ISSN: 2454-1907

International Journal of Engineering Technologies and Management Research
June 2020, Vol 7(06), 141 – 151
DOI: https://doi.org/10.29121/ijetmr.v7.i6.2020.712

EFFICIENT OPERATIONS IN LARGE FINITE FIELDS FOR ELLIPTIC CURVE CRYPTOGRAPHIC

Yan-Haw Chen [1], Chien-Hsing Huang [*2]

[*1, 2] Department of Department of Information Engineering I-Shou University, Kaohsiung, Taiwan 84008, Republic of China

DOI: https://doi.org/10.29121/ijetmr.v7.i6.2020.712

Article Citation: Yan-Haw Chen, and Chien-Hsing Huang. (2020). EFFICIENT OPERATIONS IN LARGE FINITE FIELDS FOR ELLIPTIC CURVE CRYPTOGRAPHIC. International Journal of Engineering Technologies and Management Research, 7(6), 141-151. https://doi.org/10.29121/ijetmr.v7.i6.2020.712

Published Date: 27 June 2020

Keywords:
Elliptic Curve
Encryption
Finte Field
Horner Rule
Dynamic Lookup Table
Multiplication

ABSTRACT

An efficient method to compute the finite field multiplication for Elliptic Curve point multiplication at high speed encryption of the message is presented. The methods of the operations are based on dynamic lookup table and modified Horner rule method. The modified Horner rule method is not only to finite field operations but also to Elliptic curve scalar multiplication in the encryption and decryption. By comparison with using Russian Peasant method and in the new proposed method, one of the advantages of utilizing the proposed algorithm is that in the Elliptic Curve point addition are reduced by a factor of three in GF (2163). Therefore, using the Algorithm 1 running on Intel CPU, computation cost of the multiplication method is above 70% faster than using standard multiplication by Russian Peasant method. Ultimately, the proposed Algorithm 1 for evaluating multiplication can be made regular, simple and suitable for software implementations.

1. INTRODUCTION

In theory, a finite field is an algebraic structure with established operations for addition, subtraction, multiplication, and division by satisfying an Abelian group. These operations have following four properties closure, associativity, commutativity, and having an inverse element. Galois Fields GF(2m) have a wide variety of applications utilized in the cryptographic standards of ANSI and error correction code (Chen authors, 2016). The industry uses Elliptic curve groups over the large finite fields of GF(2m) and GF(p), Koblitz EC groups in GF(2m) (Koblitz, 1987) faster than GF(p). The operation modulo is using binary irreducible polynomial in finite field (Scott, 2007; Hasan authors, 1992) that is suitable for resource-constrained systems, such as cellphone, networked wireless sensors, and smart cards. The most efficient and secure cryptographic system in use today is known as elliptic curve cryptography (ECC) and is based on the concept of elliptic curves built over Galois fields (Savas & Ko_c, 2010). NIST recommended curves: Koblitz GF(2m), where m is 163, 233, 283, 409, and 571. Elliptic curves are a type of cubic equation of the form $y^2=x^3+ax+b$, where a and b represent coefficients. When elliptic curves are operation in Galois Field, the points on the curve can be form an Abelian group making it operations to addition of two points on the curve, or the point doubling. ECC encryption and decryption data in Galois Field that is popular the form $y^2+xy=x^3+ax^2+b$, where a and b represent coefficients. The operation on elliptic curves is scalar multiplication (Ansari & Hasan, 2011) which refers to multiplying a Point P by an integer k, resulting in the Point k*P scalar multiplication not only dominates the

execution time of ECC algorithms, but it is also essential to the security in systems. In GF(2m), the addition and the subtraction are the same XOR instruction in the processor. However, both multiplication and inversion operations are complicate in cryptographic systems; they are due to finite field of size m (Kobayashi & Takagi, 2008; Jing authors, 2006; Luo authors, 2012; Wang authors, 1983; Mahboob & Ikram, 2005; Brwon, 1971). The multiplication and inverse are required while using the Diffie-Hellman key exchange protocol on an elliptic curve (Dong & Li, 2008), as specified in ANSI X9.62 is required many multiplications and inverses. Therefore, to develop efficient arithmetic operations have high-speed computation that needs for using available technologies. In the past, the multiplication algorithm can be used to look-up tables, which have proposed in (Mahboob & Ikram, 2005). The lookup table method is pre-compute to reduce the number of operations required during the computation through the pre-computation and to reduce the effective computation time for multiplication by suitable width of the registers of the processor to achieve higher computation speed. The new algorithm has two properties: First, it utilizes the dynamic lookup table by precomputing input data and save memory. Second, it uses modified Horner rule for iteration loop and can determine the table entries quickly. The operation of inverse usually utilizes the polynomial modular mathematic called Euclidean algorithm (Brwon, 1971; Dong & Li, 2008), in which is hard to be known the worst cases execution time. The Fermat's Little Theorem also uses to compute inverse because the worst cases execution time can found. The adaptation of Itoh-Tsujii method (Guajardo & Paar, 2002) for standard basis, particularly for Optimal Extension Fields has been effective in achieving fast inversion. However, despite recent improvements, inversion is still the slowest operation in elliptic curve implementations (Agnew authors, 1993; Choi authors, 2002; Kumar, 2006). Finial, this issue is addressed by proposing multiplication methods that the execution time is faster than which measured for others multiplication execution time and it can be speed up establish a shared security over an insecure channel, Elliptic curve Diffie–Hellman (ECDH) (Diffie & Hellman, 1976).

