Malaysian Journal of Mathematical Sciences 9(1): 91-110 (2015)

MALAYSIAN JOURNAL OF MATHEMATICAL SCIENCES

Journal homepage: http://einspem.upm.edu.my/journal

Speeding up the Elliptic Curve Scalar Multiplication Using the Window- w Non Adjacent Form

[1,2*] **Najlae Falah Hameed Al Saffar and** [1] **Mohamad Rushdan Md Said**

[1,2] Institute for Mathematical Research, Universiti Putra Malaysia,
43400 UPM Serdang, Selangor, Malaysia

[2] Department of Mathematics, Faculty of Mathematics and
Computer Science, Kufa University, Iraq

E-mail: najlae_falah@yahoo.com and mrushdan@upm.edu.my

**Corresponding author*

ABSTRACT

Nowadays, elliptic curve based cryptosystem is an efficient public key cryptosystem, The very expensive operation in this cryptographic protocol is the elliptic curve scalar multiplication (elliptic curve point multiplication). Efforts have been mainly focused on developing efficient algorithms for representing the scalar which is involved of elliptic curve scalar multiplication. One of these is using the window- w non adjacent form method. In the present work, the accelerating elliptic curve scalar multiplication using the window- w non adjacent form method is proposed, where the number of operations in the elliptic curve scalar multiplication has been reduced. The expected gain is about 20%, 14% and 7.6% comparing with using the anther methods to compute the elliptic curve scalar multiplication. 20%

Keywords: Elliptic Curve Cryptosystems, Elliptic Curve Scalar Multiplication, Non Adjacent Form, Window- w Non Adjacent.

1. INTRODUCTION

Elliptic curves have wealthy and nice history, having been studied by mathematicians for over a hundred years, it used to solve a diverse rang of problems but not cryptographic problems. Until 1985, elliptic curves were first proposed for use in public-key cryptography by Neal Koblitz

(1987) and Victor Miller (1986) independently requires smaller key sizes than the other public cryptosystems such as RSA (Ronald L. Rivest *et al.* (1978)) at the same level of security. The security of the elliptic curve cryptosystem (ECC) is based on the computational intractability of solving the discrete logarithm problems ($ECDLP$) which is the problem of finding scalar k , given Q and P in the operation $Q = kP$, such as ElGamal encryption (Taher ElGamal (1985)) and the DSA (Patrick Gallagher (2009)).

Since the mid of 1980s and so far, researchers always try to improve the efficiency of ECC. Elliptic curve scalar multiplication is the main operation in ECC, which is the process of computing

$$Q = kP = \underbrace{P + P + ... + P}_{k\ times}$$

where k is a positive integer called scalar and P , Q are points on elliptic curve, so that a fast and secure elliptic curve scalar multiplication algorithm is required.

Many algorithms had been introduced to improve the efficiency of this elliptic curve scalar multiplication. These algorithms have mainly forced on improving efficient numeric expansions for the scalar k that by reducing the number of operations required for the computation of the elliptic curve scalar multiplication. In 2001, Gallant, Lambert and Vanstone (2001) has proposed a method to compute the elliptic curve scalar multiplication using the application of the endomorphism function defined on the group of elliptic curve over binary field (F_{p^n}) for $n \geq 1$ and p is prime. This algorithm is to compute kP by decomposing k into k_1 and k_2 but with condition that k_1 and k_2 bounded by $\sqrt{n}$. Another example of these algorithms is by transferring the scalar k to the Non Adjacent Form (NAF) or the Window- w Non Adjacent Form (Window $w - NAF$) such like (Marc Joye and Christophe Tymen (2001); Franois Morain and Jorge Olivos (1989) and Katsuyuki Okeya and Tsuyoshi Takagi (2003). The story started since 1951 when Booth (1951) proposed a new scalar representation called signed binary method which is the NAF , after ten years when Rietweisner (1960) proved that every integer could be represented in this form and it is unique.

The aim of the present work is to speed up the elliptic curve scalar multiplication using the window $w-NAF$ of integer. This achieved by representing the scalar $k = NAF(k_1)_w + NAF(k_2)_w$. This computation is simple and efficient when compared to other traditional methods.

The contributions of this paper are organized as follows:

- The preliminaries section describes an abstract analysis of the additive group of elliptic curves over prime field, with some definition related with our work such as signed digit representation of an integer k to the base b.
- The next section is about the main operation on ECC, which is the elliptic curve scalar multiplication. This section has been divided according to the method used to account this operation.
- The proposed algorithm section contains illustrated the main idea of this work together with its algorithm and the complexity of it. Comparison between the three methods mentioned in this work is also in the complexity subsection.
- Finally, the conclusion section.

