Efficient Window-Based Scalar Multiplication on Elliptic Curves Using Double-Base Number System

Rana Barua, Sumit Kumar Pandey, and Ravi Pankaj

Indian Statistical Institute
205, B.T. Road
Kolkata, India
{rana,mtc0518,mtc0520}@isical.ac.in

Abstract. In a recent paper [10], Mishra and Dimitrov have proposed a window-based Elliptic Curve (EC) scalar multiplication using double-base number representation. Their methods were rather heuristic. In this paper, given the window lengths w_2 and w_3 for the bases 2 and 3, we first show how to fix the number of windows, ρ, and then obtain a Double Base Number System (DBNS) representation of the scalar n suitable for window-based EC scalar multiplication. Using the DBNS representation, we obtain our first algorithm that uses a small table of precomputed EC points. We then modify this algorithm to obtain a faster algorithm by reducing the number of EC additions at the cost of storing a larger number of precomputed points in a table. Explicit constructions of the tables are also given.

1 Introduction

The efficiency of Elliptic Curve Cryptography (ECC) implementation largely depends upon how fast one can compute the point $[n]P = \sum_{i=1}^{n} P$, given a point P on the curve and the integer (scalar) n. Several efficient algorithms for computing $[n]P$ have been proposed. See Avanzi et al [1] and Hankerson et al [8] for detailed discussion on these methods. Several window-based methods have also been proposed, of them w-NAF methods seem to be very efficient.

Recently, Mishra and Dimitrov[10] have proposed a new window-based scalar multiplication algorithm by suitably representing the scalars in DBNS. The DBNS has recently been exploited to compute exponentiation (or scalar multiplication) efficiently[5]. The sparseness of the representation leads to fewer point additions than the usual double-and-add or NAF methods. In fact, one can have a DBNS representation of n having $O(logn/loglogn)$ terms. This together with the fact that $2^a3^b[P]$, for an EC point P, can be computed efficiently([6]) gives rise to some very efficient algorithms for scalar multiplication. However, the method in [10] is quite heuristic and an explicit method for finding the *partition length* ρ is not given, nor any explicit expression for the cost of scalar multiplication in terms of EC addition, doubling or tripling.

K. Srinathan, C. Pandu Rangan, M. Yung (Eds.): Indocrypt 2007, LNCS 4859, pp. 351–360, 2007.

In this paper, we first show how to fix ρ, the length of the partition i.e. the number of windows, given w_2, w_3, the *lengths of the window corresponding to the bases 2 and 3* respectively. We then obtain a DBNS representation of the scalar n suitable for window-based scalar multiplication. We obtain the DBNS representation more efficiently than in [10] using a much smaller search space. Using our DBNS representation we obtain our scalar multiplication algorithm using a table look-up that stores $(w_2+1)(w_3+1)$ EC points. Explicit construction of the table is also given. Using a larger table that stores $(2^{w_2} \times 3^{w_3})/2$ points, we modify the above algorithm that considerably reduces the number of EC point additions. We also obtain an expression for the average number of EC additions, doubling, tripling required for computing the scalar multiplication.

2 Double-Base Number System

The double base number system (DBNS) [6] is a representation scheme in which every integer n is represented as the sum or difference of numbers of the form $2^a 3^b$(called $\{2,3\}$-integers) i.e.

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

This representation is very short and the representation scheme is highly redundant. It has been shown that (cf [5], every positive integer n can be represented as the sum of at most $O(logn/loglogn)$ $\{2,3\}$-integers. A simple Greedy algorithm ensures nearly shortest representation for a given integer. A modified Greedy was proposed in [10] suitable for window-based scalar multiplication. Here we propose a more efficient algorithm suitable for window-based method that uses a search space consisting only of $2^{w_2} 3^{w_3}$ integers.

3 Proposed Window-Based Method for Scalar Multiplication

Unlike earlier proposed methods ([10], [6], [11]), by choosing the window sizes, we obtain natural bounds on maximum exponents of bases 2 and 3, and propose a window method so that it reduces the overall storage.

