Survey of Elliptic Curve Scalar Multiplication Algorithms

Dr. E.Karthikeyan
Department of Computer Science. Government Arts College, Udumalpet – 642126. India.
Email: e_karthi@yahoo.com

ABSTRACT
The standard bodies accepted elliptic curve cryptography as an efficient and alternative public key cryptosystem for the next generation cryptographic applications. The most dominant operation in this cryptosystem is the scalar multiplication. In this paper, we study various scalar multiplication algorithms with respect to the efficiency, average hamming weight and memory requirements etc. This paper gives an idea and the areas for which the researchers can proceed further.

Keywords: Elliptic curve cryptography, binary method, non-adjacent form, scalar multiplication

Date of Submission : July 10, 2012 Date of Acceptance : August 23, 2012

1. INTRODUCTION

In the year 1985, Miller [1] and Koblitz [2] independently proposed elliptic curve cryptography (ECC) and it is gaining a wide acceptance as an alternative to the conventional cryptosystems like RSA [3], DSA [4] and DH [5]. The primary reason for the attractiveness of ECC over the conventional systems is that it offers equivalent security using far smaller key sizes. For example, 160-bits of ECC and 1024-bits of RSA/DSA/DH offer the same level of security. These advantages are particularly beneficial in applications where bandwidth, processing capacity, power availability and storage is limited. Such applications are smart cards, cellular phones, PDA, sensor networking and pagers (refer [6] for more details).

Elliptic curve based protocols such as Elliptic Curve Diffie-Hellman (ECDH), Elliptic Curve Digital Signature Algorithm (ECDSA) and Elliptic Curve Integrated Encryption Scheme (ECIES) involves scalar multiplications. The speed of scalar multiplication plays a vital role in deciding the efficiency of the whole system. In particular, fast multiplication is more crucial in some environments such as e-commerce servers where the large number of key agreements or signature generations occurs, and handheld devices with low computational power. There has been extensive research to compute scalar multiplication efficiently based on various representations of the multiplier [7].

2. MATHEMATICAL BACKGROUND OF ECC

An elliptic curve can be described as the set of solutions for the equation $y^2 \equiv x^3 + ax + b \pmod p$ where a, b ϵ Zp such that $4a^3 + 27b^2$ # 0, including the point infinity O. The efficiency of elliptic curve algorithm is determined by various factors like selecting finite field (prime / binary), coordinate representations (Affine, Projective, Jacobian, Chudnovsky Jacobian and Modified Jacobian coordinate), elliptic curve arithmetic, scalar representation etc. More details about the mathematical aspect of ECC are available in **[25, 26]**

The two basic elliptic curve arithmetic operations are elliptic curve addition (ECADD) and elliptic curve doubling (ECDBL). The formula to compute the same is given in the Table-1 with respect to the affine coordinate system. Let P = (x1, y1) and Q = (x2, y2). Point addition will be made when P # Q, and if P = Q, then point doubling operation will be carried out.

Operation	Formula (Affine Coordinate System)
ECADD	$x_3 = \lambda^2 - x_1 - x_2$, $y_3 = \lambda(x_1 - x_3) - y_1$, and $\lambda = \frac{y_2 - y_1}{x_2 - x_1}$.

ECDBL	$x_3 = \lambda^2 - 2x_1$, $y_3 = \lambda(x_1 - x_3) - y_1$, and $\lambda = \frac{3x_1^2 + a}{2y_1}$.

The result of "addition" (P+Q) or "doubling" (2P) of points on the elliptic curve will always be another point on the curve. One elliptic curve addition operation over E(Fp) requires one inversion, two multiplications, one squaring and six addition operations. Similarly, doubling requires one inversion, two multiplications, two squaring and eight addition operations. The number of arithmetic operations is varying for each coordinate system.

Let us consider the elliptic curve over Fp where a = 1, b = 6, p = 11 with the equation $y^2 \equiv x^3 + x + 6 \pmod{11}$. The set of solutions are E = {(2,4), (2,7), (3,5), (3,6), (5,2), (5,9), (7,2), (7,9), (8,3), (8,8), (10,2), (10,9), O}, including the point infinity O.

Choose P = (2, 4) and Q = (10, 9) and the elliptic curve point addition is performed as follows.

