Simple Countermeasure to Cryptanalysis against Unified ECC Codes

Yoo-Jin Baek

Abstract: **As a countermeasure to simple power attack, the unified point addition codes for the elliptic curve cryptosystem were introduced. However, some authors proposed a different kind of power attacks to the codes. This power attack uses the observation that some internal operations in the codes behave differently for addition and doubling. In this paper, we propose a new countermeasure against such an attack. The basic idea of the new countermeasure is that, if one of the input points of the codes is transformed to an equivalent point over the underlying finite field, then the code will behave in the same manner for addition and doubling. The new countermeasure is highly efficient in that it only requires $27(n-1)/3$ extra ordinary integer subtractions (in average) for the whole n-bit scalar multiplication. The timing analysis of the proposed countermeasure is also presented to confirm its SPA resistance.**

Index Terms: **Countermeasure, elliptic curve cryptosystem (ECC), simple power attack (SPA), unified code.16.**

I. INTRODUCTION

While increasingly being used in practice due to its shorter key size and lower processing time, the elliptic curve cryptosystem (ECC) [1], [2] is also known to be vulnerable to various side channel attacks, among which the power attack [3], [4] tries to find out some secret information from the power consumption signals. Several kinds of power attacks against ECC have been introduced so far and various countermeasures have been proposed accordingly. Especially, this paper mainly concerns the unified point addition codes which were introduced as a countermeasure against the simple power attack (SPA) [5], [6] to ECC. The main idea of the unified codes is to implement the ECC point addition and doubling in the same formula. Note that the ordinary ECC addition and doubling have different equation forms, whence they can easily be distinguished by SPA.

However, authors in [7] and [8] proposed a different kind of power attacks to these unified codes. They investigated the property that even though a point operation is performed in a unified way, some of its internal operations behave differently for addition and doubling, which can be analyzed by SPA to reveal (part of) the secret key. The authors also proposed some countermeasures for the attack, but they are not satisfactory in the sense that they require much computational overhead and/or are vulnerable to another kind of power attacks. In this paper, we propose a new countermeasure to these attacks. The basic idea of the new countermeasure is that, if one of input points of the unified codes is manually transformed to an equivalent point over the underlying finite field, then the internal operations of the codes will behave in the same manner for doubling and addition. The big advantage of the new countermeasure lies at its efficiency: It only requires $27(n-1)/3$ extra integer subtractions for an n-bit scalar multiplication, which is negligible compared with the whole effort of the ECC scalar multiplication.

This paper is organized as follows. In Section II, we briefly give an overview of elliptic curves. The basics of power attacks and their countermeasures will also be introduced in the same section. Section III is the place in which we present the new countermeasure and its computational cost. The brief security analysis of the new countermeasure will also be given in the section. The conclusion can be found in Section IV.

Manuscript received June 09, 2008; approved for publication by Sarah Kate Wilson, Division I Editor, February 19, 2009.

The author is with Samsung Electronics, Korea, email: yoojin.baek@samsung.com.

II. PREREQUISITES

A. Elliptic Curves

For a prime p, let F_p denote the finite field with p elements. An elliptic curve E over F_p is the set consisting of points $(x, y) \in F_p \times F_p$ which satisfy the following nonsingular Wierstrass equation

$$y^2 = x^3 + ax + b, \quad a, b \in F_p \tag{1}$$

plus a point at infinity $\mathcal{O}$. It is well known that E forms an Abelian group under a special addition rule given by: $\mathcal{O}$ acts as the identity element and for $P = (x_1, y_1) \neq \mathcal{O}$, $-P$ is defined to be $(x_1, -y_1)$. Also, for $P = (x_1, y_1) \neq \mathcal{O}$ and $Q = (x_2, y_2) \neq \mathcal{O}, -\mathcal{P}$, $P + Q = (x_3, y_3)$ is given by:

$$\begin{aligned} x_3 &= \begin{cases} (\frac{y_2 - y_1}{x_2 - x_1})^2 - x_1 - x_2, & \text{if } P \neq Q \\ (\frac{3x_1^2 + a}{2y_1})^2 - 2x_1, & \text{if } P = Q, \end{cases} \\ y_3 &= \begin{cases} (\frac{y_2 - y_1}{x_2 - x_1})(x_1 - x_3) - y_1, & \text{if } P \neq Q \\ (\frac{3x_1^2 + a}{2y_1})(x_1 - x_3) - y_1, & \text{if } P = Q. \end{cases} \end{aligned} \tag{2}$$

Note that (2) has a different form for addition ($P \neq Q$) and doubling ($P = Q$), which is the main source of the simple power attack vulnerability.

