Efficient Countermeasures against RPA, DPA, and SPA*

Hideyo Mamiya, Atsuko Miyaji, and Hiroaki Morimoto**

Japan Advanced Institute of Science and Technology
{hmamiya, miyaji,h-morimo}@jaist.ac.jp

Abstract. In the execution on a smart card, side channel attacks such as simple power analysis (SPA) and the differential power analysis (DPA) have become serious threat [15]. Side channel attacks monitor power consumption and even exploit the leakage information related to power consumption to reveal bits of a secret key d although d is hidden inside a smart card. Almost public key cryptosystems including RSA, DLP-based cryptosystems, and elliptic curve cryptosystems execute an exponentiation algorithm with a secret-key exponent, and they thus suffer from both SPA and DPA. Recently, in the case of elliptic curve cryptosystems, DPA is improved to the Refined Power Analysis (RPA), which exploits a special point with a zero value and reveals a secret key [10]. RPA is further generalized to Zero-value Point Attack (ZPA) [2]. Both RPA and ZPA utilizes a special feature of elliptic curves that happens to have a special point or a register used in addition and doubling formulae with a zero value and that the power consumption of 0 is distinguishable from that of an non-zero element. To make the matters worse, some previous efficient countermeasures are neither resistant against RPA nor ZPA. Although a countermeasure to RPA is proposed, this is not universal countermeasure, gives each different method to each type of elliptic curves, and is still vulnerable against ZPA [30]. The possible countermeasures are ES [3] and the improved version [4]. This paper focuses on countermeasures against RPA, ZPA, DPA and SPA. We show a novel countermeasure resistant against RPA, ZPA, SPA and DPA without any pre-computed table. We also generalize the countermeasure to present more efficient algorithm with a pre-computed table.

Keywords: Elliptic curve exponentiation, ZPA, RPA, DPA, SPA.

1 Introduction

Koblitz [14] and Miller [20] proposed a method by which public key cryptosystems can be constructed on the group of points of an elliptic curve over a finite field. If elliptic curve cryptosystems satisfy both MOV-conditions [19] and

* This work is partially supported by National Institute of Information and Communications Technology (NICT).

** The author is currently with Hitachi Systems & Services, Ltd.

FR-conditions [7], and avoid p-divisible elliptic curves over $\mathbb{F}_{p^r}$ [31, 1, 29], then the only known attacks are the Pollard $\rho-$method [26] and the Pohlig-Hellman method [25]. Hence with current knowledge, we can construct elliptic curve cryptosystems over a smaller definition field than the discrete-logarithm-problem (DLP)-based cryptosystems like the ElGamal cryptosystems [9] or the DSA [8] and RSA cryptosystems [27]. Elliptic curve cryptosystems with a 160-bit key are thus believed to have the same security as both the ElGamal cryptosystems and RSA with a 1,024-bit key. This is why elliptic curve cryptosystems have been attractive in smart card applications, whose memory storage and CPU power is very limited. Elliptic curve cryptosystems execute an exponentiation algorithm of dP for a secret key d and a publicly known P as a cryptographic primitive. Thus, the efficiency of elliptic curve cryptosystems on a smart card depends on the implementation of exponentiation.

In the execution on a smart card, side channel attacks such as the simple power analysis (SPA) and the differential power analysis (DPA) have become serious threat. Side channel attacks, first introduced in [15, 16], monitor power consumption and even exploit the leakage information related to power consumption to reveal bits of a secret key d although d is hidden inside a smart card. Thus, it is a serious issue that the implementation should be resistant against SPA and DPA, and many countermeasures have been proposed in [3, 4, 13, 16, 21, 22, 24]. We may note here that almost public key cryptosystems including RSA and DLP-based cryptosystems also execute an exponentiation algorithm with a secret-key exponent, and, thus, they also suffer from both SPA and DPA in the same way as elliptic curve cryptosystems. However, recently, in the case of elliptic curve cryptosystems, DPA is further improved to the Refined Power Analysis (RPA) by [10], which exploits a special point with a zero value and reveals a secret key. An elliptic curve happens to have a special point $(0, y)$ or $(x, 0)$, which can be controlled by an adversary because the order of basepoint is usually known. RPA utilizes such a feature that the power consumption of 0 is distinguishable from that of an non-zero element. Although elliptic curve cryptosystems are vulnerable to RPA, RPA are not applied to RSA or DLP-based cryptosystems because they don't have such a special zero element. Furthermore, RPA is generalized to Zero-value Point Attack (ZPA) by [2]. ZPA makes use of any zero-value register used in addition formulae. ZPA utilizes a special feature of elliptic curves that addition and doubling formulae need a lot of each different operations stored in auxiliary registers, one of which happens to become 0. To make the matters worse, some previous efficient countermeasures of the randomized-projective-coordinate method (RPC)[6] or the randomized-curve method (RC)[13] are neither resistant against RPA nor ZPA. Because, a special point $(0, y)$ or $(x, 0)$ has still a zero value even if it is converted into $(0, ry, r)$ or $(rx, 0, r)$ by using RPC or RC. A countermeasure to RPA is proposed in [30], but this is not a universal countermeasure, gives each different method to each type of elliptic curves, and is still vulnerable against ZPA. The only possible countermeasure is the exponent-splitting method (ES) in [3, 4], which splits an exponent and computes $dP = rP + (d - r)P = \lfloor d/r \rfloor rP + (d \bmod r)P$ by using a random number r. ES computes dP by the same cost as the add-and-double-always algorithm with an extra point for computation.

