A New Finite Field Multiplication Algorithm to Improve Elliptic Curve Cryptosystem Implementations

Abdalhossein Rezai*
Electrical Engineering, Ph.D. Student, Electrical and Computer Engineering Faculty, Semnan University
rezaie@acecr.ac.ir
Parviz Keshavarzi
Electrical Engineering, Associate Professor, Electrical and Computer Engineering Faculty, Semnan University
pkeshavarzi@semnan.ac.ir

Received: 21/Jan/2013 Accepted: 15/Jul/2013

Abstract

This paper presents a new and efficient implementation approach for the elliptic curve cryptosystem (ECC) based on a novel finite field multiplication in GF(2^m) and an efficient scalar multiplication algorithm. This new finite field multiplication algorithm performs zero chain multiplication and required additions in only one clock cycle instead of several clock cycles. Using modified (limited number of shifts) Barrel shifter; the partial result is also shifted in one clock cycle instead of several clock cycles. Both the canonical recoding technique and the sliding window method are applied to the multiplier to reduce the average number of required clock cycles. In the scalar multiplication algorithm of the proposed implementation approach, the point addition and point doubling operations are computed in parallel. The sliding window method and the signed-digit representation are also used to reduce the average number of point operations. Based on our analysis, the computation cost (the average number of required clock cycles) is effectively reduced in both the proposed finite field multiplication algorithm and the proposed implementation approach of ECC in comparison with other ECC finite field multiplication algorithms and implementation approaches.

Keywords: Computational Complexity, Network Security, Cryptography, Elliptic Curve Cryptosystem (ECC), Finite Field Multiplication, Scalar Multiplication.

1. Introduction

Elliptic curve cryptosystem (ECC) [1,2] has drawn more attentions in the network security issues due to its higher speed, lower power consumption and smaller key length in comparison with other existing public key cryptosystems [3,4]. These properties also make ECC more suitable for using in limited environments such as wireless sensor networks (WSNs) [3,5]. In ECC implementations, the total execution time and the power consumption are lowered by reducing the number of required clock cycles [3].

The most important operation in ECC is the scalar multiplication [6,7]. This operation is the most time-consuming operation and it takes 85% of the cryptosystem execution time [6,7].

Hardware implementation of ECC usually passes through three computational levels: Scalar multiplication, point operations and finite field operations that will be described in section 2.

There are many attempts to increase the efficiency of the elliptic curve scalar multiplication algorithm by increasing the computational efficiency of these three levels such as developing signed-digit scalar representation [8,9,10,11,12,13], sliding window method in scalar representation [7,9,12,13], parallel architecture in point operations [9], parallel architecture in finite field operations [6,14,15,16] and scalable modular multiplication [17,18]. A comprehensive review is also presented in [19].

High performance implementations of ECC depend heavily on the efficiency in the computation of finite field operations. Most popular finite fields which are commonly used in ECC are the prime fields GF(p) and the binary extension fields GF(2^m). Usually the binary extension fields GF(2^m) leads to a smaller and faster hardware [6,18].

In our previous work [20], the scalar multiplication is improved by using a novel finite field multiplication algorithm in GF(p). This paper presents a novel finite field multiplication algorithm in GF(2^m) based on the finite field multiplication in [20]. This new finite field multiplication uses a new signed-digit multiplier representation and multi bit scan-multi bit shift technique. Using this new signed-digit representation, the average Hamming weight of

* Corresponding Author

the multiplier is effectively reduced. Moreover, the zero chain multiplication is performed in only one clock cycle instead of several clock cycles. Therefore, the average number of required clock cycles (the computation cost) is considerably reduced in the proposed finite field multiplication algorithm. In addition, the proposed finite field multiplication algorithm is applied to the scalar multiplication algorithm in [9]. So, the computation cost of the elliptic curve scalar multiplication is also reduced considerably.

The rest of this paper is organized as follows: section 2 describes the recoding technique, the ECC over GF(2^m), the methods of the scalar multiplication and the finite field multiplication algorithm. The proposed implementation approach of ECC is presented in section 3. Section 4 evaluates the proposed algorithms. Finally, conclusion is given in section 5.

2. Preliminaries

2.1 The Recoding Technique

A signed-digit representation of an integer k_{CR} is the sequence of digits $k_{CR}=(d_m,d_{m-1},\ldots,d_1,d_0)_{SD}$ such that $k_{CR}=\sum_{i=0}^{m} d_i 2^i$ where $d_i \in \{-1,0,1\}$. Booth recoding [21] and canonical recoding (CR) [22,23] are two well-known conversions which can reduce the average Hamming weight of the integer representation. Algorithm 1 shows the canonical recoding algorithm.

```
Algorithm 1: The canonical recoding (CR) algorithm
Input: k= $(k_{m-1}k_{m-2}\ldots k_1k_0)_2$
Output: $k_{CR}$= $(d_md_{m-1}\ldots d_1d_0)_{SD}$
$c_0$:= 0;
For i = 0 to m-1
    $c_{i+1}:=\lfloor(k_i + k_{i+1} + c_i)/2\rfloor$;
    $d_i := k_i + c_i - 2c_{i+1}$;
Return $k_{CR}$;
```

In this algorithm, the input is the scalar k and the output is k_{CR}. It should be noted that, the CR representation, which is also called non-adjacent form (NAF), guarantees the minimal Hamming weight of the integer representation. The average Hamming weight of an m-bit canonical recoded integer is about $\frac{m}{3}$ [11, 22, 23].

2.2 ECC Over GF(2^m)

As described in the previous section, the hardware implementation of ECC usually involves three computational levels: scalar multiplication, point operations and finite field operations [6,20]. These three computational levels are shown in figure 1.

