A Timing Attack on Hyperelliptic Curve Cryptosystems

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. Recent works show that the performance of hyperelliptic curve cryptosystem (HECC) is competitive to that of elliptic curve cryptosystem (ECC). However, secure implementation of HECC has been little discussed as compared with ECC. In this paper, we report an experimental result of a timing attack on HECC in software. The time difference for computing the degenerated divisors with smaller degree is mounted to the timing attack. The proposed attack assumes that the secret key is fixed and the base point can be freely chosen by the attacker. Therefore, it is applicable to ElGamal-type decryption and single-pass Diffie-Hellman — SSL using a hyperelliptic curve could be vulnerable to the proposed attack. From our experimental results on a Xeon processor, one bit of the secret key for a 160-bit HECC can be recovered by calling the decryption oracle 1,000 times.

Keywords. hyperelliptic curve cryptosystems, scalar multiplication, timing attacks, weight of divisor, SSL

1 Introduction

In 1989, Koblitz proposed hyperelliptic curve cryptosystem (HECC) [Kob89]. HECC has the advantage of shorter operand length than elliptic curve cryptosystem (ECC), and it has been expected that HECC attains a faster encryption comparing with ECC. Recently some efficient addition formulas of HECC have been proposed (See, for example, [Lan02a-c]), and an implementation result in hardware shows the performance of HECC is competitive to that of ECC [PWG+03].

When we implement a cryptosystem in a real security system, we have to care side channel attacks such as timing attack [Koc96] and power analysis [KJJ99]. A simple timing attack on HECC can detect a hamming weight of secret scalar by measuring the whole computation time of the scalar multiplication. The timing attack using the final subtraction of Montgomery multiplication [Sch00,Sch02,SKQ01] can be also applicable to HECC. Other possible attacks are to use so-called exceptional procedures of the addition formula [AT03,Ava03,Gou03,IT03]. The addition formula of HECC contains a lot

of exceptional procedures, so that we have many possibilities to mount them to the 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 encryption and single-pass HEC Diffie-Hellman but not HEC DSA.

In this paper, we propose the first experimental result of a timing attack on HECC using some exceptional procedures. The algorithm of recovering the secret key follows the previous works [AT03,Ava03,Gou03,IT03]. We mainly discuss the attack on hyperelliptic curve with genus 2, especially Harley algorithm [Har00a,Har00b] and Cantor algorithm [Can87,Kob89]. Several explicit exceptional procedures suitable for the timing attack in the setting are investigated. We examine the exceptional procedures arisen from divisors with weight 1, whose computational time is faster than the ordinary one. Then we show how to apply them 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 the ordinary operation. The proposed timing attack analyses the timing difference with many sampling number. Our experiment on a Xeon processor using a 160-bit HECC shows that the timing difference is about 0.04 ms for the exceptional case of Harley algorithm. The difference is quite small, so that we apply the sampling technique proposed by Dhem et al. [DKL$^+$98]. We gather three different types of timing: the timing with a randomly chosen divisor, the timing with divisor D_b that cause the exceptional procedure if the guessed bit is $b = 0, 1$. If the guess b is correct, the timing difference from the random one becomes larger than the other case. Then our experiment successfully recovered one bit of secret key with 1,000 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. We propose the new timing attack of HECC in section 4 and show the experimental results of genus 2 HECC in section 5. Finally we conclude in section 6.

2 Hyperelliptic Curve Cryptosystems

In this section we review hyperelliptic curve cryptosystems related to our attack.

2.1 Hyperelliptic Curve

Let g be a positive integer and let $K = \mathbb{F}_{2^n}$ be a finite field with 2^n elements, where n is a positive integer. A hyperelliptic curve C of genus g over $\mathbb{F}_{2^n}$ is defined by equation $y^2 + h(x)y = f(x)$, where $f(x)$ is a monic polynomial over $\mathbb{F}_{2^n}$ with degree at most $2g + 1$ in x and $h(x) \in \mathbb{F}_{2^n}[x]$ with $\deg h \leq g$. 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 a point P satisfies $P = -P$ as a ramification point.