This paper focuses on countermeasures against both RPA and ZPA, which are also resistant against both SPA and DPA. Our countermeasure makes use of a random initial point R, computes $dP + R$, subtracts R, and gets dP. By using a random initial point at each execution of exponentiation, any point or any register used in addition formulae changes at each execution. Thus, it is resistant against DPA, RPA, and ZPA. In order to be secure against SPA, we have to compute $dP + R$ in such a way that it does not have any branch instruction dependent on the data being processed. The easiest way would be to compute $dP + R$ in the add-and-double-always method[6]. However, it does not work so straightforwardly if we execute it from MSB as we will see below. Our remarkable idea lies in the computation method of $dP + R$ that uses the binary expansion from MSB and not LSB and is resistant against SPA. The binary expansion from MSB has an advantage over that from LSB in that it is more easily generalized to a sophisticated method with a pre-computed table like the window method[18] or the extended binary method [32]. Let us remark that the computation of $dP + R$ based on the binary expansion from LSB is realized in the straightforward way: change an initial value $\mathcal{O}$ to R in the binary expansion from LSB as follows [12]:

$$dP + R = R + d_0P + d_1 2P + d_2 2(2P) + \cdots + d_{n-1} 2(2^{n-2})P$$

with the binary expansion of $d = (d_{n-1}, \cdots, d_0)_2$. We can easily change the algorithm to the add-and-double-always method. However, the computation of $dP + R$ based on the binary expansion from MSB (see Algorithm 1) is not straightforward: if we change an initial value $\mathcal{O}$ to R in the binary expansion from MSB, then it computes $2^{n-1}R + dP$, and we thus have to subtract $2^{n-1}R$ to get dP. Apparently, it needs more work than the straightforward way of binary expansion from LSB.

In this paper, we first show the basic computation method of $dP + R$ that uses the binary expansion from MSB and is resistant against SPA. This is called BRIP in this paper. Next we apply the extended binary method[32] and present more efficient computation method of $dP + R$ with a pre-computed table, which is still resistant against SPA. This is called EBRIP in this paper. EBRIP is a rather flexible algorithm that can reduce the total computation amount by increasing the size of a pre-computed table. BRIP can get dP in the computation of approximately 24.0 M in each bit without using a pre-computed table, where M shows the computation amount of 1 modular multiplication on the definition field. EBRIP can get dP in the computation of approximately 12.9 M in each bit with using a pre-computed table of 16 points. Compared with the previous RPA-, ZPA-, and SPA-resistant method ES, the computation amount of BRIP is the same as that of ES without an extra point for computation and the computation amount of EBRIP can be reduced to only 54 % of that of ES.

This paper is organized as follows. Section 2 summarizes some facts of elliptic curves like coordinate systems and reviews power analysis of SPA, DPA, RPA, and ZPA together with some known countermeasures. Section 3 presents our new countermeasures, the basic countermeasure (BRIP) and the generalized countermeasure with a pre-computed table (EBRIP). Section 4 presents the performance of our strategy compared with the previous RPA-, ZPA-, and SPA-resistant countermeasure.

2 Preliminary

This section summarizes some facts of elliptic curves like coordinate systems and reviews power analysis of SPA, DPA, RPA, and ZPA together with some known countermeasures.