For computational efficiency, elliptic curves can be represented in the projective coordinate. In the projective coordinate, a point (x, y) is converted into the point (X, Y, Z) with $x = X/Z \bmod p, y = Y/Z \bmod p$ and an elliptic curve consists of all points $(X, Y, Z) \in F_p^3$ which satisfy the equation $Y^2Z = X^3 + aXZ^2 + bZ^3$. The explicit addition formula for the projective elliptic curves can be found in [9].

One of the most time-consuming operations in ECC is the multiplication by a scalar which is defined by: For $P \in E$ and a positive integer k, the scalar multiplication kP is the operation

adding k copies of P. Also, for a negative integer n, nP is defined to be $(-n)(-P)$. The following algorithm is usually used for implementing the scalar multiplication:

Algorithm (L-R scalar multiplication)

Input: $P \in E, k = \sum_{i=0}^{n-1} k_i 2^i$ with $k_{n-1} \neq 0$.
Output: $kP \in E$.
1. $Q \leftarrow P$;
2. For $i = n-2$ to 0, do
   (a) $Q \leftarrow 2Q$;
   (b) If $k_i = 1$, then $Q \leftarrow Q + P$;
3. Return Q.

B. Power Attack and Unified ECC Code

The power attack, which was first introduced by Kocher *et al.* in [4], tries to recover some secret information from power consumption signals. The basic reason why the power attack works is that the power consumption of cryptographic devices is strongly related to the internal state of the devices. Hence, by investigating the power consumption profiling, one can get valuable information about the internal operations and the (keying-) parameters in the devices. Various power attack methods have been proposed so far, which include the SPA and the differential power attack (DPA) [3].

SPA tries to distinguish various cryptographic primitives (for example, point addition and doubling in ECC) by observing power signals for a single execution of cryptographic operations. Accordingly, the double-and-add always method [3] and the Montgomery method [10] have been introduced as its countermeasure. On the other hand, DPA collects the power consumption data and uses statistical tools to get some useful information from these data. As its countermeasure, one can use various random blinding techniques in [3].

Especially, the unified point addition codes were introduced as an ECC-countermeasure against SPA [5], [6]. The main idea of the unified codes is to implement the EC point operation with the same formula. More precisely, the unified formula in [6] calculates $P + Q = (x_3, y_3)$ for $P = (x_1, y_1) \neq \mathcal{O}$ and $Q = (x_2, y_2) \neq \mathcal{O}, -\mathcal{P}$ by: If $y_1 + y_2 \neq 0$, then $x_3 = \lambda^2 - x_1 - x_2$ and $y_3 = \lambda(x_1 - x_3) - y_1$ where

$$\lambda = \frac{(x_1 + x_2)^2 - x_1 x_2 + a}{y_1 + y_2}.$$

The projective version of the code can explicitly be given by: With $x_i = X_i/Z_i, y_i = Y_i/Z_i$,

$$X_3 = 2FW,\ Y_3 = R(G - 2W) - L^2,\ Z_3 = 2F^3$$

for $U_1 = X_1 Z_2, U_2 = Z_1 X_2, S_1 = Y_1 Z_2, S_2 = Z_1 Y_2, Z = Z_1 Z_2, T = U_1 + U_2, M = S_1 + S_2, F = ZM, L = MF, G = LT, R = T^2 - U_1 U_2 + aZ^2$, and $W = R^2 - G$, where all the operations are performed modulo p, hence the multiplications means the modular multiplication.

