SPA-resistant Scalar Multiplication on Hyperellipitc Curve Cryptosystems Combining Divisor Decomposition Technique and Joint Regular Form

Toru Akishita[1], Masanobu Katagi[1], and Izuru Kitamura[1]

Information Technologies Laboratories, Sony Corporation,
6-7-35 Kitashinagawa, Shinagawa-ku, Tokyo, 141-0001 Japan
akishita@pal.arch.sony.co.jp,
{Masanobu.Katagi,Izuru.Kitamura}@jp.sony.com

Abstract. Hyperelliptic Curve Cryptosystems (HECC) are competitive to elliptic curve cryptosystems in performance and security. Recently efficient scalar multiplication techniques using a theta divisor have been proposed. Their application, however, is limited to the case when a theta divisor is used for the base point. In this paper we propose efficient and secure scalar multiplication of a general divisor for genus 2 HECC over $\mathbb{F}_{2^m}$. The proposed method is based on two novel techniques. One is divisor decomposition technique in which a general divisor is decomposed into two theta divisors. The other is joint regular form for a pair of integers that enables efficient and secure simultaneous scalar multiplication of two theta divisors. The marriage of the above two techniques achieves both about 19% improvement of efficiency compared to the standard method and resistance against simple power analysis without any dummy operation.

Keywords: hyperelliptic curve cryptosystems, scalar multiplication, theta divisor, signed binary representation, simple power analysis

1 Introduction

Elliptic Curve Cryptosystems (ECC) have increased their importance in public key cryptosystems because of their higher efficiency than RSA cryptosystems. Hyperelliptic Curve Cryptosystems (HECC) are generalization of ECC: ECC just correspond to HECC of genus 1. The security of HECC whose genus is smaller than 4 is thought to match that of ECC of the same group size. On the other side, the performance of HECC was believed to be slower than that of ECC due to their complex group operations. However, since Harley proposed an efficient group addition and doubling algorithm, so-called Harley algorithm, for genus 2 curves of odd characteristics in 2000 [7], optimizations and generalizations of Harley algorithm have been carried out [2], and at present HECC are competitive to ECC also in performance.

In recent years, a new class of attacks has been proposed to extract some secret information from a cryptographic device using its power consumption: so-called power analysis. This paper deals with only Simple Power Analysis (SPA), which utilizes a power consumption trace during a single execution. Differential Power Analysis (DPA) is the more sophisticated attack that requires many power consumption traces with statistical tools.

In regard to HECC, the countermeasure against SPA must be considered when an ephemeral and secret scalar is used for scalar multiplication. The standard countermeasure is the double-and-add-always method that *always* repeats a divisor class doubling and a divisor class addition per bit of the scalar [3]. Recently the useful countermeasure for ECC, Montgomery ladder, was applied to HECC [4], but underlying curves of the HECC version are limited.

On the contrary, efficient and SPA-resistant scalar multiplication techniques peculiar to HECC have been recently proposed, which use a *theta* divisor [9, 8]. A theta divisor has weight smaller than the genus of the underlying curve. For a genus 2 hyperelliptic curve over $\mathbb{F}_q$, a theta divisor is represented as $D = (x + u_0, v_0)$, whereas a *general* divisor is represented as $D = (x^2 + u_1 x + u_0, v_1 x + v_0)$, where $u_i, v_i \in \mathbb{F}_q$. The cost of an addition of a theta divisor is smaller than that of a general divisor due to its simple representation, so that scalar multiplication of a theta divisor is faster than that of a general divisor. This efficiency, however, can be utilized in the limited case, when the base point is a theta divisor and scalar multiplication of the base point is carried out, for example, in HEC Diffie-Hellman phase 1 or HEC-DSA signature generation.

In this paper we enhance the efficient use of a theta divisor to scalar multiplication of a general divisor for a genus 2 curve over $\mathbb{F}_{2^m}$. The enhancement is based on the following two novel techniques. The first one is *Divisor Decomposition Technique* (DDT). A general divisor $D = (U(x), V(x))$ can be decomposed into two theta divisors D_1 and D_2 if $U(x)$ is reducible over $\mathbb{F}_{2^m}$. The second one is *Joint Regular Form* (JRF), which is a new signed binary representation of a pair of integers such that one is even and the other is odd. Any signed bits at the same position of JRF satisfy that one is 0 and the other is ± 1.