λ = (9-4)/(10-2) mod 11 = 2
P+Q = (2,4) + (10,9)
$x_3 = 2^2 - 2 - 10 = -8 = 3$
$y_3 = 2(2-3) - 4 = -2 - 4 = -6 = 5$
P+Q = (2,4) + (10,9) = (3, 5)

Select the point P = (8,8) and the doubling operation is done as follows
$\lambda = (3 * 8^2 + 1)/(2 * 8) \bmod 11 = (6 / 5) \bmod 11 = (50 / 5) \bmod 11 = 10$
$x_3 = 10^2 - 2 * 8 = 84 \bmod 11 = 7$
$y_3 = 10(8-7) - 8 = 10 - 8 = 2$
2P = (8,8)+ (8,8) = (7,2)

Note that the result of addition and doubling is (3,5) and (7,2), because the elliptic curve points are Abelian group. In case of affine coordinate system, every ECADD / ECDBL requires an inversion operation which is costlier than other operations such as addition, subtraction and multiplication. The projective coordinate doesn't requires any inversion; instead only one inversion is required at last but it takes some extra memory for storing temporary values. The other coordinate systems and its properties can be had from **[14]** and they are almost the improved version of projective coordinate system.

3. BINARY METHOD

Scalar multiplication is the computation of the form Q= k P where P and Q are the elliptic curve points and k is an integer. This is achieved by repeated point addition and doubling operations. To calculate the above, integer k is represented as $k=k_{n-1}2^{n-1}+k_{n-2}2^{n-2} + \ldots + k_1+k_0$ where $k_{n-1}=1$ and $k_i \in \{0,1\}$, i = 0,1,2…,n-1. This method is called *binary method* [8] which scan the bits of *k* either from left-to-right or right-to-left. The Algorithm-1 given below illustrate the computation of kP using binary method.

```
Algorithm-1: Binary Method
Input   : Binary representation of k and point P
Output  : Q = kP
Q=P
For i = n-1 to 0 do
    Q = 2Q     (Doubling)
    If $k_i$ = 1 then
        Q = Q + P  (Addition)
Return Q
```

The cost of multiplication depends on the length of the binary representation of k and the number of 1s in this representation. The number of non-zero digits is called the *Hamming Weight* of scalar. In an average, binary method requires (n-1) doublings and (n-1)/2 additions. For each bit '1', we need to perform ECDBL and ECADD, if the bit is '0', we need only ECDBL operation. So if we reduce the number of 1s in the scalar representation or hamming weight, the speed of elliptic curve scalar multiplication will improve.

4. ADDITION-SUBTRACTION METHOD

In 1951, Booth [9] proposed a new scalar representation called *signed binary method* and later Rietweisner [10] proved that every integer could be uniquely represented in this format. The property of this representation is that, of any two consecutive digits, at most one is non-zero. Here the integer k is represented as $k = \sum_{j=0}^{l-1} k_j 2^j$, where each $k_j \in \{-1, 0, 1\}$. Rietweisner's canonical representation is called *non-adjacent form* (NAF) [11]. NAF of a positive integer is at most one bit longer than the binary representation of the same. The Algorithm-2 is used for converting an integer k into the signed binary representation of the same.

```
Algorithm-2: Computation of NAF of an integer
Input  : Positive integer k
Output : s (NAF representation of k)

  c = k ; l = 0
  While (c > 0)
      If (c is odd)
         s[l] = 2 – (c mod 4)
         c = c – s[l]
      Else
         s[l] = 0
      EndIf
      c = c/2 ; l = l + 1
  End While
Return s
```

The average hamming weight of signed binary representation is (n/3) and it has the lower hamming weight than the binary representation. For example, the binary representation of 2927 is $(101101101111)_2$ and the hamming weight is 9 and NAF of 2927 is $(01100\text{-}100\text{-}1000\text{-}1)_2$ and the hamming weight is only 5. The hamming weight of k is reduced from 9 to 5 which leads to the improvement in the scalar multiplication.

One notable property of elliptic curve group is that the inverse of a point can be computed virtually free. This is the reason why a signed representation of scalar is meaningful. The binary method is revised accordingly and the new algorithm is called *addition-subtraction method* [12, 13] given below.

```
Algorithm-3: Addition-Subtraction method
Input  : k and P
Output : Q = kP

s[ ] =NAF(k)  /* The NAF of k is stored in s */
Q = P
For j = n–1 to 0
        Q = 2Q
        If ($s_j$ = 1)
              Q = Q + P
        ElseIf ($s_j$ = –1)
              Q = Q – P
        EndIf
Return Q
```

This algorithm performs *n* doublings and *n/3* additions in an average. The disadvantage of the *addition-subtraction* method is that it is necessary to complete the recoding and store them before starting left-to-right evaluation stage. Hence it requires additional *n*-bit memory for the right-to-left exponent recoding.

