Fast Multiple Point Multiplication on Elliptic Curves over Prime and Binary Fields using the Double-Base Number System

Jithra Adikari[†], Vassil S. Dimitrov[†], and Pradeep Mishra[‡]

[†]Department of Electrical and Computer Engineering
Schulich School of Engineering
[‡]Department of Mathematics and Statistics

The University of Calgary
2500 University Drive N.W.
Calgary AB
Canada T2N 1N4

{jithra.adikari, dimitrov}@atips.ca, pradeep@math.ucalgary.ca

***Abstract*— Multiple-point multiplication on elliptic curves is the highest computational complex operation in the elliptic curve cyptographic based digital signature schemes. We describe three algorithms for multiple-point multiplication on elliptic curves over prime and binary fields, based on the representations of two scalars, as sums of mixed powers of 2 and 3. Our approaches include sliding window mechanism and some pre-computed values of points on the curve. A proof for formulae to calculate the number of double-based elements, doublings and triplings below 2^n is listed. Affine coordinates and Jacobian coordinates are considered in both prime fields and binary fields. We have achieved upto 24% of improvements in new algorithms for multiple-point multiplication.**

I. Introduction

The public key cryptography which was proposed by Diffie and Hellman in [5] in 1976, changed directions, in the field of cryptography. Thereafter, the RSA[23] algorithm dominated the commercial world for public key cryptography. In 1985, Koblitz[18] and Miller[21] proposed the elliptic curve cryptography (ECC) which is based *elliptic curve discrete logarithm problem* (ECDLP). The main advantage of the elliptic curve cryptography is the relatively small key-size. For an example, a 160-bit elliptic curve key provides the same security as a 1024-bit RSA key[15].

The concept of *Digital signature* which is derived from public key cryptography, is equivalent to that of handwritten signature in paper-based communications. RSA signature[23], Elgamal signature[13], and Digital Signature Algorithm (DSA) [1][2] are widely used in today's commercial applications. Elliptic curve digital signature algorithm (ECDSA)[2] and elliptic curve Korean certificate-based digital signature algorithm (EC-KCDSA)[16] are the most common two elliptic curve cryptographic digital signature algorithms.

Generally, elliptic curve point multiplication consumes high computational power in elliptic curve cryptography. Number of techniques have been introduced to overcome this problem [3], [15], [19]. In addition to that, in elliptic curve digital signature verification, the multiple point multiplication consumes huge amount of computational power. Shamir method simultaneous multiple point multiplication[13], fast Shamir method, non-adjacent form (NAF)[25], joint sparse form[24] and interleaving with NAF and sliding window methods[15] have been used for improving the efficiency of the multiple point multiplication. In this paper, we propose double-base number system techniques to improve the efficiency of verifying the digital signature in both prime and binary fields. The coordinate systems that we considered, are Affine coordinates and Jacobian projective coordinates over both fields.

The double-base number system (DBNS) was introduced in [9] for cryptographic applications and later extended for elliptic curve cryptography in [7] and [8]. We apply double-base number system for multiple point multiplication because it inherits sparsity and redundancy in the DBNS number representation. We propose three new algorithms for multiple point multiplication derived from double-base number system. Moreover, combination of doublings and triplings with point additions improves performance in prime and binary fields.

The sequel of the paper is organized as follows: In the Section II, we review theory of ECC and DBNS. The existing multiple point multiplication algorithms are briefly reviewed in the Section III. We propose all three new algorithms for multiple point multiplication based on DBNS in Section IV and presents and analyze the experimental results in Section V. Finally, we discuss the significant improvements using this new algorithms and conclude the paper in Section VI with a discussion in implementation and future work.

II. Background

Theory of elliptic curve cryptography and double-base number system are briefly reviewed in this section.

A. Elliptic Curve Cryptography

Elliptic curves have been introduced to public key cryptography by Koblitz and Miller independently in 1985.

1) Elliptic Curve Cryptography: -

Definition 1 (elliptic curve): *An elliptic curve E over a field K is defined by the Weierstrass equation, given by*

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \quad (1)$$

where, $\{a_1, a_2, a_3, a_4, a_6\} \in K$ *and* $\Delta \neq 0$*, where* Δ *is the discriminant of E and is defined as follows:*

$$\begin{aligned} \Delta &= -{d_2}^2 d_8 - 8{d_4}^3 - 27{d_6}^2 + 9d_2d_4d_6 \\ d_2 &= {a_1}^2 + 4a_2 \\ d_4 &= 2a_4 + a_1a_3 \\ d_6 &= {a_3}^2 + 4a_6 \\ d_8 &= {a_1}^2a_6 + 4a_2a_6 - a_1a_3a_4 + a_2{a_3}^2 - {a_4}^2 \end{aligned} \quad (2)$$

E is defined over *K*, and coefficients a_1, a_2, a_3, a_4, a_6 are elements of *K*. The condition $\Delta \neq 0$ ensures that elliptic curve is *smooth*, which means that there are no points at which the curve has two or more distinct tangent lines.

The field *K* can be either prime, binary, ternary or optimal extension field. All of these fields are used for cryptographical purposes. Properties of the elliptic curve depends on the field and the characteristics of the Eqn. 1.

If *K* is a prime field ($\mathbb{F}_p$), *E* transforms into the curve which is represented in Eqn. 3.

$$y^2 = x^3 + ax^2 + b \quad (3)$$

where, $a, b \in K$ and the discriminant of this curve is $\Delta = -16(4a^3 + 27b^2)$. The Weierstrass equation, *E* can be reduced to Eqn. 4, if *K* is a binary field ($\mathbb{F}_{2^m}$) and $a_1 \neq 0$.

$$y^2 + xy = x^3 + ax^2 + b \quad (4)$$

where, $a, b \in K$ and $\Delta = b$. Such a curve is said to be non-supersingular.