In order to utilize both DDT and JRF, we compute $dD_1 + (d+1)D_2$ and then subtract D_2 as compensation instead of the scalar multiplication dD. The simultaneous scalar multiplication $dD_1 + (d+1)D_2$ with JRF of $(d, d+1)$ repeats a divisor class doubling and an addition of a theta divisor $\pm D_1$ or $\pm D_2$ per bit of d. Its cost is almost equal to the cost of the double-and-add-always method of a theta divisor, and is smaller than that of a general divisor. Moreover, SPA-resistance is guaranteed because of regularity without any dummy operation, which causes the possibility of fault-based attacks. Even if D is unable to be decomposed into D_1 and D_2, we update D by repeating a divisor class doubling of D until D can be decomposed into D_1 and D_2. Then, after computing $(dD_1 + (d+1)D_2) - D_2 = dD$, we repeat a divisor class halving [10] of dD the corresponding times. The proposed method is 18.7% faster than the double-and-add-always method of a general divisor.

The rest of paper is organized as follows. In next two sections, we briefly introduce HECC mainly focused to theta divisors and scalar multiplication. In Section 4 and 5, we present two novel techniques: Divisor Decomposition Technique (DDT) and Joint Regular Form (JRF). In Section 6, we show the efficient and secure scalar multiplication of a general divisor by combining DDT and JRF. Section 7 analyzes the computational efficiency of the proposed method. Finally, we draw our conclusion and discuss further work in Section 8.

2 Hyperelliptic Curve Cryptosystems

We give only a brief introduction of Hyperelliptic Curve Cryptosystems (HECC) because of space limitation. More details can be found, for example, in [1, 2, 11].

In this paper, we discuss genus 2 HECC over $\mathbb{F}_{2^m}$. A hyperelliptic curve over $\mathbb{F}_{2^m}$ is defined by $C: y^2+h(x)y = f(x)$, where $h(x) = x^2+h_1x+h_0 \in \mathbb{F}_{2^m}[x]$ and $f(x) = x^5+\sum_{i=0}^{3} f_i x^i \in \mathbb{F}_{2^m}[x]$. In contrast to ECC, points P on a hyperelliptic curve C do not form a group. The group law is defined over Jacobian variety $\mathcal{J}_C$. $\mathcal{J}_C$ is isomorphic to a divisor class group which forms an additive group, and each divisor class is uniquely represented as a reduced divisor. A reduced divisor $D = \sum m_i P_i - (\sum m_i)P_\infty$, where $P_i = (x_i, y_i)$ and P_∞ is a point at infinity, can be represented by two polynomials $(U(x), V(x))$ [16],

$$U(x) = \prod_i (x + x_i)^{m_i}, \quad V(x_i) = y_i,$$
$$\deg V < \deg U \leq 2, \quad V^2 + hV + f \equiv 0 \bmod U.$$

The $\deg U$ of a reduced divisor is called weight. We denote the weight of a reduced divisor D by $w(D)$.

2.1 Theta Divisor and General Divisor

In the case of genus 2 hyperelliptic curves, a reduced divisor has weight smaller than or equal to 2. A divisor is called a *theta* divisor[1] if its weight is smaller than 2 [12]. On the other hand, we call a divisor of weight 2 a *general* divisor. A theta divisor is represented as $D = (x + u_0, v_0)$, whereas a general divisor is represented as $D = (x^2 + u_1x + u_0, v_1x + v_0)$.

Let $D_1 = (U_1, V_1)$, $D_2 = (U_2, V_2) \in \mathcal{J}_C(\mathbb{F}_{2^m})$ be reduced divisors. The computational cost of a divisor class doubling $D_3 = (U_3, V_3) = 2D_1$ and a divisor class addition $D_3 = D_1 + D_2$ depends on the conditions that D_1, D_2 and D_3 satisfy. We list some conditions as follows:

DBL $w(D_1) = 2,\ w(D_3) = 2,\ \gcd(h, U_1) = 1,$
ADD $w(D_1) = 2,\ w(D_2) = 2,\ w(D_3) = 2,\ \gcd(U_1, U_2) = 1,$
TDBL $w(D_1) = 1,\ w(D_3) = 2,\ \gcd(h, U_1) = 1,$
TADD $w(D_1) = 2,\ w(D_2) = 1,\ w(D_3) = 2,\ \gcd(U_1, U_2) = 1.$

[1] A theta divisor is called a degenerate divisor in [9]

DBL and ADD correspond to so-called *most frequent cases* of a divisor class doubling and an addition, respectively. TDBL denotes the doubling of a theta divisor. TADD denotes the addition of a general divisor and a theta divisor.

The computational cost of TDBL and TADD is smaller than that of DBL and ADD, respectively, because of simple representation of a theta divisor. We summarize their cost in Table 1. M, S, and I denote the required time of multiplication, squaring, and inversion, respectively.