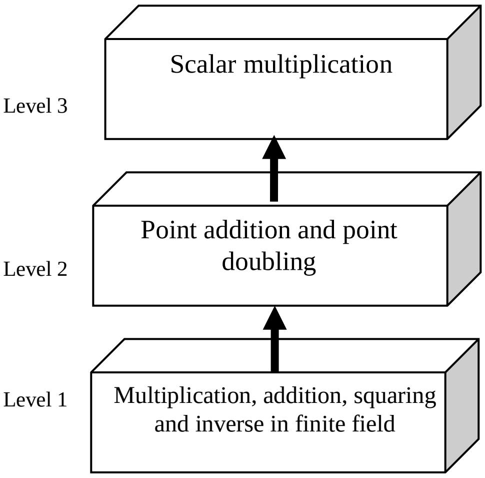
Level 3
Scalar multiplication
Level 2
Point addition and point doubling
Level 1
Multiplication, addition, squaring and inverse in finite field

Figure 1: The three-level model for elliptic curve scalar multiplication [6,20]

The scalar multiplication at the top of the hierarchy computes Q=kP with repeated point addition (Q=R+P) and point doubling (Q=2P) operations where k is a positive integer, and P and Q are elliptic curve points. The middle level of the hierarchy includes the point addition and point doubling operations, which are based on the coordinates used to represent the points. In the lowest level of the hierarchy, the finite field arithmetic includes four operations: finite field multiplication, finite field squaring, finite field addition and finite field inversion [6].

An elliptic curve E over GF(2^m) in affine coordinates is defined as the set of solutions of the reduced Weierstrass equation

$$\mathrm{E}: \quad y^2 + xy = x^3 + ax^2 + b \qquad (1)$$

where $a, b \in GF(2^m), b \neq 0$, together with the point at infinity O [24,25]. Note that for b=1, equation (1) shows especial curves which are commonly called Koblitz curves [12].

The point addition operation $\mathrm{Q} = (x_q, y_q) = \mathrm{R} + \mathrm{P} = (x_r, y_r) + (x_p, y_p)$ is defined by GF(2^m) operations as the following equations [24,25]:

$$\begin{cases} \lambda = (y_r + y_p)/(x_r + x_p) \\ x_q = \lambda^2 + \lambda + x_r + x_p + a \\ y_q = (x_p + x_q)\lambda + x_q + y_r \end{cases} \qquad (2)$$

Similarly, the point doubling operation $\mathrm{Q} = (x_q, y_q) = 2\mathrm{P} = 2(x_p, y_p)$ is defined by GF(2^m) operations as follows [24,25]:

$$\begin{cases} \lambda = x_p + \dfrac{y_p}{x_p} \\ x_q = \lambda^2 + \lambda + a \\ y_q = (x_p + x_q)\lambda + x_q + y_p \end{cases} \qquad (3)$$

These point operations involve finite field operations [24,25].

It should be noted that, the use of signed-digit representation for finite field operation in GF(2^m) is considered in [18]. The multiple-precision arithmetic for finite field operation in GF(2^m) is also investigated in [26].

2.3 The Methods of Scalar Multiplication

The most common method for performing an elliptic curve scalar multiplication (Q=kP) is the binary method which scans the bits of the scalar k either from left to right (the L2R binary method) or from right to left (the R2L binary method) [19,24]. The proposed implementation approach is based on the R2L binary method in GF(2^m) and its algorithm is shown in algorithm 2.

Algorithm 2: The R2L binary scalar multiplication algorithm

```
INPUT: $k=(k_{m-1}k_{m-2}\ldots k_0)_2$, P=(x,y);
OUTPUT: Q=(x',y')=kP;
1. Q ← 0;
2. For i= 0 to m-1 do
3.     If $k_i$=1 then Q ← Q+P;
4.     P←2P;
5. Return Q;
```

In algorithm 2, the inputs are the scalar k and the elliptic curve point P. The output is the elliptic curve point Q=kP. The computation cost in the binary multiplication method depends on the Hamming weight and the length of the binary representation of the scalar k (for m-bit scalar k, the binary multiplication method requires m point doubling operations and $\frac{m}{2}$ point addition operation on average).

The efficiency of the binary method may be enhanced by scanning w bits at a time as with the sliding window method [7,13] or reducing the Hamming weight as with the signed-digit recoding technique [10]. One of the efficient efforts to reduce the computation cost in ECC is the window scalar multiplication algorithm based on interleaving (IW algorithm) on Koblitz curves [9] which is shown in algorithm 3.

Algorithm 3:The window scalar multiplication algorithm based on interleaving (IW algorithm)[9]

```
INPUT: w; $k=(k_{m-1}k_{m-2}\ldots k_0)_2$; $P \in GF(2^m)$;
OUTPUT: Q=kP;
1. Use algorithm 3 in appendix [9] to compute ρ'=k partmod δ;
2. Use algorithm 4 in appendix [9] to compute $TNAF_w(\rho')= \sum_{i=0}^{l-1} u_i \tau^i$ ;
3. For $u \in U=\{1,3,5,\ldots,2^{w-1}-1\}$, let $Q_u \leftarrow 0$;
4. For i=l-1 to 0 do
   4.1. If $u_i \neq 0$ then
          Let u satisfy $a_u=u_i$ or $a_{-u}=-u_i$;
          If u>0 then $Q_u \leftarrow Q_u+P$;
          Else $Q_{-u} \leftarrow Q_{-u}-P$;
   4.2. P ← τP;
5. Compute $Q \leftarrow Q + \sum_{u \in U} u_i Q_u$ ;
6. Return Q;
```

The inputs of this algorithm are the scalar k, window width w, and elliptic curve point P. The output is the elliptic curve point Q=kP. Moreover, $\tau = \frac{\mu + \sqrt{-7}}{2}$, $\mu = (-1)^{1-a}$, $a = \{0,1\}$ and $\delta = \frac{\tau^m - 1}{\tau - 1}$ [9,12]. In the IW algorithm, the multiplication cost is reduced by using the sliding window method and the signed-digit representation (steps 1 and 2). In this algorithm, when $u_i \neq 0$, the point Q_u is computed in which u satisfies $a_u=u_i$ or, $a_{-u}=-u_i$ [9].

2.4 The Finite Field Multiplication Algorithm

The performance of ECC is primarily determined by the efficient realization of the arithmetic operations in the underlying finite field [6].