2) Elliptic Curve Arithmetic: -

Let *E* be an elliptic curve defined over the field *K* and *P*, *Q* be two points in *E/K*. The third point in *E/K* which is derived by adding two points in *E/K* by using the chord-and-tangent rule [3], [15]. The set of points *E/K* forms an abelian group with ∞ serving as its identity. The sum of *P* and *Q* is defined as follows: draw a line through *P* and *Q*; the intersection of this line and the elliptic curve is the negation of the resultant point; take the reflection of that point about the *x*-axis. This point gives the resultant point as depicted in the Fig. 1. The double of a point, *P* is defined: draw the tangent line to the point *P*; intersection of this tangent line and the elliptic curve is the negation of the double of the point; take the reflection about *x*-axis to get the resultant point. Given

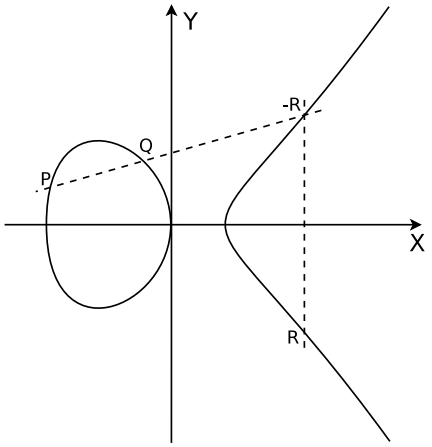

Fig. 1. Elliptic Curve Addition

a point $P \in E(K)$ and $k \in \mathbb{N}$, the operation of computing the new point $k \times P$ is called point multiplication or scalar multiplication. This operation is computationally dominant in ECC.

Digital signature algorithms need multiple point multiplication for the verification process. In this process, much of the computational power is spent on multiple point multiplication. If both points *P*, *Q* are on the elliptic curve and both integers *k*, *l* are less than the order of the elliptic curve over the field, the operation of computing the new point *kP + lQ* is called the multiple point multiplication.

TABLE I

Operation counts for point addition, doubling & tripling over prime field using Affine coordinates & Jacobian projective coordinates [8], [12], [15].

Curve operation	Affine	Jacobian
Addition	1*I* + 2*M* + 2*S*	12*M* + 4*S*
Doubling	1*I* + 2*M* + 1*S*	4*M* + 6*S*
Tripling	2*I* + 3*M* + 3*S*	10*M* + 6*S*

Key: I - inversion, M - multiplication, and S - squaring.

Elliptic curve addition, doubling and tripling are involved in these multiple point multiplication algorithms. When the point addition, doubling and tripling are calculated in the field *K*, it requires field inversions, multiplications and squarings. Refer Table II-A.2 and II-A.2 for the operational costs for curve additions, doublings and triplings in terms of feild operations.

It may be advantageous to represent points using projective coordinates, when the inversion in *K* is significantly more expensive than multiplication. For an example, one inversion is equal to 40 multiplications in prime fields. Further, in binary fields the inversion is equivalent to 3 to 10 multiplications. We eliminate inversions by using Jacobian projective coordinates. However, in our approaches we consider the both Affine

TABLE II

OPERATION COUNTS FOR POINT ADDITION, DOUBLING & TRIPLING OVER BINARY FIELD USING AFFINE COORDINATES & JACOBIAN PROJECTIVE COORDINATES [8], [12], [15].

Curve operation	Affine	Jacobian
Addition	1*I* + 2*M* + 1*S*	6*M* + 3*S*
Doubling	1*I* + 2*M* + 1*S*	5*M* + 5*S*
Tripling	2*I* + 6*M* + 3*S*	15*M* + 7*S*

Key: I - inversion, M - multiplication, and S - squaring.

coordinates and Jacobian projective coordinates. Table II-A.2 & II-A.2 gives the number of field operations involved in elliptic curve operations over prime and binary fields in Affine coordinates and Jacobian projective coordinates.

Jacobian coordinates are a special class of projective coordinates, where the point $(X : Y : Z)$ corresponds to the affine point $\left(X/Z^2 : Y/Z^3\right)$, when $Z \neq 0$. The point at infinity is represented as (1:1:0). The opposite of $(X : Y : Z)$ is $(X : -Y : Z)$. For further details on projective coordinates, reader can refer to [8], [12], [15].

3) Elliptic Curve Digital Signature Algorithms: -

Digital signatures serve authentication, data integrity, and non-repudiation in the digital space [3], [15], [20], [22]. A signature scheme consists of four segments: namely, domain parameter generation, key generation, signature generation and signature verification algorithms [20]. Two standardized signature schemes based on elliptic curve cryptography are Elliptic Curve Digital Signature Algorithm (ECDSA) and Elliptic Curve Korean Certificate-based Digital Signature Algorithm (EC-KCDSA) [15].

ECDSA is the most widely standardized elliptic curve-based signature scheme. ANSI X9.62, FIPS 186-2, IEEE 1363-2000 and ISO/IEC 15946-2 standards have included ECDSA. EC-KCDSA appears in the ISO/IEC 15946-2 standard. Both signature schemes need to perform multiple point multiplication in the signature verification process.

Domain parameters define the elliptic curve, E which is defined over a finite field, $\mathbb{F}_q$. These parameters should be chosen in a way that the elliptic curve discrete logarithm problem (ECDLP) is resistant to all known attacks [15]. In both signature schemes we have to use domain parameters, field order (q), field representation (FR), a seed if the elliptic curve is randomly generated (S), two coefficients of the equation ($a, b \in \mathbb{F}_q$), a finite point ($P = (x_p, y_p) \in \mathbb{F}_q$), order of finite point (n) and the cofactor ($h = \#E(\mathbb{F}_q)/n$). Further details on digital signatures are available in [5], [13], [20] and elliptic curve digital signature schemes can be found in [15].

B. Double-Base Number System

The double-base number system (DBNS) has been studied extensively, due to its applicability in signal processing and cryptography. We will begin DBNS background study with following definition from [4], [6], [9], [10], [11].