5. WINDOW METHOD

When we are allowed to use extra memory, the *window method* further enhances the efficiency of scalar multiplication by using table of pre-computed points. A window is a combination of consecutive columns such that the number of columns is less than or equal to *w* where *w* is the width. In this method *w* consecutive bit of binary representation is scanned and get replaced by pre-computed table value. Let us consider the window method with width *w*=4, the necessary pre-computation is made according to the digits set T={1,3,5,7} that is 1….2^{w-1}-1. During the recoding stage, the binary exponent is getting replaced as follows: $1|1 \rightarrow 0|3$, $1|0|1 \rightarrow 0|0|5$, and $1|1|1 \rightarrow 0|0|7$. The conversion from binary to the window can be performed left-to-right or right-to-left as well. The result may differ syntactically, but there is no change in the non-zero density of those representations. The window based signed binary representation is called *w*NAF, first described in [15] has minimal hamming weight than any other scalar representations. The property of *w*NAF is that

- o The most significant non-zero bit is positive
- o Among any *w* consecutive digits, at most one is non-zero
- o Each non-zero digit is odd and less than 2^{w-1} in absolute value

The Algorithm-4 is used to generate *w*NAF of given integer *k* from least significant bit that is right-to-left.

```
Algorithm-4: Generation of wNAF of integer
Input  : width w, an n-bit integer k
Output : wNAF of k ($sk_n|sk_{n-1}|...|sk_0$)
    i = 0
    While d ≥ 1 do
             If k is even then
                     $sk_i$ = 0
             Else
```

```
                $sk_i$ = k mods $2^w$
                k = k - $sk_i$
            Endif
        k = k/2; i = i + 1
    Return ($sk_n$, $sk_{n-1}$, ..., $sk_0$)
```

The average density of non-zero bits is asymptotically 1/ (w+1) for $n \to \infty$, and the digit set equals T= $\{\pm 1, \pm 3, ..., \pm(2^{w-1}-1)\}$ which seems to be minimal. For example, the binary representation of 2927 is $(101101101111)_2$ and *w*NAF representation of 2927 is $(005005005007)_2$. The the hamming are 9 and 4 respectively. So the wNAF is an optimal representation of the scalar. The Algorithm-5, [15] does the necessary pre-computations and then kP is computed using window method.

Algorithm-5: Scalar multiplication using window method

```
Pre-computation Stage:
Input   : point P and width w
Output  : Pi = iP; i = 1…$2^{w-1}$ – 1

P1 = P
X = ECDBL(P1)
For i = 3 to $2^{w-1}$ - 1, step 2 do
    Pi = ECADD(X, $P_{i-2}$)
Return Pi; i = 1 … $2^{w-1}$ – 1
```

```
Evaluation Stage:
Input  : point P and k (k is an integer)
         wNAF = $sk_n|sk_{n-1}|...|sk_0$ of $d$
         Pi := iP; i = 1… $2^{w-1}$ - 1
Output : dP

X = O
for i = n down to 0 do
    X = ECDBL(X)
    if $k_i > 0$ then
        X = ECADD(X, $P_{ki}$)
    else if $k_i < 0$ then
        X = ECADD(X, $-P_{|kj|}$)
return X
```

*w*NAF is computed only from the least significant bit, that is right-to-left. So we need to compute and store the *w*NAF representation of the multiplier before starting scalar multiplication. The drawback of *w*NAF is that it is not possible to merge the exponent recoding and the evaluation stage and it seems impossible to compute *w*NAF left-to-right. However, in connection with memory constraint devices left-to-right recoding schemes are by far more valuable.

6. MUTUAL OPPOSITE FORM (MOF)

The left-to-right recoding method eliminates the need for recoding and storing the multiplier in advance. Joye and Yen [16] first proposed the left-to-right recoding algorithm in the year 2000. In CRYPTO 2004, Okeya [17] proposed a new efficient left-to-right recoding scheme called *mutual opposite form* (MOF). The property of new singed binary representation is that

- o Signs of adjacent non-zero bits (without considering 0 bits) are opposite
- o Most non-zero bit and the least non-zero bit are 1 and -1, respectively
- o All the positive integers can be represented by unique MOF

The following Algorithm-6 is a simple and flexible conversion from n-bit binary string k to (n+1)-bit MOF.

