An Efficient Window-Based Countermeasure to Power Analysis of ECC Algorithms

Fan Zhang and Zhijie Jerry Shi
Computer Science and Engineering Department
University of Connecticut, Storrs, CT 06269, USA

Abstract

Elliptic curve cryptography (ECC) has been adopted in many systems because it requires shorter keys than traditional public-key algorithms in primary fields. However, power analysis attacks can exploit the power consumptions of ECC devices to retrieve secret keys. In this paper, we propose an efficient window-based countermeasure that is secure against existing power analysis attacks. Compared to previously proposed countermeasures, our method has low memory overhead, requiring only a table of w+1 entries for a window size of w bits. It also has better performance than many algorithms that perform one point addition or subtraction for each bit in the scalar.

1. Introduction

Since Miller and Koblitz proposed the elliptic curve cryptography (ECC) in the middle 1980s, ECC has attracted a lot of attention because compared to traditional public-key algorithms working in a prime field, ECC algorithms provide equivalent security strengths with shorter keys. Smaller key sizes have many advantages including higher performance, smaller storage space, and lower energy consumption. These advantages are especially important to resource-constrained systems.

Many implementations of ECC algorithms have been found vulnerable to power analysis attacks. Power analysis is one type of side channel attacks that exploit the power consumptions of cryptographic devices to reveal the secret data used in cryptographic computations [3]. There are two main types of power analyses: Simple Power Analysis (SPA) [3] and Differential Power Analysis (DPA) [3]. SPA exploits the correlation between the power outputs and the operations. By observing power traces of cryptographic computations, adversaries can learn what operations are performed and then figure out part or all of the secret data. DPA exploits the relationship between power consumptions and values generated during computation. Adversaries collect a set of power traces and use statistical methods to check whether a specific value is generated during cryptographic computations. They can then deduce the secrets by observing how input data affect the watched value. Both of SPA and DPA can be utilized to attack ECC implementations.

In this paper, we proposed an efficient window-based countermeasure. To defeat SPA, we propose an extended signed binary form to represent all possible values in a window with the same Hamming weight. We adopt the random initial point and randomized projective coordinate techniques to defeat DPA attacks. The way we utilize the random initial point requires only one additional subtraction to compute the correct output. Our countermeasure is also secure against other attacks. Moreover, it requires a small amount of memory, e.g., much less than the methods proposed in [14][15], and has a better performance than the methods proposed in [9] [12].

The rest of the paper is organized as follows. Section 2 describes the basic operations in ECC and reviews existing power analysis attacks on ECC and previous countermeasures. Then we present our countermeasure in Section 3 and discuss its security properties in Section 4. We conclude the paper in Section 5.

2. Past work

This section first gives a brief description of ECC operations and then discusses previous power analysis attacks on ECC and some existing countermeasures.

2.1. Elliptic Curve Cryptography

An elliptic curve can be defined over either a prime field or a binary field. Both have similar properties in terms of power analysis. Here we use elliptic curves

over a binary field to illustrate main operations in ECC. An elliptic curve over a binary field is defined by a bivariate cubic equation:

$$y^2 + xy = x^3 + ax^2 + b. \quad (1)$$

where a and b are constant, x, and $y \in \mathrm{GF}(2^n)$.

The points on an elliptic curve, together with a special point O called the point at infinity and an addition operation, form an abelian group [5]. If a point $P = (x_1, y_1) \neq O$, $-P = (x_1, x_1+y_1)$. Given another point $Q = (x_2, y_2) \neq O$ and $Q \neq -P$, the addition $(x_3, y_3) = P + Q$ is defined as:

$$\begin{aligned} x_3 &= \lambda^2 + \lambda + x_1 + x_2 + a. \\ y_3 &= \lambda(x_1 + x_3) + x_3 + y_1. \end{aligned} \quad (2)$$

where $\lambda = \dfrac{y_2 + y_1}{x_2 + x_1}$ if $P \neq Q$, or $\lambda = \dfrac{y_1}{x_1} + x_1$ if $P = Q$.

A subtraction can be performed with a negation followed by an addition, i.e., $P - Q = P + (-Q)$. Since the negation of a point is fast, the performance of subtraction and additions is similar.