Definition 2 (s-integer): *An s-integer is a positive integer, whose largest prime factor does not exceed the s-th prime number.*

Definition 3 (double-based number system): *The double-based number system (DBNS) is a representation scheme in which every positive integer, n, is represented as the sum or difference of 2-integers that is, numbers of the form $2^a 3^b$.*

$$n = \sum_{i=1}^{M} s_i 2^{b_i} 3^{t_i} \tag{5}$$

where, $s_i \in \{-1, 1\}$, and $b_i, t_i \geq 0$.

The DBNS representation is highly sparse and redundant. If only the positive signs ($s_i = 1$) are considered for the DBNS representations; 100 has exactly 402 different DBNS representations and 1000 has exactly 1,295,579 different DBNS representations. The following theorems give an important result about DBNS[10].

Theorem 1: *Every positive integer, n, can be represented as the sum of at most $\mathcal{O}\left(\frac{\log n}{\log \log n}\right)$ 2-integers.*

Theorem 2: *There exists an absolute constant, C, such that there is always a number of the form $2^b 3^t$ between n and $n - \frac{n}{(\log n)^C}$.*

Algorithm 1: A greedy algorithm to convert integers to DBNS

Input : A positive integer n.
Output : the sequence of exponents (b_n, t_n) leading to one DBNS representation of n.

1. while $n > 0$ do.
2. find $z = 2^a 3^b$, the largest 2-integer less than or equal to n.
3. print (b, t).
4. $n \leftarrow n - z$.

Some of these representations have the minimal number 2-integer representation. An integer can be represented as the sum of m 2-integers, but cannot be represented as the sum of $(m-1)$ 2-integers. That is called canonic representation and very sparse. The Algorithm 1 gives a nearly canonical unsigned DBNS representation of a number (in this case, $s_i = 1$ only).

In the new point multiplication algorithm we need to calculate number of DBNS elements below some 2^n number. This helps us to calculate the required number of doublings and triplings involved in the algorithms. The total number of DBNS elements below 2^n (d_n) is given by the following expression:

$$d_n = \sum_{i=1}^{n} \lceil \log_3 2^i \rceil \tag{6}$$

The inductive method is used to prove Eqn. 6 in Proof 1. We have extended Eqn. 6 to calculate number of doublings and triplings involved in calculating all DBNS elements below 2^n.

Proof 1: Number of DBNS Elements less than 2^n

Consider the expression for $n = 1$

$$\begin{aligned} d_1 &= \sum_{i=1}^{1} \lceil \log_3 2^i \rceil \\ &= \lceil \log_3 2^1 \rceil \\ &= 1 \end{aligned}$$

d_n is the number of elements less than 2^1. The only DBNS element less than 2^1 is 1.
Assuming the Eqn. 6 is true for $n-1$, then;

$$d_{n-1} = \sum_{k=1}^{n-1} \lceil \log_3 2^k \rceil$$

Find the number of DBNS elements in $\left[2^{n-1}, 2^n\right)$:
First element = 2^{n-1}
Second element = $2^{n-2} \cdot 3$
Third element = $2^{n-4} \cdot 3^2$
...
Last element = $2^x \cdot 3^{\lfloor \log_3 2^n \rfloor}$, In this case $0 \leq x < n$.
According to the above calculation we have $\lceil \log_3 2^n \rceil$ number of elements in $\left[2^{n-1}, 2^n\right)$.
Therefore,

$$\begin{aligned} d_n &= d_{n-1} + \lceil \log_3 2^n \rceil \\ &= \sum_{k=1}^{n} \lceil \log_3 2^k \rceil \end{aligned}$$

We use tripling algorithm to calculate all 3^i ($i \in [1, \lfloor \log_3 2^n \rfloor]$) DBNS elements less than 2^n. To calculate other DBNS elements we use doublings. We can calculate number of doublings (DBL_n) and triplings (TPL_n) that we need to calculate all DBNS elements less than 2^n in the following manner:

$$\begin{aligned} TPL_n &= \lfloor \log_3 2^n \rfloor \\ DBL_n &= d_n - TPL_n - 1 \\ &= \sum_{k=1}^{n} \lceil \log_3 2^k \rceil - \lfloor \log_3 2^n \rfloor - 1 \\ &= \sum_{k=1}^{n} \lceil \log_3 2^k \rceil - \lceil \log_3 2^n \rceil \\ &= \sum_{k=1}^{n-1} \lceil \log_3 2^k \rceil \end{aligned} \tag{7}$$

We have minimized calculation of tripling operations in DBNS element calculations, because tripling operation utilizes more cost than doubling operation.

III. Multiple Point Multiplication Algorithms

In elliptic curve digital signature algorithms, we consider the resultant, $kP + lQ$ for multiple point multiplication. Where k and l are t-bit numbers, both P and Q are two points on the elliptic curve. There are number of multiple point multiplication algorithms have been proposed, namely simultaneous multiple point multiplication based on Shamir's trick, sliding window and non-adjacent form (NAF), joint sparse form (JSF) and interleaving with NAF [14], [15], [24].

A. Simultaneous Multiple Point Multiplication (SMPM)

In this method, we represent k and l in w-bit, $d = \lceil t/w \rceil$ number of blocks and do the pre-computation for $iP + jQ$ for $0 \leq i, j < 2^w$. Then, k and l are represented $K^{d-1} \| .... \| K^1 \| K^0$ and $L^{d-1} \| .... \| L^1 \| L^0$ respectively. Calculating $R \leftarrow 2^w R + (K^i P + L^i Q)$ for $d-1$ times, the final answer can be obtained. See Algorithm 2.

Algorithm 2: Simultaneous Multiple Point Multiplication

Input : Window width $w, k, l, P, Q \in E(\mathbb{F}_q)$
Output : $kP + lQ$

1. compute $iP + jQ$ for all $i, j \in [0, 2^w - 1]$.
2. write $k = K^{d-1} \| .... \| K^1 \| K^0$ and $l = L^{d-1} \| .... \| L^1 \| L^0$
   where all K^i, L^i are w-bit long and $d = \lceil t/w \rceil$.
3. $R \leftarrow \infty$.
4. for $i$from $d-1$ downto 0 do
   4.1 $R \leftarrow 2^w R$.
   4.2 $R \leftarrow R + (K^i P + L^i Q)$.
5. return R.