Denote by $J_c(\mathbb{F}_{2^n})$ the Jacobian variety of C defined over $\mathbb{F}_{2^n}$. It is known that $J_c(\mathbb{F}_{2^n})$ is isomorphic to the ideal class group. 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 $i \neq j$. The weight of divisor D is defined by $\sum_i m_i$, and we denote it by $w(D)$. Mumford reported that the semi-reduced divisor can be expressed by two polynomials (a, b) of $\mathbb{F}_{2^n}[x]$, which satisfy the following conditions [Mum84]:

1. $a(x) = \prod_i (x + x_i)^{m_i}$,
2. $b(x_i) = y_i$,
3. $\deg b < \deg a$,
4. $f + hb + b^2 \equiv 0 \bmod a$.

A semi-reduced divisor with $\deg a \leq g$ is called a reduced divisor. Any divisor class of $J_c(\mathbb{F}_{2^n})$ is uniquely represented by a reduced divisor. Hereafter we denote $D \in J_c(\mathbb{F}_{2^n})$ by a reduced divisor $D = (a, b)$. The unit element of additive group $J_c(\mathbb{F}_{2^n})$ is $(1, 0)$ and the inverse of an divisor $D = (a, b)$ is $-D = (a, a + b + h)$.

In this paper, we deal with hyperelliptic curves that are suitable for cryptographic purposes, for example, the order of Jacobian has only a small cofactor, say 2.

2.2 Scalar Multiplication

The inevitable 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 an integer d. Denote by $(d_{n-1} \cdots d_1 d_0)_2$ the n-bit binary representation of d. The standard method of computing scalar multiplication dD is the binary method described as follows:

Algorithm 1 *Binary Method*

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

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

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

2.3 Addition Formula

In order to implement a group operation in $J_c(\mathbb{F}_{2^n})$, we deploy addition formulas assembled by the operations of polynomial ring $\mathbb{F}_{2^n}[x]$. 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, and we are able to compute both HECDBL and HECADD with one formula (the computation times of HECDBL and HECADD are not same). However, Cantor algorithm is quite slow due to its versatility. Harley algorithm aims at efficiently implementing the group operations for small genus. The arithmetic of HECDBL and HECADD is independently optimized based on their explicit operations.

In the following, we describe the Cantor algorithm for even characteristic [Can87,Kob89]. Let $D_i = (a_i(x), b_i(x)) \in J_c(\mathbb{F}_{2^n})$ be the reduced divisors, where c is the curve defined by $c : y^2 + h(x)y = f(x)$ and $i = 1, 2$. The reduced divisor D_3 of addition $D_1 + D_2$ is computed as follows:

Algorithm 2 *Cantor Algorithm*

Input: $D_1 = (a_1, b_1), D_2 = (a_2, b_2)$
Output: $D_3 = (a_3, b_3) = D_1 + D_2$

1. $d = \gcd(a_1, a_2, b_1 + b_2 + h) = s_1a_1 + s_2a_2 + s_3(b_1 + b_2 + h)$
2. $a \leftarrow a_1a_2/d^2$
3. $b \leftarrow (s_1a_1b_2 + s_2a_2b_1 + s_3(b_1b_2 + f))/d \bmod a)$
4. while $\deg(a) > g$
   $a' \leftarrow (f + hb + b^2)/a$, $b' \leftarrow (h + b) \bmod a'$ $a \leftarrow \mathrm{MakeMonic}(a'), b \leftarrow b'$
5. $a_3 \leftarrow a$, $b_3 \leftarrow b$
6. return (a_3, b_3)

From Step 1 to Step 3 is called a composition part and Step 4 is called a reduction part. The composition part computes the semi-reduced divisor $D = (a, b)$ that is equivalent to $-D_3$. The reduction part finds the reduced divisor $D_3 = (a_3, b_3)$ equivalent to D.

Next, we describe the outline of 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 Harley algorithm, respectively. We assume that $D_1 = (u_1, v_1), D_2 = (u_2, v_2)$ are the reduced divisors of curve defined by equation $y^2 + h(x)y = f(x)$. 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. $U \leftarrow u_1u_2$	1. $U \leftarrow u_1^2$
2. $S \leftarrow (v_2 + v_1)/u_1 \bmod u_2$	2. $S \leftarrow h^{-1}(f + hv_1 + v_1^2)/u_1 \bmod u_1$
3. $V \leftarrow Su_1 + v_1 \bmod U$	3. $V \leftarrow Su_1 + v_1 \bmod U$
4. $U \leftarrow (f + hV + V^2)/U$	4. $U \leftarrow (f + hV + V^2)/U$
5. Make U monic	5. Make U monic
6. $V \leftarrow V \bmod U$	6. $V \leftarrow V \bmod U$
7. $u_3 \leftarrow U, v_3 \leftarrow U + V + h$	7. $u_3 \leftarrow U, v_3 \leftarrow U + V + h$
8. return (u_3, v_3)	8. return (u_3, v_3)

