Efficient Window-Based Elliptic curve Scalar Multiplication using Double base Number Representation

Ravi Pankaj

mtc0520

Professor Rana Barua

Indian Statistical Institute,
Kolkata, 700 108

CERTIFICATE

This is to certify that the thesis entitled "***Efficient Window-Based Elliptic curve scalar Multiplication using Double base Representation***" is submitted in partial fulfillment of the requirement for the award of the degree of the Masters in Technology in ***Computer science*** at Indian Statistical Institute, Kolkata.

It is a faithful record of bona fide research work carried out by Mr. ***Ravi Pankaj (mtc0521)*** under my supervision and guidance. It is further certified that no part of thesis has been submitted to any other University or Institute for the award of any Degree or Diploma.

(Professor Rana Barua)
Supervisor

Countersigned
External Examiner

Date: of July, 2007.

Acknowledgements

With great pleasure and sense of obligation I express my heartfelt gratitude to my guide and supervisor **Prof. Rana Barua** of Stat-Math unit, Indian Statistical Institute, Kolkata. I am highly indebted to him for his invaluable guidance and ever ready support. His persisting encouragement, perpetual motivation, everlasting patience and excellent expertise in discussions, during progress of Project Work, have benefited to an extent, which is beyond expression.

The chain of my gratitude would be definitely incomplete without expressing my gratitude to all my batch mates, for their support and encouragement throughout the entire M.Tech course. Lastly I sincerely thank all my friends and well wishers who helped me directly or indirectly towards the completion of this work.

Ravi Pankaj,
mtc0521,
Indian Statistical Institute,
Kolkata 700 108.

M.Tech. (Computer Science) Dissertation Series

Efficient Window-Based Elliptic curve Scalar Multiplication using Double base Number Representation

a dissertation submitted in partial fulfillment of the requirements for the M. Tech. (Computer Science) degree of the Indian Statistical Institute

By

Ravi Pankaj

mtc0520

under the supervision of

Professor Rana Barua

INDIAN STATISTICAL INSTITUTE

203, Barrackpore Trunk Road

Kolkata- 700 108

Abstract

Exponentiation is the most basic operation in the implementation of all discrete logarithm based cryptosystems. It translates to scalar multiplication, where the underlying group is additive in nature. Window based methods are very efficient methods to compute exponentiation. Double-base number system (DBNS) is a non standard number representation scheme with many interesting and useful properties. In this work we have proposed method for scalar multiplication where scalar (integer) is represented in DBNS format. We have adopted new way to represent the scalar in DBNS format. For that we have new algorithm to have representation in DBNS. This algorithm is faster than the previous conversion algorithms. The proposed method is more efficient than its single base counter part as well as straight double base part.

Synopsis of dissertation titled "*Efficient window-based elliptic curve Scalar Multiplication using Double base number representation"*

Submitted by- Ravi Pankaj (MTC0520)

Under Supervision of Prof. Rana Barua

1 Introduction

Elliptic curve cryptography *(ECC)* has occupied the centre stage of public key cryptography research because of its relatively small key-length and enhanced theoretical robustness. The reason behind these is the fact that there is no known algorithm to solve elliptic curve discrete logarithm problem (*ECDLP)* even in sub exponential time. Hence it is believed that a 160-bit *ECC* key can provide the same level of security as an 80-bit symmetric – key for block ciphers or a 1024-bit RSA modulus. This is a major device with limited hardware resources as smart cards, cell phones or PDAs. The efficiency of an *ECC* implementation depends largely on the *scalar multiplication* computed. It is the computation of the point $mP = \sum_{i=1}^{m} P$, for a given point *P* on the curve and an integer *m*. Enormous efforts have been devoted to accelerate and secure this operation.

Among the various methods proposed for efficient and secure implementation of exponentiation, the window-based methods occupy a special place. In the present work we propose a new window-based scalar multiplication algorithm where the scalar is represented in *double-base number system (DBNS).* DBNS is a number representation scheme, which uses 2 radii to represent integers. In this work we use 2 and 3 as radii. In this work we show that the terms of DBNS representation can be grouped together in small windows to reduce the number of addition further.

We use 2 and 3 as the base of the DBNS representation. That is because, in ECC, we have efficient formulas to compute doubling and tripling of elliptic curve points. Thus the exponent is represented as a sum of terms of the form $\pm 2^b 3^t$. In this work we have proposed algorithms to convert an integer in to a suitable DBNS representation. Though we use greedy algorithm, but it will be used for those integer which falls in a window. That will decrease search time up to great extent. We store some pre computed points

which can be used while calculation of *[n] P*. Finally for our representation of *n* we have proposed algorithm to compute *[n] P*.