Table 1. Cost of group operations (genus 2, $C/\mathbb{F}_{2^m}$)

Group operations	Cost
DBL [13]	$1I + 22M + 5S$
ADD [13]	$1I + 22M + 3S$
TDBL [8]	$1I + 5M + 2S$
TADD [13]	$1I + 10M + 1S$

3 Scalar Multiplication on HECC

3.1 Double-and-Add-Always Method

In order to construct HECC, it is necessary to compute scalar multiplication dD, where d is a non-negative integer and D is a reduced divisor. Let $d = (d_{n-1} \cdots d_0)_2$ be the binary representation of d, where $d_{n-1} = 1$. The most standard SPA-resistant method to compute dD is called the Double-and-Add-Always method (DAA), shown in Algorithm 1.

Algorithm 1 *Double-and-add-always Method* (DAA)

Input: a non-negative integer d, a reduced divisor $D \in \mathcal{J}_C(\mathbb{F}_{2^m})$
Output: dD

1. $Q[0] \leftarrow D$
2. For $i = n - 2$ downto 0 do:
   2.1 $Q[0] \leftarrow 2Q[0]$
   2.2 $Q[1] \leftarrow Q[0] + D$
   2.3 $Q[0] \leftarrow Q[d_i]$
3. return($Q[0]$)

In step 2.1 and step 2.2, a divisor class doubling and a divisor class addition are computed, respectively. If the input divisor D is a general divisor, a doubling in step 2.1 corresponds to DBL and an addition in step 2.2 corresponds to ADD with very high probability. On the other hand, if D is a theta divisor, ADD is replaced by TADD. From Table 1, we estimate the computational cost of scalar multiplication dD of both a general divisor and a theta divisor. DAA of a General Divisor, which we call DAA_GD, takes $(1I + 22M + 3S) + (1I + 22M + 5S) =$

$2I + 44M + 8S$ per bit of the scalar d. On the other hand, DAA of a Theta Divisor, which we call DAA_TD, takes only $(1I+10M+1S)+(1I+22M+5S) = 2I+32M+6S$ per bit of d [8]. Therefore the idea choosing a theta divisor as the input divisor can achieve about 20% improvement of efficiency under $I = 8M$ and $S = 0.1M$. Its cryptographic application, however, is limited; even if we choose a theta divisor as the base point of HECC, we must often compute scalar multiplication of a general divisor.

DAA repeats the identical sequence of a doubling and an addition by inserting dummy additions for $d_i = 0$ in step 2.2. Thus, an SPA attacker cannot guess any bit information of d. The insertion of dummy operations, however, causes the possibility of fault-based attacks.

3.2 Simultaneous Scalar Multiplication

Let us consider the sum of scalar multiplication kD_1 and lD_2, where k, l are non-negative integers and D_1, D_2 are reduced divisors. It is necessary to compute the sum $kD_1 + lD_2$ in HEC-DSA signature verification. The efficient method to compute $kD_1 + lD_2$ is known as Shamir's method. This method computes $kD_1 + lD_2$ simultaneously instead of computing kD_1 and lD_2 independently, so that it is called *simultaneous scalar multiplication*.

Let $k = (k_{n-1} \cdots k_0)_2$ and $l = (l_{n-1} \cdots l_0)_2$ be the binary representations of k and l, respectively, where $k_{n-1} = 1$ or $l_{n-1} = 1$. The simultaneous scalar multiplication $kD_1 + lD_2$ is shown in Algorithm 2.

Algorithm 2 *Simultaneous Scalar Multiplication*

Input: non-negative integers k, l and reduced divisors D_1, D_2
Output: $kD_1 + lD_2$

1. (pre-computation) compute $D_1 + D_2$
2. $Q \leftarrow k_{n-1}D_1 + l_{n-1}D_2$
3. For $i = n-2$ downto 0 do:
   3.1. $Q \leftarrow 2Q$
   3.2. if $(k_i, l_i) \neq (0, 0)$ then
        $T \leftarrow Q + (k_iD_1 + l_iD_2)$
4. return Q

Algorithm 2 can reduce the number of doublings to half compared to computing kD_1 and lD_2 separately. Furthermore we define the following terms in order to evaluate the number of additions.

Definition 1. *Let $\langle k_{n-1} \cdots k_0 \rangle$ and $\langle l_{n-1} \cdots l_0 \rangle$ be signed binary representations of non-negative integers k and l, respectively. The number of i ($0 \le i \le n-1$) satisfying $(k_i, l_i) \neq (0, 0)$ is called* Joint Hamming Weight *of (k, l).*

Definition 2. *Let $\langle k_{n-1} \cdots k_0 \rangle$ and $\langle l_{n-1} \cdots l_0 \rangle$ be signed binary representations of non-negative integers k and l, respectively. The ratio of Joint Hamming Weight of (k, l) to n is called* Joint Hamming Density.

