Vol. 47 No. 8 IPSJ Journal Aug. 2006

Regular Paper

Fixed-Hamming-Weight Representation for Indistinguishable Addition Formulae☆

Hideyo Mamiya†,†† and Atsuko Miyaji†

In the execution of signature on a smart card, side channel attacks such as simple power analysis (SPA) have become serious threat[12]. There are the fixed procedure method and the indistinguishable method for SPA resistant methods. The indistinguishable method conceals all branch instructions by using indistinguishable addition formulae but may reveal the hamming weight when an addition chain with the un-fixed hamming weight is used. In the case of hyper-elliptic curve, the indistinguishable method has not been proposed yet. In this paper, we give an indistinguishable addition formulae of hyper-elliptic curve. We also give algorithms which output the fixed-hamming-weight representation for indistinguishable addition formulae and works with or without computation table, which can dissolve the above mentioned problem on the indistinguishable method and are also applied to an elliptic curve scalar multiplication.

1. Introduction

Elliptic curve cryptosystems and hyperelliptic curve cryptosystems chosen appropriately to avoid known attacks[1),7),15),29),31)] are vulnerable only to the Pollard $\rho-$method[21)] and the Pohlig-Hellman method[20)]. As a result, they can be constructed over a smaller definition field than discrete-logarithm-problem (DLP)-based cryptosystems[8),9)] and RSA cryptosystems[22)]. Elliptic curve cryptosystems (ECC) and hyperelliptic curve cryptosystems (HECC) with a 160-bit key are thus believed to have the same security as both the DLP-based cryptosystems and RSA with a 1,024-bit key. This is why ECC and HECC have been attractive in smart card applications, whose memory storage and CPU power is very limited. ECC and HECC execute a scalar multiplication of dP for a secret key d and a publicly known P as a cryptographic primitive, which determines the performance of a smart card.

In executions on a smart card, side channel attacks such as the simple power analysis (SPA) and the differential power analysis (DPA) have become a serious threat. Side channel attacks[12),13)] monitor power consumption and even exploit the leakage information related to power consumption to reveal bits of secret key d even though d is hidden inside the smart card. Thus, developing resistance to SPA, DPA, and specific DPAs, called refined power analysis (RPA)[10)] or zero-register analysis (ZRA)[2)]. Many countermeasures[3),4),11),13),14),16),17),19),30)] have been proposed so far.

This paper focuses on countermeasures against SPA, which consists of two main countermeasures: fixed procedure method[6)] and indistinguishable method[3)]. A fixed procedure method deletes any branch instruction conditioned by a secret d; some examples are the add-and-double-always method[6)], the Montgomery-ladder method[18)], and the SPA-resistant wNAF[19)]. The indistinguishable method conceals all branch instructions of scalar multiplication by using indistinguishable addition formulae[26),28),32)]. No indistinguishable addition formulae for a hyperelliptic curve has been proposed yet. An SPA attack against indistinguishable addition formulae that uses the difference in power consumption between multiplication and squaring in the Montgomery multiplication, was proposed[33)], but it can be easily avoided by modifying the Montgomery multiplication or reconstructing indistinguishable addition formulae on the condition that multiplication and squaring are assumed to be different. However, there is another problem with the indistinguishable addition formula in that the hamming weight of d is revealed when it is used to execute dP in an addition chain that depends on the hamming weight. As a result, to make the execution of dP secure, the indistinguishable addition formula needs such an addition chain[6),18)]

☆ This study was partly supported by Grant-in-Aid for Scientific Research (B) 17300002 and also partly supported by YAZAKI Corporation.

† School of Information Science, Japan Advanced Institute of Science and Technology

†† He joined to this work when he was in JAIST.

that outputs a fixed hamming weight for any d. Thus, in order to have an advantage against the fixed procedure method, the indistinguishable method needs an addition chain that outputs a fixed-hamming weight but does not run in a fixed procedure. In the case of binary representation, the add-and-double-always method[6] is only a method that works with the fixed hamming weight for any d. Therefore, it is useless for indistinguishable addition formulae to be used in such addition chains. On the other hand, in the case of signed-binary representation, no method has been proposed that works in a fixed-hamming weight and a fixed procedure. Furthermore, the SPA-resistant window method works in the fixed hamming weight but requires additional points for a precomputed table compared with the ordinary window method[5]. Therefore, these methods are open to further investigation for indistinguishable addition formulae.

In this paper, we first give an indistinguishable addition formula for a hyperelliptic curve. The formula deals with multiplication and squaring differently and thus is secure against the above attack[33]. Then, we present both signed-binary and signed-window algorithms, which work in a fixed hamming weight without additional memory. Our signed binary algorithm always computes dP in $(\lfloor \frac{n'}{2} \rfloor - 1)$ additions and $(n'-1)$ doublings without any precomputed point, where n' is the length of NAF representation of d. Compared with the add-and-double-always method, our method can reduce the computation amount by 0.2% or 21.3% with the same memory in the case of elliptic curve or hyperelliptic curve, respectively. Our signed w-window algorithm always computes dP in $(\lfloor \frac{n''}{w} \rfloor + 2^{w-2} - 2)$ additions and n'' doublings with 2^{w-2} precomputed points, where n'' is the length of wNAF representation of d. Compared with the SPA-resistant wNAF[19], our method can reduce the computation amount by 2.8% with the same memory in the case of a hyperelliptic curve.

This paper is organized as follows. Section 2 summarizes some facts of hyperelliptic curves such as coordinate systems and reviews SPA and DPA with some known countermeasures. Section 3 presents an indistinguishable addition formulae for hyperelliptic curves. Section 4.1 describes the relationship between a hamming-weight leak and the maximum computation amount of an addition chain and then presents the signed-binary algorithm and the signed-window algorithm of 2^{w-2} precomputed points in a fixed hamming weight.

2. Preliminary

This section summarizes some facts about hyperelliptic curves such as coordinate systems, and also reviews SPA, DPA, and RPA together with some known countermeasures.