There are many techniques to efficiently implement modular multiplications. Especially, this paper assumes that the Montgomery multiplication method (MMM) [10] is used for implementing the above modular multiplications. However, we emphasize that the basic idea of the paper can also be applied to other multiplier schemes.

MMM, given inputs A, B, p and a constant R with $R > p$ and $(R, p) = 1$, first computes $C = (AB + Qp)/R\ (= ABR^{-1} \bmod p)$ for some $0 \leq Q < R$ and then outputs $C - p$ if $C > p$ or C, otherwise. However, as noted in [7] and [8], the final conditional subtraction $C - p$ in MMM occurs with a biased probability for squaring and multiplication. Also, if $P_1 = P_2$, we have $U_1 = U_2$ and $S_1 = S_2$ in the above code. Hence, if a final subtraction occurs in only one of U_1 (S_1) and U_2 (S_2), then one may conclude that the operation must not be a doubling, which serves as the main point of attack in [8]. A more sophisticated attack to the unified codes of [5] was also proposed in [7], but the new countermeasure can easily adjusted to avoid the attack as well.

III. NEW COUNTERMEASURE

In this section, a new countermeasure to the attacks in [7], [8] is proposed. The idea is very simple: to overcome the SPA vulnerabilities in Section II, we manually change one of input points of the codes to an equivalent point modulo p. Hence, even though the point operation is originally intended for doubling, it will behave like an addition.

However, this idea must be implemented with care not to cause much computational overhead. For example, suppose that we change a point by adding a multiple of p to each coordinate of the point. Clearly, the resulting point is equivalent to the original point modulo p. However, noting that common cryptographic libraries use an array of words (or bytes) to represent a big number, this change inevitably causes an increase of the array size for holding the new point. Moreover, even though the bit size of resulting coordinates is increased by 1, the library must add one word to the array since the field size of commonly recommended elliptic curves is exactly fitting to the multiple of word size [11]. These clearly impose undesirable penalties for efficiency.

On the other hand, if we change a point by subtracting p from each coordinate, the memory overhead mentioned above may be avoided, while the resulting point is clearly equivalent to the original point. Moreover, since usual cryptographic libraries adopt an independent word (or byte) for holding the sign symbol of big numbers, it is not required to increase the array size for holding the new point. We will use this simple observation to develop a new countermeasure.

In addition, noting that the resulting point has all the negative coordinates and the usual Montgomery multiplication algorithm works on two nonnegative inputs, we must slightly modify it to work on one nonnegative and the other negative inputs. However, this task can be accomplished without any additional computational overhead as follows:

Algorithm 1 (new Montgomery multiplication method)

Input: $p = \sum_{i=0}^{n-1} p_i r^i, A = \sum_{i=0}^{n-1} a_i r^i, B = -\sum_{i=0}^{n-1} b_i r^i$, and $R = r^n$ such that $(p, r) = 1, 0 \leq A < p$, and $-p \leq B < 0$.
Output: $C = ABR^{-1} \bmod p$.
1. $Q \leftarrow P$;
2. $C \leftarrow 0$;
3. For $i = 0$ to $n-1$, do
   (a) $q_i \leftarrow (b_i a_0 - c_0) p_0^{-1} \bmod r$ for $c_0 = C \bmod r$;
   (b) $C \leftarrow (C - b_i A + q_i p)/r$;

4. If $C < 0, C \leftarrow C + p$;
5. Return C.

Note that, in Algorithm 1, the first multiplicand A is nonnegative and the second one B is negative, which will be used as a convention in the sequel.

To justify Algorithm 1, we only have to prove that C lies in the interval $(-p, p)$ after Step 2 is completed: After Step 2, C is of the form $C = (AB + Qp)/R$ with $0 \leq Q = \sum_{i=0}^{n-1} q_i r^i < r^n = R$. Since we also have $-Rp < AB + Qp < Rp$, C consequently lies in the interval $(-p, p)$. With this justification, Step 3 is required to confirm that the return value of the algorithm lies in the interval $[0, p)$.