The average Joint Hamming Density is $3/4$ for the binary representations of k and l. As a result, the number of additions required in Algorithm 2 is about $3n/4$ on average.

In order to speed up simultaneous scalar multiplication, Solinas gave the efficient signed binary representation of two non-negative integers [17]. The representation is called Joint Sparse Form (JSF), and any pair of non-negative integers has unique JSF. The average Joint Hamming Density of JSF is $1/2$, so that simultaneous scalar multiplication with JSF can reduce the number of additions to about $n/2$ on average.

4 Divisor Decomposition Technique

In this section we propose a novel technique called *Divisor Decomposition Technique* (DDT).

We now consider a general divisor $D = (x^2+u_1x+u_0, v_1x+v_0)$, where $u_i, v_i \in \mathbb{F}_{2^m}$ for $i = 0, 1$. D can be decomposed into two theta divisors $D_1 = (x+x_1, y_1)$ and $D_2 = (x+x_2, y_2)$ if $x^2+u_1x+u_0$ is factored to $(x+x_1)(x+x_2)$ over $\mathbb{F}_{2^m}$. It depends on only the reducibility of $x^2+u_1x+u_0$ over $\mathbb{F}_{2^m}$ whether D can be decomposed or not. Consequently, $Tr(u_0/u_1^2) = 0$ is the only condition of divisor decomposition, where $Tr(c)$ is trace of $c \in \mathbb{F}_{2^m}$. We show the procedure of DDT in Algorithm 3, where $Hr(c)$ is half-trace of c. T and H denote the computational cost of trace and half-trace, respectively.

Algorithm 3 *Divisor Decomposition Technique* (DDT)

Input: a general divisor $D = (x^2+u_1x+u_0, v_1x+v_0)$
Output: theta divisors $D_1, D_2, s.t. D = D_1 + D_2$ or FAILURE

Step	Procedure	Cost
1.	if $Tr(u_0/u_1^2) = 1$ return FAILURE	$1I + 1M + 1S + 1T$
2.	$x_1 \leftarrow u_1 Hr(u_0/u_1^2),\ x_2 \leftarrow x_1 + u_1$	$1M + 1H$
3.	$y_1 \leftarrow v(x_1),\ y_2 \leftarrow v(x_2)$	$2M$
4.	$D_1 \leftarrow (x+x_1, y_1),\ D_2 \leftarrow (x+x_2, y_2)$	
5.	return D_1, D_2	

The question we have to ask here is whether DDT contributes to the efficiency of scalar multiplication dD. As we have seen, each scalar multiplication dD_1, dD_2 is faster than dD. The direct computation of $dD_1 + dD_2$, however, is slower than scalar multiplication dD because Table 1 shows that TADD is not twice as fast as ADD.

5 Joint Regular Form

In this section we propose the other novel technique called *Joint Regular Form* (JRF).

We define a signed binary representation for a pair of non-negative integers as follows.

Definition 3. *Let* $\langle k_{n-1} \cdots k_0 \rangle$ *and* $\langle l_{n-1} \cdots l_0 \rangle$ *be signed binary representations of* k *and* l*, respectively, satisfying* $k+l \equiv 1 \pmod 2$*.* $\langle k_{n-1} \cdots k_0 \rangle$ *and* $\langle l_{n-1} \cdots l_0 \rangle$ *is called* Joint Regular Form (JRF) *of* (k,l)*, if* k_i *and* l_i *satisfy* $k_i + l_i = \pm 1$*, that is,* $(k_i, l_i) = (0, \pm 1)$ *or* $(\pm 1, 0)$ *for any* i*.*

Example 1. JRF of $(52, 39)$ is represented as follows, where $\bar{1}$ denotes -1.

$$\begin{aligned} 52 &= \langle\ 1\ 0\ \bar{1}\ 0\ 1\ 0\ 0\ \rangle \\ 39 &= \langle\ 0\ 1\ 0\ 1\ 0\ \bar{1}\ 1\ \rangle \end{aligned}$$

The Joint Hamming Density of JRF is always 1. JRF has the following properties:

- A pair of non-negative integers (k,l) satisfying $k+l \equiv 1 \pmod 2$ has a JRF.
- JRF of a certain length is unique.

We first prove the uniqueness of JRF.

