Elliptic Curve Scalar Point Multiplication Algorithm Using Radix-4 Booth's Algorithm

Sangook Moon, Non-member

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-most level of GF 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 multiplication method than the *double-and-add* by applying redundant recoding which originates from the radix-4 modified Booth's algorithm. We call the novel algorithm *quad-and-add*. After deriving the algorithm, we created a new GF 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 elliptic curve scalar point multiplication can be utilized in many elliptic curve security applications for handling efficient and fast calculations.

Keywords: elliptic curve cryptosystem, scalar point multiplication, GF, HDL, security

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 (GF) [1][2].

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 [3].

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

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

Figure 1 shows the hierarchical structure of an ECC operation. In general, as we intend to perform one *scalar point multiplication* [4], we need a couple of *point addition* operation (if two points are different) and a couple of *point double* operation (if two points are identical). The most important factor required in the speed-effective implementation of a *scalar point multiplication* is proper handling of expression (1). *Double-and-add* algorithm has been traditionally prevalent in this area, which is recently being challenged by NAF algorithm [5]. 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 elliptic curve *scalar point multiplication* operation in Chapter 2. In Chapter 3 we discuss our evaluation and validation about our proposed algorithms, and will conclude in Chapter 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 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.

04PSI05: Manuscript received on December 21, 2004 ; revised on June 3, 2005.

The author is with the Department of Information Security and Electronic Engineering, Mokwon University, Daejeon, Korea. E-mail:smoon@mokwon.ac.kr

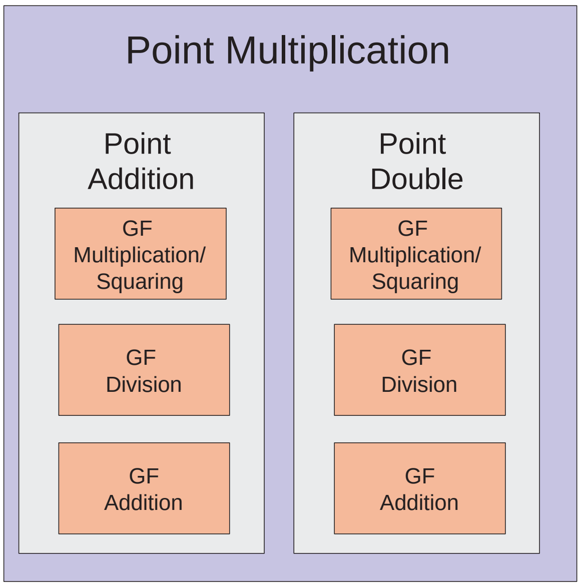

Fig.1: *Hierarchical structure of an elliptic curve operation.*

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* and *pointdouble* 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:

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 ***point double*** *routine.*
If $x_1 = x_2$ and $y_1 y_2, P + Q = O$.
If $P \neq Q$, then $P + Q = R(x_3, y_3)$, where

$$\begin{aligned} x_3 &= \lambda^2 + \lambda + x_1 + x_2 + a, \\ y_3 &= \lambda(x_1 + x_3) + x_3 + y_1, \\ \text{and } \lambda &= \left(\frac{y_1 + y_2}{x_1 + x_2}\right) \end{aligned} \quad (2)$$

Point double:

Let $P(x_1, y_1) = Q(x_1, y_1)$ 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

$$\begin{aligned} x_3 &= \lambda^2 + \lambda + a, \\ y_3 &= x_1^2 + (\lambda + 1)x_3, \\ \text{and } \lambda &= \left(x_1 + \frac{y_1}{x_1}\right) \end{aligned} \quad (3)$$

Point negation:

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

$$(x_3, y_3) = -(x_1, y_1) = (x_1, x_1 + y_1) \quad (4)$$

From the rules above, we can discern the number of field operations required to carry out the routine. In the *point addition* routine, 8 additions, 1 multiplication, 1 division, and 1 squaring are required. We should check that the divider of λ , or (x_1+x_2) is not zero. The *point 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 *point negation* routine requires just one addition. This operation is needed 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 *scalarpointmultiplication* 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 [7]. *Double-and-add* is similar to the *square-and-multiply* algorithm in the RSA cryptosystem [8], in which modular exponentiation is implemented with the algorithm. *Double-and-add* algorithm is represented in expression (5) 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. The point addition operation will be represented as *add( )* and the *point double as double( )*.