2. PRELIMINARIES

Let $A(x) = \sum_{i=0}^{m-1} a_i x^i$ and $B(x) = \sum_{i=0}^{m-1} b_i x^i$ be polynomial over the finite field $GF(2^m)$, where $a_i, b_i \in \{0, 1\}$. The multiplication in the finite field is defined as follows:

$$C(x) \equiv A(x) \cdot B(x) \bmod F(x), \tag{1}$$

where the polynomial $F(x)$ is the irreducible polynomial. There is common method utilizing Horner rule for computing multiplication, which is rewritten (1) in the following form, where the $B(x)$ polynomial is represented as binary vector $B = (b_{m-1}, b_{m-2}, \ldots, b_0)$, where $b_i \in GF(2)$.

$$C(x) \equiv (\cdots((a_{m-1}B)x + a_{m-2}B)x + \cdots + a_1 B)x + a_0 B \bmod F(x). \tag{2}$$

The ECC with multiplication is irreducible polynomial $f = x^m + x^{m-1} + \cdots + x^2 + x + 1$. In (1), the Russian Peasant method can be written as a function in C programming as follows:

Russian Peasant method

```
GFM(a, b){
c = 0;
cbit=1<<m;
for  i = 0; i < m;  i=i+1
   if (b & 1)
     c ^= a;
   if (a & cbit)
     a = (a << 1) ^ f;
   else
     a <<= 1;
```

```
  b >>= 1;
  }
 return c;
}
```

The inverse element of A in finite field $GF(2^m)$ is derived by Fermat's Little Theorem, which is given by

$$A^{-1} = A^{2^m-2} = A^{2^{m-1}+2^{m-2}+\cdots+2}. \tag{3}$$

Many multiplications are required in common method of the inverse, which need 2^m-3 multiplications for calculating inverse element. In others word, it need more computation time for computing inverse but using number theory can reducing multiplication that is only m-1 multiplications. The inverse operation also can use the extended Euclidean algorithm.

$$F(x)s(x) + A(x)t(x) = 1, where\ A(x)t(x) \equiv 1 \bmod F(x). \tag{4}$$

Thus, the remainder of division $t(x)$ by mod $F(x)$, where $F(x)$ is irreducible polynomial, that is the multiplicative inverse of $A(x)$ mod $F(x)$. The compute the inverse element over finite field $GF(2^m)$ and the extended Euclidean algorithm also can use for computing multiplicative inverse in finite field. However, Euclidean division operation is hard to know the maximum execution time when the number is large. Points P=(x_1, y_1) and Q=(x_2, y_2) on the curve, assume $x_1 \neq x_2$. Let $y = \lambda x + d$ be the line that intersects P and Q, the value of the λ slope needs calculating inverse.

$$f(x,y) = y^2 + xy = x^3 + ax^2 + b$$
$$E_{2^m}(a,b) \to f(x,y), GF(2^m)$$

If $\neq Q$, $P + Q = R = (x_3, y_3)$

$$\begin{aligned} \lambda &= \frac{y_2 + y_1}{x_2 + x_1} \\ x_3 &= \lambda^2 + \lambda + x_1 + a \\ y_3 &= y_1 + (x_1 + x_3)\lambda + x_3 \end{aligned} \tag{5}$$

If $P = Q$, $P + Q = R = 2P = (x_3, y_3)$

$$\begin{aligned} \lambda &= x_1 + \frac{y_1}{x_1} \\ x_3 &= \lambda^2 + \lambda + a \\ y_3 &= y_1 + (x_1 + x_3)\lambda + x_3 \end{aligned} \tag{6}$$

For example, $GF(2^3), p(x) = x^3 + x + 1$.

$E_{2^3}(\alpha^3, \alpha^0) \to f(x,y) \to y^2 + xy = x^3 + \alpha^3 x^2 + \alpha^0$.

Let P and Q be curve point, where $P = (x_1, y_1), Q = (x_2, y_2)$.

$R = (x_3, y_3)$, the point of R is equal P point addition Q point.

We know that the slope is $\lambda = \frac{y_2 + y_1}{x_2 + x_1}$. The function of line is $y = \lambda x$.