Theorem 1. *A pair* (k,l) *of non-negative integers has at most one Joint Regular Form of a certain length.*

Proof. Assume that there are two distinct JRFs of length n as

$$\begin{aligned} k &= \langle k_{n-1} \cdots k_0 \rangle = \langle k'_{n-1} \cdots k'_0 \rangle \\ l &= \langle l_{n-1} \cdots l_0 \rangle = \langle l'_{n-1} \cdots l'_0 \rangle. \end{aligned}$$

Let j be the minimal value satisfying $k_i \neq k'_i$ or $l_i \neq l'_i$, and

$$\begin{aligned} s &= \langle k_{n-1} \cdots k_j \rangle = \langle k'_{n-1} \cdots k'_j \rangle \\ t &= \langle l_{n-1} \cdots l_j \rangle = \langle l'_{n-1} \cdots l'_j \rangle. \end{aligned}$$

We may assume that $k_j \neq k'_j$ by exchanging k and l if necessary. It follows that k_j and k'_j have value 1 and -1. We assume that $k_j = 1$ and $k'_j = -1$ without loss of generality. By the definition of JRF, $l_j = 0$ and $l'_j = 0$.

Suppose that $s \equiv 1 \pmod 4$. $k_{j+1} = 0$ and $k'_{j+1} = \pm 1$ since $k_j = 1$ and $k'_j = -1$. It follows that $l_{j+1} = \pm 1$ and $l'_{j+1} = 0$. The former indicates that $t \equiv 2 \pmod 4$ and the latter indicates that $t \equiv 0 \pmod 4$. This contradiction shows that the initial assumption must be wrong. Supposing $s \equiv 3 \pmod 4$, the similar contradiction occurs. $\square$

The most straightforward way to prove the existence of JRF for any pair of non-negative integers (k,l) satisfying $k+l \equiv 1 \pmod 2$ is to present an algorithm for constructing it. We explain how to construct JRF from the least significant bit. Let $(k_{n-1} \cdots k_0)_2$ and $(l_{n-1} \cdots l_0)_2$ be the binary representations of k and l, respectively. Firstly, $(k_0, l_0) = (0,1)$ or $(1,0)$ by $k + l \equiv 1 \pmod 2$.

Next, we notice (k_1, l_1). If $(k_1, l_1) = (0,0)$, either of the following transformations is carried out according to (k_0, l_0).

$$\begin{array}{cc} k_1 & k_0 \\ 0 & 1 \\ 0 & 0 \\ l_1 & l_0 \end{array} \Rightarrow \begin{array}{cc} k_1 & k_0 \\ 1 & \bar{1} \\ 0 & 0 \\ l_1 & l_0 \end{array} \qquad \begin{array}{cc} k_1 & k_0 \\ 0 & 0 \\ 0 & 1 \\ l_1 & l_0 \end{array} \Rightarrow \begin{array}{cc} k_1 & k_0 \\ 0 & 0 \\ 1 & \bar{1} \\ l_1 & l_0 \end{array}$$

If $(k_1, l_1) = (1, 1)$, one performs either of the following transformations according to (k_0, l_0).

$$\begin{array}{cc} k_1 & k_0 \\ 1 & 1 \\ 1 & 0 \\ l_1 & l_0 \end{array} \Rightarrow \begin{array}{ccc} & k_1 & k_0 \\ {\scriptstyle +1} & 0 & \bar{1} \\ & 1 & 0 \\ & l_1 & l_0 \end{array} \qquad \begin{array}{cc} k_1 & k_0 \\ 1 & 0 \\ 1 & 1 \\ l_1 & l_0 \end{array} \Rightarrow \begin{array}{ccc} & k_1 & k_0 \\ & 1 & 0 \\ {\scriptstyle +1} & 0 & \bar{1} \\ & l_1 & l_0 \end{array}$$

${\scriptstyle +1}$ means 1 is carried over to either $(k_{n-1} \cdots k_2)_2$ or $(l_{n-1} \cdots l_2)_2$. If $(k_1, l_1) = (0, 1)$ or $(1, 0)$, one needs no transformation. In all cases of (k_1, l_1), it is possible to satisfy the following conditions: $(k_0, l_0) = (0, \pm 1)$ or $(\pm 1, 0)$, and $(k_1, l_1) = (0, 1)$ or $(1, 0)$.

By applying this transformation from the least significant bit, we construct the signed binary representations $\langle k_{n-1} \cdots k_0 \rangle$ and $\langle l_{n-1} \cdots l_0 \rangle$ satisfying $(k_i, l_i) = (0, \pm 1)$ or $(\pm 1, 0)$ for any i. The detailed algorithm is shown in Algorithm 4.