Double-and-add algorithm for computing kP

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

      for $i$ from $m-1$ downto 0 do
          $Q := double\{Q)$
          if $b_i = 1$ then
```

```
        $Q := add(P,Q)$
$end\ (Q = kP)$                                  (5)
```

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 expression (6).

Binary NAF method for computing kP

kP:

$$NAF(k) = \sum_{i=0}^{t-1} k_i 2^i$$

```
$Q := 0$
    $for\ i\ from\ t-1\ downto\ 0\ do$
        $Q := 2Q$
        $if\ k_i = 1\ then\ Q := Q + P$
        $if\ k_i = -1\ then\ Q := Q - P$
$end\ (Q := kP)$                                 (6)
```

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. Expression (7) 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 $\left[\frac{m}{2}\right]-1$. According to the result 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 multiplication result in $\left[\frac{m}{2}\right]-1$ cycles, which is 2 times as fast as the *double-and-add* algorithm.

Quad-and-add algorithm using radix-4 redundancy

kP:

$k = \sum_{i=0}^{\left[\frac{m}{2}\right]-1} r_i 4^i$ *(r_i is the value of redundancy recoding)*

```
$P := P(x_1, y_1)$
```

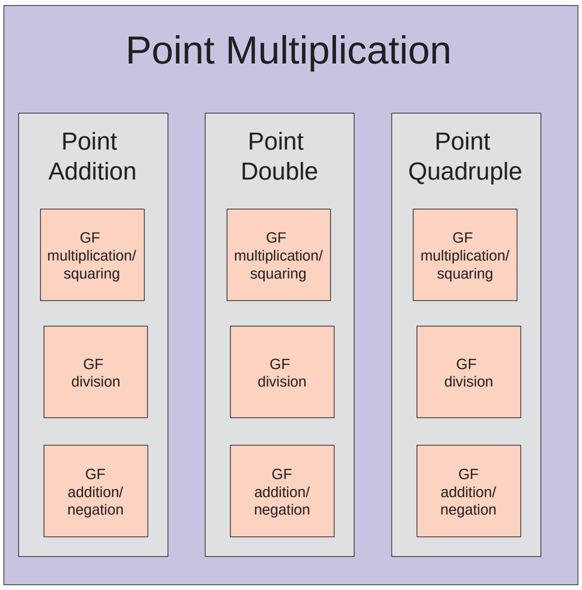

Fig.2: *Hierarchical structure of elliptic curve operations in suggested algorithm.*

