Novel Efficient Implementations of Hyperelliptic Curve Cryptosystems using Degenerate Divisors

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

[1] Sony Corporation, 6-7-35 Kitashinagawa Shinagawa-ku, Tokyo, 141-0001 Japan
{Masanobu.Katagi, Izuru.Kitamura}@jp.sony.com, akishita@pal.arch.sony.co.jp
[2] Technische Universität Darmstadt, Fachbereich Informatik,
Alexanderstr.10, D-64283 Darmstadt, Germany
takagi@informatik.tu-darmstadt.de

Abstract. It has recently been reported that the performance of hyperelliptic curve cryptosystems (HECC) is competitive to that of elliptic curve cryptosystems (ECC). However, it is expected that HECC still can be improved due to their mathematically rich structure. We consider here the application of degenerate divisors of HECC to scalar multiplication. We investigate the operations of the degenerate divisors in the Harley algorithm and the Cantor algorithm of genus 2. The timings of these operations are reported. We then present a novel efficient scalar multiplication method using the degenerate divisors. This method is applicable to cryptosystems with fixed base point, e.g., ElGamal-type encryption, sender of Diffie-Hellman, and DSA. Using a Xeon processor, we found that the double-and-add-always method using the degenerate base point can achieve about a 20% increase in speed for a 160-bit HECC. However, we mounted an timing attack using the time difference to designate the degenerate divisors. The attack assumes that the secret key is fixed and the base point can be freely chosen by the attacker. Therefore, the attack is applicable to ElGamal-type decryption and single-pass Diffie-Hellman — SSL using a hyperelliptic curve could be vulnerable to the proposed attack. Our experimental results show that one bit of the secret key for a 160-bit HECC can be recovered by calling the decryption oracle 500 times.

Keywords. hyperelliptic curve cryptosystem, scalar multiplication, timing attack, degenerate divisor, efficient computation, SSL

1 Introduction

In 1989, Koblitz proposed hyperelliptic curve cryptosystems (HECC) [Kob89]. HECC has the advantage of shorter operand length than elliptic curve cryptosystems (ECC), and it has been expected that HECC can attain a faster encryption compared to ECC due to their rich algebraic structure. Recently some efficient addition formulae of HECC have been proposed (see, for example, [Lan02a-c]), and implementation in software shows the performance of HECC to be competitive to that of ECC [PWG$^+$03,Ava03b].

When we implement a cryptosystem in a real security system, we have to prepared for side channel attacks such as timing attack [Koc96] and power analysis [KJJ99]. A simple timing attack on HECC can reveal the hamming weight of the secret scalar by measuring the computation time of the scalar multiplication. The timing attack using the final subtraction of Montgomery multiplication [Sch00,Sch02,SKQ01] is also applicable to HECC. Other possible attacks use exceptional procedures of the addition formula [AT03,Ava03a,Gou03,IT03]. The addition formula of HECC involves many exceptional procedures, so that we have many possibilities to use them in a timing attack. Note that these attacks assume that the base point D can be freely chosen by the attacker and the secret scalar d is fixed: we can apply our attack to HEC ElGamal-type decryption and single-pass HEC Diffie-Hellman but not HEC DSA.

In this paper, we focus on the degenerate divisors of HECC, which has a lower weight than g, where g is the genus of the underlying curve. We investigate possible degenerate divisors appearing in the Harley algorithm [Har00a,Har00b] and Cantor algorithm [Can87,Kob89] of genus 2. The addition formulae to compute these degenerate divisors have a different running time than the standard divisors. We estimated the time required to compute the doubling and addition using the degenerate divisors. In general, the probability that these procedures will have to be used in the addition formula is $\mathcal{O}(1/q)$, where q is the definition field of HECC [Nag00]. However, in some cryptographic protocols we can use the degenerate divisor in both positive and negative ways.

As a positive application, we present an algorithm to efficiently perform scalar multiplication using a fixed base point. The degenerate divisor, which allows faster hyperelliptic curve addition in the Harley algorithm, is used for the base point. We found that the security of the underlying discrete logarithm problem is not decreased due to the random self reducibility. We also found that a randomly chosen divisor with weight 1 achieves about 20% faster scalar multiplication using the double-and-add-always method for a 160-bit HECC. Efficient scalar multiplication with a fixed base point can be applied to ElGamal-type encryption, sender of Diffie-Hellman, and DSA.

As a negative application, we propose a timing attack on HECC using the degenerate divisors. The algorithm of recovering the secret key draws on previous work [AT03,Ava03a,Gou03, IT03]. We will consider the attack on the hyperelliptic curve with genus 2, especially the Harley algorithm and Cantor algorithm. Several explicit exceptional procedures suitable for the timing attack in the setting are investigated. We examine the exceptional procedures which arise from the divisors with weight 1, whose computational time is faster than the ordinary one in the Harley algorithm. Then we show how to apply these procedures to the timing attack. The exceptional procedures appear with negligible probability in the scalar multiplication for a randomly chosen base point. Thus the exceptional procedures cause a timing difference in the scalar multiplication from ordinary operation. The proposed timing attack analyses the timing difference with many sampling numbers. Our experiment on a Xeon processor using a 160-bit HECC shows that the timing difference is about 0.02 ms for the exceptional case of the Harley algorithm. The difference is quite small, so that we apply the sampling technique proposed by Dhem et al. [DKL+98]. We consider three different types of timing: the timing with a randomly chosen divisor, timing with divisor D_b that brings about an exceptional procedure if the supposed bit is $b = 0, 1$. If b is correct, the timing difference from the random divisor becomes larger than otherwise. Our experiment successfully recovered one bit of secret key with 500 samples for a 160-bit HECC.

This paper is organized as follows: In Section 2, we describe the basic properties of HECC. In Section 3, we review the side channel attacks. In Section 4, the degenerate divisors are investigated, and we present an efficient scalar multiplication using the degenerate divisors.

In Section 5, we propose the new timing attack of HECC and show the experimental results of genus 2 HECC.

2 Hyperelliptic Curve Cryptosystems

In this section we review hyperelliptic curve cryptosystems related to this paper.

2.1 Hyperelliptic Curve

Let g be a positive integer, and let $K = \mathbb{F}_q$ be a finite field of characteristic p, $q = p^n$ where n is a positive integer. Hyperelliptic curve C of genus g over $\mathbb{F}_q$ is defined by equation $y^2 + h(x)y = f(x)$, where $f(x)$ is a monic polynomial over $\mathbb{F}_q$ with degree $2g+1$ in x. $h(x)$ is a polynomial over $\mathbb{F}_q$ with $\deg h \leq g$ for even characteristic and 0 for odd characteristic. Let $P_i = (x_i, y_i)$ be a rational point on curve C and P_∞ be a point at infinity. The inverse of $P = (x, y)$ is the point $-P = (x, -y - h(x))$. We define point P as satisfying $P = -P$ as a ramification point.