2.1 Hyperelliptic Curve

Let $\mathbb{F}_p$ be a finite field, where $p > 3$ is a prime. A hyperelliptic curve $C/\mathbb{F}_p$ with genus 2 is described as

$$\begin{aligned} C : Y^2 &= F(X) \\ &= X^5 + f_4X^4 + f_3X^3 + f_2X^2 + f_1X + f_0, \end{aligned}$$

where $F(X)$ is in $\mathbb{F}_p[X]$. In the case of $p \neq 5$, we can set $f_4 = 0$. The divisors of a hyperelliptic curve are defined as the free abelian group of points $P_1, \ldots, P_r \in C$,

$$D = \sum_{P_i \in C} m_i P_i,\ m_i \in \mathbb{Z}.$$

The degree of D is defined as $\sum_{P_i \in C} m_i$ and order at P_i in C is defined as $m_i = ord_{P_i}(D)$. The Jacobian variety J_C is defined as D^0/D^l, where D^0 is a group of divisors with degree 0 and D^l is a group of divisors of functions. Any divisor $D \in J_C$ is equivalent modulo D^l to a divisor called a semi-reduced divisor as follows,

$$D \sim \sum_{P_i \in C} m_i' P_i - (\sum_{P_i \in C} m_i') P_\infty \ (m_i' \geq 0).$$

A semi-reduced divisor is further equivalent modulo D^l to a divisor called a reduced divisor,

$$D \sim \sum_{P_i \in C} m_i'' P_i - rP_\infty \ (r = \sum_{P_i \in C} m_i'' \leq g),$$

where g is a genus of C. Any divisor in J_C is uniquely represented by a reduced divisor. To compute an addition of divisors, Mumford-representation is useful. In Mumford-representation, $D \in C$ with the genus 2 is described by $D = (u_1, u_0, v_1, v_0)$, where

$$\begin{aligned} U &= \prod_{P_i} (X - x_i)^{m_i''} \\ &= X^2 + u_1X + u_0 \in \mathbb{F}_p[X], \\ V &= v_1X + v_0 \in \mathbb{F}_p[X], \\ V^2 &\equiv F(X) \pmod{U(X)}. \end{aligned}$$

Hyperelliptic curve cryptosystems (HECC) are defined over J_C, whose security depends on a hyperelliptic curve discrete logarithm problem (HECDLP), that is, a problem to find x for a given $D_1, D_2 \in J_C(\mathbb{F}_p)$ such that $D_1 = xD_2$.

2.2 Power Analysis

There are two types of power analysis[12),13)]: simple power analysis (SPA) and differential power analysis (DPA). In the case of elliptic curve and also hyperelliptic curve, DPA is further improved to use a special point with a zero value, the refined power analysis (RPA)[10)] and the zero-register analysis (ZRA)[2)].

2.2.1 Simple Power Analysis

SPA makes use of such an instruction performed during a scalar multiplication algorithm that depends on the data being processed. Apparently, Algorithm 1 has a branch instruction conditioned by a secret d, and thus it reveals the secret d. To be resistant to SPA, any branch instruction of an exponentiation algorithm should be eliminated. There are two main countermeasures: the fixed procedure method[6)] and the indistinguishable method[3)]. The fixed procedure method deletes any branch instruction conditioned by a secret exponent d such as the add-and-double-always method[6)](Algorithm 2), the Montgomery-ladder method[18)], and the SPA-resistant wNAF[19)]. The indistinguishable method conceals all branch instructions of a scalar multiplication algorithm by using indistinguishable addition and doubling formulae. However, use of the indistinguishable addition formula may leak the number of additions if we use an exponentiation algorithm whose number of additions depend on d. Therefore, we need an algorithm that always computes dP in the fixed number of additions and that has a branch instruction to make the indistinguishable addition formula secure and worthy, respectively.

Algorithm 1

```
Binary algorithm (from MSB)
Input:  d, P  ( n is length of d )
Output:  dP
1.  T0 = O, T1 = P.
2.  for i = n − 2 to 0
        T0 = 2T0
        if di = 1 then T0 = T0 + T1
3.  output T0.
```

Algorithm 2

```
Add-and-double-always algorithm
Input:  d, P
Output:  dP
1.  T0 = P and T2 = P.
2.  for i = n − 2 to 0
        T0 = 2T0.   T1 = T0 + T2.
        if di = 0 then T0 = T0.
        else T0 = T1.
3.  output T0.
```

2.2.2 Differential Power Analysis

DPA uses correlation between power consumption and specific key-dependent bits. Algorithm 2 reveals d_{n-2} by computing the correlation between power consumption and any specific bit of the binary representation of $4P$. In order to be resistant against DPA, power consumption should be changed at each new execution of the scalar multiplication. There are mainly 4 types of countermeasures, the randomized-projective-coordinate method (RPC)[6)], the randomized curve method (RC)[11)], the exponent splitting (ES)[3),4)], and the randomized initial point (RIP)[14),25)]. RPC uses the Jacobian or Projective coordinate to randomize a point $P = (x, y)$ into (r^2x, r^3y, r) or (rx, ry, r) for a random number $r \in \mathbb{F}_p^*$, respectively. RC maps an elliptic curve into an isomorphic elliptic curve by using an isomorphism map of (x, y) to (c^2x, c^3y) for $c \in \mathbb{F}_p^*$. However, both RC and RPC are vulnerable against RPA and ZRA, which uses a special elliptic-curve point with a zero value of $(x, 0)$ or $(0, y)$ or a register of addition or doubling formula with a zero value. These special points still have a zero value even if it is converted by RPC or RC. The same discussion holds in ZRA. This is why both RC and RPC are vulnerable against RPA and ZRA. The only method secure against RPA and ZRA is ES and RIP, where ES splits an exponent and computes $dP = rP + (d - r)P$ for a randomly integer r and RIP adds a random pint R and computes $dP = (dP + R) - R$.

3. Indistinguishable hyperelliptic-curve Addition Formulae

Indistinguishable addition formulae that execute addition and doubling in a unified procedure have been proposed in elliptic curve cryptosystems[26),28),32)]. However, because of complicated addition formulae, there has been no indistinguishable addition formula of hyperelliptic curve cryptosystems. Unifying both addition and doubling of hyperelliptic curves is difficult without any dummy execution. This is why we aim to derive indistinguishable addition formulae in hyperelliptic curves.

An addition formula consists of an inversion, multiplication, squaring, addition and subtraction in $\mathbb{F}_p$, whose power consumption is decreased in order of

$$\text{inversion} \gg \text{multiplication} > \text{square} \gg \text{addition} \simeq \text{subtraction}.$$

We revise addition formulae[27] in such a way that both addition and doubling are executed in the same procedure with negligible additional registers and computation. We assume that the computation amount of $2x$ is equal to that of $x + x$; that parallelizing is not used; and that registers can be shared between addition and doubling. Our strategies of indistinguishable hyperelliptic-curve addition formulae are as follows.

Dependency analysis. To analyze dependencies between formulae, we use a program dependence graph (PDG)[23], which is a program data flow graph to make the dependency of each variable clear. Figure 1 illustrates one example.

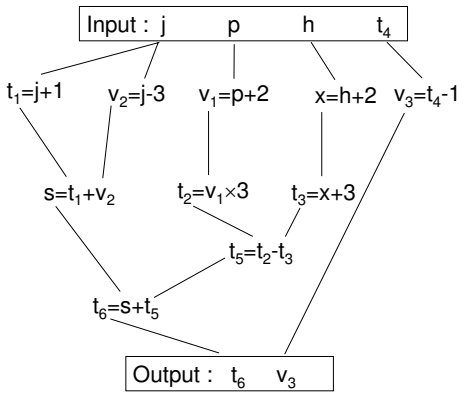


Fig. 1 Program Dependence Graph

Register allocation and coding. Allocate registers in addition and doubling of formulae[27] by as-late-as-possible (ALAP) scheduling policy[24]. The formulae do not consider the registers' allocation, and many registers are used wastefully. ALAP allocates registers to variables just before they are used. Let sets of operations of each formula be $\mathcal{ADD} = \{F_{a.1}, ..., F_{a.i}\}$ and $\mathcal{DBL} = \{F_{d.1}, ..., F_{d.j}\}$. For two input registers in addition, one register can be used for execution, while the other maintains the input value during execution, and can be used in the next procedure. For doubling, one input register also maintains the input value during execution.