```
$2P := double(P)$
 $Q := one\ of\ \{0P, +P, +2P, -P, -2P\}$
$for\ i\ from\ \left[\frac{m}{2}\right]-1\ downto\ 0\ do$
 $Q := quad(Q)$
  $if\ (r_i == +P)\ then$
 $Q := add(P,Q)$
  $if\ (r_i == +2P)\ then$
 $Q := add(2P,Q)$
  $if\ (r_i == -P)\ then$
$tempP := neg(P)$
  $if\ (r_i == -2P)\ then$
$tempP := neg(2P)$
 $Q := add(tempP,Q)$
$end\ (Q := kp)$                                 (7)
```

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 *point addition* and *point double* operation, as in expression (8). Then, the hierarchy shown in Figure 1. becomes slightly modified as Figure 2.

Point quadruple operation (quad( ))

$\cdot P(x_1, y_1) = Q(x_1, y_1)$ is identical on an EC
· if $x_1 = 0$, the result 4P is O (zero at infinity)
· if $x_1 \neq 0$, the result $4P(x_1, y_1) = R(x_3, y_3)$, where x_3 and y_3 are as follows,

$$x_3 = \acute{\lambda}^2 + \acute{\lambda} + a,$$

$$\begin{aligned} y_3 &= x_2 + (\acute{\lambda} + 1)x_3, \\ \acute{\lambda} &= x_2 + \lambda + 1 + \frac{x_1^2}{x_2}, \qquad (8) \\ x_2 &= \lambda^2 + \lambda + a, \\ \lambda &= \left(x_1 + \frac{y_1}{x_1}\right). \end{aligned}$$

From this formula, we can determine the number of field operations. The *quad( )* routine will require 10 additions, 1 multiplication, 2 divisions, and 4 squarings. Fig. 3 shows a simple example of comparison between the traditional double-and-add algorithm and our proposed new algorithm using radix-4 redundant recoding.

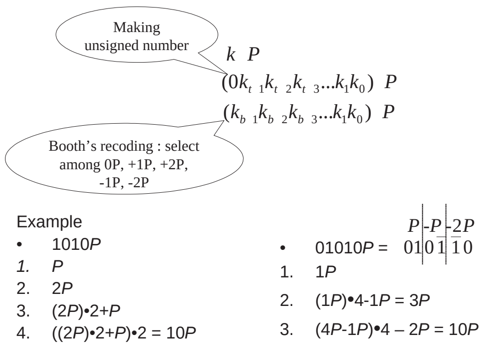

Fig.3: *Comparison example of two algorithms.*

The number of iterations decreases from m to $\left\lceil \frac{m}{2} \right\rceil + 1$ steps. Table 1 summarizes the improvement in the number of steps and required EC operations. The number of operations in Table 1 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. The new algorithm exhibits a reduction of about 12.5% in handling *Add* operations. Furthermore, the new algorithm is also advantageous because of using *Quad* operations. The *Quad* routine is induced from manipulating the expressions in the *Double* routine resulting in a reduction of 1 field multiplication, and the proposed algorithm can be far more efficient by enhancing the *Quad* routine using higher mathematics in future. The proposed algorithm requires Booth's recoding circuit and memory space for storing the values of $P, 2P, -P$, and $-2P$ additionally. The number of field operations for calculating kP are represented in detail in Table 2, which demonstrates the efficiency of our proposed algorithm. We achieved performance improvement of about 19% in multiplication considering that our GF multiplier can be used as a *GF* squarer, and about 9% in *GF* division.

Table 1: *Comparison of Number of GF operations.*

	# of steps	Add	Double	Neg	Quad
Double-and-add	m	$\frac{1}{2}m$	m	0	0
Proposed algorithm	$\left\lceil \frac{m}{2} \right\rceil + 1$	$\frac{3}{8}m$	1	2	$\left\lceil \frac{m}{2} \right\rceil + 1$

3. EVALUATION AND VALIDATION

We divided the entire scalar multiplication process into 3 sub-blocks to verify the derived expressions of *quad( )* operation algorithm. Sub-blocks were roughly categorized as GF multiplier block and GF divider block. Entire prototype processor was simulated using digital control unit designed from finite state machines, integrating temporary register blocks. Overall performance of the pilot EC processor was verified through simulation using both C test bench programs and HDL at algorithmic level.

Firstly, we implemented Mastrovito's serial GF multiplier [9], which is undoubtedly reliable for verification, in C language level and applied 193-bit random binary numbers as test vectors. C language-described serial multiplier does not produce error due to its algorithmic originality. Result was confirmed by checking the value of partial results during each step of algorithmic process. Also, we used GF divider which was also verified in [1]. We used the fact that if we multiply a point by the order of given EC, we get the *point at infinity* (O), as a special characteristic of EC operations [10], to verify the entire *scalar point multiplication* process. The upper part of Figure 3 represents the process of obtaining the point at *infinity* using *double-and-add* algorithm at 192^{nd} level. In the lower part of Figure 4, we can see that we get the expected point at *infinity* at 96th step using *quad( )* operation.

Evaluation was performed at the level of highest hierarchy, or *scalar point multiplication*, in the HDL-described 193-bit elliptic curve cryptoprocessor. Figure 5 shows the block diagram of HDL-described 193-bit elliptic curve processor. We evaluated the performance focusing on the suggested *quad( )* GF operation. Table 2 represents the performance improvement. Measurement on our 193-bit cryptoprocessor showed more than 40% reduction of GF multiplication operation and a small amount of increase in the number of GF division and addition applying *sect193r2* as EC parameters which is suggested by SEG2 [4] in 193 bit scalar multiplication. Overall performance gain over the architecture using *double-and-add* algorithm was about 30%, considering speed and area.