A divisor $D = \sum m_i P_i$, $m_i \in \mathbb{Z}$, is defined as a formal sum of points P_i on C. The set $\mathbb{D}$ of the divisors form an Abelian group. The degree (weight) of a divisor D is defined as $\sum_i m_i$, and we denote it by $w(D)$. The set $\mathbb{D}^0$ of the degree zero divisors is a subgroup of $\mathbb{D}$. The divisor of a rational function on C, called the principal divisor, is a finite formal sum of the zeros and poles. The set $\mathbb{P}$ of principal divisors is a subgroup of $\mathbb{D}^0$. Denote by $J_c(\mathbb{F}_q)$ the Jacobian variety of C defined over $\mathbb{F}_q$. The divisor class group $\mathbb{D}^0/\mathbb{P}$ is isomorphic to the Jacobian variety $J_c(\mathbb{F}_q)$. A semi-reduced divisor of a hyperelliptic curve is represented by $D = \sum_i m_i P_i - (\sum_i m_i) P_\infty$, where $m_i \geq 0$ and $P_i \neq -P_j$ for $i \neq j$. Mumford reported that the semi-reduced divisor can be expressed by two polynomials (u, v) of $\mathbb{F}_q[x]$, which satisfy the following conditions [Mum84]:

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

A semi-reduced divisor with $\deg u \leq g$ is called a reduced divisor. Any divisor class of $J_c(\mathbb{F}_q)$ is uniquely represented by a reduced divisor. Hereafter we denote $D \in J_c(\mathbb{F}_q)$ by a reduced divisor $D = (u, v)$. The unit element of additive group $J_c(\mathbb{F}_q)$ is $(1, 0)$ and the inverse of divisor $D = (u, v)$ is $-D = (u, v + h)$ where the second polynomial is reduced modulo u.

In this paper, we deal with hyperelliptic curves that are suitable for cryptographic purposes, for example, the order of Jacobian $\#J_c(\mathbb{F}_q)$ has only a small cofactor, say c. Based on Hasse-Weil range, the size of $\mathbb{F}_q$ have to satisfy at least $g \log_2 q \approx 2^{160}$. (We call 160-bit HECC.)

2.2 Scalar Multiplication

The basic operation for implementing HECC is the scalar multiplication, which computes $dD = D + \cdots + D$ (d times) for a divisor $D \in J_c(K)$ and integer d. Denote by $(d_{m-1} \cdots d_1 d_0)_2$ the m-bit binary representation of d. The standard method of computing scalar multiplication dD is the following binary method.

Algorithm 1 *Binary Method*

Input: $d = (d_{m-1} \cdots d_1 d_0)_2,\ D \in J_c(K),\ (d_{m-1} = 1)$
Output: dD

1. $D_1 \leftarrow D$
2. *for* i *from* $m-2$ *to* 0 *do*
3. $\quad D_1 \leftarrow HECDBL(D_1)$
4. $\quad$ *If* $d_i = 1$ *then* $D_1 \leftarrow HECADD(D_1, D)$
5. *Return*(D_1)

We denote by HECDBL and HECADD hyperelliptic doubling and addition, namely $HECDBL(D_1) = 2D_1$ and $HECADD(D_1, D) = D_1 + D$, respectively. The binary method requires $(m-1)$ HECDBL and $(m-1)/2$ HECADD on average for randomly chosen d.

2.3 Addition Formulae

In order to implement a group operation in $J_c(\mathbb{F}_q)$, we deploy addition formulae assembled by the operations of polynomial ring $\mathbb{F}_q[x]$. There are two basic algorithms, namely Cantor algorithm [Can87,Kob89] and Harley algorithm [Har00a,Har00b]. Cantor algorithm is a universal addition formula. It is used for all hyperelliptic curves with any genus g, so we are able to compute both HECDBL and HECADD with one formula (however, the computation times of HECDBL and HECADD are not the same). However, the Cantor algorithm is quite slow due to its versatility. The Harley algorithm aims at efficiently implementing the group operations for a small genus. The arithmetic of HECDBL and HECADD is independently optimized based on their explicit operations.

In the following, we describe the Cantor algorithm [Can87,Kob89]. Let $D_i = (u_i(x), v_i(x)) \in J_c(\mathbb{F}_q)$ be the reduced divisors, $i = 1, 2$. The reduced divisor D_3 of addition $D_1 + D_2$ is computed as follows:

Algorithm 2 *Cantor Algorithm*

Input: $D_1 = (u_1, v_1), D_2 = (u_2, v_2)$
Output: $D_3 = (u_3, v_3) = D_1 + D_2$

1. $d = \gcd(u_1, u_2, v_1 + v_2 + h) = s_1 u_1 + s_2 u_2 + s_3(v_1 + v_2 + h)$
2. $u \leftarrow u_1 u_2 / d^2$, $v \leftarrow (s_1 u_1 v_2 + s_2 u_2 v_1 + s_3(v_1 v_2 + f))/d \bmod u)$
3. while $\deg(u) > g$
   $u' \leftarrow (f - hv - v^2)/u$, $v' \leftarrow -h - v \bmod u'$, $u \leftarrow \mathrm{MakeMonic}(u'), v \leftarrow v'$
4. $u_3 \leftarrow u,\ \ v_3 \leftarrow v$
5. return (u_3, v_3)

Step 1 and Step 2 are called the composition part and Step 3 is called the reduction part. The composition part computes the semi-reduced divisor $D = (u, v)$ that is equivalent to D_3. The reduction part finds the reduced divisor $D_3 = (u_3, v_3)$.

We now present the Harley algorithm [Har00a,Har00b]. We denote by *HarleyADD* and *HarleyDBL* the addition $D_3 = D_1 + D_2$ and the doubling $D_3 = 2D_1$ of the Harley algorithm, respectively. We assume that $D_1 = (u_1, v_1), D_2 = (u_2, v_2)$ are the reduced divisors of $J_c(\mathbb{F}_q)$.

These formulas are as follows:

HarleyADD	*HarleyDBL*
Input: $D_1 = (u_1, v_1), D_2 = (u_2, v_2)$, $\deg u_1 = \deg u_2 = 2, \gcd(u_1, u_2) = 1$	Input: $D_1 = (u_1, v_1), \deg u_1 = 2$, $\gcd(u_1, h) = 1$
Output: $D_3 = (u_3, v_3)$	Output: $D_3 = (u_3, v_3)$
1. $U \leftarrow u_1 u_2$	1. $U \leftarrow u_1^2$
2. $S \leftarrow (v_2 - v_1)/u_1 \bmod u_2$	2. $S \leftarrow (2v_1 + h)^{-1}(f + hv_1 - v_1^2)/u_1 \bmod u_1$
3. $V \leftarrow Su_1 + v_1 \bmod U$	3. $V \leftarrow Su_1 + v_1 \bmod U$
4. $U \leftarrow (V^2 + hV - f)/U$	4. $U \leftarrow (V^2 + hV - f)/U$
5. Make U monic	5. Make U monic
6. $V \leftarrow V \bmod U$	6. $V \leftarrow V \bmod U$
7. $u_3 \leftarrow U, v_3 \leftarrow -(V + h) \bmod u_3$	7. $u_3 \leftarrow U, v_3 \leftarrow -(V + h) \bmod u_3$
8. return (u_3, v_3)	8. return (u_3, v_3)

In the HarleyADD, from Step 1 to Step 3 is the composition part and from Step 4 to Step 7 is the reduction part. The composition part computes the semi-reduced divisor $D = (U, V)$ equivalent to D_3. In Step 2 and Step 3, we compute V such that $V^2 + hV - f \equiv 0 \bmod U$, which can be obtained by $V \equiv v_1 \bmod u_1$ and $V \equiv v_2 \bmod u_2$ via the Chinese remainder theorem. The reduction part finds the reduced divisor $D_3 = (u_3, v_3)$ in the same way as the Cantor algorithm. *HarleyDBL* is similar to *HarleyADD*, but the Chinese remainder theorem is replaced by the Newton iteration.

3 Side Channel Attacks

In this section we review the side channel attacks which HECC is vulnerable to.

