A Binary Redundant Scalar Point Multiplication in Secure Elliptic Curve Cryptosystems

Sangook Moon

School of Electronic and Information Engineering, Mokwon University
800 Doan-dong Seo-gu Daejeon 302-729 Korea. (Email: smoon@mokwon.ac.kr)

(Received Aug. 4, 2005; revised and accepted Sept. 14 & Oct. 4, 2005)

Abstract

The main back-bone operation in elliptic curve cryptosystems is *scalar point multiplication*. The most frequently used method implementing the *scalar point multiplication* which is performed in the top level of GF (Galois Field) multiplication and GF division, has been the *double-and-add* algorithm, which is being recently challenged by NAF (Non-Adjacent Format) algorithm. In this paper, we propose a more efficient and novel approach of a *scalar point multiplication* method than existing *double-and-add* by applying redundant recoding which originates from radix-4 Booth's algorithm. We call the novel algorithm *quad-and-add*. Along with the algorithm, we have created a new EC (Elliptic Curve) point operation, named *point quadruple*, and verified with calculations of a real-world application to utilize it. Derived numerical expressions were verified using both C programs and HDL (Hardware Description Language). Proposed method of EC *scalar point multiplication* can be utilized in many EC security applications for handling efficient and fast calculations.

Keywords: Elliptic curve cryptosystem, Galois field, scalar point multiplication

1 Introduction

As an indispensable component of information technologies, security applications, such as IC cards used for personal authentication and domestic network applications, play an important role. In fact, such data security receives constant attention, since people tend to communicate with each other by various electronic devices over networks. Security applications are based upon intensive computations of cryptographic algorithms, which generally involve in arithmetic operations in large Galois fields [1, 8].

Polynomial basis offers good solutions to most GF computational problems. Also, polynomial basis is the easiest to use among other representations. Therefore, we focus on using the polynomial basis throughout this document [7].

The most important and time-consuming operation in calculating EC operations is the *scalar point multiplication*, which repeatedly performs *point addition* GF operation as in Equation (1). In Equation (1), k is an arbitrary integer number on a finite field GF(2^m) and P is an arbitrary point on an EC de-fined on the finite field GF(2^m).

$$kP = \sum_{i=1}^{k} P \quad (k \text{ times of } \textit{point addition}) \tag{1}$$

Figure 1 shows the hierarchical structure of an ECC operation. In general, in order to perform one *scalar point multiplication* [4], we need to calculate *point addition* operations (if two points are different) along with *point double* operations (if two points are identical). The most important factor required in the speed-effective implementation of a *scalar point multiplication* is proper handling of Equation (1). *Double-and-add* algorithm has been traditionally prevalent in this area, which is recently being challenged by NAF algorithm [3]. In this paper, we propose a *scalar point multiplication* algorithm with a novel approach applying radix-4 Booth's recoding and derive numerical expressions on the *point quadruple* operation [6]. We evaluated and verified the algorithms using real applications. Derived expressions were described with both C program and HDL to be proven, measuring its performance improvement. The outline of the paper is as follows: We start by introducing the concept of EC *scalar point multiplication* operation in Section 2. In Section 3 we discuss our evaluation and validation about our proposed algorithms, and will conclude in Section 4.

2 Elliptic Curve Scalar Point Multiplication Operation Algorithms

In this contribution, we will propose a new approach of obtaining the *scalar point multiplication* product based on an EC group. First, we'll introduce the fundamental mathematics of the ECC-based cryptosystem, especially