2.1 Elliptic Curve

Let $\mathbb{F}_p$ be a finite field, where $p > 3$ is a prime. The Weierstrass form of an elliptic curve over $\mathbb{F}_p$ is described as

$$E/\mathbb{F}_p : y^2 = x^3 + ax + b \quad (a, b \in \mathbb{F}_p, 4a^3 + 27b^2 \neq 0).$$

The set of all points $P = (x, y)$ satisfying E, together with the point of infinity $\mathcal{O}$, is denoted by $E(\mathbb{F}_p)$, which forms an abelian group. Let $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ be two points on $E(\mathbb{F}_p)$ and $P_3 = P_1 + P_2 = (x_3, y_3)$ be the sum. Then the addition formulae in affine coordinate are given as follows [5].

•Addition formulae in affine coordinate($P \neq \pm Q$)

$$x_3 = \lambda^2 - x_1 - x_2, \quad y_3 = \lambda(x_1 - x_3) - y_1,$$

where $\lambda = (y_2 - y_1)/(x_2 - x_1)$.

•Doubling formulae in affine coordinate($P = \pm Q$)

$$x_3 = \lambda^2 - 2x_1, \quad y_3 = \lambda(x_1 - x_3) - y_1,$$

where $\lambda = (3x_1^2 + a)/(2y_1)$.

Let us denote the computation time of an addition (resp. a doubling) in the affine coordinate by $t(\mathcal{A}+\mathcal{A})$ (resp. $t(2\mathcal{A})$) and represent multiplication (resp. inverse, resp. squaring) in $\mathbb{F}_p$ by M (resp. I, resp. S). Then we see that $t(\mathcal{A} + \mathcal{A}) = I + 2M + S$ and $t(2\mathcal{A}) = I + 2M + 2S$. Both addition and doubling formulae need one inversion over $\mathbb{F}_p$, which is much more expensive than multiplication over $\mathbb{F}_p$. Therefore, we transform affine coordinate(x, y) into other coordinates, where the inversion is free. We give the addition and doubling formulae with Jacobian coordinate, which are widely used.

In the Jacobian coordinates [5], we set $x = X/Z^2$ and $y = Y/Z^3$, giving the equation

$$E_{\mathcal{J}} : Y^2 = X^3 + aXZ^4 + bZ^6.$$

Then, two points (X, Y, Z) and (r^2X, r^3Y, rZ) for some $r \in \mathbb{F}_p^*$ are recognized as the same point. The point at infinity is represented with $(1, 1, 0)$. Let $P_1 =$

$(X_1, Y_1, Z_1), P_2 = (X_2, Y_2, Z_2)$, and $P_3 = P_1 + P_2 = (X_3, Y_3, Z_3)$. The doubling and addition formulae can be represented as follows.

•Addition formulae in Jacobian coordinate($P \neq \pm Q$)

$X_3 = -H^3 - 2U_1H^2 + R^2$,
$Y_3 = -S_1H^3 + R(U_1H^2 - X_3)$,
$Z_3 = Z_1Z_2H$,

where $U_1 = X_1Z_2^2$, $U_2 = X_2Z_1^2$, $S_1 = Y_1Z_2^3$, $S_2 = Y_2Z_1^3$, $H = U_2 - U_1$, and $R = S_2 - S_1$.

•Doubling formulae in Jacobian coordinate($P = \pm Q$)

$$X_3 = T,\ Y_3 = -8Y_1^4 + M(S - T),\ Z_3 = 2Y_1Z_1,$$

where $S = 4X_1Y_1^2$, $M = 3X_1^2 + aZ_1^4$, and $T = -2S + M^2$.

The computation times in the Jacobian coordinate are $t(\mathcal{J} + \mathcal{J}) = 12M + 4S$ and $t(2\mathcal{J}) = 4M + 6S$, where $\mathcal{J}$ means Jacobian coordinates.

Elliptic curve cryptosystems often execute the elliptic curve exponentiation of $dP = P + P + \cdots + P$, where $P \in E(\mathbb{F}_p)$ and d is an n-bit integer. The simple method to compute dP is a so-called binary algorithm. Algorithm 1 shows the binary algorithm to compute dP from MSB, where the binary expansion of d is $d = (d_{n-1}, \cdots, d_0)$. Average computing complexity of Algorithm 1 is $nD + n/2A$, where A and D denotes the computation amount of addition and doubling, respectively. When we compute dP from LSB, we have to keep another point 2^iP instead of $T_1 = P$ but can apply the iterated doubling formulae in Jacobian coorinate [11], which computes 2^kP for $k \geq 1$ by $4kM + (4k+2)S$. However, the binary algorithm from LSB is not easily generalized to a sophiticated method with a pre-computed table.