At Crypto'96 Kocher introduced the timing attack (TA), which considers the secret key in cryptographic devices [Koc96]. TA measures the computation time for various inputs and analyzes the difference between these times. For example, if we compute a scalar multiplication dD using the binary method (Algorithm 1), then TA can reveal the hamming weight of secret key d. The standard way to resist this attack is the following double-and-add-always method:

Algorithm 3 *Double-and-Add-Always Method*

Input: $d = (d_{m-1} \cdots d_1 d_0)_2,\ D \in J_c(K),\quad (d_{m-1} = 1)$
Output: dD

1. $D[0] \leftarrow D$
2. *for* i *from* $m-2$ *to* 0 *do*
3. $D[0] \leftarrow$ *HECDBL(*$D[0]$*)*, $D[1] \leftarrow$ *HECADD(*$D[0]$,D*)*, $D[0] \leftarrow D[d_i]$
4. *Return(*$D[0]$*)*

The double-and-add-always method always computes HECADD whether $d_i = 0$ or 1. Therefore, TA cannot compute the bit hamming weight of d. There is another type of timing attack. In the RSA cryptosystem, the attacker can utilize the existence of the final subtraction in Montgomery multiplication [Sch00,Sch02,SKQ01]. The same argument can be applied to HECC.

At Crypto '99 Kocher et al. introduced simple power analysis (SPA) and differential power analysis (DPA) to reveal the secret key by measuring the power consumption of cryptographic devices [KJJ99]. SPA uses only a single observation of the power to obtain information, and DPA uses many observations together with statistical tools. Algorithm 1 is vulnerable to SPA. The operation HECADD is computed only if the corresponding bit is 1, although HECDBL is always computed. HECDBL and HECADD show different power consumption curves because they are different operations, as is described in Section 2. Thus the SPA can detect the secret bits. In order to defend against SPA, we must eliminate the relation between the addition formulas and the bit information. The double-and-add-always method in the previous section can be used to defend against SPA. Even if a scheme is secure against SPA, it might be insecure against DPA. The standard method to resist DPA is randomizing the parameters of the curve [Cor99,JT01]. However, Goubin proposed an extension of DPA to an elliptic curve cryptosystem [Gou03]. He pointed out that the point $(0, y)$ is not fully randomized by the standard countermeasures against DPA. Recently, Avanzi extended his attack to a hyperelliptic curve cryptosystem [Ava03a]. He noted that the divisors with zero coefficient could be used in a Goubin-type attack, in which one of the coefficients of u or v for divisor (u, v) would be zero.

Izu and Takagi proposed an exceptional procedure attack by using the exceptional procedure in the addition formula of ECC [IT03]. The standard addition formula of ECC bring about an exceptional procedure only if either the input or output is infinity point $\mathcal{O}$. The order of elliptic curve $\#E$ is usually chosen such that $\#E$ is the product of a large prime and a very small integer. When scalar d is smaller than the order of elliptic curve $\#E$, the exceptional procedure occurs only if the order of the processing point is small. Thus, we can detect this attack by checking if the base point does not belong to small group before scalar multiplication. Avanzi also mentioned the possibility of extending this attack to hyperelliptic curve cryptosystems [Ava03a]. However, the details of the extended attack require further discussion.

4 Degenerate Divisors of HECC

We assume here that the genus of hyperelliptic curves is equal to 2 and the characteristic is even for the sake of convenience. However, all discussions are applicapble to higher genus > 2 and an odd prime characteristic.

We deal with the divisor with weight 1, and we define it as the degenerate divisor.

Definition 1. *Let C be a hyperelliptic curve cryptosystem over $\mathbb{F}_{2^n}$, let $J_c(\mathbb{F}_{2^n})$ be the Jacobian of curve C. We call reduced divisor $D = (u, v) \in J_c(\mathbb{F}_{2^n})$ degenerate, if the degree of D is smaller than g, namely* $\deg u < g$.

The degenerate divisor can be easily generated. For example, a random divisor with weight 1 can be generated as follows:

1. Choose a random point (x_0, y_0) on C.
2. Set $u = x + x_0, v = y_0$.
3. Check the order of divisor $D = (u, v)$.

Let $D_1 = (u_1, v_1), D_2 = (u_2, v_2)$ be the reduced divisors of Jacobian $J_c(\mathbb{F}_{2^n})$. Denote by D_3 the addition of $D_1 + D_2$. There is the following possible group operation with degenerate divisors:

$\mathsf{ExHarADD}^{2+2\to 1}$: $w(D_1) = 2$, $w(D_2) = 2$, $w(D_3) = 1$, $D_1 \neq D_2, gcd(u_1, u_2) = 1$,
$\mathsf{ExHarADD}^{1+2\to 2}$: $w(D_1) = 1$, $w(D_2) = 2$, $w(D_3) = 2$, $D_1 \neq D_2, gcd(u_1, u_2) = 1$,
$\mathsf{ExHarDBL}^{1\to 2}$: $w(D_1) = 1$, $w(D_3) = 2$, $D_1 = D_2$, $\gcd(h, u_1) = 1$,
$\mathsf{ExHarDBL}^{2\to 1}$: $w(D_1) = 2$, $w(D_3) = 1$, $D_1 = D_2$, $\gcd(h, u_1) = 1$.

Similarly, there could exist other exceptional procedures using degenerate divisors, for instance, $\mathsf{ExHarADD}^{1+2\to 1}$ and $\mathsf{ExHarADD}^{1+1\to 2}$. However, these cases are not suitable for application to scalar multiplication, because the combination of divisors D_1, D_2, D_3 is not freely chosen. In this paper we do not consider these other exceptional procedures.

The computational cost of these exceptional procedures strongly depends how to implement the addition formulae. In the following we assume $g = 2$. Table 1 shows the cost of the Harley algorithm and its degenerate algorithms improved by Lange [Lan02a] and Sugizaki et al. [SMC+02]. The explicit algorithms for *HarleyDBL* and its degenerate variations are shown in Appendix A. We evaluate the computational cost according to the time of one multiplication M and one inversion I. The exceptional cases are clearly faster than the ordinary cases.

Table 1. Number of multiplication and inversion of Harley Algorithm

Addition Formula	Cost
HarleyADD	$1I + 25M$
HarleyDBL	$1I + 27M$
$\mathsf{ExHarADD}^{2+2\to 1}$	$1I + 14M$
$\mathsf{ExHarADD}^{1+2\to 2}$	$1I + 11M$
$\mathsf{ExHarDBL}^{2\to 1}$	$1I + 17M$
$\mathsf{ExHarDBL}^{1\to 2}$	$1I + 7M$

For $\mathsf{ExHarDBL}^{2\to 1}$ shown in the Algorithm 5, the same algorithm requires less computation time compared to that of *HarleyDBL* because the weight of the output divisor is 1. When the weight of the output divisor is 1, t_1 is always 0; therefore after Step 3, Step 5' is executed. The algorithm in Step 6 and after can be expressed by equations with less computation requirements, as shown in Step 6' and after. For the same reason as $\mathsf{ExHarDBL}^{2\to 1}$, $\mathsf{ExHarADD}^{2+2\to 1}$ requires a less computation time than *HarleyADD*. The computation of $\mathsf{ExHarADD}^{1+2\to 2}$ can be performed using the algorithm of *HarleyADD*. Because the input divisors's weight is 1 for $\mathsf{ExHarADD}^{1+2\to 2}$, however, the degree of the polynomials dealt with in each step is smaller than that for *HarleyADD*, thus saving computational time. The computation time for $\mathsf{ExHarDBL}^{1\to 2}$ is less than that for *HarleyDBL* because the divisor of weight 1, $D = P - P_\infty$, can be reduced to a simple algorithm that gives the tangent to the curve C at point P.