2. PRELIMINARIES

For cryptographic purposes, the reader refer to Hankerson, Menezes and Vanstone (2004); Menezes *et al.* (1997) and Yan (2002). In this work, the consider elliptic curve E over a field K denoted by $E(K)$ is given by the equation

$$y^2 = x^3 + ax + b \tag{1}$$

which is the Weierstrass equation (1) over prime field F_p, that is $a,b \in F_p$ and $4a^3 + 27b^2 \neq 0$. Let E be an elliptic curve over a finite prime field F_p. Then the set of pairs (x, y) that solve the elliptic curve equation (1) where $x,y \in F_p$, and the point at infinity O_∞ form an abelian additive group. Which contains all the possible elements for computations on EC over F_p. All algorithms to compute an addition for any two points is given in Yan (2002). For two points $P_1,P_2 \in E\left(F_p\right)$ with $P_1 \neq P_2$, we consider an operation $P_1 + P_2$ as $ECADD$, and an operation $2P$ with $P \in E\left(F_p\right)$ as $ECDBL$.

In the field of real number R, the elliptic curve is defined on equation (1) but with $a,b\in R$ and $3a^3+27b^2\neq 0$. Now to explain how the *CEADD* is preformed, it is defined as finding the line between two points (P and Q), to get the result which is the inverse of the point $-R$ as shown in Figure 1.

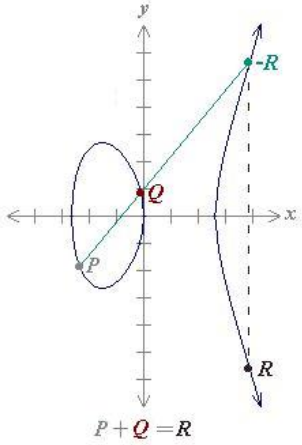

Figure 1: Elliptic Curve Addition

The operation which computes the $kP = \underbrace{P+P+...+P}_{k\ times}$ where k is an integer and P is a point on an elliptic curve, is called elliptic curve scalar (point) multiplication .

Definition: A signed digit representation of an integer k to the base b (denoted by $(k)_b$ is an ordered sequence of integers $k_0k_1k_2...k_r$ with $|k_i|<b$ for $i=0,1,...,r$, such that $k=\sum_{i=1}^{r}k_ib^i$.

Signed digit representation is not unique, for example,

$$\begin{aligned} 18 &= (11\bar{1}\bar{1}0) = (1)2^4+(1)2^3+(-1)2^2+(-1)2^1+(0)2^0 \\ &= (10010) = 2^4+2^1 \\ &= (1\bar{1}0010) = 2^5-2^4+2 \end{aligned}$$

where $\bar{1}=-1$.

In 2007, Ebeid and Hasan (2007) proposed an algorithm to generate all possible signed digit representation of any integer k .

Definition: The hamming weight of an integer k (denoted by $h(k)$) is the number of 1s in the signed digit representation.

Definition: The length of the expression $(k)_b$ (denoted by $l(k)$) is the number of its digits.

3. ELLIPTIC CURVE SCALAR MULTIPLICATION

In this section we discuss the common methods for performing scalar multiplication on an elliptic curve. These methods are to represent the scalar k in different ways to compute the main operation in ECC which is kP elliptic scalar multiplication.

The idea of this operation (kP) is adding a point P to itself k times, where P is a point with order n on an elliptic curve $E\left(F_p\right)$, and $k \in [1, n-1]$ is an integer. The simplest way to perform the elliptic curve scalar multiplication kP is the binary algorithms, which is the analogue of the square and multiply process for fast modular exponentiations (Yan (2002)).

3.1 Binary Method

The basic technique for elliptic scalar multiplication is the $ECADD$ and $ECDBL$. It is based on the binary method of the coefficient k . The integer k is represented as a signed digit representation or as $k = k_{l-1}2^{n-1} + k_{l-2}2^{l-2} + ... + k_0$ where $k_{l-1} = 1$ and $k_i \in \{0,1\}$, $i = 0,1,2,...,l-1$. That is $k = \sum_{i=0}^{l-1} k_i 2^i$, where $k_i \in \{0,1\}$. This method scans the bits of the bits of k from the left to right, if the bit is 1 then perform a $ECDBL$ and $ECADD$, otherwise, the $ECDBL$ will perform (the first bit is always 1 which is used as initialization). This method is called binary method (IEEE (2000)), the process of it for computation of kP is given in the following Algorithm 1.