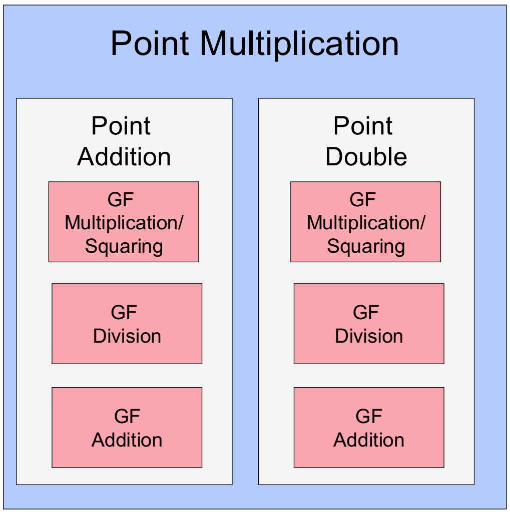

Figure 1: Hierarchical structure of an elliptic curve operation

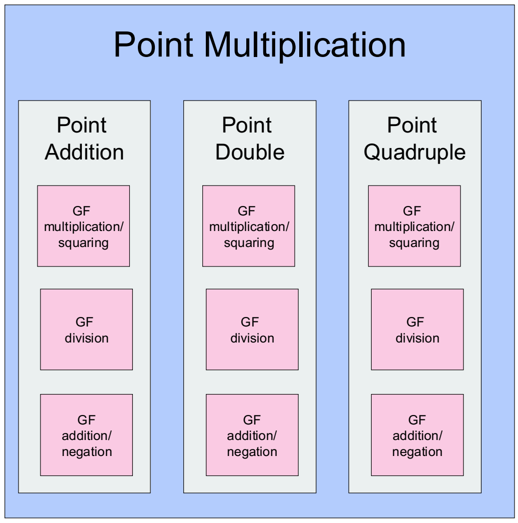

Figure 2: Hierarchical structure of elliptic curve operations in suggested algorithm

for polynomial basis arithmetics. In Section 2.2, we discuss the previous studies which have been researched to improve the complex EC *point multiplication* operation calculation. After that, we propose the algorithm and a few complementary formulas in Section 2.3.

2.1 Mathematics of the ECC-based Cryptosystem

Two main operations are required to multiply an EC group element by a constant when encrypting a message: *point addition* (hereafter $add()$) and *point double* ($double()$) operations. We also include *point negation* ($neg()$) as a miscellaneous operation and *point quadruple* ($quad()$) operation, which is about to be suggested for fast implementation algorithm of kP.

The elliptic curve E is defined as the set of all solutions (x, y) to the equation $y^2+xy = x^3+ax^2+b$ together with the point at infinity O, where b is not 0. This extra point O is needed to represent the group identity. Rules for the above mathematical operation routines except for $quad()$ operation are presented below. Rules for the $quad()$ operation are given in Section 2.3.

Point addition (add( )):

Let $P(x_1, y_1)$ and $Q(x_2, y_2)$ be two different points on the curve.
If either point is O, the result is the other point.
If $P = Q$, use **double( )** *routine.*
If $x_1 = x_2$ and $y_1 \neq y_2$, $P + Q = O$.
If $P \neq Q$, then $P + Q = R(x_3, y_3)$, where

$$x_3 = \lambda^2 + \lambda + x_1 + x_2 + a,$$
$$y_3 = \lambda(x_1 + x_3) + x_3 + y_1,$$
$$and\ \lambda = (\frac{y_1 + y_2}{x_1 + x_2})$$

Point double (double( )):

Let $P(x_1, y_1)$ and $Q(x_1, y_1)$ be a point on the curve.
If $x_1 = 0$, the result of $2P$ is O.
If $x_1 \neq 0$, $2(x_1, y_1) = R(x_3, y_3)$, where

$$x_3 = \lambda^2 + \lambda + a,$$
$$y_3 = x_1^2 + (\lambda + 1)x_3,$$
$$and\ \lambda = (x_1 + \frac{y_1}{x_1}).$$

Point negation (neg( )):

Let $P(x_1, y_1)$ be a point on the curve
$-P = R(x_3, y_3)$, or

$$(x_3, y_3) = -(x_1, y_1) = (x_1, x_1 + y_1).$$

From the rules above, we can discern the number of field operations required to carry out the routine. In the *add( )* routine, 8 additions, 1 multiplication, 1 division, and 1 squaring of GF operations are required. We should check that the divider of λ, or $(x_1 + x_2)$ is not zero. The *double( )* routine requires 4 additions, 1 multiplication, 2 squarings, and 1 division. Also, we should check that the divider of λ or x_1 is not zero.The $neg()$ routine requires just one GF addition. This operation is needed only when implementing the fast algorithm for the calculation of kP. As explained later in Section 2.3, the values of $(-P)$ and $(-2P)$ are needed in the algorithm we developed.