The total cost for computing the scalar multiplication with the double-and-add-always method is $318I + 8268M$ on average for a 160-bit HECC, if there is no exceptional procedure during the computation. For example, it is $10112.4M$ for $1I = 5.8M$.

If we implement a hyperelliptic curve cryptosystem with the Harley algorithm, we have to contend with exceptional cases. For genus 2, there are more than 10 exceptional cases, so that it is a complicated task for implementers. Although the exceptional cases appear with negligible probability for a randomly chosen divisor, the implementer should implement them in order to avoid an error in an exceptional case. Some implementations employ the Cantor algorithm for exceptional cases. If genus is more than 2, there are many exceptional cases, and it is better to implement them using the Cantor algorithm. In the appendix, we describe the timings related to the Cantor algorithm.

4.1 Efficient Scalar Multiplication Using Degenerate Divisors

In this section we present efficient scalar multiplication using degenerate divisors.

The main thrust of our improvement is to choose a degenerate divisors as the base point. As we noted in the previous section, the computational cost of group operation between standard divisors and degenerate divisors are quite different. (See Table 1 and 4.) For example, $\mathsf{ExHarADD}$ is faster than *HarleyADD*. Note that the scalar multiplication is usually computed from the most significant bit because we can utilize efficient mixed coordinates for the affine base point D [CMO98]. Indeed, the binary method (Algorithm 1) in Section 2.2 is a left-to-right method, and thus the base point D is added to D_1 during the scalar multiplication dD. In our case, the base point D is chosen as the degenerate divisor (e.g., $w(D) < g$), so $\mathsf{ExHarADD}$ can be calculated more efficiently than the standard HECADD. Therefore we are able to achieve efficient scalar multiplication using the degenerate base point D.

Here we have a question about the security of choosing the degenerate divisor as the base point. The following theorem can be easily proven thanks to random self reducibility. (See Appendix C for the proof.)

Theorem 2. *Let J be the Jacobian of a hyperelliptic curve of genus g, where $\frac{\#J}{c}$ is prime. We assume that $\bar{D} = (u, v)$ is the degenerate divisor, where $\deg u < g$. Solving the discrete logarithm problem with base point $\bar{D}$ is as intractable as using a random divisor of J.*

From this theorem, the special base point has no influence the security of the underlying discrete logarithm problem.

The presented efficient scalar multiplication with the fixed base point can be applied only to ElGamal-type encryption, the sender of Diffie-Hellman, and DSA. The scalar of these schemes is usually an ephemeral random number, and thus we focus only on SPA, not DPA. A standard countermeasure against SPA is the double-and-add-always method (Algorithm 3) in Section 3. Note that SPA-resistant addition chains (e.g., window-based methods) are not used for the efficiency improvement using the degenerate base point, because it is difficult to generate two degenerate divisors D, aD with previously known integer a.

We assume that the scalar multiplication is computed using the double-and-add-always method (Algorithm 3). We choose the divisor with the weight 1 as the base point D, and then $\mathsf{ExHarADD}^{1+2\to 2}$ is used for HECADD. The power consumption curve of the scalar multiplication shows the fixed pattern, namely the repeats of the power consumption curve of *HarleyDBL* and $\mathsf{ExHarADD}^{1+2\to 2}$. However, HECADD $2D + D$ cannot be computed using $\mathsf{ExHarADD}^{1+2\to 2}$, so we must use different addition formula $\mathsf{ExHarADD}^{1+2\to 2}_{D+2D}$ because $D = P - P_\infty$ and $2D = P + P - 2P_\infty$ have a common factor. This means that only the first pattern of the power consumption is different from the others, but the difference is independent from the bit information of the secret scalar.

We compare the computational cost of the scalar multiplication for a divisor with weight 1, with that for a divisor with weight 2. For the divisor with weight 1, the first HECDBL and HECADD costs $1I + 7M$ ($\mathsf{ExHarDBL}^{1\to 2}$) and $1I + 27M$ ($\mathsf{ExHarADD}^{1+2\to 2}_{D+2D}$) respectively. The other HECDBL and HECADD costs $1I + 27M$ (*HarleyDBL*) and $1I + 11M$ ($\mathsf{ExHarADD}^{1+2\to 2}$) respectively. Therefore the total cost of the scalar multiplication is $(1I + 7M) + (1I + 27M) + ((1I + 27M) + (1I + 11M)) \times 158 = 318I + 6038M$. On the other hand, for a divisor with weight 2, HECDBL and HECADD costs $1I + 27M$ (*HarleyDBL*) and $1I + 25M$ (*HarleyADD*), respectively. The total cost of the scalar multiplication is $((1I + 27M) + (1I + 25M)) \times 159 = 318I + 8268M$. Thus, the proposed scheme can achieve about 22% improvement under $1I = 5.8M$.

Table 2. Improved timing of scalar multiplication

Base Point	Timing
random weight 2	13.36 ms
random weight 1	10.92 ms

In order to demonstrate the improvement of our algorithm, we implemented the proposed scheme (See Table 2). For our experiment we chose the following hyperelliptic curve with genus 2 from [HSS00]: Let $\mathbb{F}_{2^{83}}$ be defined as $\mathbb{F}_2[t]/(t^{83} + t^7 + t^4 + t^2 + 1)$ and $y^2 + h(x)y = f(x)$ over $\mathbb{F}_{2^{83}}$,

$$h(x) = x^2 + \text{2b770d0d26724d479105f}x + \text{540efb4e1010a0fc69f23},$$
$$f(x) = x^5 + \text{2cc2f2131681e8fe80246}x^3 + \text{53b00bad6fbb8f6ea5538}x + \text{54f5f3b4f4fc25898ee4}.$$

The order of the Jacobian is:

$$2 \times 46768052394588893382517909320120991667183740867853.$$

The experiment was implemented on an Intel Xeon Processor 2.80GHz using operation system Linux 2.4 (RedHat). We employed compiler gcc 3.3 and number theoretic library NTL5.3 with GMP4.0 [NTL]. The precise measurement of the timing difference of scalar multiplication on PC is difficult due to many other processes running on the PC, so that we use a CPU clock as the measurement of timing for the test codes. In this computational environment, the timing ratio of the inversion by the multiplication is estimated to be $I/M = 5.78$ from 10 million random samples.

5 Timing Attack on HECC

In this section, we propose a timing attack using degenerate divisors.

5.1 Target of Timing Attack

We explain the target system of the proposed timing attack.

Note that the probability, which a randomly chosen divisor in Jacobian $J_c(\mathbb{F}_{2^n})$ causes an exceptional procedure in the addition formula, is $\mathcal{O}(1/2^n)$ [Nag00]. Therefore, the exceptional procedure appears with negligible probability during the scalar multiplication for a randomly chosen base point. The attacker has to choose appropriate divisors in order to achieve the timing attack.

The proposed attack is categorized as a chosen ciphertext attack on a public-key cryptosystem. We assume that the secret key d is fixed during the attack and the base point P can be freely chosen by the attacker. This scenario has been used for several attacks, namely an exceptional procedure based attack [AT03,Ava03a,Gou03,IT03]. The protocols for which our proposed attack works are HEC ElGamal-type decryption (e.g. HECIES) and single-pass HEC Diffie-Hellman.