A multiplication dP of a scalar d and a point P, referred to as a *scalar multiplication*, is defined as adding d copies of P together.

2.2. Power analysis attacks of ECC and countermeasures

Scalar multiplication is the basic operation in ECC algorithms. It can be performed with the binary algorithm BINALG as shown in Figure 1(a) where $d_{l-1} = 1$ and l is the number of bits in d. In BINALG, point doubling $2Q$ is performed in every iteration while point addition $Q + P$ is performed only when $d_i = 1$. BINALG is vulnerable to SPA [6]. Since point doubling and point addition are performed with different formulae (see Section 2.1), they have different power outputs. Adversaries can monitor the power traces of a device performing BINALG and identify the point addition and doubling operations. A doubling followed by an addition indicates a 1 bit and two consecutive doublings indicate a 0 bit. The adversaries can easily retrieve the value of the secret d.

To make the algorithm SPA-resistant, one can remove the correlations between operations and data values. Coron [6] proposed a countermeasure BINALG′, as shown in Figure 1(b). The algorithm performs the point addition for every bit in d no matter whether it is 0 or 1. BINALG′ is secure against SPA. However, Coron discovered that it is vulnerable to DPA [6]. Oswald and Aigner further pointed out that any similar algorithms are also vulnerable to DPA attacks [4].

BINALG
Input : P and d
Output: $Q = dP$
$Q \leftarrow P$
For i from $l-2$ to 0 do
$Q \leftarrow 2Q$
If $d_i = 1$ then $Q \leftarrow Q+P$
Return Q

(a) Binary algorithm

BINALG′
Input: P and d
Output: $Q[0] = dP$
$Q[0] \leftarrow P$
For i from $l-2$ to 0 do
$Q[0] \leftarrow 2Q[0]$
$Q[1] \leftarrow Q[0]+P$
$Q[0] \leftarrow Q[d_i]$
Return $Q[0]$

(b) Double-and-add algorithm

Figure 1: Scalar multiplication algorithms

Oswald and Aigner proposed two randomized automata (Automata 1 and 2) trying to thwart both SPA and DPA [4]. When performing a scalar multiplication dP, an automaton starts from the initial state and accepts the bits in d from the least significant bit. For every bit, the automaton performs proper operations and advances to the next state. In order to thwart power analysis attacks, random variables are introduced to decide some state transitions. As a result, the automaton travels through different states and performs different operations even if the scalar d remains the same. However both automata are not secure. Many attacks have been designed to attack randomized automata [1][2][7][8][16][17][18]. In [7], Okeya *et al* proposed an attack on Automaton 1. In [8], Han *et al* proposed an attack that can break both automata. In [1][2], an attack based on unique sequences was proposed, which can be extended to all randomized algorithms by exploiting the distribution of power outputs. In [16], Walter showed an efficient attack that requires only tens of traces. In [17][18], a general framework of attack was proposed, which can be applied to all randomized algorithms.

Besides Oswald *et al*'s randomized algorithms, many other countermeasures have also been proposed [6][9][10][11][14][15]. In [6], Coron described three countermeasures that prevent DPA: (1) Randomizing the private scalar d. This method first computes $d' = d + kE$, where k is a random number and E is the number of the points on the elliptic curve. Then $Q = d'P = dP$ is computed. (2) Blinding the point P. This method adds a random point R to P and computes $Q' = d(R+P)$. The final result is computed as $Q = Q' - S$, where $S = dR$. (3) Randomizing projective coordinates. This method represents a point $P = (x,y)$ with projective coordinates (X, Y, Z) and the conversion is determined by a parameter θ, which can be randomly chosen.