From Step 1 to Step 3 is a composition part and from Step 4 to Step 7 is a 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 $f+hV+V^2 \equiv 0 \bmod U$, 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 $D_3' = (u_3', v_3')$ equivalent to D. We transform $u_3' = (f + hV + V^2)/u_1u_2$ to the monic polynomial, and then we compute $v_3' \equiv V \bmod u_3'$. Finally, we output a divisor $D_3 = -D_3'$ as: $D_3 = (u_3, v_3) = (u_3', u_3' + v_3' + h)$. HarleyDBL is similar to HarleyADD, but the Chinese Remainder Theorem is replaced by the Newton iteration. In these algorithms, the Karatsuba algorithm is used to reduce the number of multiplications.

3 Side Channel Attack

In this section we review side channel attacks related to our attack on HECC.

3.1 Timing Attack

At Crypto'96 Kocher introduced timing attack (TA), which tries to expose the secret key in cryptographic devices [Koc96]. TA measures the computation times for many inputs and analyzes the difference of these times. For example, if we compute a scalar multiplication dD with the binary method (Algorithm 1), then TA can detect 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: $d = (d_{n-1} \cdots d_1 d_0)_2,\ D \in J_c(K),\quad (d_{n-1} = 1)$
Output: dD

1. $D_1[0] \leftarrow D$
2. *for* i *from* $n-2$ *to* 0 *do*
3. $\quad D_1[0] \leftarrow$ *HECDBL*$(D_1[0])$
4. $\quad D_1[1] \leftarrow$ *HECADD*$(D_1[0], D)$
5. $\quad D_1[0] \leftarrow D_1[d_i]$
6. *Return*$(D_1[0])$

The double-and-add-always method always compute HECADD whether $d_i = 0$ or 1. Therefore, the TA cannot guess the bit hamming weight of d.

There is another type of timing attack. For 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.

3.2 Power Analysis

At Crypto '99 Kocher et al. introduced the simple power analysis (SPA) and differential power analysis (DPA) to expose the secret key by observing 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 features of power consumption curve because they are different operations as described in section 2. Thus the SPA can detect the secret bits. In order to resist SPA, we must eliminate the relations between addition formulae and the bit information. The double-and-add-always method in the previous section can be used for resisting 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 elliptic curve cryptosystem [Gou03]. He pointed out the point $(0, y)$ is not randomized by the standard countermeasures against DPA. Very recently, Avanzi extended his attack to hyperelliptic curve cryptosystems [Ava03]. He noted that the divisors with zero coefficient could be used for Goubin-type attack, namely one of the coefficient of a or b for divisor (a, b) is zero.

3.3 Exceptional Procedure Attack

Izu and Takagi proposed the exceptional procedure attack by using the exceptional procedure in the addition formula of ECC [IT03].

The standard addition formula of ECC causes the exceptional procedure only if either 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 the base point does not belong to small group before scalar multiplication.

Avanzi also mentioned a possibility of extending this attack to hyperelliptic curve cryptosystems [Ava03]. However, the details of the extended attack require further discussions.

4 Timing Attack on HECC

In this section, we propose a new timing attack using exceptional procedures of addition formulas. Roughly speaking, the timing difference of computing the exceptional procedure from the ordinary one is mounted to the proposed timing attack. We assume that the genus of hyperelliptic curve is equal to 2 for the sake of convenient. However, all discussions can be adapted to hyperelliptic curves with higher genus.

4.1 Target of Timing Attack

We explain the target system of the proposed timing attack.

The proposed attack is categorized to 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 exceptional procedure based attack [AT03,Ava03,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 asks a ciphertext including base point D to 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 comparing with that of computing the padding/hash value, so that the timing of the primitive part directly reflects that of receiving the rejection. In this scenario, Boneh et al. showed an experimental result of the remote timing attack on RSA-CRT using OpenSSL [BB03].