Let n be an r-bit integer. Let w_2, w_3 be the *dimension* of the window. Let max_2, max_3 be integers satisfying $2^{max_2} 3^{max_3} \geq n$ and such that $max_2/w_2 = max_3/w_3 = \rho$, say (as in [10]). Then, we have

$$max_2 + max_3 \log_2 3 \geq \log_2 n \tag{1}$$

Substituting $max_2 = \rho w_2$, $max_3 = \rho w_3$ in (1), we get

$$\rho \geq \frac{\log_2 n}{w_2 + w_3 \log_2 3} \tag{2}$$

For fix window lengths, we can obtain the number ρ by choosing ρ to be the least positive integer satisfying inequality (2). If n is r-bit, then we may also choose $\rho = \lceil r/(w_2 + w_3 \log_2 3) \rceil$. *Henceforth, we fix such a number* ρ.

3.1 Representation of n

There is no unique representation of n in double base number system. Finding a canonical representation of n, i.e. having least number of terms, is extremely difficult. A short and sparse representation of n results in less computation for scalar multiplication. We propose a representation of n which can be obtained very efficiently and will be suitable for our window-based method. Since we are particularly interested in window method, we will first obtain a DBNS representation of integers m *lying in the window* i.e. $0 \leq m \leq 2^{w_2}3^{w_3}$ using (distinct) terms in the window.

Proposition 3.1. *Every integer* $0 \leq m \leq 2^{w_2}3^{w_3}$ *can be represented as* $\sum_j s_j 2^{b_j} 3^{t_j}$*, where* $s_j \in \{-1, 1\}$ *and* $0 \leq b_j \leq w_2, 0 \leq t_j \leq w_3$*.*

To find a DBNS representation of m lying in a window, we will use a table T such that $T(a,b) = 2^a 3^b$ where $0 \leq a \leq w_2$ and $0 \leq b \leq w_3$. With the help of table T, we can easily find the nearest $\{2,3\}$-integer lying in a window and hence the double base representation of any integer m, lying in a window.

Algorithm 1 gives the method to find a DBNS representation of n by greedy approach which is almost the same as in [10].

Algorithm 1. Conversion into DBNS

Input : an integer m such that $0 \leq m \leq 2^{w_2}3^{w_3}$ for a given window lengths w_2, w_3 for 2, 3 respectively and table T, where $T(a,b) = 2^a 3^b$ and $0 \leq a \leq w_2, 0 \leq b \leq w_3$.

Output : The sequence (s_i, b_i, t_i) such that $m = \sum_{i=1}^{l} s_i 2^{b_i} 3^{t_i}$, where $s_i \in \{-1, 1\}, 0 \leq b_i \leq w_2, 0 \leq t_i \leq w_3$

$i \leftarrow 1$
$s_i \leftarrow 1$
$A[i] \leftarrow (0,0,0)$
while $m > 0$ do
  define $X = 2^{b_i}3^{t_i}$, the best approximation of m in T with $0 \leq b_i \leq w_2$ and $0 \leq t_i \leq w_3$. If there are two choices, choose nearest integer smaller to m.
  $A[i] \leftarrow (s_i, b_i, t_i)$
  if $m < X$ **then**
    $s_{i+1} \leftarrow -s_i$
  $m \leftarrow |m - X|$
  $i \leftarrow i + 1$
return A.

Now, for any integer n, by our choice of ρ we have $0 \leq n \leq (2^{w_2}3^{w_3})^{\rho}$. The propositon below gives a way to represent n suitable for our purpose.

Proposition 3.2. *Every integer* $0 \leq n \leq (2^{w_2}3^{w_3})^{\rho}$ *can be represented as* $n = M_{\rho-1}(2^{w_2}3^{w_3})^{\rho-1} \pm M_{\rho-2}(2^{w_2}3^{w_3})^{\rho-2} \pm \cdots \pm M_0$ *s.t.* $0 \leq M_{\rho-1} \leq 2^{w_2}3^{w_3}$ *and* $0 \leq M_j \leq (2^{w_2}3^{w_3})/2$ *for* $0 \leq j \leq \rho - 1$*.*