Unifying from the last operation of $\mathcal{ADD}$. Both the addition and doubling are unified from the last operations of $\mathcal{ADD}$ because $\mathcal{ADD}$ uses more registers than $\mathcal{DBL}$, and both are similar to each other from the middle to the last operation. As a result, we can reduce the number of registers. Let a set of operations of unified formulae be $\mathcal{U} = \{U_1, ..., U_k\}$, where $U_1 \cdots U_k$ are described in execution order and U_i consists of addition and doubling operations such as $\{F_{a.i}, F_{d.j}\}$.

Unifying from the first operations. Operations of $\mathcal{ADD}$ and $\mathcal{DBL}$ are unified from the first operation based on the following order, where one additional register for dummy operations are allowed.

- (1) Unifying without dummy operation.
  ◦ Choose an executable operation $F_{a.i} \in \mathcal{ADD}$. If not, go to (2).
  ◦ Search all executable-operation sequences of $\mathcal{DBL}$ with the same operation as $F_{a.i}, F_{a.i+1}, ...$, choose the longest sequence, and renew $\mathcal{U}$. Retry (1) again.
- (2) Unifying with dummy operations.
  ◦ Choose $\mathcal{ADD} \ni F_{a.i}$ with the smallest un-unified i. If not, go to (3).
  ◦ If there exist operation sequences of executable $F_{d.i}$ in $\mathcal{DBL}$ that are the same operation as $F_{a.i}$, then choose the smallest sequence, insert dummy operations $F_{a.dummy}$ in $\mathcal{ADD}$ if necessary, and renew unified formula $\mathcal{U}$. Retry (2) again.
  ◦ Otherwise, put dummy operations $F_{d.dummy}$ with the same operation as $F_{a.i}$ in $\mathcal{DBL}$ and renew $\mathcal{U}$ by using $U_t = \{F_{a.i}, F_{d.dummy}\}$. Retry (2) again.
- Repeat steps (1) and (2) above by changing the order of operations of $\mathcal{ADD}$ as long as they can run.

Table 1 shows the indistinguishable addition formulae. The efficiency is shown in Table 2, where M, S, I, Add, and Sub respectively indicate the computation times for multiplication, squaring, inversion, addition, and subtraction. Indistinguishable addition formulae can be derived by adding only a few dummy operations and one register.

4. Addition Chain with a Fixed Hamming Weight

The computation amount of the addition chain of dP usually depends on "d", which means that an addition chain does not usually work with a fixed hamming weight. In this sec-

Table 1 Hyperelliptic-curve indistinguishable addition formulae

Addition: Input	$(R_{00},\ R_{01},\ R_{02},\ R_{03}) \leftarrow (u_{20},\ u_{21},\ v_{20},\ v_{21}) = D_1$
	$(R_{04},\ R_{05},\ R_{06},\ R_{07}) \leftarrow (u_{10},\ u_{11},\ v_{10},\ v_{11}) = D_2$
Addition: Output	$D_2 = (u'_0,\ u'_1,\ v'_0,\ v'_1) \leftarrow (R_{00},\ R_{01},\ R_{02},\ R_{09})$
Doubling: Input	$(R_{00},\ R_{01},\ R_{02},\ R_{03}) \leftarrow (u_0,\ u_1,\ v_0,\ v_1) = D_1$
Doubling: Output	$D_2 = (u'_0,\ u'_1,\ v'_0,\ v'_1) \leftarrow (R_{09},\ R_{06},\ R_{04},\ R_{08})$
	R_d, R'_d is any register ($\neq 0$)