Remark 1. As a related research, Manger proposed a reject timing attack on RSA-OAEP [Man01], and Kim et al. showed a memory dump attack on several provably secure cryptosystems [KCJ$^+$01]. The cryptosystem, which checks the integrity before the primitive operation (e.g., Cramer-Shoup cryptosystem [CS98]), is not vulnerable to these attacks. In the same reason, our attack is also infeasible for Cramer-Shoup cryptosystem.

4.2 Basic Observation

We explain which exceptional procedure is used for the proposed timing attack.

There are several different exceptional procedures in the explicit addition formulas (See, for example, [Lan02a-c]). These exceptional cases occurs, if the divisor for inputting or outputting to the addition formula is not ordinary, e.g. the weight of the divisor is 1, the gcd of two input divisors is not one, the constant term of the divisor is zero, divisors including a ramification point, etc.

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 carefully choose appropriate divisors in order to achieve the timing attack.

In this paper we deal with the divisor with weight 1, and we define it as the degenerated divisor.

Definition 2. *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 a reduced divisor $D = (a, b) \in J_c(\mathbb{F}_{2^n})$ is degenerated, if the degree of D is smaller than g, namely* $\deg a < g$.

The degenerated divisor can easily be generated using a point P over C because any divisor with weight 1 is represented as $D = P - P_\infty$. We only choose a random point P_r and check the order of $D_r = P_r - P_\infty$.

Let $D_1 = (a_1, b_1), D_2 = (a_2, b_2)$ 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 degenerated divisors:

ExHarADD$^{2+2\to1}$: $w(D_1) = 2$, $w(D_2) = 2$, $w(D_3) = 1$, $D_1 \neq D_2, gcd(a_1, a_2) = 1$,
ExHarADD$^{1+2\to2}$: $w(D_1) = 1$, $w(D_2) = 2$, $w(D_3) = 2$, $D_1 \neq D_2, gcd(a_1, a_2) = 1$,
ExHarDBL$^{1\to2}$: $w(D_1) = 1$, $w(D_3) = 2$, $D_1 = D_2$, $\gcd(h, D_1) = 1$,
ExHarDBL$^{2\to1}$: $w(D_1) = 2$, $w(D_3) = 1$, $D_1 = D_2$, $\gcd(h, D_1) = 1$.

Similarly, there could exist other exceptional procedures using degenerated divisors, for instances, **ExHarADD**$^{1+2\to1}$ and **ExHarADD**$^{1+1\to2}$. However, these cases are not suitable for the proposed attack, because the combination of divisors D_1, D_2, D_3 are not freely chosen. In this paper we exclude them from our attack.

The computational cost of these exceptional procedures strongly depend how to explicitly implement the addition formula. Table 1 shows the cost of Haley algorithm and its degenerated algorithms improved by Sugizaki et al. [SMC$^+$02]. The explicit algorithms for HarleyDBL and its degenerated variation are shown in the appendix. 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. The difference of their computational costs is a crucial point of the proposed attack.

Table 1. Number of multiplication and inversion of Harley Algorithm

Addition Formula	Cost
HarleyADD	$1I + 25M$
HarleyDBL	$1I + 27M$
ExHarADD$^{2+2\to1}$	$1I + 14M$
ExHarADD$^{1+2\to2}$	$1I + 11M$
ExHarDBL$^{2\to1}$	$1I + 17M$
ExHarDBL$^{1\to2}$	$1I + 7M$

For **ExHarDBL**$^{2\to1}$ shown in the Alg. 5, the same algorithm requires a less computation amount compared to 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 amounts as shown in Step 6' and after. For the same reason as **ExHarDBL**$^{2\to1}$, **ExHarADD**$^{2+2\to1}$ requires a less computation amount than HarleyADD. The computation of **ExHarADD**$^{1+2\to2}$ can be performed using the algorithm of HarleyADD. Because the weight of one of the input divisors is 1 for **ExHarADD**$^{1+2\to2}$, however, the degrees of the polynomials dealt with in each step are smaller than those for HarleyADD, thus saving the computational cost. The computation amount of **ExHarDBL**$^{1\to2}$ is

smaller than that of HarleyDBL because the divisor of weight 1, $D = P - P_\infty$, can reduce to a simple algorithm that gives the tangent to the curve C at the 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 $9540M$ for $1I = 4M$.

4.3 Recovering Secret Scalar

We describe 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 them with ordinary case.