Sliding window method can be used to improve this algorithm [15]. In Table VI & VII, cost of SMPM and that of sliding SMPM are compared. Further note that storage required for the sliding SMPM is less than storage required for SMPM. $9 + 15t/32$ and $2 + t$ give the number of additions and doublings involved in the elliptic curve additions of two *t*-bit numbers, under the Simultaneous Multiple Point Multiplication.

B. Simultaneous Multiple Point Multiplication with NAF (NAF-SMPM) and Joint Sparse Form (JSF)

For fixed-window, we have $3t/4$ point additions on average for $kP + lQ$. When the non-adjacent form (NAF) representation of both k and l are considered the average point addition comes down to $5t/9$. The joint sparse form (JSF) of two integers introduced by Solinas in [24] reduces the average point addition to $t/2$. Algorithm 3 explains the JSF and it needs only 4 storage positions for the calculation.

P, Q, $P + Q$ and $P - Q$ is pre-computed and stored in both algorithms. Advantage of JSF over NAF method is that the bit distribution for both numbers in JSF is determined, after considering the relationship between two numbers.

Algorithm 3: Joint Sparse Form

Input : Positive integers k^1 and k^2
Output : $JSF(k^1, k^2)$

1. $l \leftarrow 0$, $d_1 \leftarrow 0$, $d_2 \leftarrow 0$.
2. while $(k^1 + d_1 > 0$ or $k^2 + d_2 > 0)$
   2.1 $l_1 \leftarrow d_1 + k^1, l_2 \leftarrow d_2 + k^2$.
   2.2 for i from 1 to 2 do
      if l_i is even then $u \leftarrow 0$;
      else
         $u \leftarrow l_i$ mods 4.
         if $l_i \equiv \pm 3 \ (mod\ 8)$ and $l_{3-i} \equiv 2 \ (mod\ 4)$
         then $u \leftarrow -u$.
      $k_l^i \leftarrow u$.
   2.3 for i from 1 to 2 do
      if $2d_i = 1 + k_l^i$ then $d_i \leftarrow 1 - d_i$.
      $k^i \leftarrow \lfloor k^i/2 \rfloor$.
   2.4 $l \leftarrow l + 1$.
3. return $JSF(k^1, k^2)$.

Under the joint sparse form, the number of additions and doublings involved in an elliptic curve addition of two *t*-width numbers are $2 + t/2$ and t, respectively.

C. Interleaving with w-NAF (I-w-NAF)

The interleaving with w-NAF method needs pre-computation of iP_j for $i \in \{1, 3, ..., 2^{w_j - 1} - 1\}$ and $1 \leq j \leq 2$. Therefore, required storage positions are $2^{w_j - 2}$ for the point P_j. The other benefit of this method is that we can use different windows, w_j for different points, P_j. Algorithm 4 describes the steps for interleaving with NAF for v number of points.

Algorithm 4: Interleaving with w-NAF

Input : v, Positive integers k^j, widths w_j and points P_j, $1 \leq j \leq v$
Output : $\sum_{j=1}^{v} k^j P_j$

1. compute iP_j for $i \in \{1, 3, ..., 2^{w_j - 1} - 1\}$, $1 \leq j \leq v$.
2. calculate $NAF_{w_j}(k^j)$, $1 \leq j \leq v$
3. let $l = max\{l_j : 1 \leq j \leq v\}$.
4. define $k_i^j = 0$ for $l_j \leq i \leq l$, $1 \leq j \leq v$
5. $Q \leftarrow \infty$.
6. for i from $l - 1$ downto 0 do
   6.1 $Q \leftarrow 2Q$
   6.2 for j from 1 to v do
      if $k_i^j \neq 0$ then
         if $k_i^j > 0$ then $Q \leftarrow Q + k_i^j P_j$;
         else $Q \leftarrow Q - k_i^j P_j$
7. return Q.

When we add two t-bit numbers with interleaving with w-NAF algorithm, there are $3 + 11t/30$ additions and $1 + t$ doublings. This algorithms can easily be extended for any other number of point multiplication.

IV. DBNS based Algorithms for Multiple Point Multiplication

We have proposed three algorithms based on DBNS, namely, sliding DBNS, sliding DBNS with simultaneous point multiplications and interleaving with signed DBNS. In all three approaches, we consider the point, P is known in advance.

A. Sliding DBNS Algorithm

In sliding DBNS algorithm, we use w-bit blocks to represent k and l numbers as in Eqn. 8 and Eqn. 9. In that case, we have maximum of $d = \lceil \frac{t}{w} \rceil$ number of blocks in a given pair of t-bit numbers. This algorithm is shown in the Fig. 2 and it represents $k = k^1$, $l = k^2$, $P = P_1$ and $Q = P_2$. Note that BIN(k) gives the binary representation of k, in Fig 2.

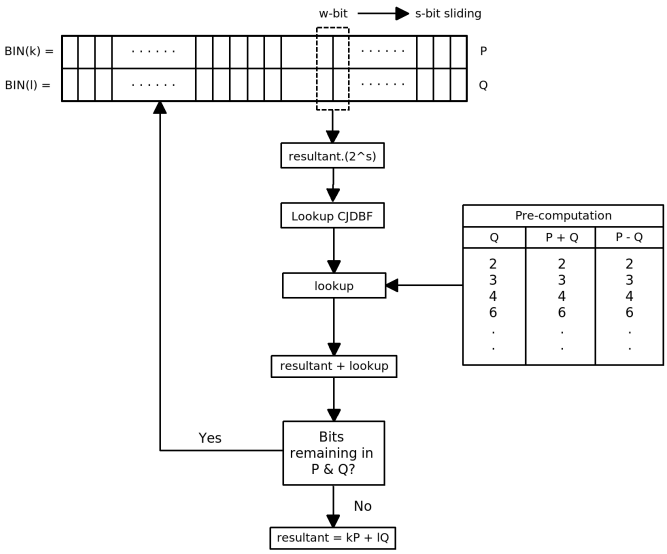

Fig. 2. Sliding DBNS Algorithm

$$\begin{aligned} k &= K^{d-1} \| .... \| K^1 \| K^0 \\ l &= L^{d-1} \| .... \| L^1 \| L^0 \end{aligned} \tag{8}$$

where,

$$\begin{aligned} K^i &= k_{w-1}^i, ..., k_1^i, k_0^i \\ L^i &= l_{w-1}^i, ..., l_1^i, l_0^i \end{aligned} \tag{9}$$