	Addition $D_2 \leftarrow D_1 + D_2$	**Doubling** $D_2 \leftarrow 2D_1$			
1	$R_{08} \leftarrow R_d^2$	$R_{04} \leftarrow R_{03}^2$	37	$R_{03} \leftarrow R_{03}^2$	$R_{04} \leftarrow R_{04}^2$
2	$R_{08} \leftarrow 2 \times R_d$	$R_{05} \leftarrow 2 \times R_{02}$	38	$R_{03} \leftarrow R_{03} \times R_{09}$	$R_{04} \leftarrow R_{04} \times R_{07}$
3	$R_{08} \leftarrow R_d + R'_d$	$R_{06} \leftarrow f_3 + R_{04}$	39	$R_{09} \leftarrow R_{10} \times R_{09}$	$R_{07} \leftarrow R_{06} \times R_{07}$
4	$R_{08} \leftarrow 2 \times R_d$	$R_{07} \leftarrow 2 \times R_{05}$	40	$R_{02} \leftarrow R_{02} \times R_{09}$	$R_{05} \leftarrow R_{05} \times R_{07}$
5	$R_{08} \leftarrow R_{01} - R_{05}$	$R_{07} \leftarrow R_{07} - R_{06}$	41	$R_{09} \leftarrow R_{10} \times R_{09}$	$R_{06} \leftarrow R_{06} \times R_{07}$
6	$R_{09} \leftarrow 2 \times R_d$	$R_{04} \leftarrow 2 \times R_{04}$	42	$R_{01} \leftarrow R_{02} - R_{01}$	$R_{08} \leftarrow R_d - R'_d$
7	$R_{09} \leftarrow R_d + R'_d$	$R_{04} \leftarrow R_{04} + R_{06}$	43	$R_{10} \leftarrow R_{05} \times R_{02}$	$R_{08} \leftarrow R_{03} \times R_{05}$
8	$R_{03} \leftarrow R_{03} - R_{07}$	$R_{04} \leftarrow R_{04} - R_{05}$	44	$R_{10} \leftarrow R_{10} + R_{04}$	$R_{08} \leftarrow R_{08} + R_{02}$
9	$R_{09} \leftarrow 2 \times R_d$	$R_{05} \leftarrow 2 \times R_{01}$	45	$R_{11} \leftarrow R_{02} - R_{08}$	$R_{09} \leftarrow R_d - R'_d$
10	$R_{09} \leftarrow R_{08} \times R_{03}$	$R_{06} \leftarrow R_{03} \times R_{05}$	46	$R_{01} \leftarrow R_{01} \times R_{11}$	$R_{09} \leftarrow R_d \times R'_d$
11	$R_{10} \leftarrow 2 \times R_d$	$R_{08} \leftarrow 2 \times R_{00}$	47	$R_{00} \leftarrow R_{01} - R_{00}$	$R_{09} \leftarrow R_d - R'_d$
12	$R_{10} \leftarrow R_{04} - R_{00}$	$R_{09} \leftarrow R_{08} - R_{06}$	48	$R_{00} \leftarrow R_{00} + R_{10}$	$R_{09} \leftarrow R_d + R'_d$
13	$R_{11} \leftarrow -R_d$	$R_{05} \leftarrow -R_{05}$	49	$R_{01} \leftarrow 2 \times R_{07}$	$R_{09} \leftarrow 2 \times R_{01}$
14	$R_{02} \leftarrow R_{02} - R_{06}$	$R_{06} \leftarrow R_{08} - R_{06}$	50	$R_{01} \leftarrow R_{01} \times R_{09}$	$R_{09} \leftarrow R_{09} \times R_{06}$
15	$R_{03} \leftarrow R_{02} + R_{03}$	$R_{10} \leftarrow R_{09} + R_{05}$	51	$R_{11} \leftarrow R_d^2$	$R_{10} \leftarrow R_{05}^2$
16	$R_{11} \leftarrow R_{01} \times R_{08}$	$R_{07} \leftarrow R_{03} \times R_{07}$	52	$R_{00} \leftarrow R_{00} + R_{01}$	$R_{09} \leftarrow R_{10} + R_{09}$
17	$R_{11} \leftarrow R_{11} + R_{10}$	$R_{07} \leftarrow R_{07} + f_2$	53	$R_{01} \leftarrow R_{09}^2$	$R_{06} \leftarrow R_{06}^2$
18	$R_{02} \leftarrow R_{11} \times R_{02}$	$R_{06} \leftarrow R_{08} \times R_{06}$	54	$R_{09} \leftarrow 2 \times R_{05}$	$R_{10} \leftarrow 2 \times R_d$
19	$R_{10} \leftarrow R_{10} \times R_{11}$	$R_{05} \leftarrow R_{04} \times R_{05}$	55	$R_{09} \leftarrow R_{09} + R_{08}$	$R_{10} \leftarrow 2 \times R_{03}$
20	$R_{11} \leftarrow R_{08} + R_{11}$	$R_{08} \leftarrow R_d + R'_d$	56	$R_{09} \leftarrow R_{09} \times R_{01}$	$R_{10} \leftarrow R_{10} \times R_{06}$
21	$R_{12} \leftarrow R_{08}^2$	$R_{08} \leftarrow R_{01}^2$	57	$R_{00} \leftarrow R_{00} + R_{09}$	$R_{09} \leftarrow R_{09} + R_{10}$
22	$R_{03} \leftarrow R_{11} \times R_{03}$	$R_{11} \leftarrow R_{02} \times R_{05}$	58	$R_{09} \leftarrow 2 \times R_{02}$	$R_{10} \leftarrow 2 \times R_{05}$
23	$R_{03} \leftarrow R_{03} - R_{02}$	$R_{07} \leftarrow R_{07} - R_{08}$	59	$R_{11} \leftarrow R_{05} + R_{02}$	$R_{05} \leftarrow R_{03} + R_{05}$
24	$R_{11} \leftarrow 4 \times R_d$	$R_{08} \leftarrow 4 \times R_{08}$	60	$R_{08} \leftarrow R_{09} - R_{08}$	$R_{11} \leftarrow R_d - R'_d$
25	$R_{11} \leftarrow R_{12} \times R_{00}$	$R_{08} \leftarrow R_{02} \times R_{08}$	61	$R_{01} \leftarrow R_{08} - R_{01}$	$R_{06} \leftarrow R_{10} - R_{06}$
26	$R_{10} \leftarrow R_{10} + R_{11}$	$R_{06} \leftarrow R_{08} + R_{06}$	62	$R_{08} \leftarrow R_{11} - R_{01}$	$R_{05} \leftarrow R_{05} - R_{06}$
27	$R_{11} \leftarrow 1 + R_{01}$	$R_{08} \leftarrow 1 + R_{03}$	63	$R_{02} \leftarrow R_{04} \times R_{02}$	$R_{07} \leftarrow R_{02} \times R_{07}$
28	$R_{11} \leftarrow R_{09} \times R_{11}$	$R_{05} \leftarrow R_{05} \times R_{08}$	64	$R_{09} \leftarrow R_{01} \times R_{08}$	$R_{10} \leftarrow R_{06} \times R_{05}$
29	$R_{09} \leftarrow R_{00} \times R_{09}$	$R_{08} \leftarrow R_{07} \times R_{09}$	65	$R_{09} \leftarrow R_{09} + R_{00}$	$R_{10} \leftarrow R_{10} + R_{09}$
30	$R_{12} \leftarrow R_d + R'_d$	$R_{04} \leftarrow R_{07} + R_{04}$	66	$R_{09} \leftarrow R_{09} - R_{10}$	$R_{08} \leftarrow R_{10} - R_{08}$
31	$R_{12} \leftarrow R_d \times R'_d$	$R_{04} \leftarrow R_{10} \times R_{04}$	67	$R_{09} \leftarrow R_{09} \times R_{03}$	$R_{08} \leftarrow R_{08} \times R_{04}$
32	$R_{12} \leftarrow R_d - R'_d$	$R_{04} \leftarrow R_{04} - R_{08}$	68	$R_{09} \leftarrow R_{09} - R_{07}$	$R_{08} \leftarrow R_{08} - R_{01}$
33	$R_{03} \leftarrow R_{03} - R_{11}$	$R_{04} \leftarrow R_{04} - R_{05}$	69	$R_{08} \leftarrow R_{00} \times R_{08}$	$R_{05} \leftarrow R_{09} \times R_{05}$
34	$R_{02} \leftarrow R_{02} - R_{09}$	$R_{05} \leftarrow R_{08} - R_{11}$	70	$R_{02} \leftarrow R_{08} - R_{02}$	$R_{05} \leftarrow R_{05} - R_{07}$
35	$R_{09} \leftarrow R_{10} \times R_{03}$	$R_{07} \leftarrow R_{06} \times R_{04}$	71	$R_{02} \leftarrow R_{02} \times R_{03}$	$R_{04} \leftarrow R_{05} \times R_{04}$
36	$R_{09} \leftarrow R_{09}^{-1}$	$R_{07} \leftarrow R_{07}^{-1}$	72	$R_{02} \leftarrow R_{02} - R_{06}$	$R_{04} \leftarrow R_{04} - R_{00}$

Table 2 Complexity and number of registers

	$\#M$	$\#S$	$\#\ I$	(1)	$\#Add$	$\#Sub$	(2)	(3)	# register
Addition	22	3	1	0	14	17	1	0	12
Doubling	22	5	1	1	19	14	1	1	11
Proposed formulae	**23**	**5**	**1**	**1**	**22**	**18**	**1**	**1**	**13**
#Dummy in Addition	1	2	0	1	8	1	0	1	-
#Dummy in Doubling	1	0	0	0	3	4	0	0	-

(1): # multiplication to a small constant, (2): # addition to a small constant, (3): # sign conversion

tion, however, we investigate a condition of an addition chain which does work with a fixed hamming weight. We present Algorithm 3, which always computes dP in the computation of $(\lfloor \frac{n'}{2} \rfloor - 1)A + (n'-1)D$ from MSB by using the NAF representation of $d^{\mathrm{n}} = \sum_{i=0}^{n'} d_i^{\mathrm{n}} 2^i$, where n' is the length of the NAF representation. We also present Algorithm 4, which always computes dP in the computation of $(\lfloor \frac{n''}{w} \rfloor + 2^{w-2} - 2)A + n''D$ with 2^{w-2} precom-

puted points, where n'' is the length of the wNAF representation.