Denote by $(d_{n-1}d_{n-2}\cdots d_1 d_0)_2$ the binary representation of d with $d_{n-1} = 1$. We assume the scalar multiplication is calculated by the double-and-add-always method (Algorithm 3). The attacker tries to cause the exceptional procedure during the scalar multiplication using the degenerated divisor aD for the base point D and some integer a. We can easily choose the base point, e.g., divisor $D = ((a^{-1}) \bmod \frac{\#J}{2})\bar{D}$ for any degenerated 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 exactly describe how to guess the second bit d_{n-2}. First, we guess the second most significant bit d_{n-2}. If $d_{n-2} = 0$, addition chain generates the following sequence $D, 2D, 3D(\text{dummy}),\ 4D, 5D(\text{dummy}), 8D$ for $d_{n-3} = 0$, and $D, 2D, 3D(\text{dummy}), 4D, 5D$ for $d_{n-3} = 1$. If divisor $4D$ is degenerated, the exceptional procedures $\mathbf{ExHarDBL}^{2\to1}(2D) \to 4D$ and $\mathbf{ExHarADD}^{1+2\to2}(4D) \to 5D$ appears. In this case we have additional exceptional procedure $\mathbf{ExHarDBL}^{1\to2}(4D) \to 8D$ only for if $d_{n-3} = 0$.

$$d_{n-2} = 0: \quad 2D \xrightarrow{HarleyDBL} 3D \begin{array}{l} \xrightarrow{\mathbf{ExHarDBL}^{2\to1}} 4D \xrightarrow{\mathbf{ExHarADD}^{1+2\to2}} 5D \\ \xrightarrow{\mathbf{ExHarDBL}^{1\to2}} 8D\ (d_{n-3}=0) \end{array}$$

Therefore the timing difference ΔT_0 for $d_{n-2} = 0$ is as follows:

$$\Delta T_0 = (\text{HarleyDBL} - \mathbf{ExHarDBL}^{2\to1}) + (\text{HarleyADD} - \mathbf{ExHarADD}^{1+2\to2}) + 1/2(\text{HarleyDBL} - \mathbf{ExHarDBL}^{1\to2}) = 34M.$$

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

$$d_{n-2} = 1: \quad 2D \xrightarrow{HarleyADD} 3D \xrightarrow{\mathbf{ExHarDBL}^{2\to1}} 6D \begin{array}{l} \xrightarrow{\mathbf{ExHarADD}^{1+2\to2}} 7D \\ \xrightarrow{\mathbf{ExHarDBL}^{1\to2}} 12D\ (d_{n-3}=0) \end{array}$$

Therefore the timing difference ΔT_1 for $d_{n-2} = 1$ is as follows:

$$\Delta T_1 = (\text{HarleyDBL} - \mathbf{ExHarDBL}^{2\to 1}) + (\text{HarleyADD} - \mathbf{ExHarADD}^{1+2\to 2}) \\ +1/2(\text{HarleyDBL} - \mathbf{ExHarDBL}^{1\to 2}) = 34M.$$

The timing differences for $d_{n_{i-2}} = 0, 1$ are exactly same for this attack. For a 160-bit HECC, the timing difference is about 0.36% of the whole scalar multiplication under $1I = 4M$.

From the observation above, the attacker is able to guess the bit by comparing the whole computation time of the scalar multiplication for $((4^{-1}) \bmod \frac{\#J}{2})\bar{D}$ or $((6^{-1}) \bmod \frac{\#J}{2})\bar{D}$, where $\bar{D}$ is any divisor with weight 1.

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

We cannot apply this observation 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 guessed by the exhaustive search.

Remark 3. Other exceptional procedures can be used for the timing attack. For example, we can use the base point D that satisfies $(3^{-1} \bmod \frac{\#J}{2})D_0$ 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 $\mathbf{ExHarADD}^{2+2\to 1}$ is used.

4.4 Implementing with Cantor Algorithm

We discuss the implementation with Cantor algorithm.

If we implement a hyperelliptic curve cryptosystem with Harley algorithm, we have to treat the 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 the error at the exceptional case. Some implementations employ Cantor algorithm for the exceptional cases. If genus is more than 2, there are plenty of exceptional procedure cases, and it is better to implement them using Cantor algorithm.