3. IMPLEMENTATION OF MULTIPLICATION IN LARGE FIELDS *GF*(2^m).

Using a two-term polynomial $a_l x + a_w$ is designed for finite field multiplication, where $a_l, a_w \in GF(2)$. Let $q \equiv m \mod 2$, where the value of the remainder q is either 0 or 1. Therefore, the computation of $A(x) = \sum_{i=0}^{m-1} a_i x^i$ can be replaced by the following equation:

$$A(x) = \left(\sum_{i=0}^{\lfloor (m-2)/2 \rfloor - 1} (a_{m-2i-1}x + a_{m-2i-2}) x^{(m-2i-2)} + a_{q+1}x + a_q \right) x^q + q \cdot a_0, \tag{7}$$

A substitution of Equation (7) into Equation (1) yields,

$$C(x) \equiv \left(\left(\sum_{i=0}^{\lfloor (m-2)/2 \rfloor - 1} (a_{m-2i-1}Bx + a_{m-2i-2}B) x^{(m-2i-2)} + a_{q+1}Bx + a_q B \right) x^q + qB \cdot a_0 \right) \mod F(x). \tag{8}$$

A two-term polynomial $a_{m-2i-1}Bx + a_{m-2i-2}B$ can be pre-computed in order to generate a table of values. To see this, let $L(a_l, a_w) = a_l Bx + a_w$ be an array as lookup table, where $l = m - 2i - 1$ and $w = m - 2i - 2$. Therefore, a substitution of $L(a_l, a_w)$ into Equation (8) yields $C(x) \equiv \left(\left(\sum_{i=0}^{\lfloor (m-2)/2 \rfloor - 1} L(a_l, a_w) x^w + L(a_{q+1}, a_q) \right) x^q + q \cdot a_0 B \right) \mod F(x)$. Note that $\lfloor x \rfloor$ denotes the largest integer less than or equal to x. This implies that $\sum_{i=0}^{\lfloor (m-2)/2 \rfloor - 1} L(a_l, a_w) x^w$ can be represented by the use of Horner's rule form. It becomes

$$\sum_{i=0}^{\lfloor (m-2)/2 \rfloor - 1} L(a_l, a_w) x^w = (\cdots((L(a_{m-1}, a_{m-2})x^2 + L(a_{m-3}, a_{m-4}))x^2 + L(a_{m-5}, a_{m-6}))x^2 + \cdots + L(a_{m-2(\lfloor (m-2)/2 \rfloor - 1)-1}, a_{m-2(\lfloor (m-2)/2 \rfloor - 1)-2}))x^2. \tag{9}$$

The product of $L(a_l, a_w)$ and x^2 is required module $F(x)$ that can be reduce polynomial degree by a module lookup table *M*. The lookup table *M* is by making the polynomial $F(x)$. An irreducible polynomial $F(x)$ is a trinomial x^m+x^k+1 or a binary pentanomial $x^m+x^k+x^{k1}+x^{k2}+1$ (Hankerson authors, 2004). First, the modulo operation with lookup table is in term of the irreducible polynomial $F(x)=f_m x^m+f_k x^k+f_{k1}x^{k1}+f_{k2}x^{k2}+1$, where f_i is belong to {0, 1}, for computing modulo polynomial. Let $f= f_k x^k+f_{k1}x^{k1}+f_{k2}x^{k2}+1$ are the value of polynomial. Scheme 1, in step 2 needs table *M* to make *L* dynamic lookup table. The table *M* is as shown in Table 1.

Note that, the reduction table method requires that $m-k \geq wb$, where wb is the number of terms polynomial in polynomial $A(x)$. If $m-k$ is less than wb, the reducing polynomial f to make lookup table cause polynomial B is dependent. It can't use lookup table *M* for reducing the polynomial degree. The pre-computing reduction table is shown in Table 1.

Table 1: Pre-computing reduction *M* table

$M[a_i, a_j]$	reduction f	Result
M[0, 0]	(0f << 1)+0f	0
M[0, 1]	(0f <<1)+1f	f
M[1, 0]	(1f <<1)+0f	f << 1
M[1, 1]	(1f <<1)+1f	M[1, 0]+ M[0, 1]

The lookup table *M* can be reduced polynomial degree, the leader coefficient of polynomial in degree *m*-1 multiplied by *x* is become of degree *m* that meaning needs the table *M* to reduce the polynomial degree. The multiplication by *x* is represented as binary form shift left one position (i.e., *B*=1101, *B* << 1=1010). The terms $a_i Bx + a_j B$ can combinations of two bits for computing dynamic lookup table in Table 2.

Table 2: Two terms dynamic Lookup table

L[ai, aj]	aiBx modF(x)+ajB	Result
L[0, 0]	0Bx+0B	0
L[0, 1]	0Bx+1B	B
L[1, 0]	1Bx+M[0, bm-1]+0B	(B << 1)+M[0, bm-1]
L[1, 1]	1Bx+ M[0, bm-1]+1B	L[1, 0]+L[0, 1]

Where b_{m-1} is leader of the polynomial $B = (b_{m-1}, b_{m-2}, \cdots + b_0)$. Let $U = L(a_l, a_w)$ be binary vector $U = (u_{m-1}, u_{m-2}, \ldots, u_0)$, the $L(a_l, a_w)x^2 \bmod F(x)$ can be represented as $U << 2 + M(u_{m-1}, u_{m-2})$, where $U << 2$ is left-shift operation the binary vector (i.e., $U << 2 = (u_{m-3}, u_{m-4}, \ldots, u_0, 0, 0)$). Consequently, the scheme of multiplication for computing the two terms polynomial called Scheme 1, is proposed as follows:

Scheme 1: Multiplication Using Look up Table (LUT) Computation

1) Compute $q \equiv m \mod 2$, where *m* is the element of bit sizes.
2) Set the initial message C to be zero.
3) To make reduction $M(a_l, a_w)$ table as shown in Table 1
4) Computing $L(a_l, a_w)$ table as shown in Table 2.
5) **for** *i* = 0 to $\left\lfloor \frac{(m-2)}{2} \right\rfloor$-1 **do**
6) Read bits a_{m-2i-1} and a_{m-2i-2} from the polynomial *A*(*x*)
7) $U \leftarrow C + L(a_{m-2i-1}, a_{m-2i-2})$
8) $C \leftarrow (U << 2) + M(u_{m-1}, u_{m-2})$
9) **end for** *i*
10) $U \leftarrow C + L(a_{q+1}, a_q)$
11) $C \leftarrow (U << q) + M(qu_{m-1}, qu_{m-2}) + q \cdot a_0 B$

Scheme 1, the lookup table require the memory of the size $2^2 \cdot m$ bits. For example, the *GF*(2^{163}) requires 4 LUTs. Those tables can be dynamic pre-calculated and can then be stored in memory. The simplest approach is to use a pair of (a_l, a_w), where a_l and a_w are bit value, for evaluating multiplication in LUTs derived from Scheme 1, which requires only 80 iterations for evaluating a multiplication.

If a three-term polynomial $a_l x^2 + a_w x + a_x$ is designed for computing multiplication, $q \equiv k \mod 3$, It can be divided into $\left\lfloor \frac{(m-3)}{3} \right\rfloor$ unites into *A*(*x*). In equation (1) can be replaced by following a three-term polynomial for multiplication. It is called Scheme 2 as follows:

Scheme 2: Multiplication LUT Computation

1) Compute $q \equiv m \mod 3$, where *m* is the element of bit sizes.
2) Set the initial the value of C to be zero.
3) To make reduction $M(a_l, a_w, a_x)$ table as shown in Table 3

4) Computing $L(a_l, a_w, a_x)$ table as shown in Table 4.
5) **for** i = 0 to $\left\lfloor \frac{(m-3)}{3} \right\rfloor$-1 **do**
6) Read elements a_{m-3i-1}, a_{m-3i-2} and a_{m-3i-3}
7) $U \leftarrow C + L(a_{m-3i-1}, a_{m-3i-2} + a_{k-3i-3})$
8) $C \leftarrow (U << 3) + M(u_{m-1}, u_{m-2}, u_{m-3})$
9) **end for** i
10) $U \leftarrow C + L(a_{q+2}, a_{q+1}, a_{q+3})$
11) $C \leftarrow U << q + M(0, u_{m-1}, u_{m-2}) + \mathrm{sgn}(q) \cdot (\sum_{j=0}^{q-1} a_j B x^j) \bmod F(x)$

Where $\mathrm{sgn}(q) = \begin{cases} 1 \text{ if } q > 0, \\ 0 \text{ if } q = 0. \end{cases}$

Table 3: Pre-computing reduction *M* table

$M[a_i, a_j]$	reduction f	Result
M[0, 0, 0]	(0f<<1)+0f	0
M[0, 0, 1]	(0f<<1)+1f	f
M[0, 1, 0]	(1f<<1)+0f	f<<1
M[0, 1, 1]	(1f<<1)+1f	M[0, 1, 0]+M[0, 0, 1]
M[1, 0, 0]	(1f<<2)+(0f<<1)+0f	f<<2
M[1, 0, 1]	(1f<<2)+(0f<<1)+1f	M[1, 0, 0]+M[0, 0, 1]
M[1, 1, 0]	(1f<<2)+(1f<<1)+0f	M[1, 0, 0]+M[0, 1, 0]
M[1, 1, 1]	(1f<<2)+(1f<<1)+1f	M[1, 0, 0]+M[0, 1, 1]

Table 4: Three term dynamic lookup table

$L[a_l, a_w, a_x]$	$a_l Bx^2 \bmod F(x) + a_w Bx \bmod F(x) + a_x B$
L[0, 0, 0]	0
L[0, 0, 1]	B
L[0, 1, 0]	(B<<1)+M[0,0, a_{m-1}]
L[0, 1, 1]	L[0, 1, 0]+L[0, 0, 1]
L[1, 0, 0]	(B<<2)+M[0, a_{m-1}, a_{m-2}]
L[1, 0, 1]	L[1, 0, 0]+L[0,0,1]
L[1, 1, 0]	L[1, 0, 0]+L[0,1,0]
L[1, 1, 1]	L[1, 0, 0]+L[0,1,1]

Scheme 2, the lookup table require the memory of the size $2^3 \cdot m$ bits. For example, the *GF* (2^{163}) requires 8 data in lookup table. Those tables can be dynamic pre-calculated data to store in memory. The simplest approach is to use three for (a_l, a_w, a_x), where a_l, a_w, and a_x are bit value, for precomputing data in Table 3 and in Table 4. Scheme 2 used Table 3 and Table 4 operation, which requires only 53 iterations for evaluating a multiplication, however, they would be increased time to make dynamic lookup table. Characteristic two in this is case due to work described in (Gaudry authors, 2000) *m* to be prime. According to Scheme 1 (i.e., using Table 1 and Table 2) and Scheme 2 (i.e., using Table 3 and Table 4), If the value of *m* is odd number, the multiplications can be designed as Algorithm 1.