4.1 The Relation between the Hamming-weight Leak and the Maximum Computation

Algorithm 1 to compute $dP = \sum_{i=0}^{n} d_i 2^i P$ executes doubling and addition if $d_i = 1$; otherwise it executes only doubling. Therefore, even if we use the indistinguishable addition formula, the number of additions, that is the hamming weight, is leaked by measuring the power consumption. This is why we have to use Algorithm 2 even with the indistinguishable addition formula. The computation amount of indistinguishable addition formulae is larger than that of conventional formulae and, thus, the indistinguishable addition formulae do not give any advantage over the conventional formulae in the case of binary algorithm. Then what algorithm is useful for the indistinguishable addition formula? Let us think about the algorithmic meaning between Algorithms 1 and 2.

As we mentioned above, the complexity of the addition chain depends on d; that is, there are worst and best cases. The worst case in an addition chain means that it works with a fixed hamming weight. Therefore, the indistinguishable addition formula can work in the worst case of an addition chain without revealing the hamming weight (HW). The worst case in Algorithm 1 is, apparently, that the hamming weight (HW) of d is full for n, in which both addition and doubling are done in each bit of d. This is exactly the case of Algorithm 2. Therefore, we can consider Algorithm 2 to be the worst case of Algorithm 1. In the same way, the SPA-resistant wNAF[19] is considered to be the worst case of wNAF.

In summary, the indistinguishable addition formula has an advantage over conventional addition formulae for an addition chain of dP that always works in the worst case of d but has a branch instruction. This is because a conventional addition formula reveals the branch if it is used on such an algorithm. The addition-subtraction chain works in $(\lfloor \frac{n}{3} \rfloor - 1)A + (n-1)D$ on the average and $(\lfloor \frac{n}{2} \rfloor - 1)A + (n-1)D$ on the maximum, where A and D mean the computation amount of addition and doubling, respectively. We remark that an addition-subtraction method that always has the maximum computation has not yet been proposed. On the other hand, the wNAF[5] computes dP in $\lfloor \frac{n}{w+1} \rfloor A + (n-1)D$ on the average and in $(\lfloor \frac{n}{w} \rfloor - 1)A + (n-1)D$ at the maximum for an n-bit d with 2^{w-2} precomputed points. The SPA-resistant wNAF[19], called S-wNAF in short, needs 2^{w-1} precomputed points and always works in $(\lfloor \frac{n}{w} \rfloor - 1)A + (n-1)D$. We can say that S-$w$NAF realizes the SPA resistance by the sacrifice of additional precomputed points. Note that the window method has not been proposed that always run with the maximum computation while maintaining 2^{w-2} precomputed points.

4.2 Fixed-hamming-weight Signed-binary Algorithm

In the NAF representation d^{n}, a non-zero bit is always next to 0. Therefore, the HW of d^{n} is less than or equal to $\lfloor \frac{n'}{2} \rfloor$. We can compute dP while increasing the HW of d^{n} to exactly $\lfloor \frac{n'}{2} \rfloor$ by the following rules:

• **Case 1** If $d_i^{\mathrm{n}} d_{i-1}^{\mathrm{n}}$ is 00, then execute addition, doubling, subtraction, and subtraction. This means that 00 is converted into $1(\bar{1} + \bar{1})$ and then HW is increased by 3.

• **Case 2** If d_i^{n} is 0, then execute addition and subtraction. This means that 0 is converted into $(1 + \bar{1})$ and then HW is increased by 2.

• **Case 3** If $d_i^{\mathrm{n}} d_{i-1}^{\mathrm{n}}$ is 01 or $0\bar{1}$, then execute addition, doubling, and subtraction, or subtraction, doubling, and addition, respectively. This means that 01 or $0\bar{1}$ is converted into $1\bar{1}$ or $\bar{1}1$, respectively, and then HW is increased by 1.

Therefore, we compute dP by executing Case 1 or 2 if $d_i^{\mathrm{n}} d_{i-1}^{\mathrm{n}} = 00$; and Case 3 if $d_i^{\mathrm{n}} d_{i-1}^{\mathrm{n}} = 0\bar{1}$ or 01 until HW $= \lfloor \frac{n'}{2} \rfloor$. The following algorithm 3 describes these steps in detail. The detailed algorithm to compute dP in the signed binary algorithm with a fixed hamming weight is given as follows, where it starts at the NAF representation d^{n} and computes dP from MSB.

Algorithm 3

```
Fixed-HW-Signed-binary algorithm
Input :  P, d = Σ_{i=0}^{n} d_i 2^i (binary representation)
Output:  dP
1.  Convert d = Σ_{i=0}^{n} d_i 2^i to the NAF
    representation d^n = Σ_{i=0}^{n'} d_i^n 2^i;
2.  Set HW of d^n to l;
3.  Q = P;
4.  for i = n' − 1 down to 0
      Q = 2Q;
      if d_i^n = ±1 then Q = (Q ± P);
      else
```

```
    if (⌊n'/2⌋ − l) ≥ 1 and d^n_{i−1} = ±1, then
     Q = Q ± P, Q = 2Q, Q = Q ∓ P,
     l = l + 1, and i = i − 1;
    else if 3 > ⌊n'/2⌋ − l > 1 and d^n_{i−1} = 0,
     then Q = Q + P, Q = Q − P, and
     l = l + 2;
    else if ⌊n'/2⌋ − l ≥ 3 and d^n_{i−1} = 0, then
     Q = Q + P, Q = 2Q, Q = Q − P,
     Q = Q − P, l = l + 3, and i = i − 1;
   next i.
5. Output Q;
```

Theorem 1 For any n-bit d, Algorithm 3 always computes dP in $(\lfloor \frac{n'}{2} \rfloor - 1)A + (n'-1)D$, where n' is the length of NAF representation d^{n}.