The typical target of this attack setting is the secure communication of a client-server model such as SSL. There is a fixed secret key d for the SSL server, and the client requests a ciphertext including base point D from the server. The server usually computes the decryption primitive dD on a hyperelliptic curve and then checks the integrity of padding/hash values. The invalid ciphertexts are rejected at the integrity check. The computation time of the primitive is quite slow compared with that of computing the padding/hash value, so that the timing of computing the primitive part directly reflects that of receiving the rejection. In this scenario, Boneh et al. reported an experimental result of a remote timing attack on RSA-CRT using OpenSSL [BB03].

5.2 Recovering Secret Scalar

We describe here how to recover the secret key d by observing the whole timing of the scalar multiplication dD using the exceptional procedures, where D is a divisor of $J = J_c(\mathbb{F}_{2^n})$. The recovering technique follows the algorithm proposed by Goubin [Gou03] and Izu et al. [IT03]. However, we have to consider where the exceptional procedure occurs and how to compare this timing with that of the ordinary case.

Denote by $(d_{m-1}d_{m-2}\cdots d_1 d_0)_2$ the binary representation of d with $d_{m-1} = 1$. We assume the scalar multiplication is calculated by the double-and-add-always method (Algorithm 3). The attacker tries to bring about an exceptional procedure during the scalar multiplication using the degenerate divisor aD for the base point D and some integer a. We can easily choose the base point, e.g., divisor $D = ((a^{-1}) \bmod \frac{\#J}{c})\bar{D}$ for any degenerate divisor $\bar{D}$, where $\#J$ is the order of $J_c(\mathbb{F}_{2^n})$. We calculate the whole time of the scalar multiplication dD and compare it with that of the scalar multiplication with random base point.

We now describe how to determine the second bit d_{m-2}. First, we determine the second most significant bit d_{m-2}. If $d_{m-2} = 0$, the addition chain generates the following sequence $D, 2D, 3D$(dummy), $4D, 5D$(dummy)$, 8D$ for $d_{m-3} = 0$, and $D, 2D, 3D$(dummy)$, 4D, 5D$ for $d_{m-3} = 1$. If divisor $4D$ is degenerate, the exceptional procedures $\mathsf{ExHarDBL}^{2\to 1}(2D) \to 4D$ and $\mathsf{ExHarADD}^{1+2\to 2}\ (4D) \to 5D$ appear. In this case we have additional exceptional procedure $\mathsf{ExHarDBL}^{1\to 2}(4D) \to 8D$ only if $d_{m-3} = 0$.

$$
\begin{aligned}
d_{m-2} = 0: \quad 2D &\xrightarrow{HarleyADD} 3D \\
&\xrightarrow{\mathsf{ExHarDBL}^{2\to 1}} 4D \xrightarrow{\mathsf{ExHarADD}^{1+2\to 2}} 5D \\
&\qquad\qquad \xrightarrow{\mathsf{ExHarDBL}^{1\to 2}} 8D\ (d_{m-3} = 0)
\end{aligned}
$$

Therefore the timing difference ΔT^0 for $d_{m-2} = 0$ is: follows:

$$\begin{aligned}\Delta T^0 = &(HarleyDBL - \mathsf{ExHarDBL}^{2\to1}) + (HarleyADD - \mathsf{ExHarADD}^{1+2\to2})\\ &+1/2(HarleyDBL - \mathsf{ExHarDBL}^{1\to2}) = 34M.\end{aligned}$$

Similarly, $d_{m-2} = 1$, the addition chain generates the following sequence $D, 2D, 3D, 6D, 7D$ or $D, 2D, 3D, 6D, 7D$(dummy), $12D$. If divisor $6D$ is degenerate, we have the following exceptional procedures:

$$\begin{aligned} d_{m-2} = 1: \quad 2D \xrightarrow{HarleyADD} 3D \xrightarrow{\mathsf{ExHarDBL}^{2\to1}} 6D &\xrightarrow{\mathsf{ExHarADD}^{1+2\to2}} 7D\\ &\xrightarrow{\mathsf{ExHarDBL}^{1\to2}} 12D\ (d_{m-3} = 0)\end{aligned}$$

Therefore the timing difference ΔT^1 for $d_{m-2} = 1$ is as follows:

$$\begin{aligned}\Delta T^1 = &(HarleyDBL - \mathsf{ExHarDBL}^{2\to1}) + (HarleyADD - \mathsf{ExHarADD}^{1+2\to2})\\ &+1/2(HarleyDBL - \mathsf{ExHarDBL}^{1\to2}) = 34M.\end{aligned}$$

The timing differences for $d_{m_{i-2}} = 0, 1$ are exactly same for this attack. For a 160-bit HECC, the timing difference is about 0.34% of the whole scalar multiplication under $1I = 5.8M$. In Appendix B, we show the timing difference for the Cantor algorithm.

In the above situation, the attacker is able to compute the bit by comparing the whole computation time of the scalar multiplication for $((4^{-1}) \bmod \frac{\#J}{c})\bar{D}$ or $((6^{-1}) \bmod \frac{\#J}{c})\bar{D}$, where $\bar{D}$ is any divisor with weight 1.

The lower bits can be recursively recovered using the above method. We explain how to compute d_i after knowing the highest bits $(d_{m-1}d_{m-2}\cdots d_{i+1})$. The attacker chooses $D_0 = ((\sum_{j=i}^{m-1} d_j 2^{j-i})^{-1} \bmod \frac{\#J}{c})\bar{D}$ or $D_1 = ((\sum_{j=i+1}^{m-1} d_j 2^{j-i})^{-1} \bmod \frac{\#J}{c})\bar{D}$ as the base point, where $\bar{D}$ is any divisor with weight 1. The base point D_0, D_1 brings about the exceptional procedure if $d_i = 0, 1$, respectively.

We cannot apply this to the least bits, because the exceptional procedures that appear before the termination of the scalar multiplication are different. However, the last few bits can be easily determined.

Remark 3. There are other exceptional procedures, e.g. the gcd of two input divisors is not one, the constant term of the divisor is zero, and some divisors include a ramification point. (See, for example, [Lan02a-c].) They can be used for proposed timing attack. For example, we can use the base point D that satisfies $(3^{-1} \bmod \frac{\#J}{c})D_0$ with a weight-1 divisor D_0. This divisor D causes the exception of HECADD from $2D$ to $3D$ when the second most significant bit is 1, namely $\mathsf{ExHarADD}^{2+2\to1}$, is used.

We have measured the average timings of the scalar multiplication for the Harley algorithm (*Harley*), the Harley algorithm with one exceptional procedure (*Harley* + ExHarley), and the Harley algorithm with one exceptional procedure of the Cantor algorithm (*Harley* + ExCantor). Table 3 shows the results with 50000 random samples.

The arithmetic of HECC was programmed only using the operations of finite field $\mathbb{F}_{2^{83}}$. The common commands of NTL library were used for both the exceptional procedure and

Table 3. Timings of scalar multiplication

Addition Formula	Timing
Harley	13.36 ms
Harley + ExHarley	13.34 ms
Harley + ExCantor	13.59 ms

the ordinary procedure. The timing of the branch condition, which switches the ordinary case to the exceptional cases, is negligible compared to that of the operations of $\mathbb{F}_{2^{83}}$.

The timing difference using one exceptional procedure ExHarley or ExCantor is 0.15% or 1.72%, respectively. These timings are comparable to the results in Section 4 and Appendix B. The exceptional procedure of the Cantor algorithm causes a larger difference than that of the Harley algorithm.

Although there is a timing difference of exceptional cases from the ordinary case, the difference is quite small. In the next section we explain how to improve the success probability of determining the secret bit.