Algorithm 1 (Binary algorithm (MSB))
```
Input: d, P
Output: dP
1. T_0 = ′, T_1 = P.
2. for i = n - 2 to 0
        T_0 = 2T_0
        if d_i = 1 then T_0 = T_0 + T_1
3. output T_0.
```

2.2 Power analysis

There are two types of power analysis, the simple power analysis (SPA) and the differential power analysis (DPA), which are described in [15, 16]. In the case of elliptic curve and also hyper elliptic curve, DPA is further improved to use a special point with a zero value, which is called the Refined Power Analysis (RPA) [10]. RPA is generalized to the Zero-value Point Analysis (ZPA) [2]. In this paper, DPA, RPA, and ZPA are called DPA variants generically.

Simple Power Analysis SPA makes use of such an instruction performed during an exponentiation algorithm that depends on the data being processed. Apparently, Algorithm 1 has a branch instruction conditioned by a secret exponent d, and thus it reveals the secret d. In order to be resistant against SPA, any branch instruction of exponentiation algorithm should be eliminated. There are mainly two types of 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 like add-and-double-always method [6], Montgomery-ladder method [23], and window-based method [18]. Add-and-double-always method is described in Algorithm 2. The indistinguishable method conceals all branch instructions of exponentiation algorithm by using indistinguishable addition and doubling operations, in which dummy operations are inserted.

Algorithm 2 (Add-and-double-always algorithm)
`Input:` d, P
`Output:` dP
`1.` $T_0 = P$ `and` $T_2 = P$`.`
`2. for` $i = n - 2$ `to` 0
    $T_0 = 2T_0$`.` $T_1 = T_0 + T_2$`.`
    `if` $d_i = 0$ `then` $T_0 = T_0$`.`
    `else` $T_0 = T_1$`.`
`3. output` T_0`.`

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 exponentiation. There are mainly 3 types of countermeasures, the randomized-projective-coordinate method (RPC) [6], the randomized curve method (RC)[13], and the exponent splitting (ES) [3, 4]. 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, all these two methods are vulnerable against RPA and ZPA, which will be described in Section 2.2. The only method secure against RPA and ZPA is ES, which splits an exponent and computes $dP = rP + (d - r)P$ for a randomly integer r.

Refined Power Analysis and Zero-value Point Attack DPA is specialized to reveal a secret key d by using a special elliptic-curve point with a zero value, which is defined as $(x, 0)$ or $(0, y)$. These special points of $(x, 0)$ and $(0, y)$ still have a zero value like $(rx, 0, r)$ and $(0, ry, r)$ even if it is converted into the projective coordinate, respectively. This is why special points can not be randomized by RPC or RC, and an adversary can thus make use of a zero value in the execution of exponentiation. A countermeasure to RPA are proposed in [30], but this is not a universal countermeasure, gives each different method to each type of elliptic curves, and is still vulnerable against ZPA, described below.

RPA is generalized to ZPA by [2], which makes use of any zero-value register in addition formulae, which is not randomized by RPC or RC. The addition and doubling formulae have a lot of each different operations stored in auxiliary registers, one of which may become zero. ZPA uses the difference in any zero value register between addition and doubling.

We may note that ES can resist both RPA and ZPA because an attacker cannot handle an elliptic curve point in such a way that any special point with zero-value value can appear during an execution of exponentiation algorithm.

3 Efficient countermeasures against SPA and DPA variants

In this section, we propose a new countermeasure against all DPA variants.

3.1 Our basic countermeasure

Here we show our basic countermeasure, called BRIP. Our method uses a random initial point (RIP) R, computes $dP + R$, and subtracts R to get dP. By using a random initial point at each execution of exponentiation, any point or any register used in addition formulae changes at each execution. Thus, it is resistant against DPA, RPA, and ZPA. In order to be secure against SPA, we have to compute $dP + R$ in such a way that it does not have any branch instruction dependent on the data being processed. Our remarkable idea lies in a sophisticated combination to compute $dP + R$ from MSB by the same complexity as Algorithm 2: first let 1 express $1 = (1\overline{1}\overline{1}\cdots\overline{1}\overline{1})_2$ and apply the extended binary method [17] to compute

$$(1\underbrace{\overline{1}\overline{1}\cdots\overline{1}\overline{1}}_{n})_2 R + (\underbrace{d_{n-1}d_{n-1}\cdots d_1 d_0}_{n})_2 P.$$

Algorithm 3 shows our idea in detail. We get dP by computing $dP + R$ and subtracting R. BRIP makes all variables T_0, T_1, and T_2 dependent on a random point R, and thus let all variables of each addition and doubling differ at each execution.