We have pre-computed all the DBNS elements less than 2^w for Q, $P + Q$ and $P - Q$. Then the canonical joint double-base format (CJDBF) for a pair of numbers is considered. This enables to get a $P + Q$ or $P - Q$ representation into the point multiplication. Our approach has a sliding window mechanism to improve the performance further. Sliding is considered from left-to-right. Once the adding is done for the i-th block, we take the first non-zero column as the most significant of the $(i-1)$-st block. Algorithm 5 illustrates the sliding DBNS, step-by-step and Table VI & VII gives the cost of point additions under this algorithm.

Algorithm 5: Sliding DBNS

Input : t-bit positive integers, k_j, width, w, and points P_j, $j = 2$
Output : $\sum_{j=1}^{2} k^j P_j$

1. compute iP_2, $i(P_1 + P_2)$, $i(P_1 - P_2)$ for
   $i \in \{1, 2, 3, ..., 2^{a_x} 3^{b_x}\}$,
   $0 \leq a_x < w$, $0 \leq b_x < \lfloor log_3 2^w \rfloor$, $2^{a_x} 3^{b_x} < 2^w$.
2. $Q \leftarrow \infty$
3. $p \leftarrow 0$
4. $s \leftarrow 0$
5. for p from $d-1$ downto 0 do
   5.1 lookup CJDBF of (K_1^p) and (K_2^p)
   for r from 0 to y do
   $lookup = d_q^r P_q$ or $d_q^r(P_1 + P_2)$ or $d_q^r(P_1 - P_2)$
   $Q \leftarrow Q + lookup$
   5.2 $p \leftarrow p + 1$
   5.3 find the next non-zero column on the right & shift s bits to right
   5.4 take leftmost bit as the MSB of K_q^p
   5.5 $Q \leftarrow 2^s \cdot Q$
   5.6 if no columns then;
   break
6. return Q.

Table III gives the average number of additions involved in the multiple point multiplication for $w-$bit width window ($w \in [4, 8]$). In this calculation we assume that we have pre-computed all the DBNS elements less than 2^w for Q, $P + Q$ and $P - Q$. The Eqn. 10 gives the average number

TABLE III
AVERAGE NUMBER OF ADDITIONS INVOLVED IN THE MULTIPLE POINT MULTIPLICATION FOR $w-$BIT WIDTH WINDOW.

window width	average additions
8	2.26
7	1.99
6	1.77
5	1.52
4	1.22

of elliptic curve additions (ADD_w) needed for the multiple point multiplication using the above algorithm with $w-$bit window. The average number of elliptic curve additions for $w-$bit window (a_w) can be taken from the Table III. Further the numbers of elliptic curve doublings (DBL_w) and triplings (TPL_w) can be obtained by equation 7. Because we have to pre-calculate Q, $P+Q$ and $P-Q$ our elliptic curve doublings and triplings as follows:

$$\begin{aligned} ADD_w &= (1 + a_w) \cdot \left(1 - \frac{1}{4w}\right) \cdot \frac{t}{w} \\ DBL_w &= 3 \times \sum_{k=1}^{w-1} \lceil \log_3 2^k \rceil \\ TPL_w &= 3 \times \lfloor log_3 2^w \rfloor \end{aligned} \tag{10}$$

Cost involved in pre-computation of P terms is excluded, assuming that P is known in advance.

B. Sliding DBNS Simultaneous Multiple Point Multiplication Algorithm (Sliding DSMPM)

In sliding DBNS simultaneous multiple point multiplication algorithm, the t-bit number is broken into d number of w-bit numbers and with the sliding mechanism, it reduces the number of w-bit numbers less than d.

Algorithm 6: Sliding DBNS Simultaneous Multiple Point Multiplication

Input : t-bit positive integers, k_j, width, w, and points P_j, $j = 2$
Output : $\sum_{j=1}^{2} k^j P_j$

1. compute iP_2 for $i \in \{1, 2, 3, ..., 2^{a_x} 3^{b_x}\}$,
   $0 \leq a_x < w$, $0 \leq b_x < \lfloor log_3 2^w \rfloor$, $2^{a_x} 3^{b_x} < 2^w$.
2. $Q \leftarrow \infty$
3. $p \leftarrow 0$
4. $s \leftarrow 0$
5. for p from $d-1$ downto 0 do
   5.1 for q from 1 to 2 do
   lookup $DBNS(K_q^p) = d_q^0 + \cdots + d_q^y$
   for r from 0 to y do
   lookup $= d_q^r P_q$
   $Q \leftarrow Q + d_q^r P_q$
   5.2 $p \leftarrow p + 1$
   5.3 find the next non-zero column on the right & shift s bits to right
   5.4 take leftmost bit as the MSB of K_q^p
   5.5 $Q \leftarrow 2^s \cdot Q$
   5.6 if no columns then;
   break
6. return Q.

In previous algorithm we had pre-computed all DBNS representation point multiplications of P_1, $P_1 - P_2$ and $P_1 + P_2$. However in this algorithm, only the DBNS element point multiplications of P_2 is pre-computed, to save memory.

TABLE IV
AVERAGE NUMBER OF ADDITIONS INVOLVED IN ALL NUMBERS IN $w-$BIT WIDTH WINDOW.

window width	average additions
10	1.32
9	1.16
8	0.98
7	0.86
6	0.75
5	0.61
4	0.47

Further, the representation of t-bit number is converted to the canonical DBNS representation which reduces the number of additions in each step. The step-by-step description of the

sliding DBNS simultaneous multiple point multiplication is described in Algorithm 6.