Algorithm–6: Generation from Binary to MOF (Left-to-Right)

```
Input  : n-bit binary string $k=k_{n-1}|k_{n-2}|\dots|k_1|k_0$
Output : MOF of k ($mk_n|...|mk_1|mk_0$)

  $mk_n = k_{n-1}$
      for i = n-1 down to 1 do
        $mk_i = k_{i-1} - k_i$
  $mk_0 = -k_0$
Return ($mk_n$, $mk_{n-1}$, ..., $mk_1$, $mk_0$)
```

This algorithm converts the binary string to MOF from the most significant bit efficiently. MOF representation of an integer is highly flexible because, the conversion from right-to-left and left-to-right is possible.

Applying window method on MOF can further minimize the non-zero density of MOF. The *w*MOF is the first window based signed recoding scheme that can be performed from the most significant bit. As in the case of elliptic curve scalar multiplication a left-to-right evaluation is the natural choice, *w*MOF enables to merge recoding and evaluation stage. Hence there is no need to store the recoded scalar earlier. The

Algorithm-7 will make use of the pre-computed table to generate wMOF of k from left-to-right.

Algorithm-7 : Left-to-Right Generation from Binary to wMOF

```
Input :
width $w$, n-bit binary string $k=k_{n-1}|k_{n-2}|...|k_1|k_0$
Output : wMOF of k $(sk_n|sk_{n-1}|...|sk_0)$

$k_{-1} = 0; k_n = 0$
i = 0
while $i \geq w - 1$ do
    if $k_i = k_{i-1}$ then
        $sk_i = 0; i = i - 1$
    else {The MOF windows begins with a non-zero digit left-hand}
        $(sk_i, sk_{i-1}, ..., sk_{i-w+1}) \leftarrow Table_{w\,SW}(k_{i-1} - k_i, k_{i-2} - k_{i-1}, ..., k_{i-w} - k_{i-w+1})$
        i = i - w
    if $i \geq 0$ then
        $(sk_i, sk_{i-1}, ..., sk_0) \leftarrow Table_{i+1SW}(k_{i-1} - k_i, k_{i-2} - k_{i-1}, ..., k_0 - k_1, - k_0)$
return $(sk_n, sk_{n-1}, ..., sk_0)$
```

The average non-zero density of wMOF is also 1/ (w+1) for $n \rightarrow \infty$. Every non-negative integer *k* has a representation as *w*MOF, which is unique except for the number of leading zeros. The Algorithm-8 merges the recoding and evaluation stage of the scalar multiplication.

Algorithm-8: Left-to-Right Scalar multiplication using wMOF(On the Fly)

```
Input : Point P, n-bit binary string $k = k_{n-1}| k_{n-2}|...|k_1|k_0$
Output : k P
  $k_{-1} =0; k_0 = 0$
  i=e+1 for the largest e with $k_e$ #0
  If $k_{i-2} = 0$ then
      Q = P ; i = i–2;
  Else {$k_{i-2} = 1$}
      Q=ECDBL(P); i=i-2;
  While $i \geq 1$ do
      If $k_{i-1} = k_i$ then
          Q=ECDBL(Q); i=i-2;
      Else { $k_{i-1}$ # $k_i$ }
          Q=ECDBL(Q)
          If $(k_i, k_{i-2}) = (1,1)$ then
              Q=ECDBL(Q);
              Q=ECADD(Q,-P)
          Else if $(k_i, k_{i-2}) = (1,0)$ then
              Q=ECADD(Q,-P); Q= ECDBL(Q)
          Else if $(k_i, k_{i-2}) = (0,1)$ then
              Q=ECADD(Q,P); Q= ECDBL(Q)
          Else if $(k_i, k_{i-2}) = (0,0)$ then
              Q= ECDBL(Q); Q=ECADD(Q,P)
      i=i-2
      If i=0 then
          Q= ECDBL(Q); Q=ECADD(Q, $-k_0$ P)
  Return Q
```

The advantage of above algorithm is that it reduces the memory requirement since it does not store the converted representation of d, instead it is used directly in the scalar multiplication.

7. SHAMIR METHOD - PARALLEL COMPUTATION

Some public key cryptographic protocols such as the verification of digital signature, self-certified signature scheme requires the computation of powers of two, three, or more. ECDSA verification requires the computation of aP + bQ, where a and b are integers and P and Q are elliptic curve points. Normally this is done by computing aP and bQ individually and add result finally. Shamir proposed a method [18] to compute aP + bQ simultaneously. The signed binary representations of a pair of integers are written one below another, the number of non-zero columns is defined as the *"joint weight"*. The *joint weight* of *a* and *b* determine the speed of the computation. For example, the joint weight of 57 and 22 in binary expansion is 6, since $57=(111001)_2$ and $22=(010110)_2$. The signed binary representation of 57 is $(100\text{-}1001)_2$ and 22 is $(10\text{-}10\text{-}10)_2$ and the joint weight is 5. The computation of aP + bQ using Shamir method is illustrated in the Table-2.

a	1	0	0	-1	0	0	1
b	0	1	0	-1	0	-1	0
Double	0	2P	4P+2Q	8P+4Q	14P+6Q	28P+12Q	56P+22Q
+P	P						**57P+22Q**
-P				7P+4Q			
+Q		2P+Q					
-Q				7P+3Q		28P+11Q	

Table–2: Shamir method to compute aP + bQ.