ALGORITHM 1: Binary Method for Elliptic Curve Scalar Multiplication

```
Input: $(k)_{10} = (k_{l-1}...k_1k_0)_2$ , $P \in E(F_p)$
Output: $Q = kP$
1. $Q = P$
2. For $i = l-1$ down to $0$ do
    2.1. $Q = 2Q$
    2.2. If $k_i = 1$ then $Q = Q + P$
3. Return $Q$
```

For example, let us assume that k is equal to $(109)_{10}$, so in the binary representation k is equal to $(1101101)_2$. The $109P$ for $P \in E\left(F_p\right)$ is compute as follows:

e_6	1	P	*initialization*
e_5	1	$2P+P$	*doubling and addition*
e_4	0	$2\left(2P+P\right)$	*doubling*
e_3	1	$2\left(2\left(2P+P\right)\right)+P$	*doubling and addition*
e_2	1	$2\left(2\left(2\left(2P+P\right)\right)+P\right)+P$	*doubling and addition*
e_1	0	$2\left(2\left(2\left(2\left(2P+P\right)\right)+P\right)+P\right)$	*doubling*
e_0	1	$\underbrace{2\left(2\left(2\left(2\left(2\left(2P+P\right)\right)+P\right)+P\right)\right)+P}_{\Downarrow \atop 109P}$	*doubling and addtion*

The cost of elliptic curve scalar multiplication using binary method depends on the $l(k)$ and the $h(k)$ in the representation of k . If $\left(k\right)_{10} = \left(k_{l-1}k_{l-2}...k_1k_0\right)_2$, then the number of $ECDBL$ is $l-1$ and the number of $ECADD$ is one less than the $h(k)$. In an average, the binary method requires $l-1$ $ECDBL$ and $\frac{l-1}{2}$ $ECADD$. For example, the cost of computation of $109P$ in the above example requires ($6ECDBL + 4ECADD$).

As a conclusion, whenever the bit is 1, two elliptic curve arithmetic operations doubling and addition will be made and if it is 0, only one operation, doubling is required. Thus, if we reduce the number of $h(k)$ in the scalar representation, we could accelerate this computation.

3.2 Non Adjacent Form Method (NAF)

The hamming weight of the scalar k can be reduced with a signed representation that uses the numbers 0 and ± 1. Among various signed representation, NAF is a canonical representation with less number of hamming weight for any integer k. The NAF representation of k has been proposed in 1951, by Booth (1951) (some time the searchers called it Addition-subtraction method according to its process). And after 10 years Rietweisner (1960) has proved that every integer could be uniquely represented in this form.

Nowadays, The NAF of a scalar k denoted by $NAF(k)$ becomes the subject of various investigations in different contexts. The property of this representation is that, of any two consecutive digits, at most one is non zero, Moreover, the length of $NAF(k)$, denoted by $l(NAF(k))$ is at most 1 more bit than its binary representation. This means, fewer point additions and therefore more efficiency when we need to compute the scalar multiplication on EC.

Now, because of the group law of elliptic curve group, we know that the inverse of $P=(x,y)\in E(F_p)$ is $-P=(x,-y)\in E(F_p)$. Therefore, computing the inverse of any point on elliptic curve is very fast in terms of computational time. That is, in the process of computing the kP, and the minus is come across, subtraction of P is performed during this computation, furthermore, it costs the same amount of $ECADD$ in the total operation.

In the example of signed digit representation, we mentioned that there is no unique signed digit representation for any integer k. To get this uniqueness one has to add some conditions on the representation. This condition will be that there are no adjacent non zeros (using NAF).

For example, the number 7 can have several signed-digit representations:

$$
\begin{aligned}
(0111)_2 &= 4+2+1=7\\
(10\bar{1}1)_2 &= 8-2+1=7\\
(1\bar{1}11)_2 &= 8-4+2+1=7\\
(100\bar{1})_{NAF(7)} &= 8-1=7
\end{aligned}
$$

But only the last representation is NAF.

Definition: A NAF of a positive integer k is a signed digit representation of k to the base $b=2$, such that $k_i k_{i+1}=0$ for $i \geq 0$. The $NAF(k)$ is written $(k_{l-1}...k_1 k_0)_{NAF(k)}$.