A draw back of using Cantor algorithm is its efficiency. This overhead of computation can be also mounted to the timing attack. We have investigated the similar analysis of the timing for original Cantor algorithm with characteristic 2 [Kob89]. Denote by *CantorDBL* and *CantorADD* the doubling and addition of Cantor algorithm. We write with bold face the exceptional cases corresponding to Harley algorithm in Section 4.2, e.g. $\mathbf{ExCanADD}^{2+2\to 1}$. It is not obvious to count the number of M, I for Cantor algorithm because of the gcd operation.

We present the maximum value in term of an experiment with randomly chosen curves. The faster variant of Cantor algorithm [Nag00] is not optimized for the degenerated case, so that we evaluated the cost of the original one [Kob89]. The estimated timings are shown in Table 2.

Table 2. Number of multiplication and inversion of Cantor Algorithm

Addition Formula	Cost
CantorADD	$4I + 72M$
CantorDBL	$4I + 68M$
$\mathbf{ExCanADD}^{2+2\to1}$	$3I + 62M$
$\mathbf{ExCanADD}^{1+2\to2}$	$2I + 41M$
$\mathbf{ExCanDBL}^{2\to1}$	$3I + 60M$
$\mathbf{ExCanDBL}^{1\to2}$	$2I + 28M$

In the four exceptional cases, the computation amounts of the reduction part, Step 4, shown in Algorithm 2 are smaller than that in the ordinary case. Among them, $\mathbf{ExCanDBL}^{1\to2}$ 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 $\mathbf{ExCanADD}^{1+2\to2}$ is 1, the degrees of the polynomials computed in Algorithm 2 are smaller; therefore, $\mathbf{ExCanADD}^{1+2\to2}$ has less computation amount than $\mathbf{ExCanADD}^{2+2\to1}$ or $\mathbf{ExCanDBL}^{2\to1}$.

We estimate the timing differences for the case that uses Cantor algorithm for the exceptional cases. Note that the exceptional case $\mathbf{ExCanDBL}^{2\to1}$ switches from Harley algorithm to Cantor algorithm only after starting to compute the first several steps of Harley algorithm — the overhead is $12M$. Therefore, we obtain the following timing differences that were defined in the previous section:

$$\begin{aligned}\Delta T_b = |&(\text{HarleyDBL} - \mathbf{ExCanDBL}^{2\to1}) + (\text{HarleyADD} - \mathbf{ExCanADD}^{1+2\to2}) \\ &+1/2(\text{HarleyDBL} - \mathbf{ExCanDBL}^{1\to2})| + 12M = 3.5I + 61.5M,\end{aligned}$$

where $b = 0, 1$. For a 160-bit HECC, the timing difference is about 0.79% of the whole scalar multiplication under $1I = 4M$. The timing difference of Cantor algorithm is much larger than that of Harley algorithm. The timing attack becomes easier.

Remark 4. Note that even if we implement the addition formula only using Cantor algorithm, the timing attack is feasible.

5 Experiments

We show an experiment of the timing attack based on the exceptional procedure in the previous section. In the experiment, we successfully recovered the secret

scalar d by attacking the test code of scalar multiplication dD implemented on a PC. This experimental result shows that the Harley algorithm is vulnerable to the proposed timing attack.

5.1 Target Curve and Experiment Environment

For our experiment we chose the following hyperelliptic curve with genus 2 from [HSS00]:

$$y^2 + h(x)y = f(x) \quad \text{over} \quad \mathbb{F}_{2^{83}},$$

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

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]. Precise measurement of the timing difference of scalar multiplication on PC is relatively difficult due to many other processes running on 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 an inversion by a multiplication is estimated as $I/M = 4.10$ from 10 million random samples. We have measured the average timings of the scalar multiplication for ordinarily Harley algorithm (*Harley*), ordinarily Harley algorithm with one exceptional procedure (*Harley* + **ExHarley**), and ordinarily Harley algorithm with one exceptional procedure of Cantor algorithm (*Harley* + **ExCantor**). Table 3 shows the results with 1000 random samples.

Table 3. Timings of scalar multiplication

Addition Formula	Timing
Harley	15.12 ms
Harley + **ExHarley**	15.08 ms
Harley + **ExCantor**	15.29 ms

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

The timing difference using one exceptional case **ExHarley** or **ExCantor** is 0.26% or 1.12%, respectively. These timings are comparable to the results in Section 4.2 and Section 4.4. The exceptional procedure of Cantor algorithm causes a larger difference than that of Harley algorithm.

