0$; Reduced divisor $\mathcal{D}_1 = (U_1, V_1)$, $U_1 = X^3 + u_{12}X^2 + u_{11}X + u_{10}$, $V_1 = v_{12}X^2 + v_{11}X + v_{10}$;	
Out.	Reduced divisor $\mathcal{D}_2 = (U_2, V_2) = 2\mathcal{D}_1$, $U_2 = X^3 + u_{22}X^2 + u_{21}X + u_{20}$, $V_2 = v_{22}X^2 + v_{21}X + v_{20}$;	
Step	Procedure	Cost
1	**Compute the resultant r of U_1 and V_1** $t_1 = u_{11}v_{10} - u_{10}v_{11}$; $t_2 = u_{12}v_{10} - u_{10}v_{12}$; $t_3 = v_{11}^2$; $t_4 = v_{11}v_{10}$; $t_5 = v_{10} + u_{12}v_{11} - u_{11}v_{12}$; $t_6 = v_{10}^2 - v_{12}t_1$; $t_7 = v_{12}t_2 - t_4$; $r = t_5t_6 + t_2(t_7 - t_4) + t_1t_3$;	$14M + 9A$
2	**If $r = 0$ then call the Cantor Algorithm**	–
3	**Compute the pseudo-inverse $I = i_2X^2 + i_1X + i_0 \equiv r/V_1 \bmod U_1$** $i_2 = t_3 - v_{12}t_5$; $i_1 = u_{12}i_2 + t_7$; $i_0 = u_{11}i_2 + u_{12}t_7 + t_6$;	$4M + 4A$
4	**Compute $Z = z_2X^2 + z_1X + z_0 \equiv (F - V_1^2)/U_1 \bmod U_1$** $t_1 = 2u_{10}$; $t_2 = 2u_{11}$; $t_3 = u_{12}^2$; $t_4 = f_4 - (t_1 + v_{12}^2)$; $t_5 = f_5 + t_3 - t_2$; $t_{10} = 2v_{12}$; $z_2 = t_5 + 2t_3$; $z_1 = u_{12}(t_2 - t_5) + t_4$; $z_0 = f_3 + t_3(t_5 - u_{11}) + u_{12}(t_1 - t_4) + u_{11}(u_{11} - f_5) - t_{10}v_{11}$;	$7M + 18A$
5	**Compute $S' = s_2'X^2 + s_1'X + s_0' = 2rS \equiv ZI \bmod U_1$ (Karatsuba, Toom)** $t_1 = i_1z_1$; $t_2 = i_0z_0$; $t_3 = i_2z_2$; $t_4 = u_{12}t_3$; $t_5 = (i_2 + i_1)(z_2 + z_1) - (t_1 + t_3 + t_4)$; $t_6 = u_{10}t_5$; $t_7 = u_{10} + u_{12}$; $t_8 = t_7 + u_{11}$; $t_9 = t_7 - u_{11}$; $t_7 = t_8(t_3 + t_5)$; $t_{11} = t_9(t_5 - t_3)$; $s_2' = t_1 + t_6 + (i_2 + i_0)(z_2 + z_0) - (t_2 + t_3 + (t_7 + t_{11})/2)$; $s_1' = t_4 + (i_0 + i_1)(z_1 + z_0) + (t_{11} - t_7)/2 - (t_1 + t_2)$; $s_0' = t_2 - t_6$;	$10M + 28A$
6	**If $s_2' = 0$ then call the Cantor Algorithm**	–
7	**Compute S, w and $w_i = 1/w$ s.t. $wS = S'/(2r)$ and S is monic** $t_1 = 2r$; $t_2 = (t_1s_2')^{-1}$; $t_3 = t_1t_2$; $w = t_2s_2'^2$; $w_i = t_1t_3$; $s_0 = t_3s_0'$; $s_1 = t_3s_1'$;	$I + 7M + A$
8	**Compute $G = X^3 + g_4X^4 + g_3X^3 + g_2X^2 + g_1X + g_0 = SU_1$ (Toom)** $t_1 = t_8(s_1 + s_0)$; $t_2 = t_9(s_0 - s_1)$; $t_3 = u_{12}s_1$; $g_0 = u_{10}s_0$; $g_1 = (t_1 - t_2)/2 - t_3$; $g_2 = u_{10} + (t_1 + t_2)/2 - g_0$; $g_3 = t_3 + u_{11} + s_0$; $g_4 = u_{12} + s_1$;	$4M + 12A$
9	**Compute $U_t = X^4 + u_{t3}X^3 + u_{t2}X^2 + u_{t1}X + u_{t0} = ((G + w_iV_1)^2 - w_i^2F)/U_1^2$** $u_{t3} = 2s_1$; $u_{t2} = s_1^2 + 2s_0$; $u_{t1} = u_{t3}s_0 + w_i(t_{10} - w_i)$; $u_{t0} = s_0^2 + 2w_i((s_1 - u_{12})v_{12} + v_{11} + w_iu_{12})$;	$7M + 10A$
10	**Compute $V_t = v_{t3}X^3 + v_{t2}X^2 + v_{t1}X + v_{t0} \equiv wG + V_1 \bmod U_t$** $t_1 = u_{t3} - g_4$; $v_{t0} = w(t_1u_{t0} + g_0) + v_{10}$; $v_{t1} = w(t_1u_{t1} + g_1 - u_{t0}) + v_{11}$; $v_{t2} = w(t_1u_{t2} + g_2 - u_{t1}) + v_{12}$; $v_{t3} = w(t_1u_{t3} + g_3 - u_{t2})$;	$8M + 11A$
11	**Compute $U_2 = X^3 + u_{22}X^2 + u_{21}X + u_{20} = (F - V_t^2)/U_t$** $t_1 = 2v_{t3}$; $u_{22} = -(u_{t3} + v_{t3}^2)$; $u_{21} = f_5 - (u_{t2} + u_{22}u_{t3} + t_1v_{t2})$; $u_{20} = f_4 - (u_{t1} + v_{t2}^2 + u_{22}u_{t2} + u_{21}u_{t3} + t_1v_{t1})$;	$7M + 11A$
12	**Compute $V_2 = v_{22}X^2 + v_{21}X + v_{20} \equiv V_t \bmod U_2$ (Karatsuba)** $v_{22} = v_{t2} - u_{22}v_{t3}$; $v_{21} = v_{t1} - u_{21}v_{t3}$; $v_{20} = v_{t0} - u_{20}v_{t3}$;	$3M + 3A$
Total		$I + 71M + 107A$