In [22], Okeya *et al* discussed potential attacks to the first two methods proposed in [6]. In method 1, although k is randomly chosen, E is fixed for an elliptic curve. Computing the distribution of bits in d' can reveal corresponding bits in d. In method 2, the total number of possible values of R is too small because of the way R is computed. Goubin proposed Refined Power Analysis (RPA) attacks in [11], which can not be prevented by method 3. In RPA attacks, adversaries guess that a value c appears in d and choose special points $P_1 = c^{-1}P_0$, where P_0 is a point that has a coordinate of 0. The 0 coordinate remains as 0 even if the point is represented with randomized projective coordinates. If the guess of c is correct, P_0 will be generated during the scalar multiplication, which can be detected from power traces because of the 0 coordinate. In [10], Akishita *et al* generalized Goubin's work and designed the Zero-Value Point Attack (ZPA) which exploits special points that generate zero values in registers if the special points are used in a doubling or addition operation. Checking whether zero values are stored in registers, adversaries can learn whether a special point is generated during the multiplication and thus confirm whether the guess of c is correct or not. ZPA does not require that special points have a 0 coordinate.

In [9], Mamiya *et al* pointed out that the attacks based on the zero-value points [10][11] can be prevented by forcing a point value check at the beginning of the scalar multiplication. Accordingly, they proposed a countermeasure BRIP that uses a random initial point R to resist DPAs. They add R to P and perform addition and subtraction for every bit in the scalar [6]. In [13], Yen *et al* proposed a chosen plaintext attack to Mamiya's countermeasure for RSA. In [12], Kim *et al* proposed a countermeasure to Yen's attack. The countermeasure requires additional random initial points that are not on the elliptic curve. It also performs an addition or subtraction for every bit.

In [14][15], Okeya *et al* proposed a countermeasure based on width-w non-adjacent form (NAF). In the countermeasure, they utilize the SPA-resistant fractional window method and randomly select the size of the partial window B where $0 < B < 2^{w-1}$. However, since the window size is not large in many implementations, especially those on resource constrained devices, the choices of B are very limited. For large window sizes, their method has large memory overhead because it requires a table of $2^{w-1} + B$ entries. The memory requirements are exponential to the window size w. Furthermore, they focused on SPA attacks in the paper. They mentioned that randomized projective coordinates may be used to thwart DPA. However, as we discussed earlier, a simple adoption of randomized project coordinates may not be enough to defeat DPA attacks.

3. The Proposed Countermeasure

In this Section, we describe a new window-based countermeasure to power analysis attacks. Since randomized scalar multiplication algorithms are vulnerable to power analysis attacks [1][2][7][8][16][17][18], we design a deterministic algorithm that performs the same operations and thus generates the same addition and doubling power traces in all runs. The power traces are also independent of the scalar value, so our algorithm is secure against SPA attacks. Furthermore, we adopt random initial points and randomized projective coordinates to thwart DPA attacks.

In addition to security, we also try to achieve better performance. In a scalar multiplication, a point doubling is always performed for each bit in the scalar. However, the number of additions varies in different algorithms and fewer additions lead to better performance. As we shall see, the number of additions is dependent on the number of non-zero bits in the representation of the scalar. Therefore, before presenting our countermeasure, we first discuss representations of a scalar and identify a representation to be used in the countermeasure.

3.1. Representations of a scalar

The performance of a scalar multiplication algorithm depends on the number of additions performed in the algorithm. For example, the number of additions performed in BINALG is the number of 1's in d while in BINALG′, it is the *total* number of the bits in d. If d is randomly chosen, only half of the bits are 1. Hence, BINALG has better performance than BINALG′.

Since a point subtraction is almost as fast as an addition, it can be used to achieve better performance. In general, we can represent an m-bit scalar d with a singed binary form:

$$d = \sum_{i=0}^{m-1} d_i 2^i$$

where $d_i \in \{-1, 0, 1\}$ and m is the number of bits in d. For simplicity, we still call a digit in this form a bit, although it has three values.

Representing d in a signed binary form, we can perform scalar multiplication similarly as in BINALG. The difference is that a subtraction is performed if $d_i = -1$. As a result, d is computed through a chain of additions, subtractions, and doublings.

Non-Adjacent Form (NAF) is a unique signed digital representation, in which non-zero digits are not next to each other. In this paper, we limit NAF to signed binary representations. In a NAF, two non-zero bits, 1 or –1, are separated by at least one 0. An important property of NAF is that its Hamming weight is minimal among all signed binary representations [20]. On average, half of the bits in binary representations are non-zero while in NAF, only one third of the bits are non-zero [20]. This implies that in a scalar multiplication, the total number of additions and subtractions is minimal if we represent d in NAF.