Proof: If $n = (2^{w_2}3^{w_3})^\rho$, then it is obvious. So let $0 \le n < (2^{w_2}3^{w_3})^\rho$. Then $n = M'_{\rho-1}(2^{w_2}3^{w_3})^{\rho-1} + R'_{\rho-1}$, where $0 \le R'_{\rho-1} < (2^{w_2}3^{w_3})^{\rho-1}$. Clearly $M'_{\rho-1} < 2^{w_2}3^{w_3}$,for otherwise $n \ge (2^{w_2}3^{w_3})^\rho$. If $R'_{\rho-1} > (2^{w_2}3^{w_3})^{\rho-1}/2$, take $M_{\rho-1} = M'_{\rho-1} + 1$ and $R_{\rho-1} = R'_{\rho-1} - (2^{w_2}3^{w_3})^{\rho-1}$, else $M_{\rho-1} = M'_{\rho-1}$ and $R_{\rho-1} = R'_{\rho-1}$. So, $n = M_{\rho-1}(2^{w_2}3^{w_3})^{\rho-1} + R_{\rho-1}$, where $0 \le M_{\rho-1} \le 2^{w_2}3^{w_3}$ and $-(2^{w_2}3^{w_3})^{\rho-1}/2 < R_{\rho-1} \le (2^{w_2}3^{w_3})^{\rho-1}/2$.

Now, take $|R_{\rho-1}|$ so that $0 \le |R_{\rho-1}| \le (2^{w_2}3^{w_3})^{\rho-1}/2$.
Let $|R_{\rho-1}| = M_{\rho-2}(2^{w_2}3^{w_3})^{\rho-2} + R_{\rho-2}$, where $0 \le M_{\rho-2} \le 2^{w_2}3^{w_3}/2$ and $-(2^{w_2}3^{w_3})^{\rho-2}/2 < R_{\rho-2} \le (2^{w_2}3^{w_3})^{\rho-2}/2$. Observe that $M_{\rho-2} \not> (2^{w_2}3^{w_3})/2$, for otherwise $|R_{\rho-1}| > (2^{w_2}3^{w_3})^{\rho-1}/2$.

Proceeding similarly, we have the result.

Algorithm 2 gives the method to find a representation of n.

Algorithm 2. To find representation of n

Input : an integer n, window dimension w_2, w_3 and ρ.
Output: a seq. of $(s_i, M_i)_{i>0}$ such that $n = \sum_{i=1}^{\rho} s_{\rho-i}M_{\rho-i}(2^{w_2}3^{w_3})^{\rho-i}$, where $s_i \in \{-1, 1\}$, $0 \le M_{\rho-1} \le 2^{w_2}3^{w_3}$ and $0 \le M_{\rho-i} \le (2^{w_2}3^{w_3})/2$ for all $2 \le i \le \rho$.

$i \leftarrow 1$
$s_{\rho-1} \leftarrow 1$
$R \leftarrow n$
$X \leftarrow (2^{w_2}3^{w_3})^{\rho-1}$
while $i \le \rho$ **do**
  $M_{\rho-i} \leftarrow \lfloor R/X \rfloor$
  $R \leftarrow R - M_{\rho-i}X$
  $s_{\rho-i-1} \leftarrow s_{\rho-i}$
  if $R > X/2$ **then**
    $M_{\rho-i} \leftarrow M_{\rho-i} + 1$
    $R \leftarrow X - R$
    $s_{\rho-i-1} \leftarrow -s_{\rho-i}$
  $X \leftarrow X/2^{w_2}3^{w_3}$
  $i \leftarrow i + 1$
  $A[\rho - i] \leftarrow (s_{\rho-1}, M_{\rho-i})$
return A

After getting a representation of n, we are in position to find $[n]P$, given an EC point P, using Horner's scheme. To calculate $[n]P$, we will use another table T^P which contains the *precomputed* values of $[2^a3^b]P$, where $0 \le a \le w_2$ and $0 \le b \le w_3$, i.e. $T^P(a, b) = [2^a3^b]P$. Observe that, since negation of an EC point can be obtained almost free, we omit its cost in our calculation. Using T^P, we can find $[n]P$ as follows.

1. Compute ρ. Then we calculate $M_j's$, where $n = \sum_{j=1}^{\rho} s_{\rho-j}M_{\rho-j}(2^{w_2}3^{w_3})^{\rho-j}$, where M_j, s_j's are as in Proposition 3.2. (Algorithm 2).
2. Now, we find out $[M_j]P$, (Algorithm 3). To obtain this we first find representation of M_j in DBNS using Algorithm 2, say $\sum_{j=1}^{l} s_j 2^{b_j} 3^{t_j}$. Then looking

at table T^P, we find the values of $s_j[2^{b_j}3^{t_j}]P$ for all $j = 1, \ldots, l$ and adding these points gives the value of $[M_j]P$.