As basic mathematics for the ECC-based cryptosystem, GF multiplication and GF division occupy indispensable positions, with the greatest importance of utilizing the *scalar point multiplication* operation, which is to discussed from below.

2.2 Recent Studies

Double-and-add algorithm has been the leading algorithm in implementing the *scalar point multiplication* in ECC [5]. *Double-and-add* is similar to the *square-and -multiply* algorithm in the RSA cryptosystem [9], in which modular exponentiation is implemented with the algorithm. *Double-and-add* algorithm is represented in Equation (2) as below, when $k = \sum_{i=0}^{m-1} b_i 2^i$ $(b_i \in 0, 1)$. From this point on, we will use some notations.

Double-and-add algorithm for computing kP

***kP*:**
$k = \sum_{i=0}^{m-1} b_i 2^i \quad (b_i \in 0, 1)$
$P := P(x_1, y_1)$
$Q := P.$

$$\begin{aligned} & for\ i\ from\ m-1\ downto\ 0\ do \\ & \quad Q := double(Q) \\ & \quad if\ b_i = 1\ then \\ & \quad Q := add(P, Q) \\ & end(Q = kP) \end{aligned} \tag{2}$$

Note that we need as many number of *add( )* operations as the number of *Hamming weight* in the binary representation of k in addition to at least $m-1$ times of *double( )*. In order to improve the performance of the algorithm above, several algorithms have been suggested. One of the algorithms is NAF (Non- Adjacent Format), as described below in Equation (3).

Binary NAF method for computing kP

***kP*:**
$NAF(k) = \sum_{i=0}^{i-1} k_i 2^i$
$Q := O$

$$\begin{aligned} & for\ i\ from\ t-1\ downto\ 0\ do \\ & \quad Q := 2Q \\ & \quad if\ k_i = 1\ then\ Q := Q + P \\ & \quad if\ k_i = -1\ then\ Q := Q - P \\ & end(Q := kP) \end{aligned} \tag{3}$$

In the above method, the concept of redundancy of the binary representation of k is used in calculating kP. However, it has a weak point that k should be converted into NAF format in advance. As an improved approach of the concept of redundancy, we propose a tricky algorithm named *quad-and-add* algorithm which utilizes *point quadruple operation*, both of which will be discussed in detail in the next section.

2.3 Quad-and-Add Algorithm

In order to obtain two times as fast calculations as *double-and-add* algorithm, we applied radix-4 redundant recoding to the binary presentation of EC point Q. Equation (4) shows the concept of using radix-4 redundancy in pseudo code representation. Due to the characteristic of radix-4 redundancy recoding, total number of steps reduces by half down to $\lceil \frac{m}{2} \rceil - 1$. According to the result

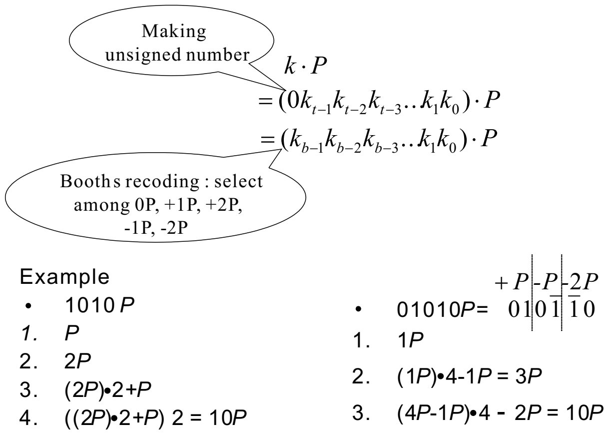

Figure 3: Comparison example of two algorithms

of radix-4 recoding of point Q in each step, one out of the adders $0P, \pm P, \pm 2P$ is chosen so that we get the final *scalar point multiplication* result in $\lceil \frac{m}{2} \rceil - 1$ cycles, which is 2 times as fast as the *double-and-add* algorithm.