```
 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_43935E74_9A872F1D
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_00000000_00000000_00000000_00000000
```

Fig.4: *Scalar multiplication comparison using double-and-add and quad-and-add .*

Table 2: *Comparison of Number of GF operations.*

	Doubld-and-add	Quad-and-add	Reduction ratio
Multiplication	$\frac{3}{2}m$	$\frac{7}{8}m+1$	≈ 0.58
Division	$\frac{3}{2}m$	$\frac{11}{8}m+1$	≈ 0.91
Square	$\frac{5}{2}m$	$\frac{19}{8}m+2$	≈ 0.95
Addition	$8m$	$8m+6$	≈ 1

4. DISCUSSION AND CONCLUSION

We proposed an improved version of a *scalar point multiplication* algorithm using the concept of radix-4 redundancy, or *point quadruple* scalar operation useful in computing complex EC operations. We applied the test environment in an Elgamal EC cryptosystem using the proposed algorithm. Designed prototype was verified with both C program language and HDL. Evaluation result showed about more than 30% performance enhancement over the algorithm using *double-and-add method.*

Fast *scalar point multiplication* operation can be used in various applications such as encryption / decryption using EC operations and electronic signature authentication as well as secure key exchange, and the importance of its versatility can be too much emphasized. Also, by utilizing the *point quadruple* operation suggested in this paper, we can expect faster and efficient computation in most finite field operations.

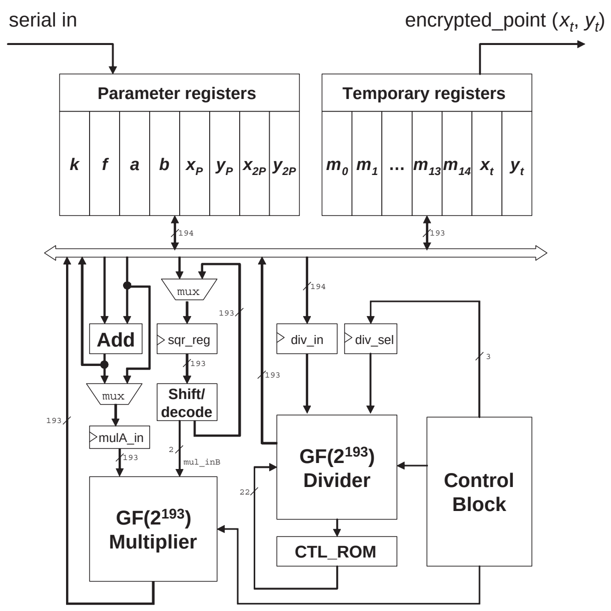

Fig.5: *193-bit EC cryptoprocessor prototype.*

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] C. Paar, P. Fleischmann, P. S-Rodriguez: "Fast Arithmetic for Public-Key Algorithms in Galois Fields with Composite Exponents," *IEEE Transactions on Computers*, October 1999, Vol. 48, No. 10, pp. 1025-1034.

[3] E. Mastrovito. : "VLSI Architectures for Computation in Galois Fields," PhD thesis, Dept. of Electrical Eng., Linkoping Univ., Sweden, 1991.

[4] Neal Koblitz, "Elliptic Curve Cryptosystems", Mathematics of Computation, 48 n.177 (1987), pp. 203-209.

[5] D. Hankerson, J. L. Hernandez, and A. Menezes, "Software Implementation of Elliptic Curve Cryptography over Binary Fields", *Crypto95* (1995).

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

[7] N. Koblitz, *A Course in Number Theory and Cryptography*, Springer-Verlag (1991).

[8] R. L. Rivest, A. Shamir, and L. M. Adleman,

"A Method for Obtaining Digital Signatures and Public-key Cryptosystems", *Communications of the ACM*, vol. 21, pp. 120-126 (Feb. 1978)

[9] E. D. Mastrovito, "VLSI Design for multiplication over finite fields GF(2^m)", in *Lecture notes in Computer Science 357*, Springer-Verlag, Berlin, pp. 297-309 (Mar. 1989).

[10] Certicom research, "SEG2: Recommended Elliptic Curve Domain Parameters" (Oct. 1999).

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 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.