So, Algorithm 1 can use reducing polynomial $f(x)$ in Table 5 and wd-term polynomial $A(x)$ in Table 6. These precomputing the table of the size is 2^{wd}, where wd is the number of terms polynomial (i.e., $a_{wd-1}x^{wd-1}+a_{wd-2}x^{wd-2}+\cdots+a_0$). The flowchart is as shown in Figure 1.

Table 5: Pre-computing 2^{wd} reduction table

$M[a_{wd-1},a_{wd-2},\ldots,a_0]$	reduction f	Result
M[0,...,0, 0, 0]	0+⋯+(0f<<1)+0f	0
M[0,...,0, 0, 1]	0+⋯+ (0f<<1)+1f	f
M[0,...,0, 1, 0]	0+⋯+ (1f<<1)+0f	f<<1
M[0,...,0, 1, 1]	0+⋯+ (1f<<1)+1f	M[0,...,0, 1, 0]+M[0,...,0, 0, 1]
M[0,...,1, 0, 0]	0+⋯+ (1f<<2)+ (0f<<1)+0f	f<<2
M[0,...,1, 0, 1]	0+⋯+ (1f<<2)+ (0f<<1)+1f	M[0,...,1, 0, 0]+M[0,...,0, 0, 1]
M[0,...,1, 1, 0]	0+⋯+ (1f<<2)+ (1f<<1)+0f	M[0,...,1, 0, 0]+M[0,...,0, 1, 0]
⋮	⋮	⋮
M[1,...,0, 0, 0]	(1f<<wd-1)+⋯+0f	f<<wd-1
⋮	⋮	⋮
M[1,...,1, 0, 1]	(1f<<wd-1)+⋯+(0f<<1)+1f	M[1,...,0, 0, 0]+M[0,...,1, 0, 1]
M[1,...,1, 1, 0]	(1f<<wd-1)+⋯+(1f<<1)+0f	M[1,...,0, 0, 0]+M[0,...,1, 1, 0]
M[1,...,1, 1, 1]	(1f<<wd-1)+⋯+(1f<<1)+1f	M[1,...,0, 0, 0]+M[0,...,1, 1, 1]

Table 6: 2^{wd}-term polynomial for dynamic lookup table

$L[a_{k_{wd-1}},\ldots,a_{k_2},a_{k_1},a_{k_0}]$	$a_{k_{wd-1}}Bx^{wd-1} \bmod F(x) + \cdots + a_{k_1}Bx \bmod F(x) + a_{k_0}B$
L[0,...,0, 0, 0]	0
L[0,...,0, 0, 1]	B
L[0,...,0, 1, 0]	(B<<1)+M[0,...,0, 0, b_{m-1}]
L[0,...,0, 1, 1]	L[0,...,0, 1, 0]+L[0,...,0, 0, 1]
L[0,...,1, 0, 0]	(B<<2)+M[0, a_{m-1}, a_{m-2}]
L[0,...,1, 0, 1]	L[0,...,1, 0, 0]+L[0,...,0, 0, 1]
L[0,...,1, 1, 0]	L[0,...,1, 0, 0]+L[0,...,0, 1, 0]
⋮	⋮
L[1,...,0, 0, 0]	(B<<wd-1)+M[0, b_{m-1}, b_{m-2},..., b_{m-wd}]
⋮	⋮
L[1,...,1, 0, 1]	L[1,...,0, 0, 0]+L[0,...,1, 0, 1]
L[1,...,1, 1, 0]	L[1,...,0, 0, 0]+L[0,...,1, 1, 0]
L[1,...,1, 1, 1]	L[1,...,0, 0, 0]+L[0,...,1, 1, 1]

Algorithm 1: Multiplication dynamic Lookup table computation

GFM (***A***, ***B***):

1) Compute $q \equiv m \mod wd$
2) Set the initial the value of C to be zero
3) To make reduction $M(a_{k_0},a_{k_1},\ldots,a_{k_{wd-1}})$ table as shown in Table 5
4) To make dynamic lookup table $L(a_{k_0},a_{k_1},\ldots,a_{k_{wd-1}})$ as shown in Table 6
5) **for** i = 0 to $\left\lfloor \frac{(m-wd)}{wd} \right\rfloor$-1 **do**
6) Read elements a_{m-3i-1}, a_{m-3i-2},..., and $a_{m-3i-wd}$