5.3 Outline of Experiment

We report an experiment of the timing attack based on the exceptional procedures in the previous section. We explain our experimental technique, which distinguishes the timing difference of the exceptional procedures from the ordinary procedure. The test codes calculate scalar multiplication dD for the base point D and n-bit secret scalar d. We assume that the scalar multiplication is computed by the double-and-add-always method (Algorithm 3).

This technique of measurement is similar to that used in [DKL+98]. Let T be the average time of N scalar multiplications with different divisors. Note that there are enough such divisors for the timing attack. The attacker aims at determining d_i, that is the i-th bit of the secret scalar d. The total bits of the secret key can be recovered by recursively applying this attack from the most significant bits. Section 5.2 shows how to generate a divisor that bring about the exceptional procedure with $d_i = 0$ or $d_i = 1$. We denote by D^{ex0} or D^{ex1} these divisors, respectively. We have the following three different timings:

T^{rand}: the average time with a randomly chosen divisor
T^{ex0}: the average time with divisor D^{ex0}
T^{ex1}: the average time with divisor D^{ex1}

The sample number of different divisors for obtaining T^{ex0} or T^{ex1} is N for each bit d_i. Timing T^{rand} is measured with N different divisors during the whole attack. The minimum number N for succeeding in the attack depends on the computational environment and distribution of divisors, and we show the minimum N for our setting in the next section.

Then we compute the differences from the random instance, more precisely $\Delta T^0 = |T^{rand} - T^{ex0}|$ and $\Delta T^1 = |T^{rand} - T^{ex1}|$. If $d_i = b$ holds for $b = 0, 1$, then $\Delta T^{\bar{b}}$ is nearly zero due to random distribution $T^{ex\bar{b}}$, that is $T^{ex\bar{b}} \approx T^{rand}$, where $\bar{b} = 1 - b$. Recall that the scalar multiplication bring about an exceptional procedure with negligible probability for a randomly chosen base point. Therefore, we can suppose $d_i = b$ if $\Delta T^b > \Delta T^{\bar{b}}$ holds for $b = 0, 1$. We summarize this as follows:

Algorithm 4 *Experiment for Determining d_i*

1. *Calculate $T^{rand}, T^{ex0}, T^{ex1}$*
2. *Calculate $\Delta T^0 = |T^{rand} - T^{ex0}|$ and $\Delta T^1 = |T^{rand} - T^{ex1}|$*
3. *Return $d_i = b$ if $\Delta T^b > \Delta T^{\bar{b}}$ for $b = 0, 1$*

5.4 Analysis of Timing Attack

We analyze the distribution of timings ΔT^b for $b = 0, 1$, and we present the experimental result of the timing attack.

The distribution of timings ΔT^b depends on the distribution of divisors D appearing in the scalar multiplication dD and the noise arising from the measurement of dD. We can average the deviation by increasing the number of base points $D_1, ..., D_N$ used for the experiment. Therefore, we can define the following distribution, which comprises k iterations of experiments for T^{rand} and T^{exi}.

Definition 4. *Let $\mathcal{T}^{rand}(N)$ and $\mathcal{T}^{exb}(N)$ be the average timing of the scalar multiplication dD_j for $j = 1, 2, ..., N$ with random base point D_j and base point D_j that brings about the exceptional procedure at target bit d_b, respectively. The (N, k)-distribution $\mathcal{T}^b_{N,k}$ is the distribution of timings $\Delta\mathcal{T}^b = \mathcal{T}^{rand}(N) - \mathcal{T}^{exb}(N)$ for k iterations of the experiment for obtaining $\mathcal{T}^{rand}(N)$ and $\mathcal{T}^{exb}(N)$, where $b = 0, 1$.*

The mean value of (N, k)-distribution $\Delta\mathcal{T}^b$ can be used for determining the secret bit b. We can determine it by comparing the mean value of $\Delta\mathcal{T}^b$ with that of $\Delta\mathcal{T}^{\bar{b}}$ for $b = 0, 1$ as we showed in Algorithm 4.

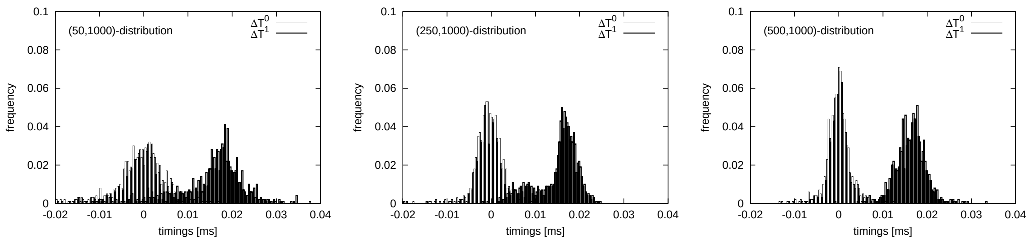

Fig. 1. (50,1000)-distribution, (250,1000)-distribution, and (500,1000)-distribution

Figure 1 shows histograms of (N, k)-distribution $\mathcal{T}^b_{N,k}$ for different $N = 50$, $N = 250$, $N = 500$ with fixed $k = 1000$. The horizontal axis and vertical axis are timing (ms) and frequency of timing, respectively. To compare the different types of (N, k)-distributions, we averaged the timings over N divisors and normalized the frequency by k. When N increases from 50 to 500, the overlapping of the two histograms between $\mathcal{T}^0_{N,k}$ and $\mathcal{T}^1_{N,k}$ becomes smaller. The proportion of the overlapping shows the probability of determining the secret bit. The success rate of our experiment is defined as the ratio of correct determines to the total number of experiments (i.e. k). Figure 2 shows the relation between the success rate for the increasing number N of random divisors with fixed $k = 1000$. If we choose $N > 500$, we achieve almost a 100% success rate.

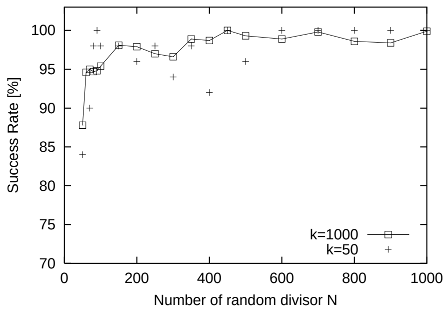

Fig. 2. Success rate of determining one bit

If k is small, the effect of the noise on the experiment becomes large and the success rate becomes smaller. For example, we show the distribution of $k = 50$ in Fig. 2, which is irregular even for increasing N. However, the choice of $k = 1000$ is large enough for eliminating the influence of the noise on our experiment, and N is considered to be the number of measurements required for the timing attack. Consequently, we conclude one bit of the secret scalar can be recovered if the attacker can measure more than 500 samples ($N > 500$) with high probability.

In order to reveal all 160 bits of the secret scalar, we recursively performed the proposed attack from the 2nd most significant bit to the 2nd least significant bit. The least significant bit could be easily revealed. In this case, we required $500 \times 158 = 79,000$ samples. Our experiment did not provide error correction similar to that used by [DKL$^+$98]. With error-correction implemented, the time required for recovery would decrease dramatically because fewer samples (for example, $N = 50$) would be needed for successful recovery.

6 Summary

We investigated the use of degenerate divisors of hyperelliptic curves in cryptography. The timing of computing addition formulas with degenerate divisors (the exceptional procedure) is in general different from that of the standard procedure. We considered the precise timing of the exceptional procedure required using the Harley algorithm and Cantor algorithm.