Table 1 lists the NAF of numbers from 0 to 63. For convenience, we use $\bar{1}$ to indicate –1. In the table, column d gives a decimal value, column NAF is the NAF of the value, and column H is the Hamming weight of the NAF. Notice that the NAF needs one more bit than the binary representation.

Table 1: NAF of numbers from 0 to 63

d	NAF	H	d	NAF	H	d	NAF	H	d	NAF	H
0	0000000	0	16	0010000	1	32	0100000	1	48	$10\bar{1}0000$	2
1	0000001	1	17	0010001	2	33	0100001	2	49	$10\bar{1}0001$	3
2	0000010	1	18	0010010	2	34	0100010	2	50	$10\bar{1}0010$	3
3	$000010\bar{1}$	2	19	$001010\bar{1}$	3	35	$010010\bar{1}$	3	51	$10\bar{1}010\bar{1}$	4
4	0000100	1	20	0010100	2	36	0100100	2	52	$10\bar{1}0100$	3
5	0000101	2	21	0010101	3	37	0100101	3	53	$10\bar{1}0101$	4
6	$00010\bar{1}0$	2	22	$010\bar{1}0\bar{1}0$	3	38	$01010\bar{1}0$	3	54	$100\bar{1}0\bar{1}0$	3
7	$000100\bar{1}$	2	23	$010\bar{1}00\bar{1}$	3	39	$010100\bar{1}$	3	55	$100\bar{1}00\bar{1}$	3
8	0001000	1	24	$010\bar{1}000$	2	40	0101000	2	56	$100\bar{1}000$	2
9	0001001	2	25	$010\bar{1}001$	3	41	0101001	3	57	$100\bar{1}001$	3
10	0001010	2	26	$010\bar{1}010$	3	42	0101010	3	58	$100\bar{1}010$	3
11	$0010\bar{1}0\bar{1}$	3	27	$0100\bar{1}0\bar{1}$	3	43	$10\bar{1}0\bar{1}0\bar{1}$	4	59	$1000\bar{1}0\bar{1}$	3
12	$0010\bar{1}00$	2	28	$0100\bar{1}00$	2	44	$10\bar{1}0\bar{1}00$	3	60	$1000\bar{1}00$	2
13	$0010\bar{1}01$	3	29	$0100\bar{1}01$	3	45	$10\bar{1}0\bar{1}01$	4	61	$1000\bar{1}01$	3
14	$00100\bar{1}0$	2	30	$01000\bar{1}0$	2	46	$10\bar{1}00\bar{1}0$	3	62	$10000\bar{1}0$	2
15	$001000\bar{1}$	2	31	$010000\bar{1}$	2	47	$10\bar{1}000\bar{1}$	3	63	$100000\bar{1}$	2

As we can see from Table 1, each number has a different Hamming weight and thus requires different numbers of additions and subtractions. Adversaries may exploit the information by SPA attacks.

If a scalar has a large number of bits, there is a simple way to prevent SPA attacks. We can perform all doublings first, save the results in a table, and then perform additions or subtractions for each non-zero bit. Power traces of this method have a sequence of doublings followed by a sequence of additions or subtractions. Adversaries know the number of non-zero bits in the representation, but do not know their locations. A problem with the method is the large memory overhead.

To reduce memory overhead, we adopt a window method in which the bits in the binary representation of the large scalar are processed window by window. In each window, we deal with a much smaller value that has only a small number of bits. We use two techniques to prevent SPA attacks. The first one is performing all doublings first. The second one is to perform the same number of additions and subtractions for all possible values in a window. This means we need to represent the values in a window in signed binary forms that have the same Hamming weight.

We first determine the smallest Hamming weight needed to represent all the values in a window. Suppose the window size is w. Then the value of the bits in a window is between 0 and 2^w–1. Let $H_{MAX}(x)$ be the largest Hamming weight of the NAF of integers between 0 and x, i.e.,

$$H_{MAX}(x) = \max\{H(y_{NAF}) \mid 0 \le y \le x\}$$

where $H(y_{NAF})$ is the Hamming weight of y's NAF. Consider a value y, $0 \le y \le 2^w$–1, we represent y in a signed binary form that has a Hamming weight equal to $H_{MAX}(2^w–1)$. Table 2 lists the values of $H_{MAX}(2^w–1)$ for w from 3 to 9.