3. After getting the values of all $[M_j]P$ in the representation of n, we evaluate $[n]P$ by applying Horner's scheme (Algorithm 4).

Algorithm 3. calculation of $[m]P$, for m in the window

Input : an integer m such that $0 \le m \le 2^{w_2}3^{w_3}$, a point P on an elliptic curve E, tables T and T^P.
Output : $[m]P$
$A \leftarrow$ **Algorithm 1**(m, w_2, w_3, T)
$L \leftarrow length(A)$
$P \leftarrow O$ (point at infinity on elliptic curve E)
$i \leftarrow 1$
while $i \le L$ **do**
  $(s_i, b_i, t_i) \leftarrow A[i]$
  $P \leftarrow P + s_i T^P(b_i, t_i)$
  $i \leftarrow i + 1$
return P

It is not hard to check that the number of terms in the DBNS representation of m lying in the window is at most $c(w_2 + log3w_3)$ for $c < 1$. Perhaps a much better estimate can be obtained. Thus we have.

Proposition 3.3. *Algorithm 3 correctly computes $[m]P$ for $0 \le m \le 2^{w_2}3^{w_3}$ using at most $c(w_2 + log3w_3)$ additions. The table T^P stores $(w_2 + 1)(w_3 + 1)$ EC points.*

Algorithm 4 calculates $[n]P$.

Algorithm 4. Calculation of $[n]P$

Input : an integer n such that $0 \le n \le (2^{w_2}3^{w_3})^\rho$, a point P on an elliptic curve E, partition length ρ, tables T and T^P.
Output : $[n]P$
$A \leftarrow$ **Algorithm 2**(n, w_2, w_3, ρ)
$P \leftarrow O$ (point at infinity on elliptic curve E)
$i \leftarrow 1$
while $i \le \rho$ **do**
  $(s_{\rho-i}, M_{\rho-i}) \leftarrow A[\rho - i]$
  $Q \leftarrow$ **Algorithm 3**$(M_{\rho-i}, w_2, w_3, P, T, T^P)$
  $P \leftarrow P + s_{\rho-i}Q$
  $P \leftarrow [3^{w_3}]P$
  $P \leftarrow [2^{w_2}]P$
  $i \leftarrow i + 1$
return P

The following is not very hard to check using Horner's scheme, since the probabily of each M_j being non-zero is $(1 - \frac{1}{2^{w_2}3^{w_3}})$.

Proposition 3.4. *Algorithm 4 correctly computes $[n]P$ using on an average $(t-1)(c(w_2+log3w_3)-1)$ EC additons, $(\rho-1)w_2$ point doublings and $(\rho-1)w_3$ point triplings, where $c<1, \rho=\lceil \frac{log_2 n}{w_2+w_3 log_2 3} \rceil$, and $t=(1-\frac{1}{2^{w_2}3^{w_3}})\rho$. Table T stores $(w_2+1)(w_3+1)$ integers, while table T^P stores $(w_2+1)(w_3+1)$ EC points.*

We can reduce the cost of computation if more precomputed points are stored. For that we construct T^P_{all} instead of T^P such that $T^P_{all}(i)=[i]P$ for $1 \leq i \leq 2^{w_2}2^{w_3}/2$. Since in the representation of n, maximum value of M_j can be $(2^{w_2}3^{w_3}/2)$, except $M_{\rho-1}$ which can have maximum value $2^{w_2}3^{w_3}$, the number of precomputed points is $(2^{w_2}3^{w_3}/2)$. If $M_{\rho-1} > 2^{w_2}3^{w_3}/2$ then $[M_{\rho-1}]P$ can be evaluated by calculating first $\lfloor [M_{\rho-1}/2] \rfloor P$ and then doubling and adding P if $M_{\rho-1}$ is odd. Hence, steps for evaluating $[n]P$ using table T^P_{all} will be same except step 3. In modified step 3, we will evaluate $[M_j]P$ by just looking at table T^P_{all}.