proof: Let HW of d^{n} be l and $a = \lfloor \frac{n'}{2} \rfloor - l$. The proof is done inductively. In the case of $a = 1$, then there exist two consecutive bits of $d^{\mathrm{n}}_i d^{\mathrm{n}}_{i-1} = 01$ or $0\bar{1}$ because a non-zero bit is next to 0 in NAF representation. Algorithm 3 executes Case 3. Then HW becomes $\lfloor \frac{n'}{2} \rfloor$. We assume that HW becomes $\lfloor \frac{n'}{2} \rfloor$ by Algorithm 3 in the case of $a = i$. When $a = i+1$, there exist two consecutive zero bits such as $d^{\mathrm{n}}_{i_0} d^{\mathrm{n}}_{i_0-1} = 00$ because $\lfloor \frac{n'}{2} \rfloor - l = i + 1 \geq 2$. We set the most significant bit to i_0, i.e $d^{\mathrm{n}}_{n'} \cdots d^{\mathrm{n}}_{i_0+1} d^{\mathrm{n}}_{i_0} d^{\mathrm{n}}_{i_0-1} = 1010 \cdots 100$. Then $d^{\mathrm{n}}_{n'} \cdots d^{\mathrm{n}}_{i_0+1}$ coincides with the NAF representation whose density of 1 is $\frac{1}{2}$. Algorithm 3 executes Case 1 to $d^{\mathrm{n}}_{i_0}$ if $i + 1 \geq 3$, or Case 2 to $d^{\mathrm{n}}_{i_0}$ if $i + 1 = 2$; then $\lfloor \frac{n'}{2} \rfloor -$ HW becomes $i - 2 < i$ or $i - 1 < i$, respectively. In both cases, $d^{\mathrm{n}}_{i_0-2} \cdots d^{\mathrm{n}}_0$ is still a NAF representation. Therefore HW becomes $\lfloor \frac{n'}{2} \rfloor$ from the assumption of induction. ∎

Corollary 1 The computation amount of Algorithm 3 is the minimum among the fixed-hamming-weight signed binary algorithms without any additional precomputed point.

proof: From Theorem 1, Algorithm 3 can always compute dP in the exact same computation amount as in the worst case of the addition-subtraction algorithm. A worst case of d exists such as $d = 10$ or 101: they cannot be transformed into an addition-subtraction algorithm that works faster than the worst case. Therefore, the computation amount of Algorithm 3 is the minimum among the fixed-hamming-weight algorithms without any additional precomputed point. ∎

4.3 Fixed-hamming-weight Signed Window Algorithm

Let a window representation with the width w and the wNAF representation of d be d^{w} and d^{wn}, respectively. The number of non-zero bits of d^{w} and d^{wn} is denoted by wHW. For example, wHW of d^{wn} is 5 when $d^{\mathrm{wn}} = 10003000\bar{3}0000070001$. An interesting property is that d^{wn} is uniquely represented for d, but neither the signed binary representation d^{s} nor the window representation d^{w} is determined uniquely. For example, the above d^{wn} can be represented by $d^{\mathrm{s}} = d^{\mathrm{s}}_{n-1} \cdots d^{\mathrm{s}}_0 = 1001100\bar{1}\bar{1}001110001$, and it can be represented by $d^{\mathrm{s}} = 11\bar{1}0\bar{1}\bar{1}10101001\bar{1}\bar{1}\bar{1}\bar{1}$, which is converted to $d^{\mathrm{w}} = d^{\mathrm{w}}_{n-1} \cdots d^{\mathrm{w}}_0 = 500\bar{3}005010001\bar{0}0\bar{7}$.

In the wNAF representation d^{wn}, a non-zero bit is always next to $(w-1)$ 0s. Therefore, wHW of d^{wn} is less than or equal to $\lfloor \frac{n''}{w} \rfloor$, where n'' is the length of the wNAF representation of $d^{\mathrm{wn}} = \sum_{i=0}^{n''} d^{\mathrm{n}}_i 2^i$. We can compute dP while increasing wHW of d^{wn} to exactly $\lfloor \frac{n''}{w} \rfloor$ by the conversion rules of Cases 1 and 2 in Section 4.2 and of the following Case 3', which is a modification of Case 3 in Section 4.2.

• **Case 3'** If $d^{\mathrm{wn}}_i = \cdots = d^{\mathrm{wn}}_{i-w+2} = 0$ and $d^{\mathrm{wn}}_{i-w+1} \neq 0$, then execute an addition or a subtraction, $(w-1)$ doublings, and subtraction of $([s]2^{w-1} - d^{\mathrm{wn}}_{i-w+1})P$ in this order, where s is the sign of d^{wn}_{i-w+1}. Namely, $d^{\mathrm{wn}}_i \cdots d^{\mathrm{wn}}_{i-w+1}$ is converted to $= [s]1 \cdots (-1)([s]2^{w-1} - d^{\mathrm{wn}}_{i-w+1})$, and thus wHW is increased by 1.

The detailed algorithm to compute dP using the signed-window method with a fixed hamming weight is given as the following Algorithm 4, where it starts at the wNAF representation d^{wn} and computes dP from MSB with the precomputation table $\{P, .., 2^{w-1}P\}$.

Algorithm 4

```
Fixed-wHW signed window algorithm
Input : P, d = Σ d_i 2^i (binary representation)
Output: dP
1. Convert d = Σ_{i=0}^{n} d_i 2^i to the wNAF
   representation d^wn = Σ_{i=0}^{n''} d_i^wn 2^i.
2. Set wHW of d^wn to l.
3. Q = d^wn_{n''} P.
4. For i = n'' − 1 down to 0
    Q = 2Q.
    if d_i^wn ≠ 0 then Q = Q + d_i^wn P;
    else
     if ⌊n''/w⌋ − l ≥ 1, d^wn_{i−1} = ··· = d^wn_{i−w+2} = 0
     and d^wn_{i−w+1} ≠ 0, then
      set the sign of d^wn_{i−w+1} to s,
      Q = Q + [s]P, Q = 2^{w−1} Q,
      Q = Q − ([s]2^{w−1} − d^wn_{i−w+1})P, l = l + 1
      and i = i − (w − 1);
```

`else if` $3 > \lfloor \frac{n''}{w} \rfloor - l > 1$, `then`
$Q = Q + P$, $Q = Q - P$, `and` $l = l + 2$;
`else if` $\lfloor \frac{n''}{w} \rfloor - l \geq 3$ `and` $d_{i-1}^{\mathsf{wn}} = 0$, `then`
$Q = Q + P$, $Q = 2Q$, $Q = Q - P$,
$Q = Q - P$, $l = l + 3$ `and` $i = i - 1$;
`next` i.
5. `Output` Q.