The reader can refer to Gordon (1998) for the proofs of existence and uniqueness of $NAF(k)$ for any integer k. Muir and Stinson (2006) in there paper have been proved that the hamming weight of the $NAF(k)$ is minimal among all signed digit representations of k. Fortunately, the number of bits in the $NAF(k)$ is at most one more than the number of bits in the binary form of k. **Algorithm 2** is for the conversion of a scalar k into the NAF.

ALGORITHM 2: Computing NAF of a Scalar k

Input: A scalar $(k)_{10}$

Output: $N=(k_{l-1}...k_1k_0)_{NAF(k)}$

```
$i=1$; $c=k$
While $c>0$
    If $c$ odd
        $N(i)=2-(c mod 4)$
        $c=c-N(i)$
    Else
        $N(i)=0$
    End if
    $c=\frac{c}{2}$; $i=i+1$
End while
Return $N$
```

Performance of **Algorithm 2** can be summarized in the following steps:

- If k is an even integer, we have to take 0, and continue with $\frac{k}{2}$.
- If $k \equiv 1 (mod 4)$, we take 1, and continue with $\frac{k-1}{2}$ which is even integer that guarantees a 0 in the next step.
- If $k \equiv 3 \equiv -1 (mod 4)$, we take -1, and continue with $\frac{k+1}{2}$ which is even integer that guarantees a 0 in the next step.

This measure produces zero after each non-zero digit, which means this signed-digit representation must has low hamming weight.

For example, the mechanism to compute $NAF(27)$, according to Algorithm 2, is shown in Table 1.

TABLE 1: Computing a $NAF(27)$

i	c	$N(i)$	N
1	27	$\bar{1}$	$(\bar{1})$
	28		
2	14	0	$(0\bar{1})$
3	7	$\bar{1}$	$(\bar{1}0\bar{1})$
	8		
4	4	0	$(0\bar{1}0\bar{1})$
5	2	0	$(00\bar{1}0\bar{1})$
6	1	1	$(100\bar{1}0\bar{1})$
	0		

Remarks:

- $NAF(k)$ for a scalar k has fewest non zero digits (hamming weight) of any signed representation of k, unless if the binary representation of k already has. For instance
$$(89)_{10} = (1011001)_2$$
$$67pt = (10\bar{1}0\bar{1}001)_{NAF(89)}.$$
- The length of $NAF(k)$ is at most one more bit than its binary representation.

- If $l(NAF(k)) = l$, then $\frac{2^l}{3} < k < \frac{2^{l+1}}{3}$.
- The average hamming weight of $NAF(k)$ (denoted by $h(NAF(k))$ when $l(NAF(k)) = l$ is $\frac{l}{3}$.

The method for computing the scalar multiplication kP using NAF expression is **Algorithm 3**.

ALGORITHM 3: NAF Method for Elliptic Curve Scalar Multiplication

```
Input: $(k)_{10} = (k_{l-1}...k_1k_0)_{NAF(k)}$ , $P \in E(F_p)$
Output: $Q = kP$
1. $Q = P$
2. For $i = l-1$ down to $0$ do
2.1. $Q = 2Q$
2.2. If $k_i = 1$ then $Q = Q + P$
2.3. If $k_i = -1$ then $Q = Q - P$
3. Return $Q$
```

According to the **Algorithm 3**, the scalar multiplication using NAF method requires $\frac{l}{3}ECADD$ and $lECDBL$, where the subtraction and addition operation have the same cost in the case of the elliptic curves group.

Example: Let $k = 127$ and P a point on the elliptic curve E. Now, the binary representation of k is $(1111111)_2$, so the cost is exactly equal to ($6ECDBL + 6ECADD$).

While, the $NAF(127)$ is $(1000000\bar{1})_{NAF(127)}$, so the cost it is equal to ($7ECDBL + 1ECADD$).

3.3 Window- w Non Adjacent Form

If the digits of representation of k are allowed to be the elements of a larger set instead of only 0 and ± 1, then the cost of Algorithms 3 is decreased.

That is mean, not only P is added or subtracted, but also some small scalar multiple of P is added or subtracted. Those values have to be computed at the beginning of the elliptic curve scalar multiplication algorithm and saved to the memory.

A generalization of the NAF is called the Window- w Non Adjacent (denoted $w-NAF$). The hamming weight of the scalar k can be reduced with this $w-NAF(k)$, where $w \geq 2$ is an integer.

Definition: A $w-NAF$ of a positive integer k is an expression $k = \sum_{i=0}^{l-1} k_i 2^i$, such that each non zero k_i is odd, $|k_i| < 2^{w-1}$ and $k_{l-1} \neq 0$. For $w \geq 2$, at most one of any w consecutive bits is non zero.