The main advantage of this algorithm is that we need less memory compared to the previous algorithm, because we calculate only all the DBNS representations of point P_2. Fig. 3 illustrates the sliding DBNS simultaneous multiple point multiplication.

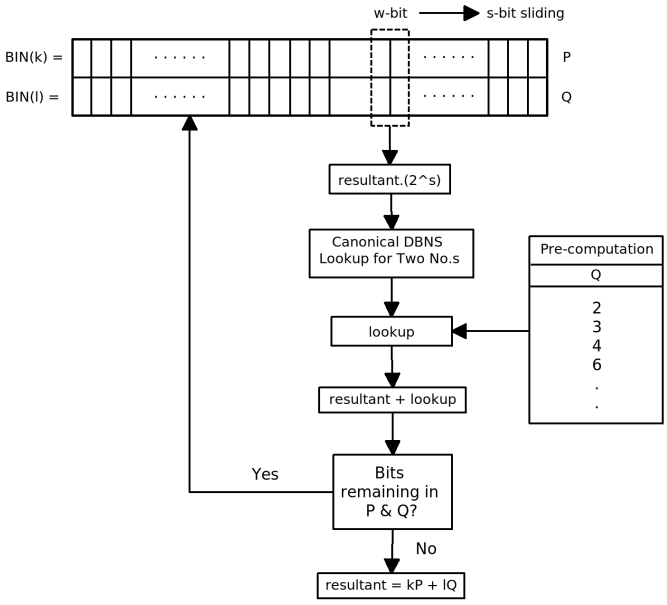

Fig. 3. Sliding DBNS Simultaneous Multiple Point Multiplication Algorithm

The Table IV gives the average number of additions that needs for the DBNS representations of all the numbers below 2^w ($w-$width window). Eqn. 11 gives the number of elliptic curve additions, doublings and triplings. Comparing with sliding DBNS, we have one more elliptic curve addition introduced to a window. Refer Eqn. 11.

$$\begin{aligned} ADD_w &= (2 + a_w) \cdot \left(1 - \frac{1}{4w}\right) \cdot \frac{t}{w} \\ DBL_w &= \sum_{k=1}^{w-1} \lceil \log_3 2^k \rceil \\ TPL_w &= \lfloor log_3 2^w \rfloor \end{aligned} \tag{11}$$

However, number of doublings and triplings has gone down in this algorithm by a factor of $2/3$. Further, number of storage also has gone down by the same factor.

C. Interleaving with DBNS (I-DBNS)

The third algorithm is based on non-adjacent form (NAF)[17] representation of numbers. When the w-NAF representation is used we need only to calculate odd numbers below 2^{w-1}. For an example, if we consider 5-NAF representation of a number, there will be odd co-efficients from -15 to 15. The w-NAF representation of a number reduces the non-zero elements in the representation. Therefore the sliding mechanism of the DBNS representation of w-NAF has less number of elliptic curve additions. Algorithm 7 demonstrates the Interleaving with double-base number system.

Algorithm 7: Interleaving with DBNS

Input : w-bit positive integers, k_j, width, w, and points P_j, $j \in \{1, 2\}$
Output : $\sum_{j=1}^{2} k^j P_j$

1. compute iP_j for $i \in \{1, 2, 3, ..., 2^{a_x}3^{b_x}\}$, $j \in \{1, 2\}$, $0 \leq a_x < w$, $0 \leq b_x < \lfloor log_3 2^w \rfloor$, $2^{a_x}3^{b_x} < 2^w$.
2. $Q \leftarrow \infty$
3. $p \leftarrow 0$
4. $s \leftarrow 0$
5. for p from $d-1$ downto 0 do
   5.1 for q from 1 to 2 do
       lookup $DBNS(K_q^p) = d_q^0 + \cdots + d_q^y$
       for r from 0 to y do
           lookup $= d_q^r P_q$
           $Q \leftarrow Q + d_q^r P_q$
   5.2 $p \leftarrow p + 1$
   5.3 find the next non-zero column on the right & shift s bits to right
   5.4 take leftmost bit as the MSB of K_q^p
   5.5 $Q \leftarrow 2^s \cdot Q$
   5.6 if no columns then;
       break
6. return Q.

Fig. 4 shows the Interleaving with DBNS algorithm for a $w-$bit sliding window. Note that we have the NAF representation of the two numbers before coming into sliding window mechanism. In Fig. 4, NAF(k) means that the non-adjacent form representation of k.

Table V gives the average number of additions that needs for the DBNS representations of all the odd numbers below 2^w ($w-$width window). We have considered only the odd numbers because in the $w-$NAF representation we need to have only the odd numbers below $(w-1)-$width.

TABLE V
AVERAGE NUMBER OF ADDITIONS INVOLVED IN ALL ODD NUMBERS IN $w-$BIT WIDTH WINDOW.

window width	average additions
11	1.5
10	1.36
9	1.13
8	1.03
7	0.91
6	0.75
5	0.63

Eqn. 12 gives the number of elliptic curve additions, doublings and triplings. Note that we have to capture only non-

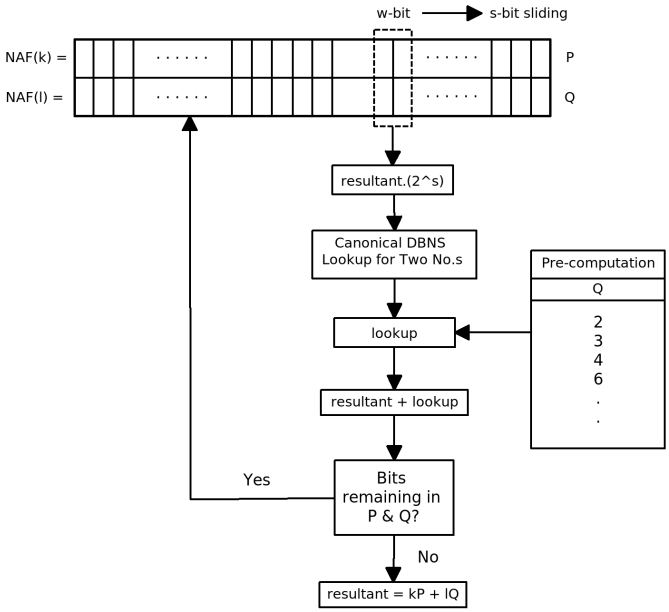

Fig. 4. Interleaving with DBNS

zero elements in each number.

$$\begin{aligned} ADD_w &= (2 + a_w) \cdot \frac{t}{(w+1)} \\ DBL_w &= \sum_{k=1}^{w-1} \lceil \log_3 2^k \rceil \\ TPL_w &= \lfloor log_3 2^w \rfloor \end{aligned} \tag{12}$$

When the additions are calculated, we assume that we have $t/(w+1)$ number of elements in each number's NAF representation. Calculating number of Doublings and Triplings are same as Sliding DSMPM algorithm.