Now, using Algorithm 1, the unified addition formula can explicitly be given by:

Algorithm 2 (unified addition formula)

Input: F_p (a finite field), a (the defining coefficient of E), $P_1 = (X_1, Y_1, Z_1), P_2 = (X_2, Y_2, Z_2) \in E$ with $0 \leq X_1, Y_1, Z_1 < p$ and $-p \leq X_2, Y_2, Z_2 < 0$.
Output: $P_3 = P_1 + P_2 = (X_3, Y_3, Z_3) \in E$.

1. $U_1 \leftarrow X_1 \cdot Z_2, U_2 \leftarrow Z_1 \cdot X_2, S_1 \leftarrow Y_1 \cdot Z_2, S_2 \leftarrow Z_1 \cdot Y_2$;
2. $U_2' \leftarrow U_2 - p$;
3. $Z \leftarrow Z_1 \cdot Z_2, Z' \leftarrow Z - p$;
4. $T \leftarrow U_1 + U_2 \bmod p, M \leftarrow S_1 + S_2 \bmod p$;
5. $T' \leftarrow T - p$;
6. $F \leftarrow M \cdot Z', F' \leftarrow F - p$;
7. $L \leftarrow M \cdot F', L' \leftarrow L - p$;
8. $G \leftarrow L \cdot T', R \leftarrow T \cdot T' - U_1 \cdot U_2' + aZ \cdot Z' \bmod p$;
9. $R' \leftarrow R - p$;
10. $W \leftarrow R \cdot R' - G \bmod p$;
11. $X_3 \leftarrow 2W \cdot F'$;
12. $Y_3 \leftarrow (G - 2W) \cdot R' - L \cdot L' \bmod p$;
13. $Z_3 \leftarrow 2(F \cdot F') \cdot F'$;
14. Return (X_3, Y_3, Z_3).

Note that, in Algorithm 2, the first input point P_1 has all nonnegative coordinates and the second input P_2 has all negative coordinates, which will be served as a convention in the sequel. Also, the multiplications in the algorithm are direct applications of Algorithm 1 and the subtractions in Step 2, 3, 5, 6, 7, 9 are ordinary integer subtractions, not the modular subtractions. With the help of these extra integer subtraction steps, all the modular multiplications in the algorithm have one nonnegative input and the other negative input. So, we can use Algorithm 1 as its subroutine.

Finally, an SPA-resistant scalar multiplication algorithm can be obtained as:

Algorithm 3 (SPA-resistant scalar multiplication algorithm)

Input: $P = (X, Y, Z) \in E, k = \sum_{i=0}^{n-1} k_i 2^i$ with $k_{n-1} = 1$.
Output: $kP \in E$.

1. $Q \leftarrow P$;
2. Multiply R in Algorithm 1 to X, Y, Z and a (the defining coefficient of E);
3. $Q \leftarrow P$;
4. For $i = n - 2$ to 0, do
   (a) $Q' \leftarrow (X' - p, Y' - p, Z' - p)$ for $Q = (X', Y', Z')$;
   (b) $Q \leftarrow Q + Q'$;
   (c) If $k_i = 1$, then do the following:
      i. $Q' \leftarrow (X' - p, Y' - p, Z' - p)$ for $Q = (X', Y', Z')$;
      ii. $Q \leftarrow P + Q'$;
5. Remove R in the coordinates of Q;
6. Return Q.

In Algorithm 3, the point additions of Steps 3.b and 3.c.ii are direct applications of Algorithm 2.

Now, we add some comments about the role of Step 3.a. While Q' is equal to Q over F_p, the two points behave differently as inputs to the point addition in Step 3.b, hence we can avoid the SPA vulnerability in [7], [8]. More precisely, the two inputs of the addition in Step 3.b are $Q = (X', Y', Z')$ and $Q' = (X' - p, Y' - p, Z' - p)$. Hence, U_1, U_2, S_1 and S_2 in Algorithm 2 are calculated as $U_1 \leftarrow X'(Z' - p), U_2 \leftarrow (X' - p)Z', S_1 \leftarrow Y'(Z' - p), S_2 \leftarrow (Y' - p)Z'$, so $U_1 \neq U_2$ and $S_1 \neq S_2$ as an integer. Also, it is probable that the events of occurring a final subtraction in computing $U_1(S_1)$ and $U_2(S_2)$ are independent to each other, which will be evidenced by the timing analysis in the subsequent section.