It is clear that the number of additions required is depends on the joint weight of a and b and the number of point doublings required is one less than the number of bits in a or b. Thus minimizing joint weight would speedup the computation. This method costs n doublings and 2n/3 additions on average. The simple pre-computations like P+Q and P-Q would improve the speed called fast shamir method given in Algorithm-9.

Algorithm-9 : Shamir Method to compute aP + bQ

```
Input  :  NAF(a), NAF(b) and P,Q
          where a,b are integers and P,Q are points
Output :  aP + bQ

   P1=P        P2=Q
   Q=0         R=P1+P2        S=P1-P2
   for i=n-1 to 0 do
   Q=2Q
               if $(a_i,b_i)$ # (0,0)
               if $(a_i,b_i)$ = (1,0) then Q=Q+P1
               elseif $(a_i,b_i)$ = (-1,0) then Q=Q-P1
               elseif $(a_i,b_i)$ = (0,1) then Q=Q+P2
               elseif $(a_i,b_i)$ = (0,-1) then Q=Q-P2
               elseif $(a_i,b_i)$ = (1,1) then Q=Q+R
               elseif $(a_i,b_i)$ = (-1,-1) then Q=Q-R
               elseif $(a_i,b_i)$ = (1,-1) then Q=Q+S
               elseif $(a_i,b_i)$ = (-1,1) then Q=Q-S
   End
   Return Q
```

8. JOINT SPARSE FORM (JSF)

Solinas [18] presented a right-to-left method called Joint Sparse Form (JSF) for computing the signed binary representation of a pair of integers, which results in minimal joint weight than shamir's method. The property of JSF is that

- The average joint weight among all JSF representations of two n-bit integers is n/2.
- Of any three consecutive positions, at least one is a double zero
- Adjacent terms do not have opposite signs, that is X_j,X_{j+1} # -1 and Y_jY_{j+1} # -1
- If X , X_{j+1} # - 1, then $Y_{j+1} = \pm 1$ and $Y_j = 0$
- If Y_j, Y_{j+1} # - 1, then $X_{j+1} = \pm 1$ and $X_j = 0$

The Algorithm-10 is used for generating joint sparse form of pair of integers called JSF of integers [18,19].

Algorithm-10: Computation of Simple Joint Sparse Form

```
Input   :  x and y are integers
Output  :  JSF of (x and y)

j=0
while x # 0 or y # 0 do
  $x_j$ = x mod 2, $y_j$ = y mod 2
  if $x_j = 1$ and $y_j = 1$
  then
    if $(x - x_j) / 2 \equiv 1 \pmod 2$ then
        $x_j = - x_j$
    end if
    if $(y - y_j) / 2 \equiv 1 \pmod 2$ then
        $y_j = - y_j$
    end if
  else if $x_j \# y_j$ then
      if $(x - x_j) / 2 \equiv (y - y_j) / 2 \pmod 2$
      then
        $x_j = - x_j, y_j = - y_j$
      end if
  end if
 $x = (x - x_j) / 2, y = (y - y_j) / 2$
 $j = j + 1$
End while
```

The output of above algorithm can be used in fast shamir method,Algorithm-9 to compute aP+bQ and now it costs *l* doublings and *w* additions, where *l* is the length and *w* is the hamming weight.

9. DIRECT DOUBLING

Among the various elliptic curve arithmetic operations, point doubling is quite costlier than point addition in the scalar multiplication over affine coordinate system. Sakai and Sakurai [20] proposed a scalar multiplication algorithm using direct computation of several doublings (which computes 2^kP directly from P) without computing the intermediate points 2P, 2^2P, $2^3P…2^{k-1}P$. The concept of direct computation of 2^kP was first suggested by Guajardo and Paar[12]. The new doubling formula is re-constructed as below.

$$A_1=x_1$$
$$B_1=3x_1^2 + a$$
$$C_1= -y_1$$
$$D_1=12A_1 C_1^2 - B_1^2$$
$$x_2 = B_1^2 – 8A_1 C_1^2 / (2C_1)^2$$
$$y_2 = 8C_1^4 – B_1 D_1 / (2C_1)^3$$

The computational complexity of this formula is (5S + 5M + I) and the existing method has the complexity of (6S + 4M + I). The complexity of the direct computation versus separate doubling is given in Table-3.