Table 3 Results and comparison of the addition algorithms on genus 3 hyperelliptic curves. "Toom" denotes the algorithm shown in Tables 1 and 2, "Karatsuba" the algorithm without Toom's multiplication, "Classical" the algorithm used only the classical multiplication.

	Addition	Doubling
Previous works		
[15]	$I + 81M + 125A$	$I + 74M + 125A$
[25]	$I + 76M + 95A$	$I + 75M + 97A$
This work		
Toom	$I + 70M + 113A$	$I + 71M + 107A$
Karatsuba	$I + 72M + 111A$	$I + 73M + 101A$
Classical	$I + 79M + 83A$	$I + 78M + 83A$

Table 4 The fastest algorithm with respect to M/A.

M/A	≤ 3.2	3.3	3.4	≥ 3.5
Addition	Classical		Toom	
Doubling	Classical			Toom

with respect to M/A in Table 4.

Note that many other variations of the algorithm can be obtained by partially using another method for each polynomial multiplication in Tables 1 and 2.

5. Implementation on a 64-Bit CPU

We implemented the proposed algorithms on Alpha EV68, which is known as a standard 64-bit CPU. The implementation uses Compaq C++ with inline assembly. The inline assembly is only used for the `umulh` instruction, which returns the high-order 64 bits of ab for 64-bit unsigned integers a and b, and the `cttz` instruction, which returns the position of the first bit to be "1" from the LSB with bit-position of the LSB as 0.

The following discusses

1. Choice of the definition field
2. Arithmetic of the definition field

for 64-bit CPUs and shows implementation results.

5.1 Definition Field

The most efficient attack against genus 3 hyperelliptic curve cryptosystems over $\mathbb{F}_p$ is Thériault's improvement [35] of Gaudry's attack [10] and its complexity is $O(p^{10/7})$, therefore a 56-bit field seems enough for the definition field in order to carry out the same security as a 160-bit elliptic curve

cryptosystem.