Algorithm 4^0. Calculation of $[n]P$

Input : an integer n such that $0 \leq n \leq (2^{w_2}3^{w_3})^\rho$, a point P on an elliptic curve E, partition length ρ, table T^P_{all}.
Output : $[n]P$

```
1: A ← Algorithm 3(n, w_2, w_3, ρ)
2: P ← O (point at infinity on elliptic curve E)
3: i ← 1
4: while i ≤ ρ do
5:     (s_{ρ−i}, M_{ρ−i}) ← A[ρ − i]
6:     if M_{ρ−i} = 0 then
7:         Q ← O (point at infinity)
8:     else
9:         if i = 1 then
10:            if M_{ρ−i} > 2^{w_2} 3^{w_3}/2
11:                M_{ρ−i} ← ⌊M_{ρ−i}/2⌋
12:                Q ← 2[T^P_{all}(M_{ρ−i})]
13:                if M_{ρ−i} is odd then
14:                    Q ← Q + P
15:            else
16:                Q ← T^P_{all}(M_{ρ−i})
17:        else
18:            Q ← T^P_{all}(M_{ρ−i})
19:    P ← P + s_{ρ−i} Q
20:    P ← [3^{w_3}]P
21:    P ← [2^{w_2}]P
22:    i ← i + 1
23: return P
```

The following is now clear.

Proposition 3.5. *Algorithm 4^0 correctly computes $[n]P$ using $(\rho-1)w_2$ point doublings, $(\rho-1)w_3$ point triplings and at most ρ point additions. Table T^P_{all} stores $2^{w_2}3^{w_3}/2$ EC points.*

4 Computation of T^P and T^P_{all}

The algorithms described so far use one or more look-up tables. If the EC point P is known in advance, then the tables can be precomputed and stored; otherwise they have to be computed online. Formation of tables T, T^P, and T^P_{all} may take much computation but it can be reduced if they are formed recursively.

Note that $T(a,b) = 2^a 3^b$ and $T^P(a,b) = [2^a 3^b]P$, so $T^P(a,b) = [T(a,b)]P$. By considering the lexicographic ordering of the tuples (a,b) we can form T, T^P as follows:

1. $T(0,0) = 1;$ $\quad T^P(0,0) = P$
2. $T(0,b) = 3T(0,b-1);$ $\quad T^P(0,b) = [3]T^P(0,b-1), b > 0.$
3. $T(a,b) = 2T(a-1,b);$ $\quad T^P(a,b) = [2]T^P(a-1,b), a > 0.$

Algorithm 5 illustrates the method to form table T^P.

Algorithm 5. Table construction for T^P

Input : window lengths w_2, w_3 and an EC point P
Output : an array $T^P(a,b)$ such that $T^P(a,b) = [2^a 3^b]P$ where $0 \le a \le w_2$ and $0 \le b \le w_3$.

$T^P(0,0) \leftarrow P$
$a \leftarrow 0$
$b \leftarrow 0$
while $b < w_3$ **do**
    $T^P(a,b+1) \leftarrow [3]T^P(a,b)$
    $b \leftarrow b+1$
$b \leftarrow 0$
while $b < w_3 + 1$ **do**
    while $a < w_2$ **do**
        $T^P(a+1,b) \leftarrow [2]T^P(a,b)$
        $a \leftarrow a+1$
    $b \leftarrow b+1$
return T^P

Proposition 4.1. *Algorithm 5 correctly computes T^P used in Algorithm 4 using w_3 triplings and $w_2(w_3+1)$ doublings.*

Remarks: If tripling is less expensive than doubling(as in the case of EC over fields of characteristic 3), we form T^P as follows:

1. $T^P(0,0) = P$
2. $T^P(a,0) = 2[T^P(a-1,0)]$, $a > 0$ $\quad T^P(a,b) = 3[T^P(a,b-1)]$, $b > 0$.

One can then appropriately modify Algorithm 5, using the above recursive relation. This will involve w_2 doubling and $w_3(w_2+1)$ triplings

Finally, Table T^P_{all} can be formed as follows, if T^P is given:

1. $T^P_{all}(1) = P$ $\quad T^P_{all}(m+1) = T^P(m+1)$, if $m+1$ is a $\{2,3\}$-integer,
2. $T^P_{all}(m+1) = T^P_{all}(m) + P$, otherwise.