Calculation	Method	Complexity			Break-Even Point
		S	M	I	
4P	Direct Doubling	9	9	1	8.6 M<I
	Separate 2 doubling	4	4	2	
8P	Direct Doubling	13	13	1	6.3 M<I
	Separate 3 doubling	6	6	3	
16P	Direct Doubling	17	17	1	5.4 M<I
	Separate 4 doubling	8	8	4	
2^kP	Direct Doubling	4k+1	4k+1	1	$\frac{3.6k+1.8}{k-1}$
	Separate k doubling	2k	2k	k	

Table-3: Computational Complexity comparison

From the Table-3, it is found that to compute 2^kP requires at most 4k+1 squaring, 4k+1 multiplication and only one inversion when compared to the separate k doubling which requires 2k squaring, 2k multiplication and k inversions. As we know that the inversion is the costliest arithmetic operation in elliptic curve. The direct computation of 2^k P using affine coordinate system is given in Algorithm-11.

```
**Algorithm-11**: Direct computation of $2^k$ P
**Input**  : $P_1=(x_1,y_1)$
**Output** : $P_{2^k} = 2^kP_1 = (x_{2^k},y_{2^k})$
$A_1=x_1$
$B_1=3x_1^2 + a$
$C_1= - y_1$
For i = 2 to k {Compute Ai,Bi and Ci }
        $A_i = B^2_{i-1} – 8A_{i-1}C^2_{i-1}$
        $B_i = 3A_i^2 + 16^{i-1} a (\prod_{j=1}^{i-1} C_j)^4$
        $C_i = -8 C^4_{i-1} - B_{i-1}(A_i – 4 A_{i-1} C^2_{i-1})$
$D_k = 12 A_kC^2_k – B^2_k$
$x_{2^k} = (B^2_k - 8 A_k C^2_k) / (2^k \prod_{i=1}^{k} C_i)^2$
$y_{2^k} = (8 C^4_k – B_k D_k) / (2^k \prod_{i=1}^{k} C_i)^3$
```

The result shows that direct computation with 160-bit size takes 18.4 ms to compute 2^kP, but binary method takes 26.8 ms for the same computation. The performance of the direct computation may be further improved by new recoding method such as MOF.

10. DOUBLE BASE NUMBER SYSTEM

In 1996, a new number representation scheme, called *double-base number system* (DBNS) has been investigated mainly due to its applicability in digital signal processing area [21]. It uses a representation of the integers as the sum of mixed powers of two and three or of the form $2^b 3^t$, where b,t are non-negative integers. The DBNS representation is highly redundant and any positive integer can be represented as

$$n=\sum_{i=1}^{m} S_i\, 2^{b_i} 3^{t_i}, \text{ with } S_i \in \{-1,1\} \text{ and } b_i,t_i \geq 0.$$

Finding one of the canonic DBNS representations, especially for very large integers, seems to be a very difficult task and this is obtained by using the following Algorithm-12.

```
Algorithm-12: Greedy algorithm to convert integer into DBNS

Input   : A positive integer n

Output  : Sequence of exponents $(b_n,t_n)$
    (leading to one DBNS representation of n)

        While n>0 do
            find $z = 2^b 3^t$, the largest 2-integer less than or equal to n
            print (b,t)
        n=n-z
```

For example, DBNS representation of 314159 is $2^{12}3^4-2^{11}3^2+2^8 3^1+2^4 3^1-2^0 3^0$. It offers some natural, elegant, protection against side-channel attacks **[x]**. It is believed that the huge redundancy of the representation can be advantageously exploited in order to randomly change the order of the operation, such that the same scalar does not give the same trace if it is processing several times and it is an additional security.

Scalar multiplication algorithm based on double-base number system and double-base chains is proposed in [22]. Moreover, further to reduce the overall complexity, all the additions were combined with doubling, tripling or quadrupling operations. In the following Table-4, the cost of the curve operations is given (see [23] [24]).