On the other hand, most of recent CPUs including Alpha EV68 can multiply (half-)word-size integers fast. Therefore using a prime finite field $\mathbb{F}_p$ as the definition field is efficient in order to carry out the most of this multiplication capability.

It is known that choosing a Mersenne prime as p leads to fast $\mathbb{F}_p$-arithmetic [1], [4] and $2^{61}-1$ is the only Mersenne prime from 56- to 64-bit, so that we choose $p = 2^{61} - 1$ for our implementation.

5.2 Arithmetic of the Definition Field

The implementation of $\mathbb{F}_p$-arithmetic is basically due to [1], [4]. However certain arithmetic can be improved to take the properties of both $p = 2^{61} - 1$ and the Harley algorithm on genus 3 hyperelliptic curves into account. Therefore the following discusses the $\mathbb{F}_p$-arithmetic suitable for genus 3 hyperelliptic curve cryptosystems using the property of $p = 2^{61} - 1$.

Note that the symbol "&" denotes the "bit-wise and" operation in the algorithms described below.

5.2.1 Addition

The implementation of addition over $\mathbb{F}_p$ is not always with a lot of care. Because the condition $A \ll M$ is satisfied in most of the public-key cryptosystem implementation. However genus 3 hyperelliptic curve cryptosystems can be implemented with the most of the multiplication capability of CPUs as described above, so that it is expected that M is small and M/A for genus 3 hyperelliptic curve cryptosystems are smaller than usual public-key cryptosystems. Therefore we must pay attention to the implementation of addition over $\mathbb{F}_p$.

An addition procedure consists of two parts, i.e. an integer addition and a reduction modulo p. The reduction usually needs larger cost than the integer addition. However an efficient reduction procedure can be obtained from the property $2^{61} \equiv 1 \bmod p$. The following shows the addition algorithm over $\mathbb{F}_p$ used in our implementation.

Input: $a, b \in [0, p-1]$
Output: $c \in [0, p-1]$ such that $c \equiv a + b \bmod p$
1: $w = a + b + 1$
2: $c = \lfloor w/2^{61} \rfloor + (w\&p) - 1$

This reduction trick can be also carried out efficient multiple addition algorithm. In fact, the trick can be used for addition of upto 8 elements. The following example shows an algorithm for multiple addition of 4 elements.

Input: $a, b, c, d \in [0, p-1]$
Output: $e \in [0, p-1]$ such that $e \equiv a + b + c + d \bmod p$
1: $w = a + b + c + d + 3$
2: $e = \lfloor w/2^{61} \rfloor + (w\&p) - 3$

Because there are many multiple addition in the Harley algorithm on genus 3 hyperelliptic curves, such kind of algorithm is efficient for speed-up the Harley algorithm. Moreover it is expected that these algorithms bring more efficient implementation on recent CPUs because those have a number of ALUs usually.

5.2.2 Multiplication

The multiplication algorithm used in our implementation is basically due to [4]. In addition, to take 64-bit word boundaries into consideration, more efficient algorithm can be obtained. Moreover a reduction trick similar to the addition's one can be also applicable. The following shows the multiplication algorithm used in our implementation.

Input: $a, b \in [0, p-1]$
Output: $c \in [0, p-1]$ such that $c \equiv ab \bmod p$
1: $w = \lfloor \texttt{mulq}(2^3 a, b)/2^3 \rfloor$
2: $w = w + \texttt{umulh}(2^3 a, b)$
3: $c = \lfloor w/2^{61} \rfloor + (w\&p)$

In the algorithm, $\texttt{mulq}(a, b)$ denotes the function that returns the low-order 64 bits of ab for 64-bit unsigned integers a, b.

5.2.3 Inversion

As efficient inversion algorithm for single-precision arithmetic, there exist extended Euclidean algorithm (EGCD), extended binary GCD algorithm (EBGCD), and $(p-2)$-powering ($p-2$ method). We implemented each algorithm in order to decide which algorithm should be used and an experimental result shows the cost for EGCD > the cost for $p-2$ method $\approx 69M$ > the cost for EBGCD, so that we used EBGCD for the inversion.