Table 2: Number of additions/subtractions in a window of size w

W	3	4	5	6	7	8	9
$2^w–1$	7	15	31	63	127	255	511
$n= H_{MAX}(2^w–1)$	2	3	3	4	4	5	5
n/w	0.67	0.75	0.6	0.67	0.57	0.63	0.56

If $H(y_{NAF}) = H_{MAX}(2^w–1)$, we can just use its NAF in the scalar multiplication. If $H(y_{NAF}) < H_{MAX}(2^w–1)$, we construct another signed binary form to represent y. We start from y's NAF and increase the Hamming weight of the representation. The Hamming weight of a signed binary form can be incremented by replacing 01 with $1\bar{1}$ or $0\bar{1}$ with $\bar{1}1$. The replacements can be done multiple times as long as 01 or $0\bar{1}$ can be found.

If no 01 or $0\bar{1}$ can be found, we extend the signed binary form and add an extra bit to one of bit positions. As a result, a bit position may have two bits. The two bits will be denoted as $\{y_i, e\}$, where y_i is the bit value in the normal signed binary form and e is the extra bit. When computing with the extended signed binary form, two additions or subtractions need to be performed at the bit position that has two bits. Note that both bits at the same position are not zero. Otherwise, the extra bit can be removed.

If we need to increase the Hamming weight by one but cannot apply the replacement method, we find a pair of bits y_{i+1} and y_i such that $y_{i+1}y_i = 10$ (or $\bar{1}0$). Then we change the pair of bits from 10 to $0\{1,1\}$ (or from $\bar{1}0$ to $0\{\bar{1}, \bar{1}\}$). An extra bit is added to the position of y_i and the values of y_{i+1} and y_i are also changed. The Hamming weight of y's representation is increased by one.

Now we look at the cases that we need to increase the Hamming weight by two or more. If the Hamming weight of a singed binary form of y is less than $H_{MAX}(2^w-1)$, and 01 or $0\,\bar{1}$ cannot be found, the least significant bit y_0 must be 0. We change y_0 to $\{1, \bar{1}\}$. Doing this, we increase the Hamming weight of the representation by two. If necessary, the Hamming weight can be increased to a larger value with the replacement method.

With the extended signed binary form, all numbers between 0 and 2^w-1 may be represented with a form that has exactly $H_{MAX}(2^w-1)$ non-zero digits. Thus exactly $H_{MAX}(2^w-1)$ additions or subtractions are performed in each window. Table 3 shows that when w = 3, all the values between 0 and 7 can be represented in a form that has a Hamming weight of two. Notice that when d = 0, we change d_0 to $\{1, \bar{1}\}$. With the new representation, all the values in a 3-bit window require two additions and subtractions.

Table 3: Representations of values in a window of 3 bits

d	NAF	H	Representations
0	0000	0	$000\{1, \bar{1}\}$
1	0001	1	$001\,\bar{1}$
2	0010	1	$01\,\bar{1}\,0$
3	$010\,\bar{1}$	2	$010\,\bar{1}$
4	0100	1	$1\,\bar{1}\,00$
5	0101	2	0101
6	$10\,\bar{1}\,0$	2	$10\,\bar{1}\,0$
7	$100\,\bar{1}$	2	$100\,\bar{1}$

3.2. Efficient window-based countermeasure

We now describe our countermeasure. To perform a scalar multiplication dP, we divided the bits d into small groups (i.e. windows) of w bits. The bits in a window have a value between 0 and 2^w-1, which can be represented in an extended signed binary form that has a Hamming weight of $H_{MAX}(2^w-1)$. Thus for each window, we perform w doublings and $H_{MAX}(2^w-1)$ additions and subtractions. Furthermore, we perform all doublings first and then perform additions or subtractions. To thwart DPA attacks, we adopt the random initial point and randomized projective coordinate techniques.