We presented two different applications of the exceptional procedures, — which can be, however, a two-edged sword. For a positive application we presented an efficient scalar multiplication using degenerate divisors as the base point. The discrete logarithm problem of the degenerate divisors is as hard as that of the random divisors due to the random self-reducibility. Our experiment shows that we can achieve about 20% improvement in speed. For a negative application, we mounted the degenerate divisors to the timing attack on the secret scalar. The attack tries to distinguish the timing of the exceptional procedure from that of the ordinary procedure. About 500 samples of the scalar multiplication enable us to break one bit of the secret key.

References

[AT03] T. Akishita and T. Takagi, "Zero-Value Point Attacks on Elliptic Curve Cryptosystem," ISC 2002, LNCS 2851, Springer-Verlag, pp.218-233, 2003.

[Ava03a] R. Avanzi, "Countermeasures against Differential Power Analysis for Hyperelliptic Curve Cryptosystems," CHES 2003, LNCS 2779, Springer-Verlag, pp.366-381, 2003.

[Ava03b] R. Avanzi, "Aspects of Hyperelliptic Curves over Large Prime Fields in Software Implementations", Cryptology ePrint Archive, 2003/253, IACR, 2003.

[BB03] D. Boneh and D. Brumley, "Remote Timing Attacks are Practical," 12th Usenix Security Symposium, USENIX, pp.1-14, 2003.

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

[CMO98] H. Cohen, A. Miyaji, and T. Ono, "Efficient Elliptic Curve Exponentiation Using Mixed Coordinates," *ASIACRYPT '98*, LNCS1514, pp.51-65, 1998.

[Cor99] J.-S. Coron, "Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems," CHES '99, LNCS 1717, Springer-Verlag, pp.292-302, 1999.

[DKL+98] J.F. Dhem, F. Koeune, P.A. Leroux, P. Mestré, J.J Quisquater and J.L. Willems, "A Practical Implementation of the Timing Attack," UCL Crypto Group Technical Report CG–1998/1, 1998.

[GMP] GMP, GNU MP Library GMP. http://www.swox.com/gmp

[Gou03] L. Goubin, "A Refined Power-Analysis Attack on Elliptic Curve Cryptosystem," PKC2003, LNCS 2567, Springer-Verlag, pp.199-211, 2003.

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

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

[HSS00] F. Hess, G. Seroussi and N. Smart, "Two Topics in Hyperelliptic Cryptography," Technical Report CSTR-00-008, Department of Computer Science, University of Bristol, 2000.

[IT03] T. Izu and T. Takagi, "Exceptional Procedure Attack on Elliptic Curve Cryptosystems," PKC2003, LNCS 2567, Springer-Verlag, pp.224-239, 2003.

[JT01] M. Joye and C. Tymen, "Protection against Differential Analysis for Elliptic Curve Cryptography," CHES 2001, LNCS 2162, Springer-Verlag, pp.377-390, 2001.

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

[Koc96] C. Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems," CRYPTO '96, LNCS 1109, pp.104-113, 1996.

[KJJ99] C. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," CRYPTO '99, LNCS 1666, pp.388-397, 1999.

[KGM+02] J. Kuroki, M. Gonda, K. Matsuo, J. Chao and S. Tsujii, "Fast Genus Three Hyperelliptic Curve Cryptosystems," Proc. of SCIS2002, 2002.

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

[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 Hyperellipitc Curve," Cryptology ePrint Archive, 2002/153, IACR, 2002.

[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.

[Nag00] N. Nagao, "Improving Group Law Algorithms for Jacobians of Hyperelliptic Curves," ANTS-IV, LNCS 1838, Springer-Verlag, pp.439-448, 2000.

[NTL] NTL: A Library for Doing Number Theory. http://www.shoup.net/ntl

[Pel02] J. Pelzl, "Hyperelliptic Cryptosystems on Embedded Microprocessors," Diploma Thesis, Rühr-Universität Bochum, 2002.

[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, Springer-Verlag, pp.351-365, 2003.

[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.

[Sch00] W. Schindler, "A Timing Attack against RSA with the Chinese Remainder Theorem," CHES 2000, LNCS 1965, pp.109-124, 2000.

[Sch02] W. Schindler, "A Combined Timing and Power Attack," PKC 2002, LNCS 2274, pp.263-279, 2002.

[SKQ01] W. Schindler, F. Koeune, J.-J. Quisquater, "Improving Divide and Conquer Attacks against Cryptosystems by Better Error Detection/Correction Strategies," Cryptography and Coding, 8th IMA Int. Conf., LNCS 2260, pp.245-267, 2001.

A Explicit Formulas of Harley Algorithm

We show the explicit description of Harley algorithm and its degenerate variations, namely *HarleyDBL*, $\mathsf{ExHarDBL}^{2\to 1}$, and $\mathsf{ExHarDBL}^{1\to 2}$.

Algorithm 5 *HarleyDBL,* $\mathsf{ExHarDBL}^{2\to 1}$

Input: $D_1 = (u_1, v_1), \deg u_1 = 2$

Output: $D_3 = (u_3, v_3) = 2D_1$

1	Compute $r = \mathrm{res}(u_1, h)$:	
	$w_1 \leftarrow h_1 + u_{11}, w_0 \leftarrow h_0 + u_{10} + u_{11}w_1, r \leftarrow u_{10}(u_{10} + h_0 + h_1w_1) + h_0w_0;$	$4M$
2	Compute $I = i_1x + i_0 \equiv rh^{-1} \bmod u_1$	
	$(i_1 \leftarrow w_1, i_0 \leftarrow w_0;)$	
3	Compute $T = t_1x + t_0 \equiv I(f + hv_1 + v_1^2)/u_1 \bmod u_1$:	
	$w_2 \leftarrow f_3 + v_{11} + u_{11}^2, w_3 \leftarrow v_{10} + v_{11}(v_{11} + h_1),$	
	$t_1 \leftarrow w_0w_2 + w_1w_3, t_0 \leftarrow (u_{11}w_0 + u_{10}w_1)w_2 + w_0w_3;$	$8M$
4	If $t_1 = 0$ then goto *5'*.	
5	Compute $S = s_1x + s_0$:	
	$w_0 \leftarrow (rt_1)^{-1}, w_2 \leftarrow w_0r, w_3 \leftarrow w_0t_1, w_4 \leftarrow w_2r, s_1 \leftarrow w_3t_1, s_0 \leftarrow w_3t_0;$	$1I$+$6M$
6	Compute $u_3 = x^2 + u_{31}x + u_{30} = s_1^{-2}(f + h(Su_1 + v_1) + (Su_1 + v_1)^2)/u_1^2$:	
	$u_{31} \leftarrow w_4(1 + w_4), u_{30} \leftarrow w_4(w_4(s_0(1 + s_0)) + w_1);$	$4M$
7	Compute $v_3 = v_{31}x + v_{30} \equiv Su_1 + v_1 + h \bmod u_3$:	
	$w_1 \leftarrow u_{11} + u_{31}, w_0 \leftarrow u_{10} + u_{30}, w_2 \leftarrow s_1w_1, w_3 \leftarrow s_0w_0,$	
	$w_4 \leftarrow (s_1 + s_0)(w_1 + w_0) + w_2 + w_3, w_2 \leftarrow w_2 + 1, w_1 \leftarrow w_4 + w_2u_{31},$	
	$w_0 \leftarrow w_3 + w_2u_{30}, v_{31} \leftarrow w_1 + v_{11} + h_1, v_{30} \leftarrow w_0 + v_{10} + h_0;$	$5M$
total	*HarleyDBL*	$1I$+$27M$
5'	Compute $S = s_0$:	
	$s_0 \leftarrow t_0/r;$	$1I$+$1M$
6'	Compute $u_3 = x + u_{30} = (f + h(Su_1 + v_1) + (Su_1 + v_1)^2)/u_1^2$:	
	$u_{30} \leftarrow f_4 + s_0 + s_0^2;$	$1M$
7'	Compute $v_3 = v_{30} \equiv Su_1 + v_1 + h \bmod u_3$:	
	$v_{30} \leftarrow u_{30}((s_0u_{11} + v_{11} + h_1) + (s_0 + 1)u_{30}) + (s_0u_{10} + v_{10} + h_0));$	$4M$
total	$\mathsf{ExHarDBL}^{2\to 1}$	$1I$+$17M$