The dominant part of the EBGCD is usually the "while-loop" shown as follows.

while t_2 is even **do**
  if t_1 is odd **then**
    $t_1 = t_1 + p$
  $t_1 = \lfloor t_1/2 \rfloor$
  $t_2 = \lfloor t_2/2 \rfloor$

However the computation of t_1 can be regarded as the w-bit cyclic right shifts of 61-bit length by using the properties of $p = 2^{61} - 1$, where w is the number of iterations of the loop, i.e. the 2-adic valuation of t_2. Therefore the loop can be simplified as follows.

$w = \texttt{cttz}(t_2)$
$t_1 = \lfloor t_1/2^w \rfloor + 2^{61-w}(t_1\&(2^w - 1))$
$t_2 = \lfloor t_2/2^w \rfloor$

Table 5 Implemented $\mathbb{F}_p$-arithmetic functions and their latency in CPU cycles on Alpha EV68. $a, \ldots, g$ denote any elements in $\mathbb{F}_p$ or $\mathbb{F}_p^\times$ for the inverse.

Arithmetic	Latency	Arithmetic	Latency
$-a$	3	a^2+b+c	15
$2a$	3	$ab+c+d+e$	16
$a/2$	3	a^2+2bc	18
$a-b$	4	$ab+cd$	18
$a+b$	5	$2ab+cd$	18
$a+b+c$	6	$ab+cd+e$	18
$2a+b$	6	$2ab+cd+e$	19
$a+b+c+d$	6	$2(ab+c)+de$	19
a^2	13	$ab+cd+e+f$	19
ab	13	$ab+cd+ef$	21
a^2+b	15	$2ab+cd+ef$	22
$ab+c$	15	$ab+cd+ef+g$	22
$ab+c+d$	15	$1/a$	564
a^2+2b	15		

5.2.4 Arithmetic of Multiple Elements

The Harley algorithm on genus 3 hyperelliptic curves needs not only multiple addition but also various arithmetic of multiple elements. The reduction trick used in the addition can be applied for these arithmetic. Therefore we implemented all of the function appeared in the proposed algorithm that needs at most 1 reduction which is realized by integer arithmetics. All implemented $\mathbb{F}_p$-arithmetic functions are shown in Table 5.

5.2.5 Timing

We timed the latency of all implemented functions. Table 5 also shows their latency.

All functions except the inversion are constant-time functions for the inputs. So each timings without for the inversion shows the minimum value, which is also the highest frequency, of every 1,000,000 operations with random inputs. For the inversion, the timing shows the average of every 1,000,000 operations with random inputs. The timing function is carefully coded as the "out-of-order" execution does not arise beyond the arithmetic functions. Note that we separately measured the latency of each function. The latency of a sequence of the functions may be different from the simple sum of the functions' one, because the compiler may schedule the instructions over the partial functions for utilizing super-scalar architecture.

5.3 Implementation Results

Table 6 shows implementation results. In the table, "Toom," "Karatsuba," and "Classical" denote the same algorithm in Table 3. We also implemented the algorithm described in [25] by using our $\mathbb{F}_p$-arithmetic functions in order to compare its performance with the proposed algorithms. The timing of this algorithm is denoted by "Pelzl" in Table 6.

We used the signed sliding-window algorithm [6, Algorithm IV.7] of the window-size 5 for the scalar multiplication and 160-bit random integers for the scalars, moreover NTL/GMP [9], [28] for the Cantor algorithm and the sliding-window algorithm. Each timing shows the average of every 1,000,000 operations on 100 random curves with irreducible F.

Table 6 Performance results on Alpha EV68 1.25 GHz.

	Addition	Doubling	Scalar mul.
Toom	919 ns	916 ns	180 μs
Karatsuba	920 ns	897 ns	177 μs
Classical	888 ns	875 ns	172 μs
Pelzl	909 ns	918 ns	180 μs

The results show the algorithm without both Toom's and Karatsuba's multiplication is the fastest in our implementation. This supports the estimation shown in Table 4, because we can estimate the M/A ratio of our implementation to be less than 3.2 from Table 5.

The result, i.e. 888 ns for an addition, 875 ns for a doubling, and 172 μs for a 160-bit scalar multiplication respectively, is the fastest one for hyperelliptic curve addition implementation as far as we know.