In Figure 2, we illustrate the countermeasure with w = 3. The algorithm first checks whether $P(x, y)$ is a valid point on the elliptic curve. It also checks whether P's coordinate is zero, to thwart RPA attacks proposed in [11]. Then, it randomly chooses a point R as the initial value of Q. R is subtracted from Q at the end of the algorithm to generate the correct result. In addition, our countermeasure adopts randomized projective coordinates, which are decided by a random variable θ. Our countermeasure can adopt any projective coordinates such as homogeneous or Jacobian projective coordinates. At the end of the algorithm, projective coordinates are converted back to affine coordinates.

After representing Q and P with projective coordinates, the algorithm adds leading 0's to d so that the number of bits in d is a multiple of three because w = 3. Then, it processes the bits window by window from the lower end to the higher end. In each window, three doublings are performed to compute $2P$, $4P$ and $8P$ (in Step 10). The results are stored in a table of four entries $(T_0, T_1, T_2, T_3) = (P, 2P, 4P, 8P)$. Depending on the value of the three bits in the current window, two additions or subtractions are performed in Steps 12–19. As shown in Table 3, a number between 0 and 7 can be represented in a signed binary form with an extra bit which has a Hamming weight of two. The power output of a window is always *DDDAA*, where *D* indicates a doubling and *A* an addition. The power output of the algorithm looks like *DDDAADDDAA…DDDAAA*, where the last *A* is the subtraction in Step 24.

Window-Based Countermeasure (w = 3)

Input: d, $P=(x, y)$
Output: $Q=dP$

```
1   If P is not a valid point on the elliptic curve, return P;
2   If x = 0 or y = 0, return P;
3   Randomly select a point R on the curve and R≠P;
4   Q = R;
5   Randomly select θ;
6   Represenet P and Q with projective coordincates determined by θ;
7   Add leading 0's in d so the number of bits in d is a multiple of 3;
8   for (i = 0; i <= k–1; i += 3) {
9     T0 = P;
10    for (j=1; j <=3; j++){ Tj = 2Tj-1};
11    switch (di+2 , di+1, di)2 {
12      case 0: Q = Q + T0, Q = Q – T0, break;
13      case 1: Q = Q + T1, Q = Q – T0, break;
14      case 2: Q = Q + T2, Q = Q – T1, break;
15      case 3: Q = Q + T2, Q = Q – T0, break;
16      case 4: Q = Q + T3, Q = Q – T2, break;
17      case 5: Q = Q + T2, Q = Q + T0, break;
18      case 6: Q = Q + T3, Q = Q – T1, break;
19      case 7: Q = Q + T3, Q = Q – T0; break;
20    }
21    P = T3;
22  }
23  Convert Q back to the affine coordinates;
24  Q = Q – R;
25  return Q;
```

Figure 2: A window-based countermeasure with a windows size of 3

3.3. Performance and memory overhead

When the window size is w, each window requires w doublings and n additions or subtractions, where $n = H_{MAX}(2^w-1)$. On average, each bit needs one doubling and n/w additions or subtractions. In Table 2, the last row gives the n/w values for w between 3 and 9. We can see that n/w does not change much when w increases. As a result, we chose a small window size of three to illustrate our countermeasure. It requires one doubling and 2/3 additions and subtractions per bit. For comparison, BINALG requires one doubling and about 1/2 additions per bit and BINGALG′ requires one doubling and one addition per bit.

The memory overhead of the algorithm is a table of $w + 1$ entries, which is linear to w. When $w = 3$, the table has only four entries. Other window-based algorithms such as [14][15] require a table size that is exponential to w.

4. Security Analysis

Security assumption. We assume that the power traces of additions and subtractions are similar and adversaries cannot tell whether an operation is an addition or subtraction from power traces. The subtraction of two points $Q - P$ can be done in two steps: first compute $-P$ and then perform a point addition $Q + (-P)$. The negative of a point $P = (x_1, y_1) \neq O$ is $-P = (x_1, x_1 + y_1)$ for a curve over a binary field and $-P = (x_1, -y_1)$ for a curve over a prime field. Compared to point addition, the overhead of computing $-P$ is small in both fields. So it is assumed in a lot of prior work such as [4] that the power consumption differences are negligible for addition and subtraction.