Algorithm 6 $\mathsf{ExHarDBL}^{1\to 2}$

Input: $D_1 = (u_1, v_1), \deg u_1 = 1$		
Output: $D_3 = (u_3, v_3) = 2D_1$		
1	Compute $u_1^2 = x^2 + u_{10}^2$:	
	$u_{30} \leftarrow u_{10}^2,\ u_{31} \leftarrow 0$	$1M$
2	Compute $v_{31} = (f'(u_{10}) + h'(u_{10})v_{10})/h(u_{10})$:	
	$w_0 \leftarrow u_{30}^2,\ w_1 \leftarrow f_3 u_{30},\ w_2 \leftarrow h_1 v_{10},\ w_0 \leftarrow w_0 + w_1 + f_1 + w_2,$	
	$w_3 \leftarrow h_1 u_{10},\ w_1 \leftarrow u_{30} + w_3 + h_0,\ v_{31} \leftarrow w_0/w_1;$	$1I{+}5M$
3	Compute $v_{30} = u_{10}v_{31} + v_{10}$:	
	$w_0 \leftarrow u_{10}v_{31},\ v_{30} \leftarrow w_0 + v_{10};$	$1M$
total	$\mathsf{ExHarDBL}^{1\to 2}$	$1I{+}7M$

B Exceptional Procedures of the Cantor Algorithm

We discuss the implementation and timing related to the Cantor algorithm.

We investigated the similar analysis of the timing for the original Cantor algorithm with characteristic 2 [Kob89]. *CantorDBL* and *CantorADD* denote the doubling and addition of Cantor algorithm. We write with bold face the exceptional cases corresponding to the Harley algorithm in Section 4, e.g. $\mathsf{ExCanADD}^{2+2\to 1}$. The number of M, I for the Cantor algorithm is not obvios because of the gcd operation. We present the maximum value in terms of an experiment with randomly chosen curves. The faster variant of Cantor algorithm [Nag00] is not optimized for the degenerate case, so that we evaluated the cost of the original algorithm [Kob89]. The estimated timings are shown in Table 4.

Table 4. Number of multiplication and inversion of Cantor Algorithm

Addition Formula	Cost
CantorADD	$4I + 72M$
CantorDBL	$4I + 68M$
$\mathsf{ExCanADD}^{2+2\to 1}$	$3I + 62M$
$\mathsf{ExCanADD}^{1+2\to 2}$	$2I + 41M$
$\mathsf{ExCanDBL}^{2\to 1}$	$3I + 60M$
$\mathsf{ExCanDBL}^{1\to 2}$	$2I + 28M$

In the four exceptional cases, the computation amount of the reduction part, Step 4 in Algorithm 2 are smaller than that in the ordinary procedure. Among them, $\mathsf{ExCanDBL}^{1\to 2}$ has the smallest computation amount due to the absence of the computation of Step 4. Because the weight of one of the input divisors for $\mathsf{ExCanADD}^{1+2\to 2}$ is 1, the degree of the polynomials computed in Algorithm 2 is smaller; therefore, $\mathsf{ExCanADD}^{1+2\to 2}$ has less computation amount than $\mathsf{ExCanADD}^{2+2\to 1}$ or $\mathsf{ExCanDBL}^{2\to 1}$.

We estimate the timing differences discussed in Section 5.2 for the exceptional procedure that uses the Cantor algorithm. Note that the exceptional case $\mathsf{ExCanDBL}^{2\to 1}$ switches from the Harley algorithm to the Cantor algorithm only after starting to compute the first several steps of the Harley algorithm — the overhead is $12M$. Therefore, we obtain the following timing differences that were defined in Sec. 5.2:

$$\Delta T^b = |(HarleyDBL - \mathsf{ExCanDBL}^{2\to 1}) + (HarleyADD - \mathsf{ExCanADD}^{1+2\to 2})$$
$$+1/2(HarleyDBL - \mathsf{ExCanDBL}^{1\to 2})| + 12M = 3.5I + 61.5M,$$

where $b = 0, 1$. For a 160-bit HECC, the timing difference is about 0.81% of the whole scalar multiplication under $1I = 5.8M$. The timing difference of the Cantor algorithm is much larger than that of Harley algorithm. The timing attack becomes easier. Note that even if we implement the addition formula only using the Cantor algorithm, the timing attack is feasible.

Figure 3 shows a comparison of the success rate of the Harley algorithm with its degenerate variations in Section 4 (Type 1) and the Harley algorithm with degenerate the Cantor algorithm in Section B (Type 2). the Cantor algorithm reaches 100% success even for small N.

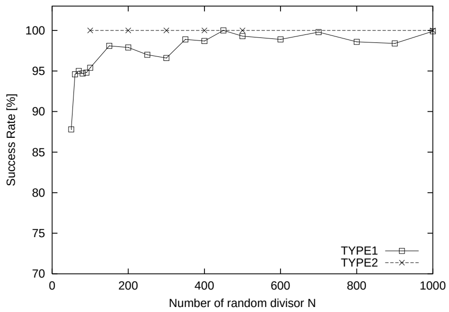

Fig. 3. Success rate of determining one bit: Type 1 vs Type 2

C Proof

Theorem. 2 *Let J be the Jacobian of a hyperelliptic curve of genus g, where $\frac{\#J}{c}$ is prime. We assume that $\bar{D} = (u, v)$ is the degenerate divisor, where $\deg u < g$. Solving the discrete logarithm problem with the base point $\bar{D}$ is as intractable as using a random divisor of J.*

Proof. ($\Leftarrow$) Let $\log_{\bar{D}} Q_0$ be the discrete logarithm problem for the base point $\bar{D}$ and a divisor Q_0. We can randomize these divisors by multiplying random scalar $r, s \in [1, \frac{\#J}{c}]$, namely let $D = r\bar{D}$, $Q = sQ_0$ be randomized divisors. Based on assumption, we can solve a discrete logarithm problem $\log_D Q$, and thus $\log_{\bar{D}} Q_0 = (\log_D Q)r/s \bmod \frac{\#J}{c}$.
($\Rightarrow$) Let A_0 be an oracle which solves the discrete logarithm problem for the base point $\bar{D}$, namely A_0 answers $\log_{\bar{D}} Q_0$ for a random divisor Q_0. We try to construct algorithm A that solves the discrete logarithm problem with a random base point. Algorithm A is going to compute $\log_D Q$ for random inputs D, Q. Algorithm A can obtain discrete logarithm $\log_{\bar{D}} D$ by asking D to oracle A_0. Similarly, algorithm A obtains $\log_{\bar{D}} Q$. Then algorithm A returns the discrete logarithm $\log_D Q = (\log_{\bar{D}} Q)/(\log_{\bar{D}} D) \bmod \frac{\#J}{c}$. □