Algorithm 4 *Joint Regular Form (JRF)*

Input: a pair of non-negative integers (k, l) s.t. $k + l \equiv 1 \pmod 2$
Output: JRF of (k, l): $\langle k_{n-1} \cdots k_0 \rangle$, $\langle l_{n-1} \cdots\ l_0 \rangle$

1. $i \leftarrow 0,\ s \leftarrow k,\ t \leftarrow l,$
2. while $s > 0$ or $t > 0$ do:
   2.1. $k_i = s \bmod 2$, $l_i = t \bmod 2$
   2.2. if $(k_i, l_i) = (0, 0)$ then
      $k_i \leftarrow k_{i-1}$, $k_{i-1} \leftarrow -k_{i-1}$, $l_i \leftarrow l_{i-1}$, $l_{i-1} \leftarrow -l_{i-1}$, $s \leftarrow s/2$, $t \leftarrow t/2$
   else if $(k_i, l_i) = (1, 1)$ then
      $k_i \leftarrow 1 - k_{i-1}$, $k_{i-1} \leftarrow -k_{i-1}$, $l_i \leftarrow 1 - l_{i-1}$, $l_{i-1} \leftarrow -l_{i-1}$,
      $s \leftarrow (s - 2k_i + 1)/2$, $t \leftarrow (t - 2l_i + 1)/2$
   else then
      $s \leftarrow (s - k_i)/2$, $t \leftarrow (t - l_i)/2$
   2.3. $i \leftarrow i + 1$
3. $n \leftarrow i$
4. return $\langle k_{n-1} \cdots k_0 \rangle$ and $\langle l_{n-1} \cdots l_0 \rangle$

If we apply JRF of (k, l) to the simultaneous scalar multiplication $kD_1 + lD_2$, we *always* compute a divisor class doubling and an addition of $\pm D_1$ or $\pm D_2$ per bit of (k, l). Consequently, we achieves the SPA-resistant simultaneous scalar multiplication without any dummy operation and pre-computation $D_1 \pm D_2$.

Remark 1. The advantages of simultaneous scalar multiplication with JRF are useful to not only HECC but also ECC. Lim-Lee method [14], GLV method [6], and BRIP [15] seem to be nice applications of JRF.

6 Combination of DDT and JRF

We show that the combination of DDT and JRF achieves efficient and secure scalar multiplication of a general divisor.

Suppose that a general divisor D can be decomposed into two theta divisors D_1 and D_2 as $D = D_1 + D_2$. We compute $(dD_1 + d'D_2) - D_2$ instead of the scalar multiplication dD, where $d' = d+1$. JRF of (d, d') is applied to the simultaneous scalar multiplication $dD_1 + d'D_2$. As we have discussed above, $dD_1 + d'D_2$ with JRF computes a divisor class doubling and an addition of a theta divisor $\pm D_1$ or $\pm D_2$ per bit of (d, d').

Indeed, JRF of (d, d'), $\langle d_n \cdots d_0 \rangle$ and $\langle d'_n \cdots d'_0 \rangle$, can be represented very easily without Algorithm 4 as follows:

1. Let $\langle d_{n-1} \cdots d_0 \rangle$ be the binary representation of d.
2. $d'_i = d_i - 1$ for $0 \le i \le n-1$.
3. Append $d_n = 0$ and $d'_n = 1$.

The validity of this representation is clearly shown by $d = \sum_{i=0}^{n-1} d_i 2^i$ and $d' = d + 1 = 2^n + \sum_{i=0}^{n-1} (d_i - 1) 2^i$. For example, JRF of $(53, 54)$ is represented as $53 = \langle 0110101 \rangle$ and $54 = \langle 100\bar{1}0\bar{1}0 \rangle$.

We present the detailed algorithm for theta divisors D_1 and D_2 in Algorithm 5. Obviously Algorithm 5 *always* computes a divisor class doubling and an addition of a theta divisor whether $d_i = 0$ or 1. Therefore, an SPA-attacker cannot guess any bit information of d.

Algorithm 5 *Simultaneous Scalar Multiplication with JRF* (SimJRF)

Input: a non-negative integer d, theta divisors D_1, D_2 s.t. $D = D_1 + D_2$
Output: dD

1. $D[0] \leftarrow -D_2$, $D[1] \leftarrow D_1$
2. $Q \leftarrow \mathsf{TDBL}(D_2)$
3. $Q \leftarrow \mathsf{TADD}(Q, D_1)$
4. for $i = n-2$ downto 0 do:
   4.1. $Q \leftarrow \mathsf{DBL}(Q)$
   4.2. $Q \leftarrow \mathsf{TADD}(Q, D[d_i])$
5. $Q \leftarrow \mathsf{TADD}(Q, D[0])$
6. return Q

As described in Section 4, a general divisor D cannot be always decomposed to two theta divisors; D can be decomposed only if $x^2 + u_1 x + u_0$ is reducible over $\mathbb{F}_{2^m}$, where $D = (x^2 + u_1 x + u_0, v_1 x + v_0)$. In order to apply DDT to any general divisor, we utilize a divisor class doubling (DBL) and its inverse operation, that is, a divisor class halving (HLV) [10]. If D cannot be decomposed, one repeats i times until $D' = 2^i D$ can be decomposed by $D' = D'_1 + D'_2$. After computing dD' using SimJRF, one then repeats a divisor class halving i times by $dD = 1/2^i (dD')$. We summarize our efficient and secure scalar multiplication algorithm for a general divisor in Algorithm 6.