Remark 1: On a 32-bit CPU, it is expected that $M \gg A$ for genus 3 hyperelliptic addition implementation, because a multiplication over a 56-bit definition field needs 32-bit word-size multiplication at least 3 times. Therefore the algorithm using both Toom's and Karatsuba's multiplication may be fastest on 32-bit CPUs. However, it seems that an implementation on 32-bit CPU is difficult to be fast as our implementation, according to our rough estimation based on the previous work [3], [14], [25] for genus three hyperelliptic curve implementation.

6. Conclusion

This paper showed improvements of the Harley algorithm on genus 3 hyperelliptic curves and obtained 3 algorithms which take $I + 70M$ for an addition and $I + 71M$ for a doubling, $I+72M$ and $I+73M$, and $I+79M$ and $I+78M$ respectively. Under the standard assumptions, the cost $I+70M$ for an addition and $I + 71M$ for a doubling is the best possible as far as we know.

Moreover this paper discusses finite field arithmetic suitable for genus 3 hyperelliptic curve cryptosystems and showed implementation results of the proposed algorithms on a 64-bit CPU Alpha EV68 1.25 GHz. The implementation results show that an addition, a doubling, and a 160-bit scalar multiplication can be done within 888 ns, 875 ns, and 172 μs respectively. This result is the fastest one for hyperelliptic curve addition implementation as far as we know. On the other hand, the fastest *elliptic* scalar multiplication over Alpha EV6, which is the almost same as EV68, is shown in [2]. By comparing our result with this, it can be seen that the hyperelliptic scalar multiplication is still slower than the elliptic scalar multiplication, even though the hyperelliptic addition is comparable with the elliptic one. The speed of algebraic curve cryptosystems immedi-

ately follows from the scalar multiplication one, so that efficient scalar multiplication algorithm on hyperelliptic curves and its implementation will be important research topics.

Acknowledgement

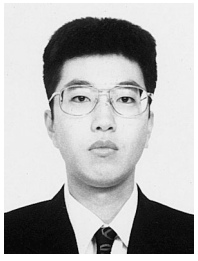
The second author is grateful to Jun-ichi Kuroki for his interesting discussions. The authors would like to thank the anonymous referees and Seigo Arita for their helpful comments.