Theorem 2 For any n-bit d, Algorithm 4 computes dP in $(\lfloor \frac{n''}{w} \rfloor + 2^{w-2} - 2)A + n''D$ with the precomputation table of $T = \{P, 3P, ..., (2^{w-1} - 1)P\}$ $(\#T = 2^{w-2})$ on the assumption that the precomputation table is constructed using a simple method of $(2^{w-2} - 1)A + D$ $(w > 2)$, where n'' is the length of the wNAF representation d^{wn}.

proof: Let HW of d^{wn} be l and $a = \lfloor \frac{n''}{w} \rfloor - l$. The proof is done inductively. In the case of $a = 1$, then there exist w consecutive bits of $d_i^{\mathsf{wn}} \cdots d_{i-w+1}^{\mathsf{wn}} = \underbrace{0 \cdots 0}_{w-1} d_{i-w+1}^{\mathsf{wn}}$ $(d_{i-w+1}^{\mathsf{wn}} \neq 0)$ because a non-zero bit is next to $(w-1)$ 0s in the wNAF representation. Algorithm 4 executes Case 3'; then HW becomes $\lfloor \frac{n''}{w} \rfloor$. We assume that wHW becomes $\lfloor \frac{n''}{w} \rfloor$ by Algorithm 4 in the case of $a = i$. When $a = i + 1$, there exist w-consecutive-zero bits like $d_{i_0}^{\mathsf{wn}} \cdots d_{i_0-w+1}^{\mathsf{wn}} = 0 \cdots 0$ because $\lfloor \frac{n''}{w} \rfloor - l = i + 1 \geq 2$. We set the most significant bit to i_0, i.e $d_{n''}^{\mathsf{wn}} \cdots d_{i_0+1}^{\mathsf{wn}} d_{i_0}^{\mathsf{wn}} d_{i_0-1}^{\mathsf{wn}} \cdots d_{i_0-w+1}^{\mathsf{wn}} = d_{n''}^{\mathsf{wn}} \cdots d_{i_0+1}^{\mathsf{wn}} \underbrace{0 \cdots 0}_{w}$. Then $d_{n''}^{\mathsf{wn}} \cdots d_{i_0+1}^{\mathsf{wn}}$ is the wNAF representation and so its density of non-zero bits is $\frac{1}{w}$. Algorithm 4 executes Case 1 to $d_{i_0}^{\mathsf{wn}}$ if $i + 1 \geq 3$, or Case 2 to $d_{i_0}^{\mathsf{n}}$ if $i + 1 = 2$; then $a = \lfloor \frac{n}{w} \rfloor - l$ becomes $i - 2 < i$ or $i - 1 < i$, respectively. In both cases, $d_{i_0-w}^{\mathsf{wn}} \cdots d_0^{\mathsf{wn}}$ is still the wNAF representation. Therefore wHW becomes $\lfloor \frac{n''}{w} \rfloor$ from the assumption of induction.

Algorithm 4 executes addition or subtraction only to points $\in T$ in any Case. In fact, $([s]2^{w-1} - d_i^{\mathsf{wn}})P$ in Case 3' is in T because $|[s]2^{w-1} - d_i^{\mathsf{wn}}| < 2^{w-1}$ and d_i^{wn} is odd. Therefore, Algorithm 4 can work with $T = \{P, 3P, ..., (2^{w-1} - 1)P\}$ of 2^{w-2} points. ∎

Corollary 2 The computation amount of Algorithm 4 is the minimum among the fixed-hamming-weight signed w-window algorithms with 2^{w-2} precomputed points.

proof: It follows from the same discussion as Corollary 1. In the w-window method, the worst case of d is given as
$(2^{w-1} - 1)\underbrace{0 \cdots 0}_{w-1}(2^{w-1} - 3)0 \cdots 02\underbrace{0 \cdots 0}_{w-1}1.$
∎

5. Comparison

Table 3 compares our methods with previous SPA-countermeasures that do not use any indistinguishable addition formula such as the add-and-double-always algorithm (ADA)[6] or the SPA-resistant wNAF (S-wNAF)[19]. Our methods use indistinguishable addition formulae, which work in $12M + 5S$ or $23M + 5S + I$ by using an elliptic-curve-addition formula[28] or our hyperelliptic-curve-addition formula in Section 3, respectively. The previous SPA countermeasures use the best coordinate of the Jacobian or Affine coordinate system[27] that works in $12M + 4S$ or $22M + 5S + I$ of addition and $4M + 6S$ or $22M + 5S + I$ of doubling in the case of elliptic or hyperelliptic curves, respectively. We assume that $S = 0.8M$ and $I = 10M$and the precomputation table is constructed by a simple method of repeating additions to $2P$.

Now let us compare Algorithm 3 with ADA. Algorithm 3 reduces the computation amount by 0.2% with an elliptic curve or 21.3% with a hyperelliptic curve. Let us compare Algorithm 4 $(w = 4)$ with S-wNAF $(w = 3)$. Algorithm 4 increases or reduces the computation amount by 43.0% or 2.8% with the same memory in the case of elliptic curves or hyperelliptic curves, respectively. We see that our novel combination of an indistinguishable addition formula and the fixed-hamming-weight representation gives a great advantage against the previous approaches of fixed procedure methods of ADA and S-wNAF in the case of hyperelliptic curve.

6. Concluding Remarks

In this paper, we have given the hyperelliptic-curve indistinguishable addition formula for the first time. We have also proposed a new countermeasure against SPA that represents any exponent d with the minimum fixed-hamming-weight HW or wHW. The novel combination of our indistinguishable addition formula and our fixed-hamming-weight representation gives an advantage over the previous approaches of fixed procedure methods such as the add-and-double-always method or the SPA-resistant wNAF in the case of hyperelliptic curve.

References

1) K. Araki and T. Satoh "Fermat quotients and the polynomial time discrete log algorithm

Table 3 Comparison ($n = 160$)

Algorithm	work	#Point	ECC($q = 160$)	HECC($q = 80$)
ADA	$(n-1)A + (n-1)D$	1	$2544M$+$1590S$ ($3816M$)	$6996M$+$1272S$ + $318I$ ($11193M$)
Alg. 3	$(\lfloor \frac{n}{2} \rfloor - 1)A + (n-1)D$	1	$2856M$+$1190S$ ($3808M$)	$5474M$+$1190S$+$238I$ ($8806M$)
S-wNAF ($w = 3$)	$(\lfloor \frac{n}{3} \rfloor + 2)A + nD$	4	$1313M$+$1184S$ ($2260M$)	$4752M$+$968S$+ $216I$ ($7687M$)
Alg. 4	$(\lfloor \frac{n}{4} \rfloor + 2)A + nD$	4	$2424M$+$1010S$ ($3232M$)	$4646M$+$1010S$+$202I$ ($7474M$)

for anomalous elliptic curves", *Commentarii Math. Univ. St. Pauli.*, vol. **47** (1998), 81–92.

2) T. Akishita and T. Takagi, "Zero-value Register Attack on Elliptic Curve Cryptosystem", IEICE, Trans. Fundamentals, vol. **E88-A**, No. 1(2005), 132–139.

3) C. Clavier and M. Joye, "Universal exponentiation algorithm - A first step towards provable SPA-resistance –", CHES2001, Lecture Notes in Computer Science, **2162**(2001), Springer-Verlag, 300–308.

4) M. Ciet and M. Joye, "(Virtually) Free randomization technique for elliptic curve cryptography", ICICS2003, Lecture Notes in Computer Science, **2836**(2003), Springer-Verlag, 348–359.