V. Comparison

In this section we compare sliding simultaneous multiple point multiplication, joint sparse form, interleaving with w-NAF with our new algorithms which are based on sliding DBNS, sliding DBNS simultaneous multiple point multiplication and Interleaving with DBNS, respectively. For the comparison, we consider 2-bit window width for the simultaneous multiple point multiplication and sliding simultaneous multiple point multiplication, while 5-bit and 4-bit window is used for Interleaving with NAF. We have used 4-bit to 8-bit window sizes for the sliding DBNS and 5-bit to 10-bit window width for sliding DBNS simultaneous multiple point multiplication. Finally, 5-bit to 12-bit window sizes are used for the Interleaving with DBNS algorithm.

Table VI gives the experimental results of operational counts for computing $kP+lQ$ over prime fields. When the operation cost is calculated, both affine and Jacobian coordinates are considered. It is assumed that field cost per a squaring over a prime field is equal to 0.8 times of a multiplication. Comparison has been carried out for 160-bit, 230-bit and 570-bit numbers. We have assumed one point is known as a domain parameter and point multiplications related to that point are pre-calculated. Similarly, Table VII gives the experimental results of operational counts for computing $kP + lQ$ over binary fields. Again this table gives the operational cost for the both affine and Jacobian coordinates. For the final calculation, we have assume that squaring is almost free in binary fields.

TABLE VI
Experimental Results of Operational Counts for Computing $kP + lQ$ over Prime fields

Algorithm	bit length	operational cost	
		Affine	Jacobian
sliding SMPM	160	10326	2608
	230	14634	3682
	570	35560	8896
JSF	160	10423	2654
	230	14945	3802
	570	36909	9378
I-*w*-NAF	160	9603	2362
	230	13728	3372
	570	33765	8276
sliding DBNS	160	11465	2860
	230	16050	3965
	570	37420	9126
sliding DSMPM	160	10323	2470
	230	14464	3446
	570	34077	8112
I-DBNS	160	9856	2349
	230	13904	3297
	570	33099	7761

In Table VI & VII, the window size and the storage required are different in the new alorithms for different bit lengths.

A. Curves defined over $\mathbb{F}_p$ with Affine Coordinates

According to the Table VI, it is noticable that sliding DBNS does not give us any improvement compared to any other algorithms in consideration. However we can notice that sliding DBNS window width is increasing with the bit length of the two numbers.

Sliding DBNS simple multiple point multiplication has an improvement with respect to sliding SMPM and JSF. The improvement is significant when the bit length is increasing. According to our calculations the improvement for 570-bit is 8% against JSF.

Interleaving DBNS with w-NAF algorithm is giving the most impressive improvements. The most existing algorithm for multiple point multiplication is interleaving with w-NAF. The new algorithm based on w-NAF and DBNS is 2% more efficienct than the interleaving with w-NAF. When it is compared with JSF, it is a 10% improvement.

B. Curves defined over $\mathbb{F}_p$ with Jacobian Coordinates

The most significant result from the Table VI and VII is that the affine coordinates need more computational power. That

is due to the inversions in the calculations using affine coordinates. When it comes to Jacobian coordinates, we eliminate inversions and it saves a lot of computational power.

In prime fields Jacobian coordinates give a some improvement with sliding DBNS over JSF. This advantage is visible when the bit length is high. For an example, 570-bit length is having a clear improvment with the new algorithm.

Considering the sliding DBNS simultaneous multip point multiplication and JSF, we can notice that 7%, 9% and 14% improvements for 160-bit, 230-bit and 570-bit numbers respectively. However, there is no notable improvements, compared to the interleaving with w-NAF.

TABLE VII
EXPERIMENTAL RESULTS OF OPERATIONAL COUNTS FOR COMPUTING $kP + lQ$ OVER BINARY FIELDS

Algorithm	bit length	operational cost	
		Affine	Jacobian
sliding SMPM	160	10072	2055
	230	14276	2886
	570	34696	6926
JSF	160	10164	2112
	230	14574	3022
	570	35994	7442
I-*w*-NAF	160	9374	1800
	230	13402	2565
	570	32966	6277
sliding DBNS	160	10958	2216
	230	15429	3052
	570	36082	6952
sliding DSMPM	160	9930	1834
	230	13944	2546
	570	32931	6007
I-DBNS	160	9502	1736
	230	13418	2419
	570	32123	5654

Interleaving DBNS with w-NAF has 6% and 17% improvement compared to interleaving with w-NAF and JSF for 570-bit numbers. Table VI gives the costs for 160-bit and 230-bit numbers using the same algorithm.

C. Curves defined over $\mathbb{F}_{2^m}$ with Affine Coordinates

In the binary fields we have the advantage in squaring because that can be considered as a no cost operation. Further, inversion costs less than ten multiplications in binary fields. This reduces overall cost in binary fields with affine coordinates.

Table VII gives the summary of the operational cost for all the algorithms. In comparison, we can notice that sliding DBNS has 9% improvement against JSF. While interleaving DBNS with w-NAF has gained 3% improvement against interleaving with w-NAF. The improvement is 11% when we compare the same algorithm with JSF. Further all of above comparisons are valid for 570-bit numbers.