Quad-and-add algorithm using radix-4 redundancy

***kP*:**
$k = \sum_{i=0}^{\lceil \frac{m}{2} \rceil - 1} r_i 4^i$ *(r_i is the value of redundancy recoding)*
$P := P(x_1, y_1)$
$2P := double(P)$
$Q := one\ of 0P, +P, +2P, -P, -2P$

$$\begin{aligned} & for\ i\ from\ \lceil \frac{m}{2} \rceil - 1\ downto\ 0\ do \\ & \quad Q := quad(Q) \\ & \quad if\ (r_i == +P)\ then \\ & \quad\quad Q := add(P, Q) \\ & \quad if\ (r_i == +2P)\ then \\ & \quad\quad Q := add(2P, Q) \\ & \quad if\ (r_i == -P)\ then \\ & \quad\quad tempP := neg(P) \\ & \quad\quad Q := add(tempP, Q) \\ & \quad if\ (r_i == -2P)\ then \\ & \quad\quad tempP := neg(2P) \\ & \quad Q := add(tempP, Q) \\ & end(Q := kP) \end{aligned} \tag{4}$$

Here, in order to get the quadruple point of a point P on the given EC without using the *double( )* operation two consecutive times, we derived the *point quadruple* operation (hereafter *quad*()) combining the *add( )* and *double( )* operation, as in Equation (5). Then, the hierarchy shown in Figure 1 becomes slightly modified as Figure 2.

Point quadruple operation (quad( ))

$P(x_1, y_1) = Q(x_1, y_1)$ *is identical on an EC*
if $x_1 = 0$*, the result* $4P$ *is* O *(zero at infinity)*

Table 1: Number of the GF operations taken calculating the value of 4P out of different EC operations

		Multiplication	Division	Square	Addition
4P	**4 · add( )**	4	4	4	32
	2 · double( )	2	2	4	8
	quad( )	1	2	4	10

if $x_1 \neq 0$, *the result* $4P(x_1, y_1) = R(x_3, y_3)$, *where* x_3 *and* y_3 *areas follows,*

$$\begin{aligned} x_3 &= \lambda'^2 + \lambda' + a, \\ y_3 &= x_1^2 + (\lambda' + 1)x_3, \\ \lambda' &= x_2 + \lambda + 1 + \frac{x_1^2}{x_2} \\ x_2 &= \lambda^2 + \lambda + a, \\ \lambda &= (x_1 + \frac{y_1}{x_1}) \end{aligned} \tag{5}$$

From this formula, we can determine the number of GF operations. The *quad*() routine will require 10 additions, 1 multiplication, 2 divisions, and 4 squarings of GF operations. In Table 1, we described how $4P$ calculation is achieved using different EC operations so as to explain the advantage of our *quad*(). Figure 3 provides a brief understanding of how two algorithms work between the traditional *double-and-add* algorithm and our new algorithm.

3 Evaluation and Validation

In order to verify the kP calculation procedure, we used the fact that if we multiply a point by the order of given EC, we get the *point at infinity (O)* [2]. The upper part of Figure 4 represents the process of obtaining the *point at infinity* using *double-and-add* algorithm at the 192nd step. In the lower part of Figure 4 we can see that we get the expected result at the 96th step using the *quad-and-add* algorithm featuring *quad*() operation.