SPA attacks. Since our countermeasure performs the same number of doublings and additions/subtractions for all windows, each window generates the same power output sequence. Hence, our countermeasure is secure against SPA attacks.

DPA attacks. To prevent DPA attacks, we use the random initial point technique [9]. Here we use it in a different way. Instead of adding the random point R to P, we assign R to Q in Step 4. Since Q starts from a random point, the adversaries cannot predict the value of Q generated in Steps 12–19.

Our countermeasure uses a 4-entry table to store the results of doublings. In order to defeat the attacks targeting the table, we adopt randomized projective coordinates. So the values stored in the table (T_0, T_1, T_2, T_3) are randomized. The attackers cannot predict the point values stored in the table and thus cannot use DPA to examine which entries are accessed in Steps 12–19.

RPA and ZPA attacks. To prevent RPA attacks that exploit special points that have a zero coordinate [11], we check the value of P at the beginning of the algorithm.

In the ZPA attacks [10], adversaries guess whether a specific scalar c appears in d and feed a special point P to a scalar multiplication algorithm. If the guess is correct, cP is generated during the multiplication and causes that zero values are written into registers in later operations. In our countermeasure, cP is never explicitly computed because of the random initial point assigned to Q. The intermediate value kept in Q is $cP + R$, represented with projective coordinates. Even if the guess of c is correct, $cP + R$ does not cause zero-value registers in later operations.

Other attacks. In [13], Yen proposed a chosen plaintext attack to break the BRIP countermeasure for RSA. They mentioned that the attack may be extended to ECC as well. When applied to ECC, their attack requires a special point P such that $2P = O$. With the special point, an attack can detect the least significant bit d_0 by checking the result Q. However, they cannot detect other bits by checking the intermediate values of Q because the values are either R or $P + R$, none of which is O. So this attack cannot break our countermeasure.

One may try to use timing attack to detect which case is executed in Steps 12–19. However, every case in Steps 12–19 can be reached by a single branch instruction in which the target address of each case is loaded from memory and stored in a register. This is a common way to implement switch statements in the C language. So the execution time of all branches is similar and can not be exploited in timing attacks. The risks can be further reduced by preloading instructions and the table into a cache.

Our countermeasure is also secure against *safe error* attacks [21], which introduce a temporary random fault on the hardware. Part of the secret key can be revealed if the device generates the correct result even when a fault is injected. There are two types of safe errors. One is *memory safe error* which modifies some bits in the storage. The other is *computational safe error*, which is introduced in computational circuits such as ALU. For example, the computational safe error attacks can break BINALG′.

When a temporary computation error is introduced on Line 4 of BINALG′, the error causes the program to generate a wrong final result when d_i is 1. However, when d_i is 0, the wrong value is not propagated to later stages and the final result is correct. By examining the final result, adversaries are able to detect the value of d_i. In our window based countermeasure, however, an error is *always* propagated to the output. So our countermeasure is secure against computational safe error attack. Our countermeasure is vulnerable to memory safe error attacks because it needs a table stored in the memory and not all the entries of the table are used in Steps 12-19. This attack can be prevented by using memory with error correction code. Adversaries cannot tell whether an error is corrected by the error correction code or the faulty entry is not used.

5. Conclusions

In this paper, we propose an efficient window-based countermeasure for ECC. We proposed an extended signed binary form that can represent all possible values in a window with the same Hamming weight. Therefore, each window performs the same number of additions and subtractions, making our countermeasure secure against SPA attacks. We also adopt the random initial point and randomized projective coordinate techniques to defeat DPA attacks. In our countermeasure, both P and Q are randomized and only one additional subtraction is required to compute the correct output. Furthermore, our countermeasure has low memory overhead, requiring a small table size that is linear to the window size. At the same time, our method provides good performance, performing about 2/3 additions and subtractions per bit.

References

[1] Z. J. Shi and F. Zhang, "New attacks on randomized ECC algorithms," *Proceedings of EITC 2006*, August 2006.