Modular addition in GF(2^m) is simple and relatively straight forward . As a result, it can be implemented by simply using XOR gates [14]. If projective coordinates are used for ECC, the inversion cost can be neglected because only one inversion operation is required to be performed at the end of the scalar multiplication. The modular squaring in GF(2^m) is simple and straight forward [6,14]. Therefore, the modular multiplication is the most important operation in ECC implementations.

The Montgomery modular multiplication algorithm [27] is widely used as an efficient algorithm [18,28]. Algorithm 4 shows the Montgomery modular multiplication algorithm for GF(2^m) [18]:

Algorithm 4: The Montgomery modular multiplication in GF(2^m)

```
Input: A(x),B(x),P(x),n;
 Output: C(x) =A(x).B(x) $x^{-n}$ mod P(x);
1. C(x)=0;
2. For i=0 to n-1
3. $q(x) = (c_0(x) + a_i(x).b_0(x)).p_0^{'}(x)(\bmod x^r)$ ;
4. $C(x) = (C(x) + a_i(x).B(x) + q(x).P(x)) / x^r$ ;
5. Return C(x)
```

The inputs of this algorithm are A(x), B(x), P(x) and n, where A(x), B(x) $\in$ GF(2^m), P(x) is the irreducible polynomial and n denotes the operand length. The output is C(x)=A(x).B(x)x^{-n} mod P(x). Moreover, r shows each digit length, $p_0^{'}(x) = p_0^{-1}(x)(\bmod x^r)$ and $a_i(x)$ shows ith digit of A(x). The output of this algorithm is computed in n-clock cycle. Therefore, it is a time- consuming operation.

3. The Proposed Implementation Approach of ECC

This section presents a novel and efficient implementation approach for the elliptic curve cryptosystem based on the parallel structure and a new and efficient finite field multiplication algorithm in GF(2^m).

3.1 Scalar Multiplication

The basic operations in all scalar multiplication algorithms are point addition and point doubling operations over an elliptic curve [20]. Using parallel structure for these point operations, the speed of the cryptosystem is increased considerably. We also used the scalar multiplication algorithm [9] to compute point addition and point doubling operations in parallel. This algorithm is shown in algorithm 3 and was described in section 2.3.

3.2 The Finite Field Arithmetic

In the finite field multiplication, zero multiplication results in zero, but this zero multiplication is performed and implemented per clock cycle. In addition, partial result is shifted one bit per clock cycle [29]. This section presents a new finite field multiplication algorithm in GF(2^m) based on a new signed-digit multiplier representation and multi bit scan-multi bit shift technique. This new finite field multiplication performs zero chain multiplication in only one clock cycle instead of several clock cycles. The proposed finite field multiplication algorithm is based on Montgomery modular multiplication algorithm in GF(2^m). The proposed algorithm is shown in algorithm 5:

```
Algorithm 5: The proposed finite filed multiplication in GF($2^m$)
Input: A(x), B(x), P(x) , n;
Output: C(x)= A(x).B(x) $x^{-n}$ mod P(x);
1. C(x)=0;
{Canonical recoding phase}
2. Compute D(x) by applying algorithm 1 to A(x);
parallel begin
    {partitioning phase}
    3.1. Building D*(x)=($u_{s-1}$(x)$u_{s-2}$(x)…$u_0$(x)) by applying CLNZ sliding window method to D(x);
    3.2. s= #D*(x) ;
    4. Compute and store table $u_i$(x).B(x)
parallel end
{multiplication phase}
5. For i = 0 to s-1
6.     C(x):= C(x) + $u_i$(x).B(x);
7.     q(x):= $P_0'(x)$ .C(x) mod $x^{l_i}$ ;
8.     C(x):= (C(x)+q(x).P(x))/ $x^{l_i}$ ;
9. Return C(x)
```

In this algorithm, the inputs are A(x), B(x), P(x) and n, where A(x), B(x) $\in$ GF(2^m), P(x) is the irreducible polynomial and m denotes the operand length. The output of this algorithm is C(x)=A(x).B(x)x^{-m} mod P(x). Moreover, $p_0'(x) = p_0^{-1}(x) \bmod x^{l_i}$, u_i(x) is the ith partition of D*(x), l_i is the ith partition length (i.e. the number of digits in ith partition) and s= #D*(x) is the number of partitions in the multiplier representation.

In step 2 of the proposed finite field algorithm, the canonical recoding algorithm is performed on the multiplier. Then the constant length nonzero (CLNZ) partitioning is performed on the signed-digit multiplier. Therefore, the average Hamming weight of the multiplier and thereby the average number of multiplication steps (or required clock cycles) in the finite field multiplication algorithm are reduced considerably. In algorithm 5, the CLNZ partitioning method scans the multiplier from the least significant digit to the most significant digit according to a finite state machine, which is shown in figure 2.

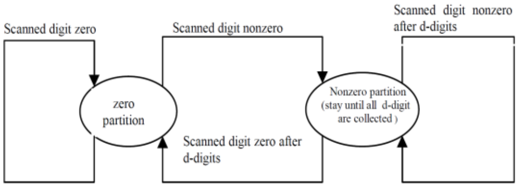
Scanned digit zero
Scanned digit nonzero
Scanned digit nonzero after d-digits
zero partition
Nonzero partition (stay until all d-digit are collected)
Scanned digit zero after d-digits

Figure 2: The finite state machine used in the CLNZ partitioning method

Using the CLNZ partitioning method, the zero partitions are allowed to have an arbitrary length, but the maximum length of the nonzero partitions should be the exact value (in figure 2, d digits). For example, for A(x) = $(011111111110001111111101)_2$, the canonical recoding of A(x) is $\mathrm{D(x)} = (01000000000\bar{1}0010000000\bar{1}01)_{CR}$ and for d=4, the partition formed will be as follows:

$$\mathrm{D^*(x)} = ((0001),(000000000),(\bar{1}001),(000000),(0\bar{1}01)) \cdot$$

As the least significant digit of the nonzero partition is either 1 or $\bar{1}$, the nonzero partition value is always an odd number. So, we only require pre-computation of u_i(x).B(x) for the odd number of u_i(x) in step 4 of the proposed finite field multiplication algorithm.