Clearly this requires $2^{w_2}3^{w_3}/2 - (w_1 w_2 + w_1 + w_2)$ EC point additions.

5 Comparison

The present method for scalar multiplication is comparable or performs better in terms of both storage and computation in many cases.

(a) Storage - Methods for scalar multiplication in [10], [11] and [6] use a table T of large size to find the nearest representation of n, but in our method the table size required to find nearest representation of n is comparatively very small.

(b) Computation - In this method, we use a table T^P or T^P_{all} of precomputed points, which reduces the overall computation in scalar multiplication. We have computed cost of $[n]P$ using existing algorithm for $[2^{w_2}]P$ ([4]) and $[3]P$ ([2]) in affine coordinates for curves over characteristic 2. Since square is almost free in affine coordinates, we have not taken the cost of squaring. On the other hand, cost for computing $[n]P$ has been calculated using algorithm for $[2^{w_2}]P$ ([9]), $[3^{w_3}]P$ ([6]) and mixed addition ([3]) in Jacobian coordinates. Table 1 summarizes the cost of operation required.

We calculated cost of field operations for different window lengths in Table 2. We compared our results with some earlier methods in Table 3.

Table 1. Cost of operation required in different point addition algorithm. Here **[I], [S]** and **[M]** denote cost of field inversion, squaring and multiplication respectively.

Operation	cost	
	Affine	Jacobian
$P + Q$	$1[\mathbf{I}] + 2[\mathbf{M}]$	$4[\mathbf{S}] + 12[\mathbf{M}]$
mixed-$(P + Q)$	-	$3[\mathbf{S}] + 8[\mathbf{M}]$ $(cf[3])$
$[2^w]P$	$1[\mathbf{I}] + (4w - 2)[\mathbf{M}]$ $(cf[4])$	$(4w + 2)[\mathbf{S}] + 4w[\mathbf{M}]$ $(cf[9])$
$[3]P$	$1[\mathbf{I}] + 7[\mathbf{M}]$ $(cf[2])$	$6[\mathbf{S}] + 10[\mathbf{M}]$ $(cf[6])$
$[3^w]P$	-	$(4w + 2)[\mathbf{S}] + (11w - 1)[\mathbf{M}]$ $(cf[6])$

Table 2. Cost of scalar multiplication for 160 bit scalar

w_2	w_3	using T^P			using T^P_{all}		
		# storage	Affine $[\mathbf{I}]/[\mathbf{M}] = 8$	Jacobian $[\mathbf{S}]/[\mathbf{M}] = 0.8$	# storage	Affine $[\mathbf{I}]/[\mathbf{M}] = 8$	Jacobian $[\mathbf{S}]/[\mathbf{M}] = 0.8$
1	1	4	2042.3 [**M**]	1973.7[**M**]	3	2066.7[**M**]	2000.5[**M**]
1	2	6	2030.0[**M**]	1966.8[**M**]	9	1878.9[**M**]	1809.6[**M**]
1	3	8	1993.5[**M**]	1932.9[**M**]	27	1750.0[**M**]	1674.6[**M**]
2	1	6	1716.0[**M**]	1812.8[**M**]	6	1679.3[**M**]	1774.7[**M**]
2	2	9	1775.0[**M**]	1823.2[**M**]	18	1665.4[**M**]	1708.4[**M**]
2	3	12	1800.3[**M**]	1822.7[**M**]	54	1584.9[**M**]	1598.6[**M**]
3	1	8	1578.7[**M**]	1766.9[**M**]	12	1490.4[**M**]	1678.8[**M**]
3	2	12	1637.8[**M**]	1760.3[**M**]	36	1504.4[**M**]	1623.8[**M**]
3	3	16	1689.4[**M**]	1774.6[**M**]	108	1459.1[**M**]	1535.0[**M**]
4	1	10	1485.2[**M**]	1732.7[**M**]	24	1310.2[**M**]	1550.7[**M**]
4	2	15	1584.4[**M**]	1764.8[**M**]	72	1362.5[**M**]	1534.0[**M**]
4	3	20	1632.3[**M**]	1768.1[**M**]	216	1385.6[**M**]	1511.6[**M**]