7) $U \leftarrow C + L(a_{m-3i-1}, a_{m-3i-2} + \cdots + a_{k-3i-wd})$
8) $C \leftarrow (U << wd) + M(u_{m-1}, u_{m-2}, \ldots, u_{m-wd})$
9) **end for** *i*
10) $U \leftarrow C + L(a_{q+wt-1}, a_{q+wt-2}, \ldots, a_q)$
11) $C \leftarrow U << q + M(0,..,0,u_{m-1}, u_{m-2}, \ldots, u_{m-q}) + (\sum_{j=0}^{q-1} a_j B x^j) \bmod F(x)$
12) Return ***C***

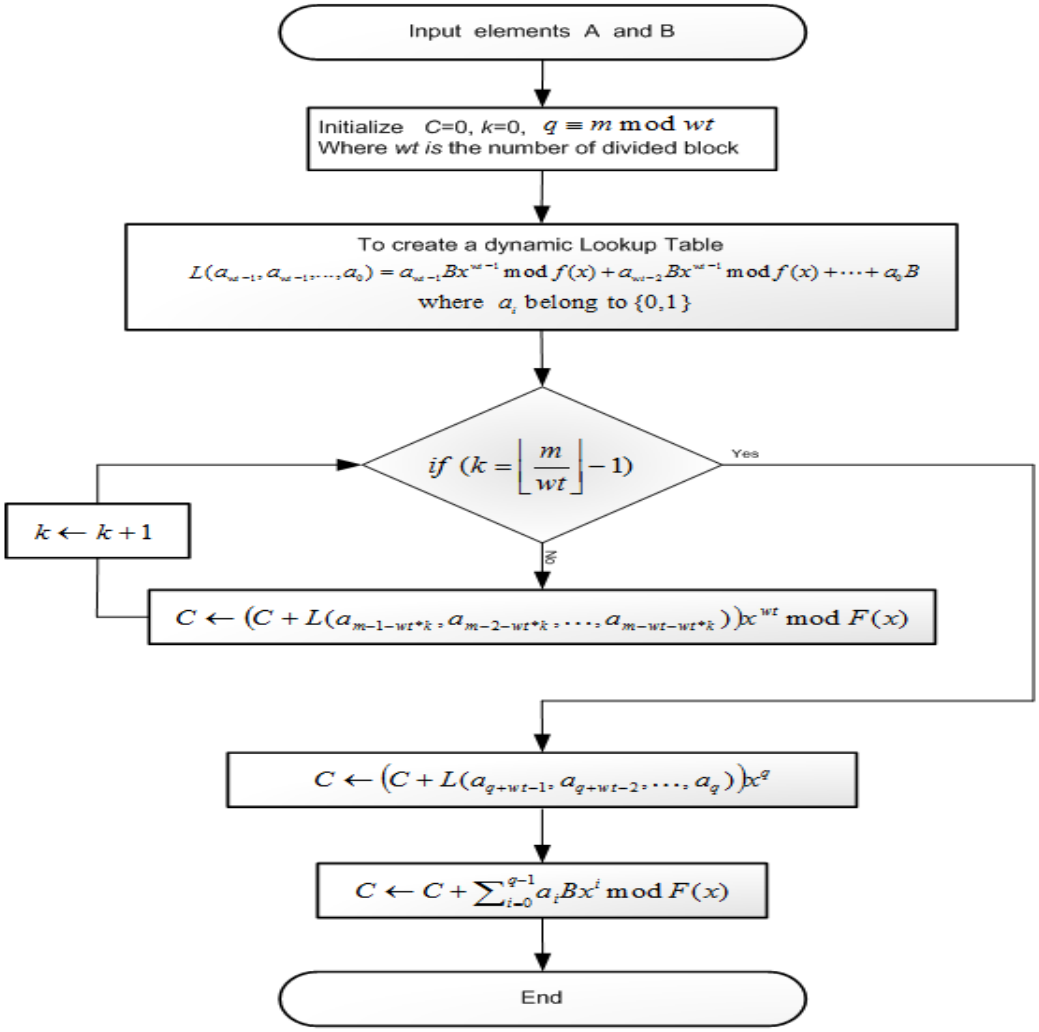

Figure 1: the flowchart is large size multiplication operation in $GF(2^m)$

4. RESULTS AND DISCUSSIONS

Regarding the efficiency of the proposed algorithm for encryption and decryption evaluation for EEC, a software simulation is performed on an Intel® Core™ i5 at 3.07 GHz Windows 7 PC using C++ program. The element in $GF(2^m)$ is implemented using an unsigned character data type. For any two elements in the finite field, the multiplication is described in (Genser *et al*. 2009), and the addition operation is implemented directly by using the bitwise C++ XOR. The multiplications size of *m* evaluating time of the 163, 233, 283, 409, and 571 listed in Table 7. The computational time of these methods over 100,000 times of message computations is listed in Table 8.