Algorithm 3 (Binary Expansion with RIP (BRIP))

`Input:` d, P

`Output:` dP

`1.` $R =$`randompoint()`

`2.` $T_0 = R, T_1 = -R, T_2 = P - R$

`3. for` $i = n - 1$ `to` 0

$\quad T_0 = 2T_0$

```
    if $d_i = 0$ then $T_0 = T_0 + T_1$
    else $T_0 = T_0 + T_2$
4. output $T_0 + T_1$
```

We discuss the security, the computation amount, and the memory amount. BRIP lets the power-consumption pattern be fixed regardless of the bit pattern of a secret key d, and thus it is resistant against SPA. The resistance against DPA depends on the method of generating a random initial point R. The simplest way to generate R is to generate the x-coordinate randomly and compute the corresponding y-coordinate if exists. It should require much work. The cheaper way is to keep one point R_0 and convert R_0 into a randomized point R by RPC [12]. If R is chosen randomly by some ways mentioned above, BRIP can be resistant against DPA, RPA, and ZPA, since any special point or zero-value register can not appear during each execution. The computation amount required for Algorithm 3 is $nD + nA$, which is the same as Algorithm 2. The number of variables necessary for computation is only 3.

3.2 Our generalized countermeasure

Our basic countermeasure BRIP can be generalized to a faster method with a pre-computed table since BRIP makes use of the binary expansion from MSB. We may note that the binary expansion from LSB can not be easily generalized to a faster method with a pre-computed table.

As for methods of using a pre-computed table, there are mainly two methods: the window method [18] and the extended binary method [17, 32]. The extended binary method is originally used to compute two exponentiations $aP+bQ$, which is applied to compute one exponentiation as follows [32]. Let $d = \sum_{i=0}^{n-1} d_i 2^i$ and n be even.

1. Divide d into two components of $d = b \parallel a$, where $b = (d_{n-1} \cdots d_{\frac{n}{2}})_2$ and $a = (d_{\frac{n}{2}-1} \cdots d_0)_2$.
2. Compute $Q = 2^{\frac{n}{2}} P$.
3. Set a pre-computed table $\{P, Q, P+Q\}$.
4. Compute $aP+bQ$ in the extended binary method by using the pre-computed table.

The detailed algorithm is shown in Algorithm 4.

Algorithm 4 (Extended-binary algorithm with 2 divisions)

```
Input: $d, P$
Output: $dP$
1. Set $d = b \parallel a$, $b = (b_{\frac{n}{2}-1} \cdots b_0)_2 = (d_{n-1} \cdots d_{\frac{n}{2}})_2$, and $a = (a_{\frac{n}{2}-1} \cdots a_0)_2 = (d_{\frac{n}{2}-1} \cdots d_0)_2$.
2. $T_1 = P$, $T_2 = 2^{\frac{n}{2}} P$, $T_3 = T_1 + T_2$, and $T_0 = \mathcal{O}$.
3. for $i = \frac{n}{2} - 1$ to 0
        $T_0 = 2T_0$.
```

```
        if (a_i, b_i) = (1,0) then T_0 = T_0 + T_1.
        elseif (a_i, b_i) = (0,1) then T_0 = T_0 + T_2.
        elseif (a_i, b_i) = (1,1) then T_0 = T_0 + T_3.
4. output T_0.
```

Going back to the countermeasure using a pre-computed table, it is necessary for both the extended binary and window methods to make power-consumption pattern same in order to be resistant against SPA. In the case of the window method, some SPA-resistant methods are proposed in [21, 22, 24]. However, all of these are not resistant against RPA or ZPA even if they are combined with the methods of RC and RPC. In the case of the extended binary method, up to the present, any SPA-resistant method has not been proposed.

Our generalized method is both SPA and DPA-variant resistant, which is able to reduce the computation amount with a pre-computed table. Our sophisticated idea lies in the length of representation of $1 = (1\overline{1}\overline{1}\cdots\overline{1}\overline{1})_2$, which is adjusted to be applied on any bit length of d and output the same executed pattern, while holding down the additional compuation and memory amount. As a result, our method is SPA-resistant naturally. In the following, two algorithms based on the extended-binary and the window methods are described. The extended-binary-based method is more efficient than window-based method although extended-binary method usually does not work on a single exponentiation as efficient as the window method.