The proof of existence and uniqueness of $w-NAF(k)$ for any integer positive integer k, where $w \geq 2$ is found in (James Muir (2004).

The length of $w-NAF(k)$ (denoted by $l(w-NAF(k))$ is at most one more than the $l(k)$. And the hamming weight of $w-NAF(k)$ (denoted by $h(w-NAF(k))$ is $\frac{1}{w+1}$. **Algorithm 4** computes the $w-NAF(k)$ for any integer k and $w>1$. Observed that when $w=2$ then $w-NAF(k) = NAF(k)$.

ALGORITHM 4: Computing $w-NAF(k)$ of a Scalar k

```
Input: A scalar $(k)_{10}$
Output: $N = (k_{l-1}...k_1k_0)_{w-NAF(k)}$

$i = 1$
While $k \geqslant 1$
  If $k$ odd
    $N(i) = 2-(k mods 2^w)$
    $k = k - N(i)$
  Else
    $N(i) = 0$
  End if
  $k = \frac{k}{2}; i = i+1$
End while
Return $N$
```

The function $mods$ in the **Algorithm 4** is an extra code designed as follows:

Coding of $mods$ Function

```
If $k\,mods\,2^{w}k \geqslant \frac{2^{w}}{2}$
 $N(i) = (k\,mods\,2^{w}) - 2^{w}$
Else
 $N(i) = (k\,mods\,2^{w})$
```

For example, $3 - NAF(27) = (3003)_{3-NAF(27)}$, since $27 = 3 \cdot 2^{3} + 3 \cdot 2^{0}$. According to Algorithm 4, the mechanism of calculating $3 - NAF(27)$ is shown in Table 2.

TABLE 2: Computing a $3 - NAF(27)$

i	c	$N(i)$	N
1	27	3	(3)
	24		
2	12	0	(03)
3	6	0	(003)
4	3	3	(3003)
	0		

Computation of elliptic curve scalar multiplication using $w - NAF(k)$ is general version of usual $NAF(k)$ elliptic curve scalar multiplication. Nevertheless, there is precomputation step.

ALGORITHM 5: $w - NAF(k)$ Method for Elliptic Curve Scalar Multiplication

```
Input: $(k)_{10} = (k_{l-1}...k_{1}k_{0})_{NAF(k)}$ , $P \in E(F_{p})$
Output: $Q = kP$
1. Calculate $2P$
2. For $i = 3$ up to $2^{w-1} - 1$ do
2.1. If $i$ is odd then $P_{i} = P_{i-1} + 2P$
3. For $i = l - 1$ down to $0$ do
3.1 If $Q = 2Q$
3.2 If $k_{i} \neq 0$
3.2.1 If $k_{i} > 0$ then $Q = Q + P_{k_{i}}$
3.2.2 Else $Q = Q - P_{k_{i}}$
4. Return $Q$
```

The cost of **Algorithm 5** is as follows: for **line 1** and **2** there is $1ECDBL$ and $2^{w-2}-1ECADD$ respectively; then as precomputaion cost there is $1ECDBL+2^{w-2}-1ECADD$. The cost of **line 2.1** is $lECDBL$ while it is $\frac{l}{w+1}ECADD$ for **lines 3.2.1** and **3.2.2**, that is, the total cost of **Algorithm 5** is $1+lECDBL+\frac{\left(2^{w-2}\right)(1+w)-2-w}{1+w}ECADD$.

Example: Let $k=27$ and P a point on the elliptic curve E.

TABLE 3: Computing a $NAF(27)P$

27	$27P$	Cost of Computing $27P$
$(11011)_2$	$2(2(2(2P+P))+P)+P$	$4ECDBL+3ECADD$
$(100\bar{1}0\bar{1})_{NAF(27)}$	$2(2(2(2(2P))-P))-P$	$5ECDBL+2ECADD$
$(3003)_{3-NAF(27)}$	$2(2(2(3P)))+3P$	$3ECDBL+1ECADD$

From the above example, we can see that the number of total operation is 7 when the form of 27 is in binary and NAF. While it is only 4 when 27 in the form of $3-NAF(27)$. For the first case, its happens because of the $l(NAF(27))$ is more than the $l(27)$. Also, $w(NAF(27))$ is less than $w(27)$ but of only one bit. Therefore, when we plan to speed up the elliptic curve scalar multiplication using $w-NAF$ algorithm, we have to look for the special scalar k with $((l(w-NAF(k))-1)+h(w-NAF(k))-1)<((l(k)-1)+h(k)-1)$.