Algorithm 6 DDT *and* SimJRF (DDT+SimJRF)

Input: a non-negative integer d, a general divisor D
Output: dD

1. $i \leftarrow 0$
2. while $\mathsf{DDT}(D)$ outputs "FAILURE" do:
   2.1. $D \leftarrow \mathsf{DBL}(D)$, $i \leftarrow i + 1$
3. $Q \leftarrow \mathsf{SimJRF}(D_1, D_2, d)$
4. while $i > 0$ do:
   4.1. $Q \leftarrow \mathsf{HLV}(Q)$, $i \leftarrow i - 1$
5. return Q

The iteration count i becomes 1 on average since DDT returns "FAILURE" in probability of about 1/2. Accordingly, we require *two* DDT and *one* DBL in step 2, and *one* HLV in step 4.

Remark 2. In HEC ElGamal-type decryption, a receiver A needs to compute $s_A D$, where s_A is A's secret key and D is a random divisor. In the case, instead of the operations DBL and HLV, we can utilize an addition of the base point G and a subtraction of $s_A G$, where $s_A G$ is A's public key.

7 Computational Efficiency

We estimate the computational cost of DDT+SimJRF proposed in Algorithm 6 and compare it to that of the Double-and-Add-Always method of a General Divisor (DAA_GD) and the Double-and-Add-Always method of a Theta Divisor (DAA_TD). The cost of divisor doublings and additions is referred to Table 1. As we have mentioned in Section 3.1, the cost of DAA_GD and DAA_TD is $(n-1)(2I + 44M + 8S)$ and $(n-2)(2I + 32M + 6S) + 2I + 28M + 3S$, respectively, where n is the bit length of d.

In order to estimate the cost of DDT+SimJRF in Algorithm 6, we evaluate the cost required in **divisor decomposition step** (step 2), SimJRF (step 3), and **compensation step** (step 4) as following.

divisor decomposition step According to our analysis in Section 6, we require *two* DDT, one of which returns "FAILURE", and *one* DBL on average. The estimated cost is $2(1I + 1M + 1S + 1T) + (3M + 1H) + (1I + 22M + 5S) = 3I + 27M + 7S + H + 2T$.

SimJRF We estimate the cost of SimJRF through Algorithm 5. In step 1, the inverse of a theta divisor $D_2 = (x + x_2, y_2)$, $-D_2 = (x + x_2, y_2 + h(x_0))$, is computed, which corresponds to $1M$. Step 2 and 3 are the main procedures of SimJRF. We then require *one* TDBL, $(n-1)$ DBL, and n TADD, which correspond to $(1I + 5M + 2S) + (n-1)(1I + 22M + 5S) + n(1I + 10M + 1S) = (n-1)(2I + 32M + 6S) + 2I + 15M + 3S$. Step 4 requires $1I + 10M + 1S$. As a result, the total cost of SimJRF is estimated to be $(n-1)(2I + 32M + 6S) + 3I + 26M + 4S$.

compensation step We require *one* HLV as we have shown in Section 6. According to [10], the cost of HLV is $1I + 19.5M + 2S + 2.5SR + 2H + 2T$, where SR denotes the cost of square root over $\mathbb{F}_{2^m}$.

Consequently, the cost of DDT+SimJRF is estimated to be $n(2I + 32M + 6S) + 5I + 40.5M + 7S + 2.5SR + 3H + 4T$ in total.

Suppose that the bit length of d is 160, that is, $n = 160$. According to [5, 10], we may assume that the following ratios of field operations to multiplication are satisfied: $I = 8M$, $S = 0.1M$, $SR = 0.5M$, $H = 0.6M$, and $T = 0$.

Table 2 summarizes the comparison of DAA_GD, DAA_TD, and DDT+SimJRF. The column 'Divisor' indicates whether each method computes scalar multiplication of a general divisor or a theta divisor; the column 'Dummy' indicates whether each method uses any dummy operation or not. The proposed method DDT+SimJRF is 18.7% faster than DAA_GD and eliminates any dummy operation. DDT+SimJRF requires no more than 1.8% increase of computational cost compared to DAA_TD that can be used only in the limited case.