Our extended-binary-based method with RIP Let us show our extended-binary-based method with RIP, which is called EBRIP for short.

1. Choose a random point R.
2. Let the number of divisions be t.
3. Adjust n to be the least common multiple n' of t and n by setting 0 to MSB of d ($n' < n + t$), where
$$d' = 0 \cdots 0\, d_{n-1} \cdots d_0.$$
4. Divide d' into t components ($\frac{n'}{t} = k$) of $d' = \alpha_{t-1} \parallel \cdots \parallel \alpha_1 \parallel \alpha_0$, that is,
$$\begin{array}{lcll} \alpha_{t-1} & = & 0 & \cdots\ d_{(t-1)k} \\ & \vdots & & \\ \alpha_1 & = & d_{2k-1} & \cdots\ d_k \\ \alpha_0 & = & d_{k-1} & \cdots\ d_0 \\ 1 & = & 1\,\overline{1} & \cdots\ \overline{1} \end{array}$$
5. Compute $P_i = 2^{ki}P$ for $i = 1$ to $t - 1$. (Set $P_0 = P$).
6. Compute a pre-computed table $T_t = \{\Sigma_{i=0}^{t-1} a_i P_i - R\ (a_i \in \{0, 1\})\}$, which consists of 2^t points.
7. Compute $\alpha_0 P_0 + \cdots + \alpha_{t-1} P_{t-1} + 1R$ in the way of the extended binary method.

Algorithm 5 shows the case of $t = 2$ and an even n for simplicity. Let us discuss the resistance, computation amount, and memory amount. As for SPA, the power-consumption pattern is not changed for any initial point R and any secret key d thanks to the expansion of 1, and EBRIP is thus secure against SPA. We may note one remarkable point that the length of expansion of 1 is not fixed to n but adjusted to $\lceil \frac{n}{t} \rceil + 1 (< n)$. As a result, it realizes more efficient computation than the window-based method. Moreover, under the assumption that an initial point R is completely random, EBRIP is secure against DPA, RPA, and ZPA, as we mentioned in Section 3.1. As for the computation amount, EBRIP consists of these parts: compute base points $P_1, \cdots, P_{t-1}$, a pre-computed table T_t, and the main routine. The computation amount for base points, T_t, or main routine is $\frac{(t-1)n'}{t} D$, $2^t A$, or $\frac{n'}{t} D + \frac{n'}{t} A$, respectively. Thus, the total computation amount is $n'D + \frac{n'}{t} A + 2^t A$. On the other hand, the number of points in T_t is 2^t, which includes a random point R. EBRIP needs one more point of variable to execute. Thus, the necessary memory is $2^t + 1$ in total.

Algorithm 5 (EBRIP (2 divisions))
`Input:` d, P
`Output:` dP
1. $R =$`randompoint()`.
1. `Set` $d = b \parallel a$, $b = (b_{\frac{n}{2}-1} \cdots b_0)_2 = (d_{n-1} \cdots d_{\frac{n}{2}})_2$, `and`
   $a = (a_{\frac{n}{2}-1} \cdots a_0)_2 = (d_{\frac{n}{2}-1} \cdots d_0)_2$.
2. $T_4 = P$, $T_3 = 2^{\frac{n}{2}} P$, $T_0 = R$, $T_1 = -R$, $T_2 = T_1 + T_4$, $T_3 = T_1 + T_3$ `and`
   $T_4 = T_3 + T_4$.
3. `for` $i = \frac{n}{2} - 1$ `to` 0
   - $T_0 = 2T_0$.
   - `if` $(a_i, b_i) = (0,0)$ `then` $T_0 = T_0 + T_1$.
   - `elseif` $(a_i, b_i) = (1,0)$ `then` $T_0 = T_0 + T_2$.
   - `elseif` $(a_i, b_i) = (0,1)$ `then` $T_0 = T_0 + T_3$.
   - `else then` $T_0 = T_0 + T_4$.
4. `output` $T_0 + T_1$.

Our window-based method with RIP Our window-based method with RIP is summarized as follow, which is is called WBRIP for short.