Evaluation was performed at the level of highest hierarchy, or *scalar point multiplication*, implemented with HDL-described 193-bit cryptoprocessor. Figure 5 shows the block diagram of the EC processor. We evaluated the performance focusing on the advantage with *quad*() operation. By adopting the algorithm of *quad-and-add*, the number of iterations decreases from m to $\lceil \frac{m}{2} + 1 \rceil$ steps. Table 2 and Table 3 summarize the advantage of the proposed algorithms, comparing with EC operations and GF operations respectively. The number of operations in Table 2 is calculated based on the probability that is dependent on the hamming weight of the prime polynomial. The probability of the existence of 1 in the binary representation of k during m steps in the *double-and-add* algorithm is 0.5, and the probability of the existence of non-zero Booth's recoding term is 6/8. Because

```
double-and-add
step# 0 : double-and-add
0_D9B67D19_2E0367C8_03F39E1A_7E82CA14_A651350A_AE617E8F
1_CE943356_07C304AC_29E7DEFB_D9CA01F5_96F92722_4CDECF6C
step# 1 : double
1_756FF0DC_810F7856_023C5F5C_B14481F3_A668572B_B1513DA3
1_071883B7_5B3044A9_217AD3AC_A9EF8CDC_89CDEBA2_3F931652
step# 2 : double
1_1549FE34_2A8980E6_C932AF6F_4C81D415_00B09840_85F3B447
1_C0DDD61E_0CD1960A_59F7FE63_A8660A53_4D9F431E_4BC9839F
.
.
step#190: double-and-add
1_2654EB57_653586DB_05FD2EBC_511BC95F_2D995691_E0E95F9F
0_9C3BCACD_837A6A81_97F97238_3D20828E_1797902E_5829F927
step#191: double
1_5AE7384C_9954F598_6475718C_069EE793_3F2AA29E_2465F8E7
1_3BC5521A_6D7AE739_4E5E2DF9_FA26FB66_2DB5D58D_13BC8CAA
step#192: double-and-add
0_00000000_00000000_00000000_00000000_00000000_00000000
0_00000000_00000000_00000000_00000000_00000000_00000000

quad-and-add
step# 0 : quad-and-add(p)
0_D9B67D19_2E0367C8_03F39E1A_7E82CA14_A651350A_AE617E8F
1_CE943356_07C304AC_29E7DEFB_D9CA01F5_96F92722_4CDECF6C
step# 1 : quad
1_1549FE34_2A8980E6_C932AF6F_4C81D415_00B09840_85F3B447
1_C0DDD61E_0CD1960A_59F7FE63_A8660A53_4D9F431E_4BC9839F
.
.
step# 94 : quad-and-add(p)
0_24771C2C_8E33F4A9_81965AC9_5FBC8DE2_4A0FC903_6208E77D
0_905C0FE8_0E90B8D0_259C0682_C561E98C_43935E7429A8
step# 95 : quad-and-add(p)
1_2654EB57_653586DB_05FD2EBC_511BC95F_2D995691_E0E95F9F
0_9C3BCACD_837A6A81_97F97238_3D20828E_1797902E_5829F927
step# 96 : quad-and-add(p)
0_00000000_00000000_00000000_00000000_00000000_00000000
0_00000000_00000000_00000000000000_00000000_00000000
```

Figure 4: Scalar point multiplication comparison using double-and-add and quad-and-add

Table 2: Comparison of the number of steps and EC operations between the algorithms

	♯ of steps	add( )	double( )	neg( )	quad( )
Double-and-add	m	$\frac{1}{2}m$	m	0	0
Quad-and-add	$\lceil \frac{m}{2} \rceil + 1$	$\frac{6}{8}\cdot\frac{1}{2}\cdot\frac{1}{2}m = \frac{3}{16}m$	1	2	$\lceil \frac{m}{2} \rceil + 1$

Table 3: Comparison of the number of GF operations between the algorithms

	Doubld-and-add	Quad-and-add	Reduction ratio
Multiplication	$\frac{3}{2}m$	$\frac{11}{16}m + 2$	≈ 0.46
Division	$\frac{3}{2}m$	$\frac{19}{16}m + 3$	≈ 0.79
Square	$\frac{5}{2}m$	$\frac{35}{16}m + 6$	≈ 0.87
Addition	$12m$	$\frac{13}{2m} + 14$	≈ 0.54