References

[1] K. Aoki, F. Hoshino, and T. Kobayashi, "The fastest ECC implementations," Proc. SCIS2000, B05, 2000.
[2] K. Aoki, F. Hoshino, and T. Kobayashi, "A cyclic window algorithm for ECC defined over extension fields," in Information and Communications Security, ed. S. Qing, T. Okamoto, and J. Zhou, LNCS 2229, pp.62–73, Springer-Verlag, 2001.
[3] R.M. Avanzi, "Aspects of hyperelliptic curves over large prime fields in software implementations," Cryptology ePrint Archive, Report 2003/253, 2003, http://eprint.iacr.org/
[4] D.V. Bailey and C. Paar, "Optimal extension fields for fast arithmetic in public-key algorithms," Advances in Cryptology—CRYPTO '98, ed. H. Krawczyk, LNCS 1462, pp.472–485, Springer-Verlag, 1998.
[5] D. Bernstein, "Multidigit multiplication for mathematicians," 2001, manuscript, ftp://koobera.math.uic.edu/pub/papers/m3.dvi
[6] I. Blake, G. Seroussi, and N. Smart, Elliptic Curves in Cryptography, LMS 265, Cambridge U.P., 1999.
[7] D.G. Cantor, "Computing in the Jacobian of hyperelliptic curve," Math. Comput., vol.48, no.177, pp.95–101, 1987.
[8] H. Cohen, A Course in Computational Algebraic Number Theory, Graduate Text in Mathematics, no.138, Springer-Verlag, 1993.
[9] F.S. Foundation, "GNU MP," http://www.swox.com/gmp/manual/, 2003.
[10] P. Gaudry, "An algorithm for solving the discrete log problem on hyperelliptic curves," Advances in Cryptology—EUROCRYPT 2000, ed. B. Preneel, LNCS 1807, pp.19–34, Springer-Verlag, 2000.
[11] P. Gaudry and R. Harley, "Counting points on hyperelliptic curves over finite fields," ANTS-IV, ed. W. Bosma, LNCS 1838, pp.313–332, Springer-Verlag, 2000.
[12] R. Harley, "adding.text, doubling.c," http://cristal.inria.fr/~harley/hyper/, 2000.
[13] A. Karatsuba and Y. Ofman, "Multiplication of multidigit numbers on automata," Soviet Physics Doklady, vol.7, pp.595–596, 1963.
[14] I. Kitamura and M. Katagi, "Efficient implementation of genus three hyperelliptic curve cryptography over $\mathbb{F}_{2^n}$," Cryptology ePrint Archive, Report 2003/248, 2003, http://eprint.iacr.org/
[15] J. Kuroki, M. Gonda, K. Matsuo, J. Chao, and S. Tsujii, "Fast genus three hyperelliptic curve cryptosystems," Proc. SCIS2002, pp.503–507, 2002.
[16] T. Lange, "Efficient arithmetic on genus 2 hyperelliptic curves over finite fields via explicit formulae," Cryptology ePrint Archive, Report 2002/121, 2002, http://eprint.iacr.org/
[17] T. Lange, "Inversion-free arithmetic on genus 2 hyperelliptic curves," Cryptology ePrint Archive, Report 2002/147, 2002, http://eprint.iacr.org/
[18] T. Lange, "Weighted coordinates on genus 2 hyperelliptic curves," Cryptology ePrint Archive, Report 2002/153, 2002, http://eprint.iacr.org/
[19] K. Matsuo and J. Chao, "Fast genus two hyperelliptic curve cryptosystems," Talk at the 2nd workshop on cryptology and algebraic curves, Chuo U., http://www.tsujii-lab.ise.chuo-u.ac.jp/hyper/pdf/mats_ohp0108.pdf, 2001.
[20] K. Matsuo, J. Chao, and S. Tsujii, "Fast genus two hyperelliptic curve cryptosystems," IEICE Technical Report, ISEC2001-31, 2001.
[21] P.K. Mishra and P. Sarka, "Parallelizing explicit formula for arithmetic in the Jacobian of hyperelliptic curves," Advances in Cryptology—ASIACRYPT 2003, ed. C.S. Laih, LNCS 2894, pp.93–110, Springer-Verlag, 2003.
[22] Y. Miyamoto, H. Doi, K. Matsuo, J. Chao, and S. Tsujii, "A fast addition algorithm of genus two hyperelliptic curves," Proc. SCIS2002, pp.497–502, 2002.
[23] D. Mumford, Tata Lectures on Theta II, Progress in Mathematics, no.43, Birkhäuser, 1984.
[24] K. Nagao, "Improving group law algorithms for Jacobians of hyperelliptic curves," ANTS-IV, ed. W. Bosma, LNCS 1838, pp.439–448, Springer-Verlag, 2000.
[25] J. Pelzl, T. Wollinger, J. Guajardo, and C. Paar, "Hyperelliptic curve cryptosystems: Closing the performance gap to elliptic curves," CHESS 2003, ed. C.D. Walter and Ç.K. Koç, LNCS2779, pp.351–365, Springer-Verlag, 2003.
[26] J. Pelzl, T. Wollinger, and C. Paar, "High performance arithmetic for hyperelliptic curve cryptosystems of genus two," Cryptology ePrint Archive, Report 2003/212, 2003, http://eprint.iacr.org/
[27] J. Pelzl, T. Wollinger, and C. Paar, "Low cost security: Explicit formulae for genus 4 hyperelliptic curves," SAC 2003, ed. M. Matsui and R.J. Zucherato, LNCS 3006, pp.1–16, Springer-Verlag, 2000.
[28] V. Shoup, "A tour of NTL," http://www.shoup.net/ntl/, 2003.
[29] H. Stichtenoth, Algebraic function fields and codes, Universitext, Springer-Verlag, 1993.
[30] H. Sugizaki, K. Matsuo, J. Chao, and S. Tsujii, "An extension of Harley addition algorithm for hyperelliptic curves over finite fields of characteristic two," IEICE Technical Report, ISEC2002-9, 2002.
[31] H. Sugizaki, K. Matsuo, J. Chao, and S. Tsujii, "A generalized Harley algorithm for genus two hyperelliptic curves," Proc. SCIS2003, pp.917–921, 2003.
[32] M. Takahashi, "Improving Harley algorithms for Jacobians of genus 2 hyperelliptic curves," Proc. SCIS2002, pp.155–160, 2002.
[33] N. Takahashi and M. Miyaji, "Efficient exponentiation on genus two hyperelliptic curves," IEICE Technical Report, ISEC2002-102, 2002.
[34] N. Takahashi, H. Morimoto, and M. Miyaji, "Efficient exponentiation on genus two hyperelliptic curves (ii)," IEICE Technical Report, ISEC2002-145, 2003.
[35] N. Thériault, "Index calculus attack for hyperelliptic curves of small genus," Advances in Cryptology—ASIACRYPT 2003, ed. C.S. Laih, LNCS 2894, pp.75–92, Springer-Verlag, 2003.
[36] A.L. Toom, "The complexity of a scheme of functional elements realizing the multiplication of integers," Soviet Mathematics Doklady, vol.3, pp.714–716, 1963.
[37] H.S. Warren, Jr., Hacker's delight, Addison Wesley, 2003.