1. Choose a random point R.
2. Set the width of window to be w.
3. Adjust n to be the least common multiple n' of w and n by setting 0 to MSB ($n' < n + w$) of d, where
$$d' = 0 \cdots 0\, d_{n-1} \cdots d_0.$$
4. Compute $R' = -(2^w - 1)R$.
5. Set a pre-computed table $T_w = \{R', P + R', 2P + R', 3P + R', \cdots, (2^w - 2)P + R', (2^w - 1)P + R'\}$, where the number of points in T_w is 2^w.

6. Compute $(\underbrace{0 \cdots 0\, d_{n-1} \cdots d_0}_{n'})_2 P + (1\,\underbrace{\overline{1}1 \cdots \overline{1}1}_{n'})_2 R$ in the way of window method by using T_w.

Let us discuss the security, the computation amount, and the memory amount. Power-consumption pattern is not changed for any random R and any secret key d thanks to the expansion of 1. This is why WBRIP is resistant against SPA. This means that WBRIP is secure against SPA without any additional modification on the window method seen in [21, 22, 24]. Furthermore, under the assumption that an initial point R is completely random, our method is resistant against DPA, RPA, and ZPA. Next we investigate the computation amount of WBRIP. WBRIP consists of three parts: compute an intermediate point R', a pre-computed table T_w, and main routine. The computation amount of R', a pre-computed table T_w, or main routine is $wD + A$, $(2^w - 1)A$, or $\frac{n'}{w}A + n'D$, respectively. Therefore, the total computation amount is $n'D + \frac{n'}{w}A + 2^w A + wD$, where $n' < n + w$. It is not as efficient as the extended-binary-based method since the length of expansion of 1 is fixed to n' to reduce the number of points in T_w, which is the same as that in T_t for $t = w$. If we change the length of expansion of 1 to a shorter length like n, then T_w must include other points and thus the size of T_w becomes larger. Finally we discuss the memory amount necessary to execute WBRIP. The number of points in T_w is 2^w, which includes a random point R. Additional one variable is necessary for computation. Thus, the necessary memory is $2^w + 1$ points in total.

As a result, compared with EBRIP, WBRIP needs more computation amount with the same memory amount.

4 Performance

From the point of view of computation and memory amount, we compare our countermeasures BRIP and EBRIP with the previous method ES [4], which are resistant against SPA and DPA variants. The previous SPA-resistant window methods[21, 22, 24] are not resistant against RPA or ZPA even if they are combined combined with RC or RPC as we mentioned before. Thus, these SPA-resistant window methods have to be combined with ES to be resistant both RPA and ZPA. As a result, the computation and memory amount would be less efficient than WBRIP, which is not so efficient as EBRIP. Table 1 shows the comparison, where M or S shows the computation amount of modular multiplication or modular square on the definition field, respectively. We assume that $S = 0.8M$ as usual. In all cases of BRIP, EBRIP, and ES, the Jacobian coordinate is the most efficient, and thus we use the Jacobian coordinate to compute the total number of modular multiplications. Table 1 shows two types of computation amount. One gives the computation amount in the case of 160-bit definition field. The other gives the average computation amount in each bit, which does not depend on the size of definition field. In order to discuss the efficiency generally, the average computation amount in each bit is useful.

We note that EBRIP can fully make use of the technique of m-repeated elliptic curve doublings [11] although it computes from MSB. Because pre-computation of base points requires m-repeated elliptic curve doublings.

BRIP can compute dP in the computation amount of $160D + 160A$ with 3 points. The computation amount in each bit is $24.0M$, which is the same as that of ES. EBRIP with $t = 2$ can compute dP in the computation amount of $160D + 84A$ with 5 points. The computation amount in each bit is $16.0M$, which is reduced to only 66% of ES. EBRIP with $t = 4$ can execute dP in the computation amount of $160D+56A$ with 17 points. In this case, the computation amount in each bit is $12.9M$, which is reduced to only 54% of ES. Note that $t = 4$ is the fastest when the size of definition filed $\mathbb{F}_p$ is 160.

Table 1. Comparison of countermeasures

	memory amount	computation amount†		computation amount
	(#points, #scalar)	#D + #A	#M + #S	in each bit
ES [4]	(4, 2)	160D+160A	$2856M + 1600S(3840M)$	$16M + 10S(24.0M)$
BRIP	(3, 0)	160D+160A	$2856M + 1600S(3840M)$	$16M + 10S(24.0M)$
EBRIP($t = 2$)	(5, 0)	160D+84A	$1648M + 1140S(2560M)$	$10.3M + 7.1S(16.0M)$
EBRIP($t = 3$)	(9, 0)	162D+62A	$1392M + 1008S(2198M)$	$8.7M + 6.3S(13.7M)$
EBRIP($t = 4$)	(17, 0)	160D+56A	$1312M + 948S(2069M)$	$8.2M + 5.9S(12.9M)$

† This shows the computation amount in the case of 160-bit definition field.