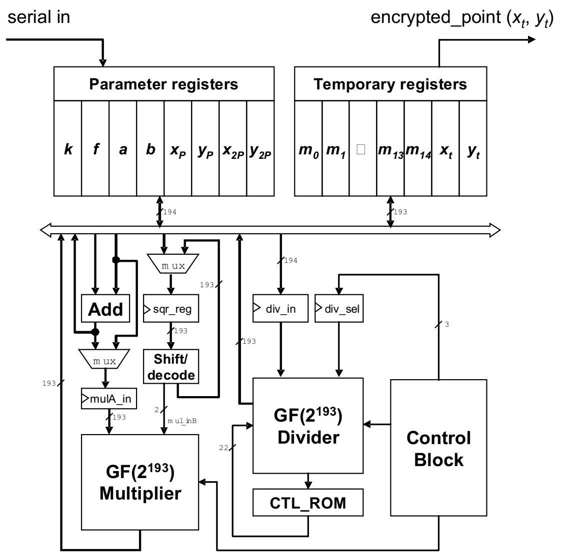

Figure 5: 193-bit EC cryptoprocessor prototype

the number of the steps has been reduced to $\lceil \frac{m}{2} + 1 \rceil$, the total number of *add( )* operations appearing in the *quad-and-add* becomes approximately $\frac{3}{16}m$. With these results in the back-ground, we can now measure how many GF operations takes in calculating kP as in Table 3.

Table 3 represents the performance improvement. Measurement on our 193-bit cryptoprocessor showed reduction percentage of 46%, 79%, 87%, and 54% in multiplications, divisions, squares, and additions of GF operations relatively. We applied as test vectors *sect193r2* EC parameters which is suggested by SEG2 [4] when the EC complexity depth is 193 bits. Mentioning in the light of hardware overhead, the proposed algorithm requires simple 3-bit Booth's recoding circuit and an m-bit register for storing the values of $P, 2P, -P$, and $-2P$ additionally, and a 194-bit shifter.

4 Discussion and Conclusion

We have proposed an improved version of *scalar point multi-plication* algorithm applying the concept of radix-4 redundancy. In order to use the concept of redundancy, we derived a new EC operation named *point quadruple*. We have tested the suggested methods in an Elgamal EC cryptosystem environment. Designed prototype was verified with both C program language and HDL. Simulation result showed drastical performance improvement over the algorithm using *double-and-add* method.

Fast *scalar point multiplication* algorithm can be used in various applications such as EC encryption and decryption, electronic signature authentication, secure key exchange, etc. The importance of its versatility cannot be too much emphasized. Also, by utilizing the *point quadruple* operation suggested in this paper, we can expect faster and efficient computation in most GF applications.

References

[1] E. R. Berlekamp, "Bit-serial reed-solomon encoders," *IEEE Transactions on Information Theory*, vol. IT-28, no. 6, pp. 869-874, nov. 1982.

[2] Certicom research, *SEG2: Recommended Elliptic Curve Domain Parameters*, Oct. 1999.

[3] D. Hankerson, J. L. Hernandez, and A. Menezes, "Software implementation of elliptic curve cryptography over binary fields", in *Crypto95*, pp. 1-24, 1995.

[4] N. Koblitz, "Elliptic curve cryptosystems", *Mathematics of Computation*, vol. 48 no. 177, pp. 203-209, 1987.

[5] N. Koblitz, *A Course in Number Theory and Cryptography*, Springer-Verlag, pp. 150-179, 1991.

[6] I. Koren, *Computer Arithmetic Algorithms*, Section 6, pp. 99-106, Prentice Hall International, 1993.

[7] E. Mastrovito, *VLSI Architectures for Computation in Galois Fields*, PhD thesis, Dept. of Electrical Eng., Lin-koping University, Sweden, 1991.

[8] C. Paar, P. Fleischmann, and P. S-Rodriguez, "Fast arithmetic for public-key algorithms in galois fields with com-posite exponents," *IEEE Transactions on Computers*, vol. 48, no. 10, pp. 1025-1034, Oct. 1999.

[9] R. L. Rivest, A. Shamir, and L. M. Adleman, "A method for obtaining digital signatures and public-key crypto-systems", *Communications of the ACM*, vol. 21, pp. 120-126, Feb. 1978.

Sangook Moon was born in Korea, in 1971. He received the B.S., M.S. and Ph.D. degree in electronic engineering from Yonsei University, Korea in 1995, 1997, and 2002 respectively. In 2004, he joined the Department of Electronic and Information Security Engineering at Mokwon University, where he is currently an assistant professor. His current research interests include VLSI, crypto-processors, microprocessors, computer arithmetic, and security-related SoC.