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

5.2 The Detail of Experiment

We explain our experimental technique of distinguishing the timing difference of the exceptional procedures from the ordinary process. The test codes calculate scalar multiplication dD for the base point D and n-bit secret scalar d.

This technique of measurement is similar to that used in [DKL$^+$98]. Let T be the average time of N scalar multiplications. The attacker aims at guessing d_i that is i-th bit of the secret scalar d. From Section 4.3 we are able to generate a divisor that causes the exception procedure with $d_i = 0$ or $d_i = 1$, we denote by D_i^{ex0} or D_i^{ex1}these divisors, respectively. He/she gathers the following three different timings:

T^{rand}: the average time with a randomly chosen divisor,
T_i^{ex0}: the average time with divisor D_i^{ex0},
T_i^{ex1}: the average time with divisor D_i^{ex1}.

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

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

We summarize the experiment as follows:

Algorithm 4 *Experiment for Guessing* d_i

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

The whole bits of the secret key can be recovered by recursively applying this attack from the most significant bits (See Section 4.3).

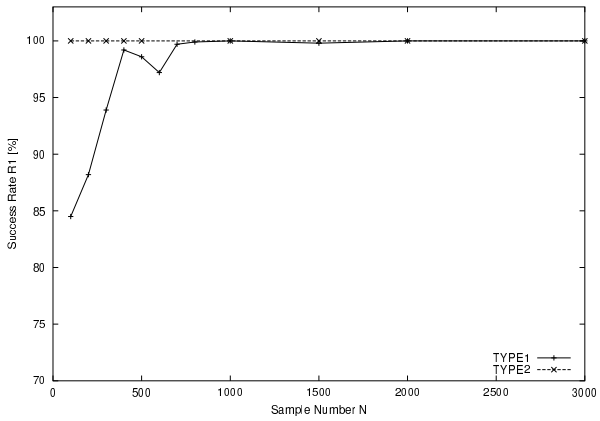

Fig. 1. Success rate of recovering 2nd most significant bit

5.3 Experimental Results

We assume that the scalar multiplication is computed by the double-and-add-always method (Algorithm 3). Harley algorithm with its degenerated variations in Section 4.2 (Type 1) and Harley algorithm with degenerated Cantor algorithm in Section 4.4 (Type 2) are examined.

We have performed our experiment with the sample number $N < 3000$ in the computational environment described in Section 5.1. We define the success rate as the number of correct guesses divided by the total number of experiments. Fig. 1 shows the relation between the success rate for recovering the 2nd most significant bit and the sample number N. If we have more than 1000 samples ($N > 1000$), we achieve 100% success rate for Type 1 and Type 2, namely no error in our attack. It means that we are able to break the bit by calling the decryption oracle 1000 times. The sample number N required to achieve high success rates is small for Type 2 compared to Type 1 because ΔT_i^b of Type 2 is larger than that of Type1.

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 by the exhaustive search. In this case, we required $1000 \times 158 = 158000$ samples.

This experiment did not perform an error correction similar to the one adopted in [DKL$^+$98]. With an error correction implemented, the time required for recovery would decrease dramatically because fewer samples (for example, $N = 100$) are needed for successful recovery.

6 Concluding Remarks

We demonstrated the first experimental result of the timing attack on hyperelliptic curve cryptosystems. The attack focuses on the exceptional procedures arisen from the divisor with weight 1, and tries to distinguish the exceptional procedure from the ordinary one. About 1,000 samples of the scalar multiplication enable us to break one bit of the secret key.

We have not examined all possible exceptional procedures. It is a further work to investigate the timing attack using other exceptional procedures.

Possible countermeasures against the proposed attack are Coron's 1st or 2nd countermeasure [Cor99]. However, it requires additional treatments, e.g. the randomization of the secret key, the security management of ephemeral random divisors.

References

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

[Ava03] R. Avanzi, "Countermeasures against Differential Power Analysis for Hyperelliptic Curve Cryptosystems," CHES 2003, LNCS 2779, Springer-Verlag, pp.366-381, 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,

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

[CS98] R. Cramer and V. Shoup, "A Practical Public-Key Cryptosystem Provably Secure against Adaptive Chosen Ciphertext Attacks," CRYPTO'98, LNCS 1462, pp.13-25, 1998.