In the proposed finite field multiplication algorithm, step 4 is performed independently and parallel with steps 3.1 and 3.2. This parallel computation also increases the speed of the finite field multiplication algorithm.

The multiplication phase of the proposed finite field multiplication algorithm is performed s times. Recall that s denotes the number of partitions in the proposed multiplier representation. In each clock cycle of the multiplication phase of the proposed finite field multiplication algorithm, l_i bits of the multiplier and m-bit multiplicand are processed.

Figure 3 shows the block diagram of the hardware implementation of the proposed finite field multiplication.

In the proposed hardware implementation approach of the finite field multiplication, the new multiplier representation D*(x) makes multi bit scan possible, but the high-radix modular multiplication (k×m multiplier) is required in $u_i(x).B(x)$ and q(x).P(x) computation. In the proposed hardware implementation approach of the finite field multiplication, LUT1 and LUT2 are used for computing $u_i(x).B(x)$ and q(x).P(x) respectively. Thus, the high-radix partial multiplication problem in each clock cycle is also solved.

In addition, the modified (limited number of shifts) Barrel shifter is proposed to execute the required multi bit shift operation in a single clock cycle in step 8 of algorithm 5. The number of required shifts in ith clock cycle (l_i) is provided from the length of the ith digit of the new multiplier representation D*(x). These two properties imply the multi bit scan-multi bit shift technique. So, the zero chain multiplication and the required addition are performed in one clock cycle instead of several clock cycles.

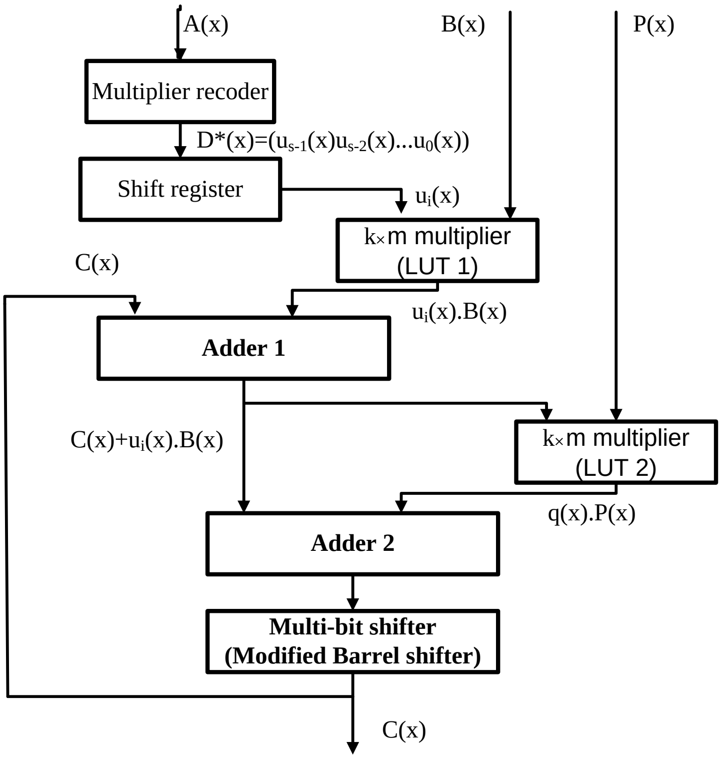

Figure 3: The block diagram of the proposed finite field multiplication

4. Evaluation

4.1 Evaluation of the Proposed Finite Field Multiplication Algorithm

In the proposed finite field multiplication algorithm, the CLNZ sliding window method is applied to the canonical recoded multiplier. So according to computation analysis of [30], the average Hamming weight of the multiplier is about $\frac{3m}{3d+4}$, where m denotes the multiplier length and d denotes the window width in the CLNZ partitioning method in the proposed finite field multiplication algorithm. Thus, the proposed finite field multiplication algorithm reduces the average number of multiplication steps by about:

$$1-\frac{\frac{6m}{3d+4}}{m}=1-\frac{6}{3d+4} \tag{4}$$

Table 1 shows the multiplication step (required clock cycle) improvement in the proposed finite field multiplication algorithm in comparison with Montgomery modular multiplication algorithm [27] for various d.

Table 1: Multiplication step improvement of the proposed finite field multiplication algorithm

d	Clock cycle improvement (%)	d	Clock cycle improvement (%)
2	40	7	76
3	53.8	8	78.6
4	62.5	9	80.6
5	68.4	10	82.4
6	72.7		

Based on our analysis which is shown in table 1, the proposed finite field multiplication algorithm reduces the average number of multiplication steps (required clock cycles) by about 40%-82.4% compared to Montgomery modular multiplication algorithm in GF(2^m) for d=2-10.

4.2 Evaluation of the Proposed Implementation Approach

According to the computational analysis of [9,20], the implementation approach of the traditional window NAF (TWN) scalar multiplication algorithm [12] will cost:

$$D+(2^{w-2}-1)A+\frac{\mathrm{m}}{w+1}A+\mathrm{m}D \qquad (5)$$

where D denotes the point doubling cost, A denotes the point addition cost, m denotes the operand length and w denotes the window width in the sliding window method in the scalar multiplication algorithm.

Moreover in the Karatsuba-Ofman method [6,14], the computation cost is computed from (5), but with different computation cost for the point addition and point doubling operations.

In addition, the implementation approach of the window scalar multiplication algorithm based on interleaving (IWN) [9] will cost:

$$\frac{\mathrm{m}}{w+1}A+\sum_{j=1}^{v}\frac{l_j}{w_j+1}A \qquad (6)$$

The proposed implementation approach of ECC is a combination of the proposed finite field multiplication algorithm and the IWN algorithm. So, the computation cost of the proposed implementation approach is computed from (6), but the cost of the point addition in the proposed implementation approach is reduced considerably based on table 1.