4. PROPOSED METHOD

Our proposed method to calculate the kP is by using basic operation addition $(+)$ between two scalars in the $w-NAF$, where the goal of speeding up the elliptic curve scalar multiplication is achieved.

The idea is to look for two integers $w-NAF(k_1)$ and $w-NAF(k_2)$ such that $k=k_1+k_2$, for which the evaluation of $w-NAF(k_1)P$ and $w-NAF(k_2)P$ require less operations than the $w-NAF(k)P$, with an extra operation which is addition where we will account it as an extra operation which will change the total cost by adding one operation. For example, if we need to compute the elliptic curve scalar multiplication $4582P$ for P a point on the elliptic curve E, the binary representation of 4582 is $(1000111100110)_2$ which means there are $12ECDBL$ and $6ECADD$, a total of 18 operation. The NAF expression of 4582 is $(1001000\bar{1}010\bar{1}0)_{NAF(4582)}$, where there are $12ECDBL$ and $4ECADD$, a total of 16 operations. The $3-NAF(4582)$ is $(1001000\bar{1}00030)_{3-NAF(4582)}$, where there are $12ECDBL$ and $3ECADD$, a total of 15 operations. In comparison,

$$4582 = 3078 + 1504$$

where

$$3-NAF(3078) = (30000000030)_{3-NAF(3078)}$$

and

$$3-NAF(1504) = (3000\bar{1}00000)_{3-NAF(1504)}$$

which means there are 14 operations as total.

Algorithm 6 is a method of calculating the elliptic curve scalar multiplication operation which ensures reduction in the number of processes.

First of all, we have to be sure that the length of $w-NAF(k)$ expression of the scalar is not equal to the length of it in the binary form, so this will be the first condition in the algorithm. Also, the algorithm will has a guarantee that the new expression of the scalar will require less operations than the original algorithm to compute the scalar multiplication operation on elliptic curve. This algorithm is as follows:

ALGORITHM 6: Computing $w-NAF(k_1)$ and $w-NAF(k_2)$

```
**Input:** $(k)_{10}$, $w$
**Output:** $w-NAF(k_1)$, $w-NAF(k_2)$

If $h(w-NAF(k)) = h((k)_2)$, take another $k_{10}$
For $k_1 = k:-1:1$
2.1 For $k_2 = k:-1:1$
2.2 If $k = k_1 + k_2$
      and
    $[h(w-NAF(k_1)) + h(w-NAF(k_2))] < [h(w-NAF(k))]$
3. $w-NAF(k_1)$, $w-NAF(k_2)$
4. End
```

When we need to use this output to compute the scalar multiplication kP, then **Algorithm 5** will perform the task, but with the input of the new expression of the scalar which is

$$w-NAF(k)+w-NAF(k)$$

If the digits of representation of the scalar k are allowed to be the elements of a larger set instead of only $\{\bar{1},0,1\}$, then the running time of above algorithms are decreased. That means, when we consider these form of k to be in the algorithm of computing the scalar multiplication, not only the point P is added or subtracted, but also there are some small scalar multiple of P that are added or subtracted. Those values have to be computed at the beginning of the scalar multiplication algorithm and saved to the memory.

4.1 Complexity Of Proposed Algorithm

In this section we will discuss the cost of the above four methods which are used to compute the elliptic curve scalar multiplication kP.
We know that $l(NAF(k))$ and $l(w-NAF(k))$ of the scalar k is at most one more bit than its binary representation with less non zero digits, this will guarantee that it will reduce the number of operations, but if the hamming weight of them is equal, there is no advantage of using $NAF(k)$ or even $w-NAF(k)$ of the scalar k, because it will affect on the number of the operations as total. Now, the proposed algorithm has the following features:

- The hamming weight of $w-NAF(k)$ expression of the scalar k is not equal to the hamming weight of it in the binary form.
- The warranty that the new expression of the scalar will require less operation than the binary, NAF and $w-NAF(k)$ expression. This is achieved in the final step. That means, the new algorithm is ensured to enhance the computing efficiency of elliptic curve scalar multiplication compared with other traditional algorithms.

In order to be able to compare the different elliptic curve scalar multiplication, we counted the number of $ECDBL$ and $ACADD$ which were required of the algorithms. Table 4 and the plots in Figure 2 shows the number of operations required by the proposed algorithm and the binary, NAF and $w-NAF(k)$ algorithms.