D. Curves defined over $\mathbb{F}_{2^m}$ with Jacobian Coordinates

The improvement due to new algorithms is noticable when they are operated in the binary field with Jacobian coordinates. In this combination, we have less cost for the squaring and field inverse. That leads to a significant improvement in these algorithms.

Considering 160-bit numbers, sliding DBNS multiple point multiplication gives 13% enhancement over JSF. This number goes upto 16% and 19% with the same algorithm when the bit length is increased to 230-bit and 570-bit numbers.

Interleaving DBNS with w-NAF has 4%, 6% and 10% improvements over Interleaving with w-NAF with 160-bit, 230-bit and 570-bit numbers, respectively. When the same algorithm is compared with JSF, it gives upto 24% improvement with 570-bit numbers. For other two bit lengths, i.e. 160-bit and 230-bit, improvement is 18% and 20% over JSF.

VI. CONCLUSIONS

In all three new algorithms, the required memory is larger than the other algorithms due to pre-computations. However the performance has been improved, specially, in the Interleaving with DBNS algorithm upto 24%. In the conclusion, we have draw the attention to the most popular multiple point multiplication algorithms, JSF and interleaving with w-NAF.

Sliding DBNS algorithm needs to have more memory and it is not giving greater performance improvements in 160-bit numbers. However, when the 230-bit and 570-bit numbers are considered it gives much improved performances. It never beats JSF and Interleaving with NAF for 4 and 5-bit widths. 4-bit width window is giving the optimum performance when the 160-bit numbers are considered. Whereas, 230-bit numbers and 570-bit numbers will have optimum performances with the sliding DBNS algorithm, when the window sizes are 6-bit and 7-bit, respectively.

Sliding DBNS simultaneous multiple point multiplication gives better performances against JSF, with Jacobian coordinates over binary fields. It has 13%, 16% and 19% improvement in 160-bit, 230-bit and 570-bit numbers, respectively and window sizes are 10-bit for all cases.

Finally, we have the most successful algorithm which is Interleaving with DBNS is giving 18%, 20% and 24% improvements with the JSF and 4%, 6% and 10% improvements with Interleaving NAF for 160-bit, 230-bit and 570-bit numbers respectively in each case, with Jacobian coordinates over binary fields. According to the comparison, we can conclude that Interleaving DBNS with w-NAF algorithm gives the optimal performance with Jacobian coordinates over binary fields.

For future work, research will be carried out to improve the calculation of multiple point multiplication of $(kP + lQ + mR)$ on elliptic curves using double-base number system. Further, we will consider the triple-base number system for multiple point multiplication. In addition to these Koblitz curves will be considered with the double-base representation in τ and $\tau - 1$. This idea will be extended to the hardware implementation.

REFERENCES

[1] FIPS 186. Digital signature standard. *National Institute of Standards and Technology*, 1994.

[2] FIPS 186-2. Digital signature standard. *National Institute of Standards and Technology*, 2000.

[3] H. Cohen, G. Frey, R. Avanzi, C. Doche, T. Lange, K. Nguyen, and F. Vercauteren. *Handbook of Elliptic and Hyperelliptic Curve Cryptography*. Chapman & Hall/CRC, July 2005.

[4] B. M. M. de Weger. Algorithms for diophantine equations. volume 65, Amsterdam, 1989. Centrum voor Wiskunde en Informatica.

[5] W. Diffie and M. Hellman. New directions in cryptography. *Information Theory, IEEE Transactions on*, 22:644–654, 1976.

[6] V. S. Dimitrov and T. Cooklev. Two algorithms for modular exponentiation using nonstandard arithmetics. *Technical report of IEICE. ISEC*, 93(525):11–17, 1994.

[7] V. S. Dimitrov, L. Imbert, and P. Mishra. Fast elliptic curve point multiplication using double-base chains. 2005.

[8] V. S. Dimitrov, L. Imbert, and P. Mishra. The double base number system and its application to elliptic curve cryptography. 77(262):10751104, April 2008.

[9] V. S. Dimitrov, G. A. Jullien, and W. C. Miller. Theory and applications for a double-base number system. In *ARITH '97: Proceedings of the 13th Symposium on Computer Arithmetic*, page 44, Washington, DC, USA, 1997. IEEE Computer Society.

[10] V. S. Dimitrov, G. A. Jullien, and W. C. Miller. An algorithm for modular exponentiation. 66:155–159, 1998.

[11] V.S. Dimitrov and G.A. Jullien. Loading the bases: a new number representation with applications. *Circuits and Systems Magazine, IEEE*, 3(2):6–23, 2003.

[12] Christophe Doche, Thomas Icart, and David R. Kohel. Efficient scalar multiplication by isogeny decompositions. 2005.

[13] T. Elgamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *Information Theory, IEEE Transactions on*, 31:469–472, 1985.

[14] D. M. Gordon. A survey of fast exponentiation methods. *J. Algorithms*, 27(1):129–146, 1998.

[15] D. Hankerson, A. J. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer, 1 edition, 2004.

[16] ISO/IEC. Information technology - security techniques - cryptographic techniques based on elliptic curves - part 2: Digital signatures. *International Organization for Standardization*, 2002.

[17] D. E. Knuth. *THE ART OF COMPUTER PROGRAMMING VOLUME 2 SEMINUMERICAL ALGORITHMS*. Addsion Weslsey Longman Publishing Group, May 1969.

[18] N. Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48:203–209, 1987.

[19] N. Koblitz. *CM-Curves with Good Cryptographic Properties*, page 279. 1992.

[20] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC, October 1996.

[21] V. S. Miller. *Use of Elliptic Curves in Cryptography*, page 417. 1986.

[22] C. P. Pfleeger and S. L. Pfleeger. *Security in Computing*. Prentice Hall PTR, 3 edition, December 2002.

[23] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21:120–126, 1978.

[24] J. Solinas. Low-weight binary representations for pairs of integers, 2001.

[25] J.H. van Lint. *Introduction to Coding Theory*. Springer, 3rd rev. and exp. ed. edition, December 1998.