Table 2. Comparison of scalar multiplication (160bit)

Method	Divisor	Dummy	Cost	
DAA_GD	general	use	$318I + 6996M + 1272S$	$(9667.2M)$
DAA_TD	theta	use	$318I + 5084M + 951S$	$(7723.1M)$
DDT+SimJRF	general	NOT use	$325I + 5160.5M + 967S$ $+2.5SR + 3H + 4T$	$(7860.3M)$

8 Conclusion and Further Work

In this paper, efficient and secure scalar multiplication of a general divisor for genus 2 HECC over $\mathbb{F}_{2^m}$ is proposed through Divisor Decomposition Technique (DDT) and Joint Regular Form (JRF). The proposed method achieves both about 19% improvement of efficiency compared to the double-and-add-always method and SPA resistance without any dummy operation.

It must be emphasized that the strategy of the proposed method is applicable to not only genus 2 HECC over $\mathbb{F}_{2^m}$. For genus 3 HECC, a general divisor whose weight is 3 might be decomposed into either three theta divisors of weight 1 or two theta divisors of weight 2. In the former case, we must generalize the concept of JRF to three non-negative integers. In the latter case, we need to develop efficient DDT in which a weight 3 divisor is decomposed into two weight 2 divisors.

References

1. D.G. Cantor, "Computing in the Jacobian of a Hyperelliptic Curve", *Mathematics of Computation*, 48, 177, pp.95-101, 1987.
2. H. Cohen, G. Frey, R. Avanzi, C. Doche, T. Lange, K. Nguyen, and F. Vercauteren, *Handbook of Elliptic Curve and Hyperelliptic Curve Cryptography*, Chapman & Hall, 2005.
3. J.-S. Coron, "Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems", *Cryptographic Hardware and Embedded Systems - CHES '99*, LNCS 1717, pp.292-302, Springer-Verlag, 1999.
4. S. Duquesne, "Montgomery Scalar Multiplication for Genus 2 Curves", *Algorithmic Number Theory - ANTS VI*, LNCS 3076, pp.153-168, Springer-Verlag, 2004.
5. K. Fong, D. Hankerson, J. López and A. Menezes, "Field inversion and point halving revised," Technical Report CORR 2003-81, 2003. http://www.cacr.math.uwaterloo.ca/techreports/2003/corr2003-18.pdf
6. R.P. Gallant, R.J. Lambert, and S.A. Vanstone, "Faster Point Multiplication on Elliptic Curves with Efficient Endomorphisms", *Advances in Cryptology - CRYPTO 2001*, LNCS 2139, pp.190-200, Springer-Verlag, 2001.
7. R. Harley, "Adding.txt, Doubling.c", 2000. http://cristal.inria.fr/ harley/hyper/
8. M. Katagi, T. Akishita, I. Kitamura, and T. Takagi, "Efficient Hyperelliptic Curve Cryptosystems Using Theta Divisors", *IEICE Trans. Fundamentals*, vol.E89-A, no.1, pp.151-160, 2006.
9. M. Katagi, I. Kitamura, T. Akishita, and T. Takagi, "Novel Efficient Implementations of Hyperelliptic Curve Cryptosystems Using Degenerate Divisors", *Information Security Application s - WISA 2004*, LNCS 3325, pp.345-359, Springer-Verlag, 2004.
10. I. Kitamura, M. Katagi, and T. Takagi, "A Complete Divisor Class Halving Algorithm for Hyperelliptic Curve Cryptosystems of Genus Two", *Information Security and Privacy - ACISP 2005*, LNCS 3674, pp.146-157, Springer-Verlag, 2005.
11. N. Koblitz, "Hyperelliptic Cryptosystems", *Journal of Cryptology*, vol.1, pp.139-150, Springer-Verlag, 1989.
12. S. Lang, "Abelian Varieties", Springer-Verlag, 1983.
13. T. Lange, "Formulae for Arithmetic on Genus 2 Hyperelliptic Curves", *Applicable Algebra in Engineering, Communication and Computing*, vol.15, pp.295-328, Springer-Verlag, 2005.
14. C.H. Lim and P.J. Lee, "More Flexible Exponentiation with Precomputation", *Advances in Cryptology - CRYPTO '94*, LNCS 839, pp.95-107, Springer-Verlag, 1994.
15. H. Mamiya, A. Miyaji, and H. Morimoto, "Efficient Countermeasure against RPA, DPA, and SPA", *Cryptographic Hardware and Embedded Systems - CHES 2004*, LNCS 3156, pp.343-356, Springer-Verlag, 2004.
16. D. Mumford, *Tata Lectures on Theta II*, Progress in Mathematics 43, Birkhäuser, 1984.
17. J.A. Solinas, "Low-Weight Binary Representations for Pairs of Integers", Technical Report CORR 2001-41, 2001. http://www.cacr.math.uwaterloo.ca/techreports/2001/corr2001-41.ps