Now, the comparison of the complexity of the algorithms which referred to above will give in the following table:

TABLE 4: Complexity of Computing Elliptic Curve Scalar Multiplication of Various Processors

Size of k	Number of Operation			
	Binary Algorithm	NAF Algorithm	$3-NAF$ Algorithm	Proposed Algorithm
10	15	14	13	12
16	23	21	20	19
27	41	36	34	33
34	48	46	44	42

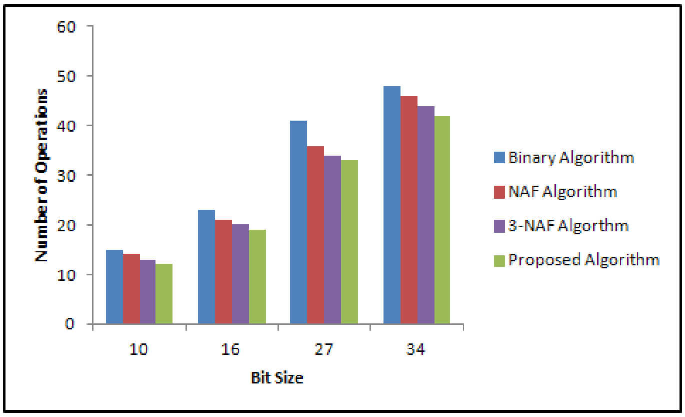

Figure 2: Complexity of Computing Elliptic Curve Scalar Multiplication of Various Processors

The running time of an algorithm is determined by how many operations are performed throughout its execution. Therefore, we followed (Lars Elmegaard-Fessel (2006)) in there assumption which is

$$1ECDBL = 1.05ECADD$$

Then the running time according to this assumption for all mentioned algorithms will be as follows:

For the binary algorithm:

$$(l(k)-1)ECDBL + (h(k)-1)ECADD = (l(k)-1)(1.05)ECADD + (h(k)-1)ECADD$$

For the NAF algorithm:

$$(l(NAF(k))-1)ECDBL+(h(NAF(k))-1)ECADD=$$
$$(l(NaF(k))-1)(1.05)ECADD+(h(NAF(k))-1)ECADD$$

Finally, for the proposed algorithm:

$$l(w-NAF(k_1))ECDBL+(h(w-NAF(k_1))-1)ECADD+(h(w-NAF(k_2))+1)ECADD=$$

$$l(w-NAF(k_1))(1.05)ECADD+(h(w-NAF(k_1))-1)ECADD+(h(w-NAF(k_2))+1)ECADD$$

For selected bit size of integer k we can make a comparison for the four mentioned methods as in the **Figure 3** and **Table 5**. These figure and table summarize our achievement with proposed algorithm of using the addition operation between two $w-NAF$ expressions with special case when $w=3$.

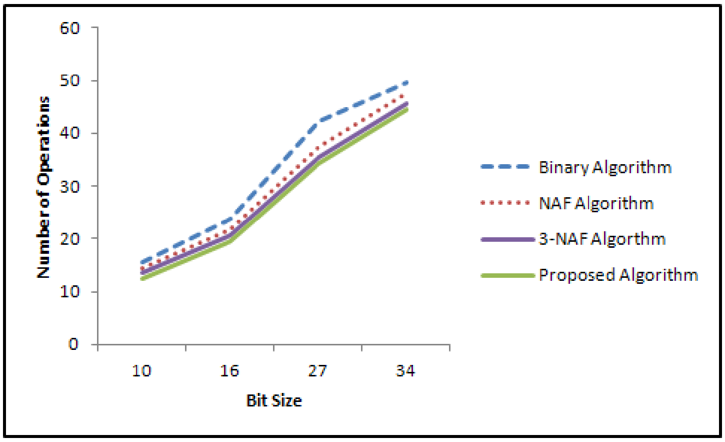

Figure 3: Running Time of the Proposed Algorithm with other Algorithms (Binary, NAF and $3-NAF$)

TABLE 5: Comparison of Running Time

Size of k	Times in Seconds by $ECADD$			
	Binary Algorithm	NAF Algorithm	$3-NAF$ Algorithm	Proposed Algorithm
10	15.45	14.05	13.5	12.4
16	23.75	21.75	20.75	19.65
27	42.3	37.35	35.35	34.25
34	49.65	47.65	45.7	44.6

All algorithms have been implemented on $Intel(R)Core(Tm)2Duo$ with processor $2.99GHz$ and and $3.00GB$ of memory using $MATLAB$ version $7.10.0.499$ ($R2010a$). In order to obtain running time, we executed all algorithms with different size bits $(10, 16, 27$ and $34)$ for the scalar k. The following is the result we have collected from the execution and its corresponding graphical explanation.