Table 7: The multiplication of the making table time with different finite filed

m	wd-term polynomial, time (ns)						
	1	2	3	4	5	6	7
163	0	1.5	1.6	1.5	6.3	10.9	20.3
233	0	1.5	1.6	3.1	6.2	12.5	23.4
283	0	1.5	1.6	4.7	7.8	15.6	31.2
409	0	1.5	3.1	4.7	10.9	20.3	39.0
571	0	1.6	3.1	6.2	12.5	25.0	49.9

Table 8: The multiplication of the looping time with different finite filed

m	wd-term polynomial, time (ns)						
	1	2	3	4	5	6	7
163	71.7	34.4	23.3	20.6	13.9	11	10.9
233	143.5	72	48.3	35.9	29.7	24.9	21.8
283	201.0	8.3	65.4	48.3	40.6	32.7	24.9
409	449.2	232.5	148.3	113.9	89.0	74.8	62.4
571	703.5	348.1	235.2	174.8	138.8	115	101.4

Table 9: The computing time of the multiplication with different finite field

m	Russian Peasant method	Diversity look table method wd-term polynomial, time (ns)						
		1	2	3	4	5	6	7
163	69.5	71.7	35.9	24.9	22.1	20.2	21.9	31.2
233	139.2	143.5	73.5	49.9	39.0	35.9	37.4	45.2
283	194.9	201.0	99.80	67.0	53.0	48.4	48.3	56.1
409	435.7	449.2	234.0	151.4	118.6	99.9	95.1	101.4
571	682.4	703.5	349.7	235.5	181.0	151.3	140.0	151.3

Table 10: The memory usage

m	Russian Peasant method needs memory size (bytes)	Diversity look table method wd-term polynomial needs memory size (bytes)						
		1	2	3	4	5	6	7
163	0	0	163	326	652	1304	2608	5216
233	0	0	223	446	892	1784	3568	7136
283	0	0	283	566	1132	2264	4528	9056
409	0	0	409	818	1636	3272	6544	13088
571	0	0	571	1142	2284	4568	9136	18272

Algorithm 1, the multiplication compute with m=163 and wd=5 that can be approximately 70% faster than Russian Peasant method. For some field, the computations for the multiplication had been executed over 100,000 data for testing. The large field m = 163 and wd=2 the proposed is used to compute multiplication, which requires 4+83=87 Left Shift (i.e., <<) operand and 1+83*2=164 XOR (i.e., ^) operand. The inverse computing process is required the number of multiplications and square are the number of m. The multiplication can be applying Algorithm 1 to evaluate scalar multiplication in Elliptic Curve. Algorithm 1 reducing time performance is better than Russian Peasant method that shown in Figure 2.