Masaki Gonda received the B.E. degree in information and system engineering from Chuo University, Tokyo, Japan, in 2002. He is currently a master student at the Graduate School of Chuo University. His research interests include public key cryptosystems, especially hyperelliptic curve cryptosystems.

Kazuto Matsuo received the B.E., M.E., and D.E. degrees from Chuo University, Tokyo, Japan in 1986, 1988, and 2001, respectively. He joined Toyo Communication Equipment Co., LTD from 1988 to 2001. He is currently a professor in the Graduate school of Information Security at the Institute of Information Security, Yokohama, Japan, and also a professor of the Research and Development Initiative of Chuo University.



Kazumaro Aoki received the B.S., and M.S., and D.S. degrees from Waseda University, Tokyo, Japan, in 1993, 1995, and 2001 respectively. He is a research engineer of NTT Information Sharing Platform Laboratories. In FY2001, he was on loan to TAO (Telecommunications Advancement Organization of Japan) and manages the cryptographic primitive evaluation project called CRYPTREC project. He was awarded the SCIS'95 and SCIS'96 paper prizes, and 1997 IEICE Young Engineer Award. Dr. Aoki is a member of the International Association for Cryptologic Research.

Jinhui Chao received the B.E. from the Department of Electronics Engineering, Xidian University, China in 1982, and the M.E. and D.E. from Tokyo Institute of Technology, Tokyo, Japan in 1985 and 1988, respectively both in electrical engineering. He was an assistant professor of Tokyo Institute of Technology in 1989, from 1992 an associate professor and from 1996 a professor in the Department of Electrical and Electronic Engineering, Chuo University, Tokyo, Japan. He is a professor of Department of Information and System Engineering, Chuo University from 2004. His current research interests include cryptography theory, 3D images, color science, artificial neural networks and nonlinear adaptive signal processing. He received the Excellent Paper Awards from IEICE in 1988 and 1990, and the Shinoda Academic Award from IEICE in 1991.

Shigeo Tsujii received the B.E. and the D.E. in electrical engineering from Tokyo Institute of Technology, Tokyo, Japan in 1958 and 1970, respectively. From 1958 to 1965 he worked for Nippon Electric Company. Between 1965 and 1971 he was an associate professor at Yamanashi University, Kofu, Japan and since 1971, after seven years as an associate professor, he was a professor at Tokyo Institute of Technology, Tokyo, Japan until 1994. From 1994 to 2004 he was a professor in the the Department of Information and System Engineering, Chuo University, Tokyo, Japan. He is currently president of the Institute of Information Security, Yokohama, Japan, a professor of the Research and Development Initiative Chuo University, and a professor emeritus of the Tokyo Institute of Technology. He has served as the president of IEICE and the chairman of the Radio Regulatory Council of MPMHAPT and has held other important posts. He is a member of the Japan Science Council. Among his awards are the IEICE Excellent Paper Prize, the Excellent Performance Prize, and the Distinguished Service Prize. He is an IEEE Fellow. He won the Memorial Prize for the Third Millennium. The PMHAPT Minister officially commended him on Radio Wave Day in 2003. He was given the 55th Broadcasting Culture Award by NHK, the Japan Broadcasting Corporation, in FY2003.