The efficiency of the proposed algorithm is clearly known, from the Table 4 and 5 and Figure 2 & 3. For instance, elliptic curve scalar multiplication using binary, NAF and $3-NAF$ algorithm require 41 , 36 and 34 respectively with respect to the number of operations, while the proposed algorithm require 33 operation when k has 27 bits. For another example, with the same size of k (27), the binary , NAF and $3-NAF$ algorithm executed within 42.3, 37.35 and 35.35 seconds of executed $ECADD$ respectively while the proposed algorithm spent 34.25 seconds of executed $ECADD$. Therefore the algorithm which we proposed saved computation and time comparing with the mentioned algorithms.

5. CONCLUSION

The elliptic curve scalar multiplication is not only the fundamental computation, but also the most time consuming operation, that is why it is interesting subject made almost all cryptographers (specialists in ECC) trying to speed up it using different methods.

In this paper, we proposed a new method to compute the elliptic curve scalar multiplication. This algorithm has guaranteed that is one of the methods that accelerate this operation on EC by 20%, 14% and 7.6% comparing with using the binary, NAF and $w-NAF$ algorithms with $w=3$ respectively. Table 4 & 5 and Figure 2 & 3 summarize our achievement with the proposed algorithm of using the addition operation between two integers in the $w-NAF$ expression with $w=3$. Moreover, the points which we listed it in the previous section were earnestly justified the conclusion.

REFERENCES

Axler, S. and Ribet, K. A. (2009). Graduate texts in mathematics. 106.

Booth, A.D. (1951). A signed binary multiplication technique. *The Quarterly Journal of Mechanics and Applied Mathematics,* **4**(2):236–240.

Lars Elmegaard-Fessel. (2006). Efficient Scalar Multiplication and Security against Power Analysis in Cryptosystems based on the NIST Elliptic Curves over Prime Fields.

Ebeid, N. and Hasan, M. A. (2007). On binary signed digit representations of integers. *Designs, Codes and Cryptography*. **42**(1):43–65.

ElGamal, T. (1985). *A public key cryptosystem and a signature scheme based on discrete logarithms.* In Advances in Cryptology, pages 10–18. Springer.

Gallagher, P. (2009). *Digital signature standard (dss).* Federal Information Processing Standards Publication, FIPS PUB, pages 186–183.

Gallant, R.P., Lambert, R. J, and Vanstone, S. A. (2001). *Faster point multiplication on elliptic curves with efficient endomorphisms*. In Advances in CryptologyCRYPTO 2001, pages 190–200. Springer.

Gordon, D. M. (1998). A survey of fast exponentiation methods. *Journal of algorithms*. **27**(1):129–146.

Hankerson, D., Menezes, A. J. and Vanstone, S. A. (2004). Guide to elliptic curve cryptography. Springer.

Joye, M. and Tymen, C. (2001). *Compact encoding of non-adjacent forms with applications to elliptic curve cryptography*. In Public Key Cryptography, pages 353–364. Springer.

Koblitz, N. (1987). Elliptic curve cryptosystems. *Mathematics of computation*. **48**(177):203–209.

Miller, V.S. (1986). Use of elliptic curves in cryptography. *In Advances in Cryptology-CRYPTO85 Proceedings,* pages 417–426. Springer.

Morain, F. and Olivos, J. (1989). *Speeding up the computations on an elliptic curve using addition-subtraction chains*.

Menezes, A. J., Van Oorschot, P.C. and Vanstone, S. A. (1997). *Applied cryptography*.

Muir, J. and Stinson, D. (2006). Minimality and other properties of the width- nonadjacent form. *Mathematics of Computation*. **75**(253):369–384.

Muir, J. (2004). *Efficient integer representations for cryptographic operations.*

Okeya, K. and Takagi, T. (2003). The width-w naf method provides small memory and fast elliptic scalar multiplications secure against side channel attacks. *Topics in CryptologyCT-RSA 2003,* pages 328–343.

Ieee standard specifications for public-key cryptography. (2000). ID: 1.

Reitwiesner, G.W. (1960). Binary arithmetic. *Advances in computers*. **1**:231–308.

Rivest, R. L., Shamir, A. and Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*. **21**(2):120–126.

Yan, S.Y. (2002). *Number theory for computing*. Springer.