The computational overhead of Algorithm 3 can be summarized as follows: First, it requires 3 integer subtractions for Step 3.a and 3.c.i and 6 integer subtractions for Step 3.b and Step 3.c.ii. Hence, additional $27(n-1)/2$ ordinary integer subtractions (in average) for the whole scalar multiplication are required. But, the cost of these extra operations is negligible, compared with that of the whole ECC scalar multiplication. More precisely, an n-bit subtraction is known to require $O(n)$ bit operations, while an n-bit multiplication needs $O(n^2)$ bit operations. Hence, the computational overhead of the new countermeasure is expected to be less than $1/n$ for an n bit scalar multiplication and this observation is justified by the following experimental timing measurement: We iteratively performed two kinds of scalar multiplications, one without any countermeasure and the other with the new countermeasure for a randomly chosen elliptic curve over a randomly chosen 192-bit prime field. The experiment platform was a desk-top PC with Intel Pentium 4 CPU 2.40 GHz, 512 MB RAM and Microsoft Windows XP operating system. We emphasize that the scalar multiplications were implemented without any optimization technique, hence the obtained timings is not adequate for the practical use, but they were intended only for comparison. The resulting timings were 471.61 milliseconds for the first one and 475.91 milliseconds for the second one, and the difference was less than 4 milliseconds.

A. Timing Analysis

To investigate the security aspects of the new countermeasure, we performed some timing analysis. For this, we used the same platform as in the previous section and analyzed the timing behaviors of computing U_1 and U_2 in Algorithm 2. For the analysis, we gathered 1,000 timing measurements of the following 4 tasks, where each task is iterated 1000 times for a randomly chosen 192-bit odd integer p and 4 randomly chosen 192-bit nonnegative integers a, b, c, and d (which are less than p). Consequently, each task was executed 1,000,000 times.

1. $a \cdot b \bmod p$ followed by $c \cdot d \bmod p$;
2. $a \cdot b \bmod p$ followed by $a \cdot b \bmod p$;
3. $a \cdot (b - p) \bmod p$ followed by $c \cdot (d - p) \bmod p$;

Table 1. Timing measurement.

	Mean	Variance
Task 1	205.49	34.04
Task 2	205.90	46.75
Task 3	206.35	35.64
Task 4	205.83	33.96

4. $a \cdot (b - p) \bmod p$ followed by $b \cdot (a - p) \bmod p$.

To measure the exact timing, we used the standard C-library function **clock()** which returns the processor time used by a program since its execution. Note that the above four tasks are simulating U_1 and U_2 in a different manner: The first and second ones are intended to simulate U_1 and U_2 of the original unified addition codes in [5] and [6] for general P_1, P_2 and for $P_1 = P_2$, respectively. The third and the forth ones are intended for U_1 and U_2 in Algorithm 2 with general P_1, P_2 and $P_1 = P_2$, respectively.

The detailed simulation result can be found in Table 1. According the table, the variance of Task 2 is bigger than the other ones, while the mean values of each task are almost same. To explain this phenomenon, we let the random variables X and Y represent the timing of the Montgomery modular multiplication and put $E(X) = E(Y) = m$ and $V(X) = V(Y) = \sigma^2$. Then, the timings of the first and the third tasks can be represented by the random variable $X + Y$ and the second one by $2X$. Also, if the two operations in the last task are independent to each other (as assumed in this paper), then the task can be represented by the random variable $X + Y$. Thus, calculating their respective mean and variance values, we can expect that all the mean values be same and the variance of the second task (which is $4\sigma^2$) be bigger than the other ones (which are $2\sigma^2$). The result in Table 1 confirms this expectation. However, we emphasize that since the simulation was performed in the PC environment, it is affected by a lot of noises, for example, the background applications which are performed in parallel with the simulation. Hence, the simulation result is not exactly matching with the expectation.