5 Concluding Remarks

In this paper, we have presented countermeasures of BRIP and EBRIP that are resistant against RPA, ZPA, DPA, and SPA. Our countermeasure BRIP does not require any pre-computed table and can get dP in the computation of approximately 24.0 M in each bit. EBRIP with $t = 4$ can get dP in the computation of approximately 12.9 M in each bit with using a pre-computed table and one more point of 17 points in total. Both RPA and ZPA are easily applied to the hyper elliptic curve cryptosystems because a divisor in a hyper elliptic curve consists of more than two parts, some of which would happen to become 0. Our countermeasure improves the addition-chain itself and not use a specific feature of an elliptic curve such as a coordinate system. Therefore, BRIP and EBRIP can also be generalized to deal with hyper elliptic curve cryptosystem. We will describe BRIP and EBRIP on a hyper elliptic curves and discuss the efficiency in our final paper.

References

1. K. Araki and T. Satoh "Fermat quotients and the polynomial time discrete log algorithm for anomalous elliptic curves", *Commentarii Math. Univ. St. Pauli.*, vol. **47** (1998), 81–92.
2. T. Akishita and T. Takagi, "Zero-value Point Attacks on Elliptic Curve Cryptosystem", ISC2003, Lecture Notes in Computer Science, **2851**(2003), Springer-Verlag, 218–233.
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. K. Itoh, M. Takenaka, N. Torii, S. Temma, and Y. Kurihara, "Fast implementation of public-key cryptography on DSP TMS320C6201", CHES'99, Lecture Notes in Computer Science, **1717**(1999), Springer-Verlag, 61–72.
12. K. Itoh, T. Izu, and M. Takenaka, "Efficient countermeasures against power analysis for elliptic curve cryptosystems", SCIS2004, 2004 (previous version). The final version will be appeared in the proceedings of CARDIS 2004.
13. 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.
14. N. Koblitz, "Elliptic curve cryptosystems", *Mathematics of Computation*, **48** (1987), 203–209.
15. 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.
16. C. Kocher, J. Jaffe, and B. Jun, "Differential power analysis", Crypto'99, Lecture Notes in Computer Science, **1666**(1999), Springer-Verlag, 388-397.
17. D. E. Knuth, *The Art of Computer Programming*, Vol. 2: Seminumerical Algorithms, 2nd ed., Addison-Wesley, 1981.
18. K. Koyama and Y. Tsuruoka, "Speeding up elliptic cryptosystems by using a signed binary window method", *Advances in Cryptology-Proceedings of Crypto'92*, Lecture Notes in Computer Science, **740** (1993), Springer-Verlag, 345–357.

19. 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.
20. V. S. Miller, “Use of elliptic curves in cryptography”, *Advances in Cryptology-Proceedings of Crypto'85*, Lecture Notes in Computer Science, **218** (1986), Springer-Verlag, 417–426.
21. B. Möller, ”Securing Elliptic Curve Point Multiplication against Side-Channel Attacks”, ISC2001, Lecture Notes in Computer Science, **2200**(2001), Springer-Verlag, 324–334.
22. 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.
23. P. L. Montgomery, ”Speeding the Pollard and elliptic curve methods for factorization”, *Mathematics of Computation*, **48**(1987), 243-264.
24. 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.
25. 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.
26. J. Pollard, “Monte Carlo methods for index computation (mod p)”, *Mathematics of Computation*, **32** (1978), 918–924.
27. 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.
28. Roberto M. Avanzi, ”On multi-exponentiation in cryptography”, Cryptology ePrint Archive, Report 2002/154, http://eprint.iacr.org/2002/154/ , 2002.
29. N. P. Smart “The discrete logarithm problem on elliptic curves of trace one”, to appear in J. Cryptology.
30. N. P. Smart “An Analysys 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. J. A. Solinas, ”Low-Weight Binary Representation for Pairs of Integers”, Centre for Applied Cryptographic Research, University of Waterloo, Combinatorics and Optimization Reseach Report CORR 2001-41, 2001.