Table 3. Comparison among different proposed methods

Algorithm	size of Table T	# Precomputed points	Affine $[\mathbf{I}]/[\mathbf{M}] = 8$	Jacobian $[\mathbf{S}]/[\mathbf{M}] = 0.8$
for 160 bit scalar				
Mishra-Dimitrov method [11]	48384	5	1469.0[**M**]	1502.0[**M**]
Mishra-Dimitrov method [10] for window length (3,2)	4332	26	-	1692.2[**M**]
w−NAF (for $w = 3$)	0	3	2016[**M**]	-
w−NAF (for $w = 4$)	0	5	1894[**M**]	-
Our method (using T^P) for window length (4,1)	10	10	1485.2[**M**]	1732.7[**M**]
Our method (using T^P_{all}) for window length (4,1)	10	24	1310.2[**M**]	1550.7[**M**]
Our method (using T^P) for window length (4,2)	15	15	1584.4[**M**]	1764.8[**M**]
Our method (using T^P_{all}) for window length (4,2)	15	72	1362.5[**M**]	1534.0[**M**]
for 200 bit scalar				
Doche-Imbert method [7] for window length (1,1)	313	3	-	2019[**M**]
Our method (using T^P) for window length (4,1)	10	10	-	2312.6[**M**]
Our method (using T^P) for window length (4,2)	15	15	-	2272.9[**M**]
Our method (using T^P_{all}) for window length (4,1)	10	24	-	1938.4[**M**]

References

1. Avanzi, R.M., Cohen, H., Doche, C., Frey, G., Langue, T., Nguyen, K., Vercauteren, F.: Handbook of Elliptic and Hyperelliptic Curve Cryptography. CRC press, Boca Raton, USA (2005)
2. Ciet, M., Lauter, K., Joye, M., Montgomery, P.L.: Trading inversions for multiplications in elliptic curve cryptography. Designs, Codes and Cryptography 39(2), 189–206 (2006)
3. Cohen, H., Miyaji, A., Ono, T.: Efficient Elliptic Curve Exponentiation Using Mixed coordinates. In: Ohta, K., Pei, D. (eds.) ASIACRYPT 1998. LNCS, vol. 1514, pp. 51–65. Springer, Heidelberg (1998)
4. Dahab, R., Lopez, J.: An Improvement of Guajardo-Paar Method for Multiplication on non-supersingular Elliptic Curves. In: SCCC 1998. Proceedings of the XVIII International Conference of the Chilean Computer Science Society, November 12-14, pp. 91–95. IEEE Computer Society Press, Los Alamitos (1998)
5. Dimitrov, V., Gullien, G.A., Miller, W.C.: An algorithm for modular exponentiation. Information Processing Letters 66(3), 155–159 (1998)
6. Dimitrov, V., Imbert, L., Mishra, P.K.: Efficient and Secure Curve Point Multiplication Using Double Base Chain. In: Roy, B. (ed.) ASIACRYPT 2005. LNCS, vol. 3788, pp. 59–79. Springer, Heidelberg (2005)

7. Doche, C., Imbert, L.: Extended Double-Base Number System with Applications to Elliptic Curve Cryptography. In: Barua, R., Lange, T. (eds.) INDOCRYPT 2006. LNCS, vol. 4329, pp. 335–348. Springer, Heidelberg (2006)
8. Hankerson, D., Menezes, A., Vanstone, S.: Guide to Elliptic Curve Cryptography. Springer, New York (2004)
9. Itoh, K., Takenaka, M., Torii, N., Temma, S., Kurihara, Y.: Fast implementation of public-key cryptography on a DSP TMS320C6201. In: Koç, Ç.K., Paar, C. (eds.) CHES 1999. LNCS, vol. 1717, p. 6172. Springer, Heidelberg (1999)
10. Mishra, P.K., Dimitrov, V.: Window-Based Elliptic Curve Scalar Multiplication using Double Base Number Representation, Short Papers, Inscrypt (2007)
11. Mishra, P.K., Dimitrov, V.: Efficient Quintuple Formulas for Elliptic Curves and Efficient Scalar Multiplication using Multibase Number Representation, ePrint archive. In report 2007/040 (2007), http://www.iacr.org