5) H. Cohen, A. Miyaji and T. Ono, "Efficient elliptic curve exponentiation using mixed coordinates", *Advances in Cryptology-Proceedings of ASIACRYPT'98*, Lecture Notes in Computer Science, **1514**(1998), Springer-Verlag, 51-65.

6) J. Coron, "Resistance against differential power analysis for elliptic curve cryptosystem", CHES'99, Lecture Notes in Computer Science, **1717**(1999), Springer-Verlag, 292–302.

7) G. Frey and H. G. Rück, "A remark concerning m-divisibility and the discrete logarithm in the divisor class group of curves", *Mathematics of computation*, **62**(1994), 865-874.

8) "Proposed federal information processing standard for digital signature standard (DSS)", *Federal Register*, **56** No. 169, 30 Aug 1991, 42980–42982.

9) T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms", *IEEE Trans. Inform. Theory*, **IT-31** (1985), 469–472.

10) L. Goubin, "A Refined Power-Analysis Attack on Elliptic Curve Cryptosystems", PKC2003, Lecture Notes in Computer Science, **2567**(2003), Springer-Verlag, 199–210.

11) M. Joye and C. Tymen, "Protections against Differential Analysis for Elliptic Curve Cryptosystem", CHES2001, Lecture Notes in Computer Science, **2162**(2001), Springer-Verlag, 377–390.

12) C. Kocher, "Timing attacks on Implementations of Diffie-Hellman, RSA, DSS, and other system", CRYPTO'96, Lecture Notes in Computer Science, **1109**(1996), Springer-Verlag, 104–113.

13) C. Kocher, J. Jaffe, and B. Jun, "Differential power analysis", Crypto'99, Lecture Notes in Computer Science, **1666**(1999), Springer-Verlag, 388-397.

14) H. Mamiya, A. Miyaji, and H. Morimoto "Efficient Countermeasures against RPA, DPA, and SPA", *CHES 2004*, Lecture Notes in Computer Science, **3156**(2004), Springer-Verlag, 343-356.

15) A. Menezes, T. Okamoto and S. Vanstone, "Reducing elliptic curve logarithms to logarithms in a finite field", *Proceedings of the 22nd Annual ACM Symposium on the Theory of Computing* (1991), 80–89.

16) B. Möller, "Securing Elliptic Curve Point Multiplication against Side-Channel Attacks", ISC2001, Lecture Notes in Computer Science, **2200**(2001), Springer-Verlag, 324–334.

17) B. Möller, "Parallelizable Elliptic Curve Point Multiplication Method with Resistance against Side-Channel Attacks", ISC2002, Lecture Notes in Computer Science, **2433**(2002), Springer-Verlag, 402–413.

18) P. L. Montgomery, "Speeding the Pollard and elliptic curve methods for factorization", *Mathematics of Computation*, **48**(1987), 243-264.

19) K. Okeya and T. Takagi, "The Width-w NAF Method Provides Small Memory and Fast Elliptic Scalar Multiplications Secure against Side Channel Attacks", CT-RSA2003, Lecture Notes in Computer Science, **2612**(2003), Springer-Verlag, 328–342.

20) S. C. Pohlig and M. E. Hellman, "An improved algorithm for computing logarithms over $GF(p)$ and its cryptographic significance", *IEEE Trans. Inf. Theory*, **IT-24** (1978), 106–110.

21) J. Pollard, "Monte Carlo methods for index computation (mod p)", *Mathematics of Computation*, **32** (1978), 918–924.

22) R. Rivest, A. Shamir and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems", *Communications of the ACM*, **21** No. 2 (1978), 120–126.

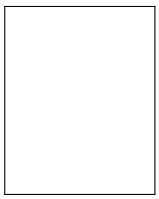
23) J. Ferrante, K. J. Ottenstein, and J. D. Warren, "The Program Dependence Graph and Its Use in Optimization", ACM Trans., Prog., Rang., Syst., Vol.**9**, No.**3** (1987), 319-349.

24) R. Govindaraian, "Instruction scheduling", CRC Press,the compiler design handbook edition, (2003).

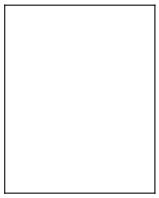
25) K. Itoh, T. Izu, and M. Takenaka, "Efficient

countermeasures against power analysis for elliptic curve cryptosystems", SCIS2004, 2004 (previous version). The final version appears in the proceedings of CARDIS 2004.
26) M. Joye and J. Quisquater, "Hessian elliptic curves and side channel attacks", CHES2001, Lecture Notes in Computer Science **2162**(2001), Springer-Verlag, 402-410.
27) T. Lange, "Formulae for Arithmetic on Genus 2 Hyperelliptic curve", Available at `http://www.ruhr-uni-bochum.de/itsc/tanja/preprints.html`.
28) E. Brier and M. Joye, "Weierstrass Elliptic Curves and Sidechannel Attacks", PKC2002, Lecture Notes in Computer Science **2274**(2002), 335-345.
29) N. P. Smart "The discrete logarithm problem on elliptic curves of trace one", to appear in J. Cryptology.
30) N. P. Smart "An Analysis of Goubin's Refined Power Analysis Attack", CHES2003, Lecture Notes in Computer Science, **2779**(2003), Springer-Verlag, 281–290.
31) I. A. Semaev "Evaluation of discrete logarithms in a group of p-torsion points of an elliptic curve in characteristic p", *Mathematics of computation*, **67** (1998), 353-356.
32) P. Y. Liardet and N. P. Smart, "Preventing SPA/DPA in ECC systems using the Jacobi form", CHES2001, Lecture Notes in Computer Science **2162**(2001), 391-401.
33) C. D. Walter, "Simple power analysis of unified code for ECC double and add", *CHES 2004*, Lecture Notes in Computer Science, **3156**(2004), Springer-Verlag, 281-290.

(Received November 28, 2005)
(Accepted February 4, 2006)



Hideyo Mamiya received a B. Eng degree from the National Defense Academy in 2000 and a M. Info. Sci. degree from Japan Advanced Institute of Science and Technology in 2005.



Atsuko Miyaji received B. Sc., M. Sc., and Dr. Sci. degrees in mathematics from Osaka University, Osaka, Japan in 1988, 1990, and 1997. She worked at Matsushita Electric Industrial Co., LTD from 1990 to 1998, where she was engaged in research and development for secure communication. She has been an associate professor at JAIST(Japan Advanced Institute of Science and Technology) since 1998. She has joined the computer science department at the University of California, Davis since 2002. Her research interests include the application of projective varieties theory into cryptography and information security. She received the IPSJ Sakai Special Researcher Award in 2002, the Standardization Contribution Award in 2003, and Engineering Sciences Society: Certificate of Appreciation in 2005. She is a member of the International Association for Cryptologic Research, the Institute of Electronics, Information and Communication Engineers, the Information Processing Society of Japan, and the Mathematical Society of Japan.