The point addition and point doubling operations have the same cost using affine coordinate, but the cost of the point addition operation is twice the cost of the point doubling operation using projective coordinate [9,24]. The computation cost of the implementation approaches in [6,9,12,14] and the proposed implementation approach are computed by analyzing (5) and (6) for various m, w and d. Figures 4-6 show the comparison of the computation cost of the proposed implementation approach with implementation approach in [6,9,12,14] for m=163 bit and various window width w using affine coordinate and projective coordinate for d=2,4,6,8 and 10.

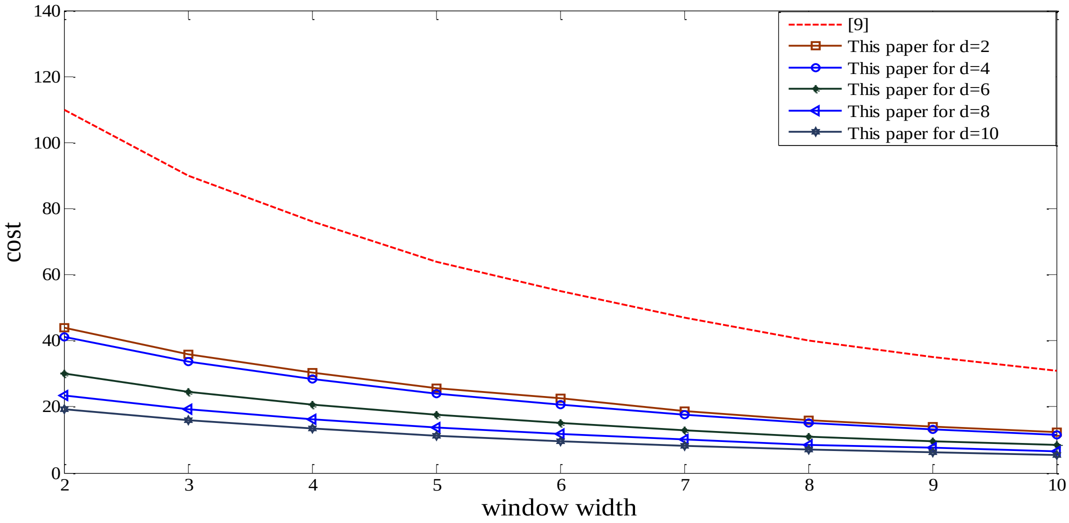

Figure 4: Comparison of the computation cost between the proposed implementation approach and the implementation approach in [9] using affine coordinate and projective coordinates

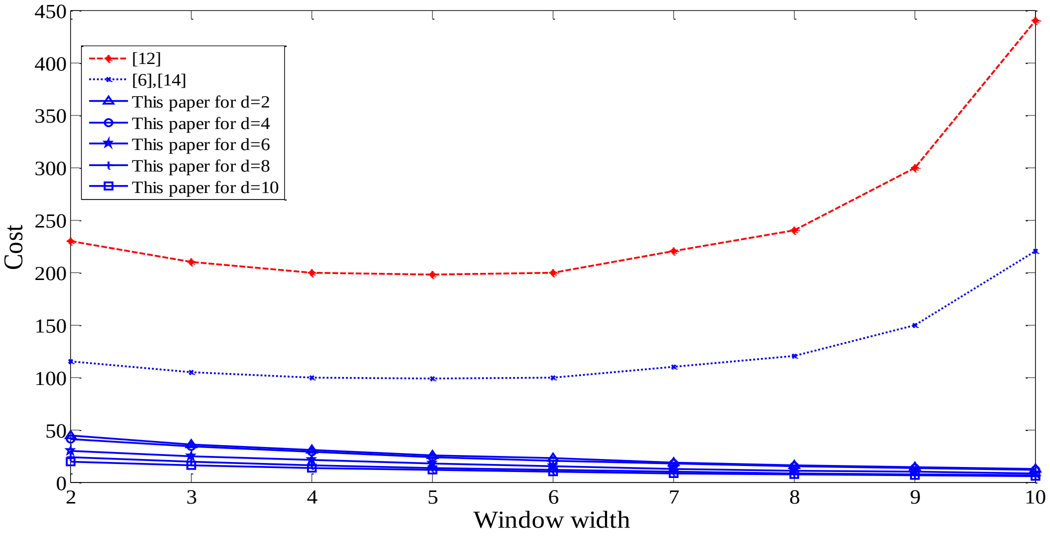

Figure 5: Comparison of the computation cost between the proposed implementation approach and the implementation approach in [6,12,14] using affine coordinate

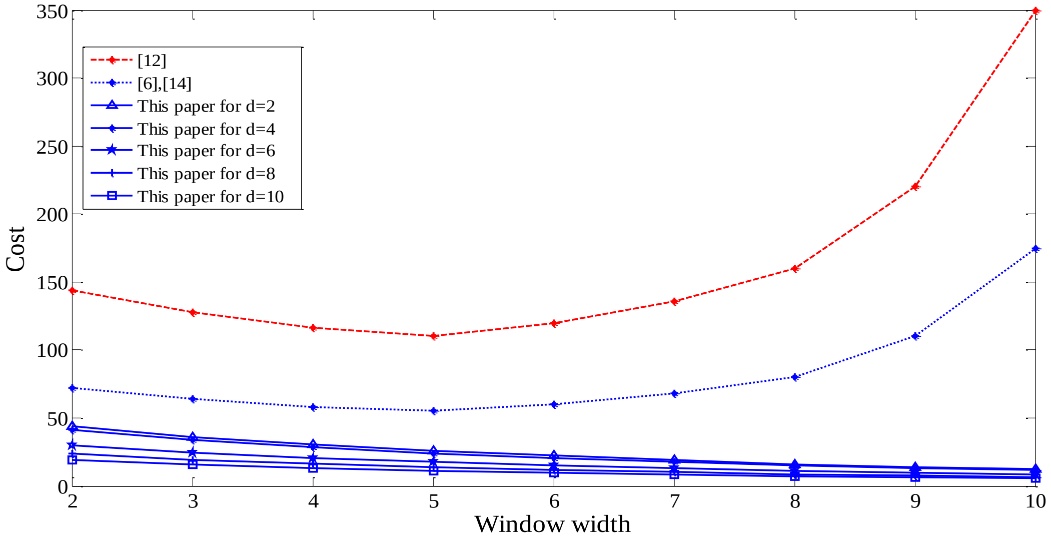

Figure 6: Comparison of the computation cost between the proposed implementation approach and the implementation approach in [6,12,14] using projective coordinate