Curve Operation	Prime field	Binary field
P+Q	1[i]+1[s]+2[m]	1[i]+1[s]+2[m]
2P	1[i]+2[s]+2[m]	1[i]+1[s]+2[m]
2P+Q	1[i]+2[s]+9[m]	1[i]+2[s]+9[m]
3P	1[i]+4[s]+7[m]	1[i]+4[s]+7[m]
3P+Q	2[i]+4[s]+9[m]	2[i]+4[s]+9[m]
4P	1[i]+9[s]+9[m]	1[i]+5[s]+8[m]
4P+Q	2[i]+4[s]+11[m]	2[i]+6[s]+10[m]

Table-4 : Number of inversions[i], squarings [s] and multiplications [m], for different curve operations

After obtaining the DBNS representation of a number, the Algorithm-13 is used for the computation of kP using double-base number system.

```
Algorithm-13: Double-Base scalar multiplication
Input :
point P and Integer $k = \sum_{i=1}^{m} S_i 2^{b_i} 3^{t_i}$, with $S_i \in \{-1,1\}$ and $b_i,t_i \geq 0$
Output : kP

Z=$S_1$P
for i=1 to m-1 do
    u=$b_i - b_{i+1}$
    v=$t_i - t_{i+1}$
    if u=0 then
        Z=$3(3^{v-1}Z) + S_{i+1}P$
    else
        Z=$3^v Z$
        Z=$4^{\lfloor (u-1)/2 \rfloor} Z$
        if u≡0 (mod 2) then
            Z=$4Z+ S_{i+1}P$
        else
            Z=$2Z+ S_{i+1}P$
Return Z
```

Though m-1 additions are required to compute kP, the above algorithm combines each addition with either a doubling (2P+Q), a tripling (3P+Q) or a quadrupling (4P+Q).

If we assume that k is a random n-bit integer, the binary algorithm requires n doublings and n/2 additions on average. In signed binary method, the average density of non-zero digits is reduced to n/3. If k is represented in a wNAF form, the average number of additions is reduced to n/(w+1). For 160-bit integers, and over prime fields, the corresponding costs can be estimated to 160[i] + 320[s] + 880[m] with the binary method, 106[i] + 320[s] + 691[m] with 2-NAF method, and 160[i] + 192[s] + 544[m] with 4-NAF method. The DBNS scalar multiplication requires 119[i] + 453[s] + 814[m] operations (See more details in [24]). It is found that that scalar multiplication algorithm using DBNS gains 20% compared to the classical binary method; 17.4% compared to 2-NAF method; and 15.25% compared to 4-NAF method.

11. CONCLUSION

Even though elliptic curve based cryptographic algorithms were widely accepted and used in many applications, still there are many areas available to improve. In this paper, the dominant operation called scalar multiplication is analyzed in many factors. Since from the inception, so many algorithms were proposed. Various representation of multiplier is presented and how it plays a role in the scalar multiplication is also discussed.

- Improve the performance of elliptic curve arithmetic operations
- Finding out efficient coordinate system and representations
- Identifying new scalar representation or reduce hamming weight
- Efficient and flexible scalar multiplication algorithm
    - Point compression and embedding and
    - Integrating in an application

REFERENCE:

[1] V.S. Miller, "Use of Elliptic Curves in Cryptography", Advances in Cryptology, Proceedings of CRYPTO’85, LNCS-218, pp. 417-426, 1986

[2] N.Koblitz, "Elliptic Curve Cryptosystem", Mathematics of Computation, Vol.48, pp.203-209, 1987

[3] R.L. Rivest, A. Shamir and L. M. Adleman, "A Method for Obtaining Digital Signatures and Public key Cryptosystems", Communications of the ACM, Vol. 21, pp. 120-126, 1978

[4] T. ElGamal, “Public key Cryptosystem and a Signature Scheme based on Discrete Logarithms”, IEEE Transactions on Information Theory, Vol. IT-31, No. 4, pp. 469-472, 1985

[5] W. Diffie and M. E. Hellman, "New Directions in Cryptography," IEEE Transactions on Information Theory, Vol. 22, No. 6, pp. 644-654, 1976

[6] J. Lopez and R. Dahab, ”An Overview of Elliptic Curve Cryptography”, Technical Report, Institute of Computing, State University of Combinas, Brazil, 2000

[7] D.M.Gordon, “A Survey of Fast Exponentiation Methods”, Journal of algorithms, vol.27, pp.129-146, 1998

[8] A.J. Menezes and S.A Vanstone, “Elliptic Curve Cryptosystems and Their Implementations”, Journal of Cryptology, Vol.6, No. 4, pp.209-224, 1993

[9] A.D.Booth, “A Signed binary multiplication technique”, Journal of Applied Mathematics, Vol. 4. No. 2, pp.236-240, 1951

[10] G.W.Reitwiesner, “Binary Arithmetic”, In Advances in computers, Academic Press, Vol.1, pp.231-308, 1960