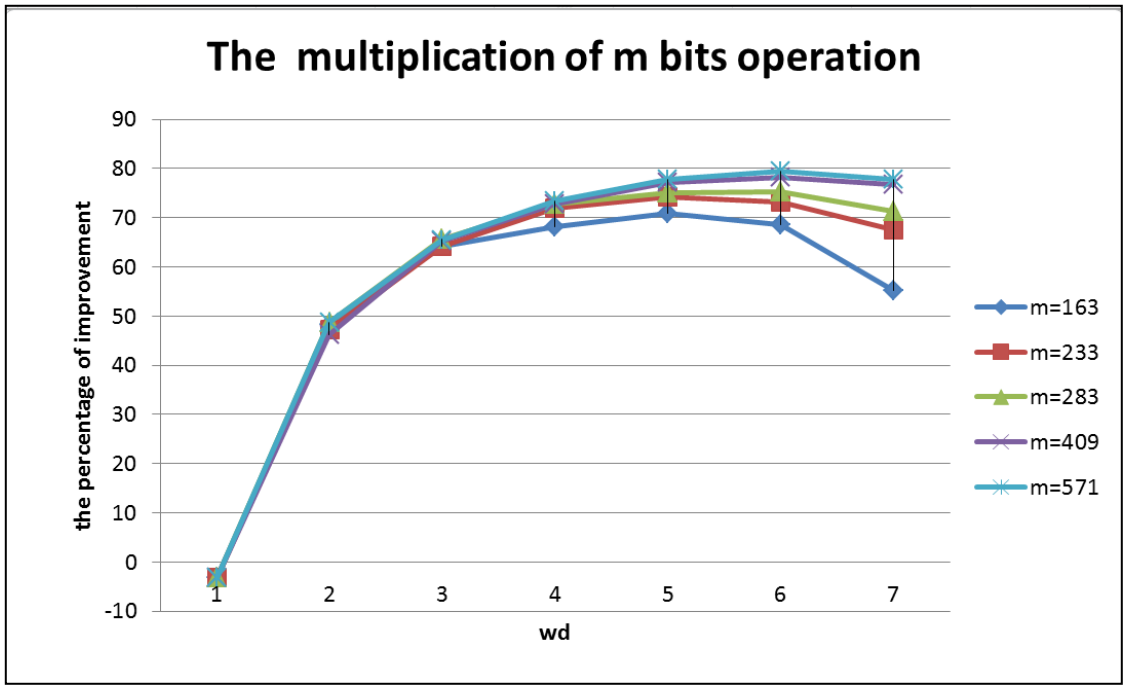

Figure 2: The multiplication of *m* bits operation in finite field

5. CONCLUSIONS AND RECOMMENDATIONS

In this paper, dynamic lookup table method using encryption and decryption in the ECC is presented. In Figure 2, Algorithm 1 is actually faster than Russian Peasant method, where $wd > 1$ in all instances m bits. The proposed multiplication method also can perform quickly inverse operation. If memory consumption in embedded systems is an acceptable range, the proposed method can be readily adaptable for speeding up and memory used the point multiplication in ECC. Thus, Algorithm 1 can use in different the value of wd to divide the polynomial $A(x)$ for encryption and decryption that can save a lot of the memory in Table 10 when the value of wd is small. It is evident that Algorithm 1 is really suitable for software applications in embedded system.

SOURCES OF FUNDING

None.

CONFLICT OF INTEREST

None.

ACKNOWLEDGMENT

None.

REFERENCES

[1] Y. H. Chen, C. F. Huang, J. Chang, Decoding of binary quadratic residue codes with hash table, IET Communications, Vol. 10, No. 1, 2016, 122-130.
[2] N. Koblitz, Elliptic curve cryptosystems, Mathematics of Computation, Issue 48, 1987, 203-209.
[3] M. Scott, Optimal irreducible polynomials for $GF(2^m)$ arithmetic. Cryptology ePrint Archive, Report 2007/192.

[4] A. Hasan, M. Z. Wang, and V. K. Bhargava, Modular Construction of Low Complexity Parallel Multipliers for a Class of Finite Fields GF(2^m), IEEE Trans. Comput., Vol. 41, No. 8, 1992, 962-971.
[5] E. Savas and C. Ko_c, Finite Field Arithmetic for Cryptography, IEEE Journals, Circuits and Systems Magazine, Vol. 10, No 2, 2010, 40-56.
[6] B. Ansari and M. Hasan, High-Performance Architecture of Elliptic Curve Scalar Multiplication, IEEE Transactions on Computers, Vol. 57, No 11, 2011, 1443-1453.
[7] Kobayashi and N. Takagi, A Combined Circuit for Multiplication and Inversion in GF(2^m), IEEE Transactions on Circuits and Systems, Vol 55, No 11,2008, 1144-1148.
[8] Jing, J. Chen, Z. Chen and Y. Chen, Low Complexity Architecture for Multiplicative Inversion in GF(2m), IEEE Asia Pacific Conference on Circuits and Systems (APCCAS 2006), 2006, 1492-1495.
[9] Luo, J., Bowers, K. D., Oprea, A., and Xu, L. Efficient software implementations of large finite fields GF (2^n) for secure storage applications. ACM Transactions on Storage (TOS), Vol 8, No1, 2012, 2
[10] C. C. Wang, T. K. Truong, H. M. Shao and L. J. Deutsch, VLSI Architectures for computing Multiplications and Inverses in GF(2^m), TDA Progress Report 42-75, 1983, 52-63.
[11] Mahboob and N. Ikram, Lookup table-based multiplication technique for GF(2^m) with cryptographic significance, IEE Proc.-Commun, Vol. 152, No. 6, 2005, pp. 965-974.
[12] W. S. Brwon, On Euclid's Algorithm and the computation of polynomial greatest common divisors, Journal of the Association for Computing Machinery, Vol. 18, 1971, 478-504.
[13] F. Dong and Y. Li, A Novel Shortest Addition Chains Algorithm Based on Euclid Algorithm, 4th International Conference on Wireless Communications, Networking and Mobile Computing, 2008, pp. 1-4.
[14] J. Guajardo and C. Paar, Itoh-Tsujii Inversion in Standard Basis and Its Application in Cryptography and Codes, Designs, Codes and Cryptography, Vol 25, No 2, 2002, 1573-7586.
[15] G. B. Agnew, R. C. Mullin, and S. A. Vanstone, An Implementation of Elliptic Curve Cryptosystems over $F_2{}^{155}$, IEEE J. Selected Areas in Comm., Vol. 11, 1993, 804-813.
[16] Y. Choi, H. W. Kim, and M. S. Kim, Implementation of elliptic curve cryptographic coprocessor over GF (2^{163}) for ECC protocols, in Proceedings of the 2002 International Technical Conference on Circuits/Systesm, Computers, and Communications, 2002, 674-677.
[17] S. Kumar Elliptic Curve Cryptography for constrained devices, Bochum Research Bibliography, 2006.
[18] W. Diffie and M. E. Hellman, New Directions in Cryptography, IEEE Trans. Inf. Theory, vol 22, 1976 664-654.
[19] D. Hankerson, A. Menezes, and S. Vanstone, Guide to Elliptic Curves Cryptogra-phy. Springer, 2004.
[20] P. Gaudry, F. Hess and N.P., Smart. Constructive and destructive facets of Weil descent on elliptic curves. Preprint, 2000.