As it is shown in figures 4-6, the computation cost of the proposed implementation approach is effectively reduced in comparison with the implementation approach in [6,9,12,14] where both window width in the scalar multiplication (w) and the window width in the proposed finite field multiplication (d) are varied from 2 to 10. Table 2 and figures 7-8 summarize the computation cost of the proposed implementation approach and the implementation approach in [6,9,12,14] for the operand length of 163, 193 and 233 in affine coordinate where w=4, d=8 and w=8, d=8.

Table 2: The comparative table for the computation cost using affine coordinate for d=8, w=4 and 8.

Operand length	Reference	Computation cost	
		w=4	w=8
163	[12]	199.6	245.1
	[6][14]	100	122.5
	[9]	65	36.1
	This paper	13.9	7.7
193	[12]	235.6	278.4
	[6][14]	117.8	139.1
	[9]	77.2	42.8
	This paper	16.5	9.2
233	[12]	283.6	322.9
	[6][14]	141.8	161.5
	[9]	93.1	51.4
	This paper	20	11

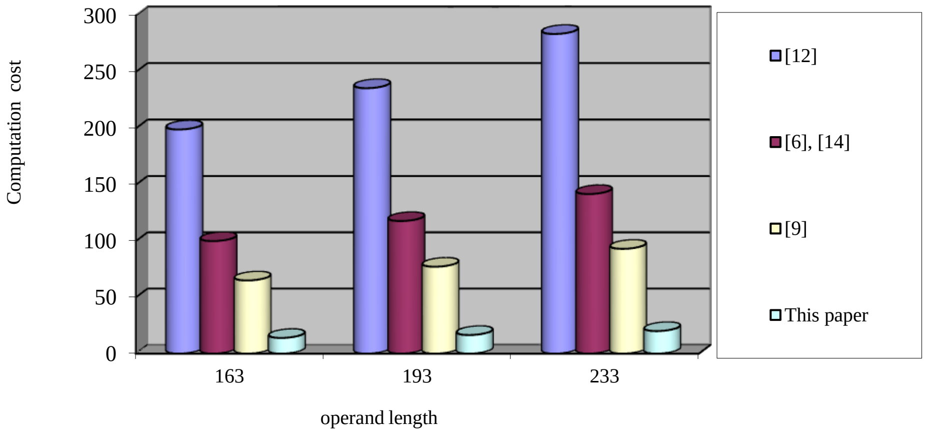

Figure 7: Comparison of the computation cost using affine coordinate for d=8, w=4

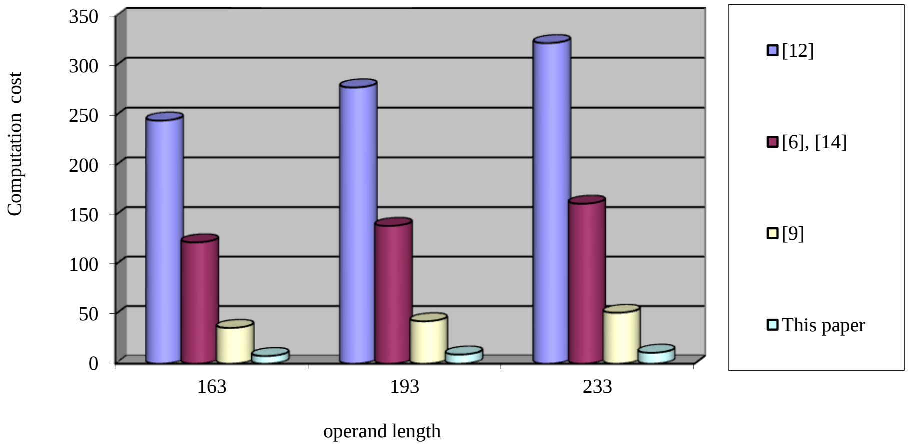

Figure 8: Comparison of the computation cost using affine coordinate for d=8, w= 8

Based on our analysis which is shown in table 2 and figures 7-8, the average computation cost of the proposed implementation approach is reduced by about 93%-96%, 86%-93% and 78.6% in comparison with the implementation approach in [12], [6] (and its extension in [14]) and [9] respectively for w=4, d=8 and w=8, d=8 using affine coordinate. Table 3 summarizes these improvements where the computation cost improvement is computed as follows:

$$\text{improvement}(\%) = (1 - \frac{\text{new cost}}{\text{old cost}}) \times 100 \qquad (7)$$

Table 3: The comparative table for the computation cost using projective coordinate for d=8

Reference	[12]		[6][14]		[9]	
Window width	w=4	w=8	w=4	w=8	w=4	w=8
Computation cost improvement (%)	93	96	86	93	78.6	78.6

As it is shown in (5), the computation cost in [12] has a multiplier as 2^w. So, by increasing the window width w, the computation cost in [12] is also increased.

In addition, table 4 and figures 9-10 summarize the computation cost of the proposed implementation approach and the implementation approach in [6,9,12,14] for the operand length of 163, 193 and 233 in projective coordinate where w=4, d=8 and w=8, d=8.

Table 4: The comparative table for the computation cost using projective coordinate for d=8, w=4 and 8.