[11] F.Morain and J.Olivos, “Speeding up the computations on an Elliptic curve using Addition-Subtraction chains”, RAIRO Theoretical Informatics and Applications, Vol.24, pp.531-543, 1990

[12] J.Gujardo and C. Paar, “Efficient Algorithms for Elliptic Curve Cryptosystems”, Advances in Cryptology-CRYPTO’97, Springer-Verlag LNCS, vol.1294, pp.342-356, 1997

[13] IEEE P1363, Standard Specifications for Public-Key Cryptography, 2000

[14] H.Cohen, A. Miyaji, and T. Ono, “Efficient Elliptic Curve Exponentiation Using Mixed Coordinates”, ASIACRYPT: Advances in Cryptology, LNCS, Vol. 1514, pp. 51-65, 1998

[15] K.Koyama and Y.Tsuruoka, “Speeding up Elliptic Cryptosystem by Using a Signed binary Window Method”, Advances in Cryptology – CRYPTO ’92, LNCS – 740, pp.345-357, 1993

[16] M.Joye and S.Yen, “Optimal Left-to-Right binary signed digit recoding”, IEEE Transactions on Computers, Vol. 49, pp. 740-748, 2000

[17] K.Okeya, “Signed binary representations revisited”, Proceedings of CRYPTO’04, pp.123-139, 2004

[18] J.A. Solinas, “Low-Weight binary representations for pairs of integers”, Technical Report CORR 2001-41, Center for Applied Cryptographic Research, University of Waterloo, Canada, 2001

[19] P.J. Grabner, C.Heuberger and H.Prodinger, “Distribution results for low-weight binary representations for pairs of integers”, Theoretical Computer Science, Vol. 319, pp. 307-331, 2004

[20] Y.Sakai and K.Sakurai, "Efficient Scalar Multiplications on Elliptic Curves with Direct Computations of Several Doublings", IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, E84-A, No.1, pp.120-129, 2001

[21] V.S.Dimitrov and G.A. Jullien, “Loading the bases: A new number representation with applications”, IEEE Circuits and Systems Magazine, Vol. 3. No. 2, pp.6-23, 2003

[22] V.S.Dimitrov, G.A. Jullien, and W.C. Miller, “An algorithm for modular exponentiation”, Information Processing letters, Vol. 66. No. 3. pp.155-159, 1998

[23] M.Ciet, M.Joye, K.Lauter, and P.L.Montgomery, “Trading inversions for multiplications in elliptic curve cryptography”, Cryptology ePrint Archive, Report 2003/257, 2003

[24] K.Eisentrager, K.Lauter, and P.L.Montgomery, “Fast elliptic curve arithmetic and improved weil pairing evaluation”, Topics in Cryptology–CT-RSA 2003, vol. 2612, Springer-Verlag LNCS, pp. 343-354, 2003

[25] Michael Rosing, Implementing Elliptic Curve Cryptography, Manning Publications, 1998

[26] D Hankerson, A. J Menezes, S. Vanstone, Guide to Elliptic Curve Cryptography, Springer, 2004

[27] Software Implementation of the NIST Elliptic Curves Over Prime Fields, M. Brown, D. Hankerson, J. Lopez, and A. Menezes, Proc. of CT-RSA'2001, Springer-Verlag, LNCS 2020, pp.250-265, 2001.

[28] Software Implementation of Elliptic Curve Cryptography Over Binary Fields, Darrel Hankerson, Julio Lopez Hernandez, Alfred Menezes, Cryptographic Hardware and Embedded Systems, CHES'2000, pp.1-24, 2000.

[29] D.V. Bailey, C. Paar C , Efficient arithmetic in finite field extensions with application in elliptic curve cryptography, Journal of cryptology (J. cryptology) 2001, vol. 14, no.3, pp. 153-176

Author (s) Biography

Dr.E.Karthikeyan completed PG degree in the year 1996 and Ph.D from Gandhigram University, Dindigul, India in 2008. He is guiding students towards Ph.D. programme and his area of research is Network Security and Cryptography and Advanced Networking. He has published 16 papers in International Journals and more than 15 conferences National and International level. He has also published a book entitled “Text Book on C: Fundamentals, Datastructures and Programming” by Prentice Hall of India. He delivered lectures and conducted workshops in various colleges. He is a life member of CSI, CRSI, IASCT etc. He is an Editor-in-Chief for an International Journal of Advanced Networking and Applications (IJANA) and reviewer in many journals and advisory committee in various conferences. He organized many workshops and conferences.