[2] F. Zhang and Z. J. Shi, "Power Analysis Attacks on ECC Randomized Automata," *Proceedings of ITNG 2007*, April 2007.

[3] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," *Proceedings of CRYPTO'99*, pp. 388-397, August 1999.

[4] E. Oswald and M. Aigner, "Randomized addition-subtraction chains as a countermeasure against power attacks," *Proceedings of CHES 2001*, pp. 39-50, May 2001.

[5] A. J. Menezes, "Elliptic curve public key cryptosystems," Kluwer Academic Publishers, 1993.

[6] J. S. Coron, "Resistance against differential power analysis for elliptic curve cryptosystems," *Proceedings of CHES 1999*, pp. 292-302, August 1999.

[7] K. Okeya, and K. Sakurai, "On insecurity of the side channel attack countermeasure using addition-subtraction chains under distinguishability between addition and doubling," *Proceedings of 7th Australasian Conference on Information Security and Privacy, ACISP 2002*, pp. 420-435, July 2002.

[8] D. Han, N. S. Chang, S. W. Jung, Y. H. Park, C. H. Kim, and H. Ryu, "Cryptanalysis of the full version randomized addition-subtraction chains", *Proceedings of 8th Australasian Conference on Information Security and Privacy, ACISP 2003*, pp. 67 – 78, July 2003.

[9] H. Mamiya, A. Miyaji, and H. Morimoto, "Efficient countermeasure against RPA, DPA, and SPA," *In Cryptographic Hardware and Embedded Systems CHES '04*, LNCS 3156, pp. 343-356, 2004.

[10] T. Akishita, and T. Takagi, "Zero-value Point Attacks on Elliptic Curve Cryptosystem," *ISC2003*, Lecture Notes in Computer Science, 2851, pp. 218–233, 2003.

[11] L. Goubin, "A Refined Power-Analysis Attack on Elliptic Curve Cryptosystems," *PKC2003*, Lecture Notes in Computer Science, 2567, pp. 199–210, 2003.

[12] C. K. Kim, J. C. Ha, S. J. Moon, S. M. Yen, W. C. Lien, S. H. Kim, "An improved and efficient countermeasure against Power Analysis Attacks," *Cryptology ePrint Archive, Report 2005/022*, January 2005.

[13] S. M. Yen, W. C. Lien, S. J. Moon, and J. C. Ha, "Power Analysis by Exploiting Chosen Message and Internal Collisions – Vulnerability of Checking Mechanism for RSA-Decryption," Mycrypt 2005, pp. 183-195, 2005.

[14] K. Okeya, and T. Takagi, "A More Flexible Countermeasure against Side Channel Attacks Using Window Method," Cryptographic Hardware and Embedded Systems - CHES 2003.

[15] K. Okeya, and T. Takagi, "The Width-w NAF Method Provides Small Memory and Fast Elliptic Scalar Multiplications Secure against Side Channel Attacks," *CT-RSA 2003*.

[16] C. D. Walter, "Issues of Security with the Oswald--Aigner Exponentiation Algorithm," *CT-RSA 2004*, pp. 208-221, 2004.

[17] C. Karlof, and D. Wagner "Hidden Markov Model Cryptanalysis," *CHES 2003*, pp. 17-34, 2003.

[18] P. J. Green, R. Noad and N. P. Smart, "Further Hidden Markov Model Cryptanalysis," *CHES 2005*, pp. 61-74, 2005

[19] D. Hankerson, J. L. Hernandez, and A. Menezes, "Software Implementation of Elliptic Curve Cryptography over Binary Fields", *CHES2000*, Lecture Notes in Computer Science, vol 1965, pp. 1-24, 2000.

[20] G. Reitwiesner, "Binary arithmetic," *Advances in Computers*, pp. 231–308, 1960.

[21] S. M. Yen, and M. Joye, "Checking before output may not be enough against fault based cryptanalysis," *IEEE Trans. on Computers*, vol. 49, no. 9, pp. 967–970, September 2000.

[22] K.Okeya, K.Sakurai, "Power analysis breaks elliptic curve cryptosystems even secure against the timing attack", Indocrypt 2000, LNCS1977, (2000),178–190.