Operand length	Reference	Computation cost	
		w=4	w =8
163	[12]	117.6	163.1
	[6][14]	58.8	81.6
	[9]	65.2	36
	This paper	13.9	7.7
193	[12]	138.6	181.4
	[6][14]	69.3	90.7
	[9]	77.2	42.8
	This paper	16.5	9.2
233	[12]	166.6	205.9
	[6][14]	83.3	103
	[9]	93.1	51.2
	This paper	20	11

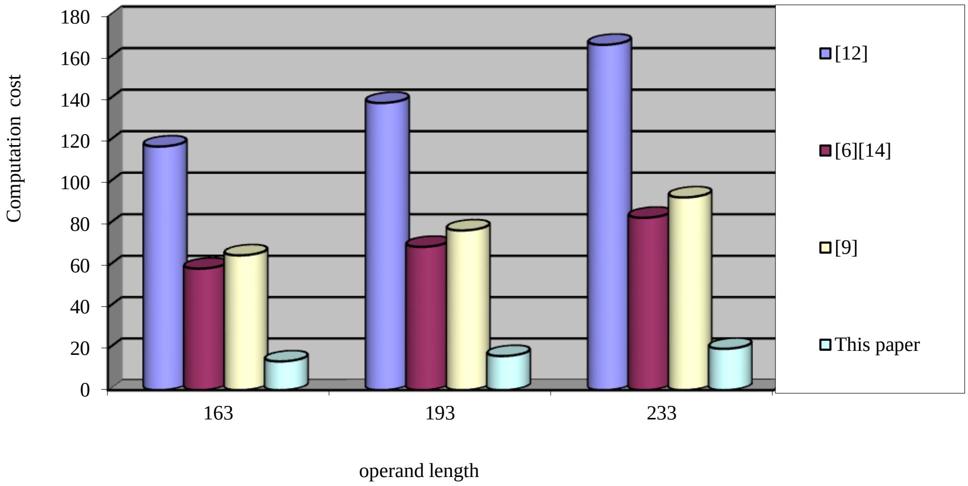

Figure 9: Comparison of the computation cost using projective coordinate for d=8, w=4.

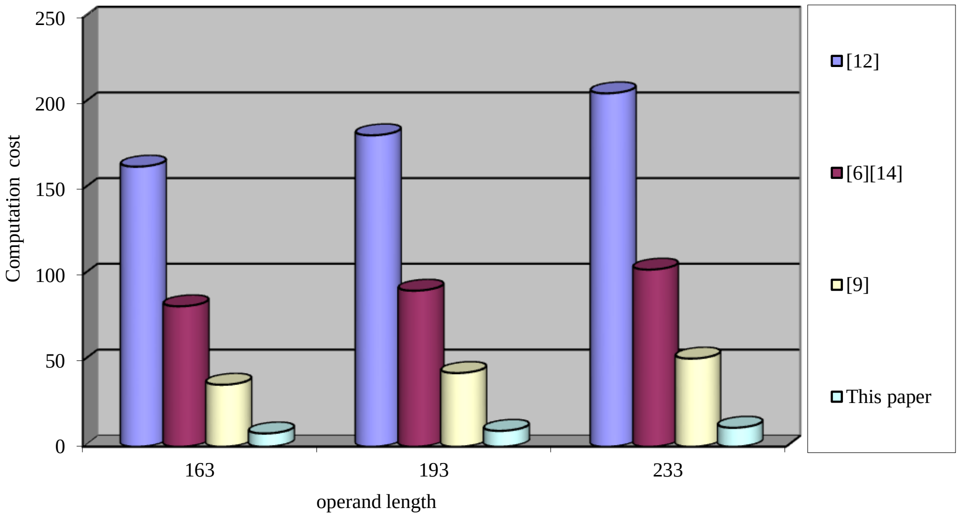

Figure 10: Comparison of the computation cost using projective coordinate for d=8 and w=8

As it is shown in table 4 and figures 9-10, the average computation cost of the proposed implementation approach is reduced by about 88%-95%, 76.1%-89.4% and 78.6% in comparison with the implementation approach in [12], [6] (and its extension in [14]) and [9] respectively using projective coordinate where w=4, d=8 and w=8, d=8. Table 5 summarizes these improvements.

Table 5: The comparative table for the cost using projective coordinate for d=8, w=4 and 8.

Reference	[12]		[6][14]		[9]	
Window width	w=4	w=8	w=4	w=8	w=4	w=8
Computation cost improvement (%)	88	95	76.1	89.4	78.6	78.6

As the computation cost of the point doubling operation using projective coordinate is half of the point addition operation, the computation cost improvement in comparison with [12] is reduced in projective coordinate compared to affine coordinate.

Therefore, using the proposed implementation approach for ECC, the efficiency of the computation cost of ECC is improved considerably.

5. Conclusion

In ECC implementation, the total execution time and the energy consumption is dependent on the required clock cycles for cryptosystem [3]. This paper presents a novel finite field multiplication algorithm in GF(2^m) based on a new signed-digit multiplier representation and multi bit scan-multi bit shift technique to reduce the required clock cycle in ECC. In this new finite field multiplication, the canonical recoding technique is used to increase probability of the zero bits in the multiplier. The CLNZ sliding window method is also applied to the signed-digit multiplier to reduce the average number of multiplication steps (required clock cycles) in the finite field multiplication algorithm. This new multiplier representation makes multi bit scan possible. The modified (limited number of shifts) Barrel shifter is also proposed to make multi-bit shift possible. Moreover, a new efficient implementation approach for the elliptic curve cryptosystem is presented by applying this new finite field multiplication to the scalar multiplication in [9]. In this new implementation approach, the point addition and point doubling operations are computed in parallel. In addition, both sliding window method and canonical recoding technique are used to reduce the computation cost considerably.

Our analysis shows that the computation cost of the proposed finite field multiplication algorithm is reduced by about 40%-82.4% in comparison with Montgomery modular multiplication algorithm for d=2-10. Moreover, the computation cost in the proposed implementation approach of the elliptic curve cryptosystem is reduced by about 88%-96%, 76%-93% and 78.6% in comparison with the implementation approach in [12], [6] (and its extension in [14]) and [9] respectively where w=4 and 8, and d=8.

References

[1] N. Koblitz, “Elliptic curve cryptosystem”, Mathematics of Computer, 1987, vol.48, pp.203-209.