The second analysis was performed for checking the independency between the sub-tasks of the above tasks. To this, we iteratively gathered 1,000 timing measurements of $a \cdot b \bmod p, c \cdot d \bmod p, a \cdot (b - p) \bmod p, c \cdot (d - p) \bmod p$ and $(a - p) \cdot b \bmod p$ and analyzed the correlation coefficients between them. The initial expectation was that all the correlation coefficients between the subtasks of Task 1, 3, and 4 lie close to 0, and the simulation result confirmed this expectation. The detailed result can be found in Table 2. Note that the correlation coefficient between sub-tasks of Task 2 is obviously 1, since they are same tasks.

Finally, we note that the simulation result is not directly applicable to the timing attack, since we cannot get a partial timing information for U_1 and U_2 computations. However, since it is possible to get the partial timing information from the power consumption signals in the power attack, we may conclude that the second task may be distinguished from the first task, while the last two tasks are not distinguishable from the power consumption information. Hence the proposed countermeasure gives some SPA-resistance, as desired.

Table 2. Analysis of correlation coefficients.

	Correlation coefficient.
Task 1	−0.0812
Task 3	−0.0957
Task 4	−0.0442

IV. CONCLUSION

In this paper, we gave a simple countermeasure to SPAs against the unified ECC codes. It changes one of input points of doubling operations into an equivalent point modulo p. Hence, the doubling is expected to behave like an addition. The new countermeasure is very efficient in that it only requires $27(n - 1)/2$ extra integer subtractions for an n-bit scalar multiplication, hence the overhead of the proposed method is negligible, compared with the whole scalar multiplication. The timing analysis of the proposed countermeasure is also presented to confirm its SPA resistance.

REFERENCES

[1] N. Koblitz, *Elliptic Curve Cryptosystems*, Mathematics Computation, vol. 48, 1987, pp. 203–209.
[2] V. S. Miller, *Use of Elliptic Curves in Cryptography*, CRYPTO'85, LNCS, vol. 218, Springer-Verlag, 1986, pp. 417–426.
[3] J.-S. Coron, *Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems*, CHES'99, LNCS, vol. 1717, Springer-Verlag, 1999, pp. 292–302.
[4] P. Kocher, J. Jaffe, and B. Jun, *Differential Power Analysis*, CRYPTO'99, LNCS, vol. 1666, Springer-Verlag, 1999, pp. 388–397.
[5] É. Brier, I. Déchène, and M. Joye, *Unified Point Addition Formulae for Elliptic Curve Cryptosystems*, Embedded Cryptographic Hardware: Methodologies and Architectures, Nova Science Publishers, 2004, pp. 247–256.
[6] É. Brier and M. Joye, *Weierstraß Elliptic Curves and Side-channel Attacks*, PKC 2002, LNCS, vol. 2274, Springer-Verlag, 2002, pp. 335–345.
[7] D. Stebila and N. Thériault, *Unified Point Addition Formulae and Side-Channel Attacks*, CHES 2006, LNCS, vol. 4249, Springer-Verlag, 2006, pp 354–368.
[8] C. Walter, *Simple Power Analysis of Unified Code for ECC Double and Add*, CHES 2004, LNCS vol. 3156, Springer-Verlag, 2004, pp. 191–204.
[9] I. F. Blake, G. Seroussi and N. P. Smart, *Elliptic Curves in Cryptography.*, Cambridge University Press, CRC Press, 1999.
[10] P. L. Montgomery, *Speeding the Pollard and Elliptic Curve Methods of Factorization*, Mathematics of Computation, 48, 1987, pp. 243–264.
[11] National Institute of Standards and Technology, *Recommended Elliptic Curves for Federal Government Use, Appendix to FIPS 186-2*, 2000.

Yoo-Jin Baek was born in Korea, on February 20, 1972. He received the B.Sc., the M.Sc. and the D.Sc. degrees in Mathematics from Seoul National University, Korea in 1997, 1999, and 2003, respectively. He is a senior engineer at Samsung Electronics, Korea. His major interests are cryptography and side-channel cryptanalysis.