[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.

[KCJ+01] S. Kim, J. Cheon, M. Joye, S. Lim, M. Mambo, D. Won, and Y. Zheng, "Strong Adaptive Chosen-Ciphertext Attacks with Memory Dump (or: The Importance of the Order of Decryption and Validation)," Cryptography and Coding, 8th IMA Int. Conf., LNCS 2260, pp.114-127, 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. Masuo, 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.

[Man01] J. Manger, "A Chosen Ciphertext Attack on RSA Optimal Asymmetric Encryption Padding (OAEP) as Standardized in PKCS #1 v2.0," CRYPTO 2001, LNCS 2139, Springer-Verlag, pp.230-238, 2001.

[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.

Appendix

In this appendix we show the explicit description of HarleyDBL and its degenerated variations, namely *HarleyDBL*, **ExHarDBL**$^{2\to1}$, and **ExHarDBL**$^{1\to2}$.

Algorithm 5 *HarleyDBL,* **ExHarDBL**$^{2\to1}$

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 $T = t_1x + t_0 \equiv I(f + hv_1 + v_1^2)/u_1 \bmod u_1$:	
	$w_2 \leftarrow f_3 + v_{11} + v_{11}^2, w_3 \leftarrow v_{10} + v_{11}(v_{11} + h_1)$,	
	$t_1 \leftarrow w_0w_2 + w_1w_3, t_0 \leftarrow (u_{11}w_0 + u_{10}w_1)w_2 + w_0w_3$;	$8M$
4	If $t_1 = 0$ then goto 5'.	
5	Compute $S = s_1x + s_0$:	
	$w_0 \leftarrow (rt_1)^{-1}, w_2 \leftarrow w_0r, w_3 \leftarrow w_0t_1, w_4 \leftarrow w_2r, s_1 \leftarrow w_3t_1, s_0 \leftarrow w_3t_0$;	
6	Compute $u_3 = x^2 + u_{31}x + u_{30} = s_1^{-2}(f + h(Su_1 + v_1) + (Su_1 + v_1)^2)/u_1^2$:	$1I$+$6M$
	$u_{31} \leftarrow w_4(1 + w_4), u_{30} \leftarrow w_4(w_4(s_0(1 + s_0)) + w_1)$;	
7	Compute $v_3 = v_{31}x + v_{30} \equiv Su_1 + v_1 + h \bmod u_3$:	$4M$
	$w_1 \leftarrow u_{11} + u_{13}, w_0 \leftarrow u_{10} + u_{30}, w_2 \leftarrow s_1w_1, w_3 \leftarrow s_0w_0$,	
	$w_4 \leftarrow (s_1 + s_0)(w_1 + w_0) + w_2 + w_3, w_2 \leftarrow w_2 + 1, w_1 \leftarrow w_4 + w_2u_{31}$,	
	$w_0 \leftarrow w_3 + w_2u_{30}, v_{31} \leftarrow w_1 + v_{11} + h_1, v_{30} \leftarrow w_0 + v_{10} + h_0$;	$5M$
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	**ExHarDBL**$^{2\to1}$	$1I$+$17M$

Algorithm 6 **ExHarDBL**$^{1\to2}$

Input: $D_1 = (u_1, v_1), \deg u_1 = 1$
Output: $D_3 = (u_3, v_3) = 2D_1$

1	Compute $u_1^2 = x^2 + u_{10}^2$:	
	$w_0 \leftarrow u_{10}^2$	$1M$
2	Compute $v_{31} = (f'(u_{10}) + h'(u_{10})v_{10})/h(u_{10})$:	
	$w_1 \leftarrow w_0^2$, $w_2 \leftarrow f_3w_0$, $w_3 \leftarrow h_1v_{10}$, $w_1 \leftarrow w_1 + w_2 + f_1 + w_3$,	
	$w_4 \leftarrow h_1u_{10}$, $w_2 \leftarrow w_0 + w_4 + h_0$, $v_{31} \leftarrow w_1/w_2$;	$1I$+$5M$
3	Compute $v_{30} = U_{10}v_{31} + v_{10}$:	
	$w_0 \leftarrow u_{10}v_{21}$, $v_{30} \leftarrow w_0 + v_{10}$;	$1M$
total	**ExHarDBL**$^{1\to2}$	$1I$+$7M$