[2] V. Miller, “Use of elliptic curves in cryptography”, in Proc. of advances in cryptology (CRYPTO), 1985, LNCS .218, 417–428.

[3] H. R. Ahmadi, and A. Afzali-kusha, “A low-power and low-energy flexible GF(p) elliptic-curve cryptography processor”, Journal of Zhejiang University-science C, 2010, Vol.11, No.9, pp.724-736.

[4] A. P. Fournaris, “Toward Flexible Security and Trust Hardware Structures for Mobile-Portable Systems”, IEEE Latin America Transactions, 2012, Vol.10, No.3, pp.1719-1722.

[5] H. Wang, and Q. Li, “Achieving distributed user access control in sensor networks”, Ad Hoc Networks, 2012, Vol.10, No.3, pp.272-283.

[6] N. Saqib, F. Rodriguez-Henriquez, and A. Diaz-perez, “A parallel architecture for fast computation of elliptic curve scalar multiplication over GF(2^m)”, in Proc. of the 18th IEEE. International parallel and distributed processing symposium, 4004, pp.144.

[7] P. Shah, X. Huang, and D. Sharma, “Sliding window method with flexible window size for scalar multiplication on wireless sensor network nodes”, in Proc. of the IEEE. International conference on wireless communication and sensor computing, 2010, pp.1-6.

[8] B. Qin, M. Li, F. Kong, and D. Li, “New left-to-right minimal weight signed-digit radix-r representation”, Computers and Electrical Engineering, 2009, Vol.35, No.1, pp.150-158.

[9] X. Yin, H. Zhu, and R. Zhao, “Window algorithm of scalar multiplication based on interleaving”, in Proc. of the IEEE. International conference on communications, circuits and systems, 2009, pp.318-321.

[10] P. Balasubramanian, and E. Karthikeyan, “Elliptic curve scalar multiplication algorithm using complementary recoding”, Applied mathematics and computation, 2007, Vol.190, No.1, pp.51-58.

[11] P. Balasubramaniam, and E. Karthikeyan, “Fast simultaneous scalar multiplication”, Applied mathematics and computation, 2007, Vol.192, No.2, pp.399-404.

[12] J. Solinas, “Efficient arithmetic on Koblitz curves”, Designs, codes and cryptography, 2000, Vol.19, No.2-3, pp.125-179.

[13] A. Rezai, and P. Keshavarzi, “CCS Representation: A new non-adjacent form and its application in ECC”, Journal of Basic and Applied Scientific Research, 2012, Vol.2, No.5, pp.4577-4586.

[14] S. Shohdy, A. Elsisi, and N. Ismail, “Hardware implementation of efficient modified Karatsuba multiplier used in elliptic curves”, International Journal of Network Security, 2010, Vol.11, No.3, pp.138-145.

[15] B. Ansari, and A. Hasan, “High-Performance architecture of elliptic curve scalar multiplication”, IEEE. Transactions on Computers, 2008, Vol.57, No.11, pp.1443-1453.

[16] Y. Dan, X. Zou, Z. Liu, Y. Han, and L. Yi, “High-performance hardware architecture of elliptic curve cryptography processor over GF(2^{163})”, Journal of Zhejiang University - Science A, 2009, Vol.10, No.2, pp.301-310.

[17] G. Orlando, and C. Paar, “A scalable GF(p) elliptic curve processor architecture for programmable hardware”, in Proc. of the third international workshop on cryptographic hardware and embedded systems (CHES2001), 2001, LNCS 2162, pp.348-363.

[18] E. Savas, and C. Koc, “Finite field arithmetic for cryptography”, IEEE. Circuits and Systems Magazine, 2010, Vol.10, No.2, pp.40-56.

[19] G. Dormale, and J. Quisquater, “High-speed hardware implementations of elliptic curve cryptography: a survey”, Journal of systems architecture, 2007, Vol.53, No.2-3,pp.72-84.

[20] A. Rezai, and P. Keshavarzi, “High-performance implementation approach of elliptic curve cryptosystem for wireless network applications”, in Proc. of the IEEE. International conference on consumer electronics, communications and networks, 2011, pp.1323-1327.

[21] A. Booth, “A signed binary multiplication technique”, Journal of mechanics and applied mathematics, 1951, Vol.4, pp.236-240.

[22] G. Reitwiesner, “Binary Arithmetic, Advances in computers”, 1960, Vol.1, pp.231-308.

[23] S. Arno, and F. Wheeler, “Signed digit representations of minimal Hamming weight”, IEEE Transactions on Computers, 1993, Vol.42, No.8, pp.1007-1010.

[24] D. Hankerson, A. Menezes, and S.Vanstone, Guide to Elliptic Curve Cryptography, New York: Springer-Verlag, 2004.

[25] G. Dormale, and J. Quisquater, “Area and time trade-offs for iterative modular division over GF(2^m): novel algorithm and implementations on FPGA”, International journal of electronics, 2007, Vol.94, No.5, pp.515-529.

[26] J. Großschädl, and G. A. Kamendje, “Instruction set extension for fast elliptic curve cryptography over binary finite fields GF(2^m)”, in Proc. of the 14th IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP 2003), 2003, pp.455-468.

[27] P. Montgomery, “Modular multiplication without trial division”, Mathematics of computation, 1985, Vol.44, No.170, pp.519-521.

[28] A. Rezai, and P. Keshavarzi, “High-performance modular exponentiation algorithm by using a new modified modular multiplication algorithm and common- multiplicand-multiplication method”, in Proc .of the IEEE. World congress on internet security, 2011, pp.192-197.

[29] A. Rezai, and P. Keshavarzi, “A new CMM-NAF modular exponentiation algorithm by using a new modular multiplication algorithm”, Trends in applied sciences research, 2012, Vol.7, No.3, pp.240-247.

[30] C. Koc, and C. Hung, “Adaptive m-ary segmentation and canonical recoding algorithms for multiplication of large binary numbers, Computers and Mathematics with Applications”, 1992, Vol.24, No.3, pp.3-12.