2 Methodology

Our goal is that for given an integer *k* and an elliptic curve point *P,* we have to compute *kP*. Efficiency of the scalar multiplication depends largely upon efficiency of the algorithms used for group arithmetic and representation of the scalar. we have proposed an efficient and secure scalar multiplication algorithm based on double-base chains. We use new window method to find double-base representation.

The whole task is divided in two parts: i) finding a representation of *n*, using windows of *2* and *3* and ii) computing scalar multiplication using the obtained representation.

2.1 Finding representation of *n*

Let max_2, max_3 be the maximum powers of *2* and *3* to occur in the *DBNS* representation. The selection of bounds of max_2, max_3 and the dimension of windows effects the *DBNS* representation greatly, so appropriate selection of max_2, max_3 and window size is needed. Thus the task is divided into two parts: i) finding appropriate maximum bounds and ii) finding appropriate window dimension.

Finding appropriate maximum bounds

We can find the suitable bounds of binary and ternary exponents. The method is heuristic, but the bounds are significantly much less and work better.

Window selection

After getting the maximum bounds for binary and ternary exponents, the window is selected in such a way that each window should be of same length. For that we break entire range (bounds) in ρ parts. Each part will be called a window. Let w_2, *and* w_3 are the window length of base 2 and 3.

As we get the maximum bounds of exponents of binary and ternary radii, and valid window size, we need to represent *n* in double- base number system. We have proposed a new way to achieve this task. For correctness of our representation we proved 3 propositions.

We represent n in (DBNS) according to our format is :

$$n = (2^{w_2}3^{w_3})^{\rho-1} M_{\rho-1} + (2^{w_2}3^{w_3})^{\rho-2} M_{\rho-2} + \ldots\ldots + M_0$$

where $0 \le M_{\rho-1}, M_{\rho-2}, \ldots, M_0 < 2^{w_2}3^{w_3}$. Here all M_i 's fall in a particular window.

We now use greedy algorithm to find the DBNS representation of M_i 's.

2.2 Calculation of complexities

We will calculate the complexity of average number of inverse, square and multiplication applied in calculating scalar multiplication. Suppose on average there are *t* number of terms needed to represent M_i 's in a given window. Writing *n* according to Horner's rule.

$$n = 2^{w_2}3^{w_2}(\ldots(2^{w_2}3^{w_3}(M_{\rho-1}) + M_{\rho-2}) + \ldots) + M_0$$

As we can see that for calculating *nP,* there will $(\rho-1)w_2$ doublings, $(\rho-1)w_3$ triplings and $\rho t - 1$ addition needed. Let there are d_i, t_i, ad_i no. of inverses, d_m, t_m, ad_m no. of multiplications and d_s, t_s, ad_s no. of square needed in doubling, tripling and addition respectively. Then for calculating *nP* the total number of inverses, multiplication, square and addition required are:

Average number of inverses = $(\rho-1)\{w_2 d_i + w_3 t_i\} + (\rho t - 1)(ad_i)$

Average number of Squares= $(\rho-1)\{w_2 d_s + w_3 t_s\} + (\rho t - 1)(ad_s)$

Average number of multiplication= $(\rho-1)\{w_2 d_m + w_3 t_m\} + (\rho t - 1)(ad_m)$.

Here we assume that there are *t* terms on an average present in M_j's, which cause and addition overhead of $\rho t - 1$ additions. If we store all values of M_j's which ranges from 1 to $2^{w_2}3^{w_3} - 1$ in a table say T^m, then we can save computation for calculating $M_j P$. Now the probability of having non-zero M_j is $(2^{w_2}3^{w_3} - 1)/2^{w_2}3^{w_3}$ .Thus we need the following costs:

Average no. of inverses = $(\rho-1)\{w_2 d_i + w_3 t_i\} + ((2^{w_2}3^{w_3} - 1)/2^{w_2}3^{w_3})(ad_i)$

Average no. of squares = $(\rho-1)\{w_2 d_s + w_3 t_s\} + ((2^{w_2}3^{w_3} - 1)/2^{w_2}3^{w_3})(ad_s)$

Average no. of Multiplication = $(\rho-1)\{w_2 d_m + w_3 t_m\} + ((2^{w_2}3^{w_3} - 1)/2^{w_2}3^{w_3})(ad_m)$

We have static storage table T^m where all the value of $m = 2^b 3^t$,

$0 \le b \le w_2, 0 \le t \le w_3$ such that $T^m(b,t) = 2^b 3^t$. We will also need to calculate $M_j P$. We can use the static table T^m to calculate $M_j P$ and store in a table say T_{pr}.

2.3 Finding Scalar multiplication $[n]P$

We use the following steps:

1. To represent n we calculate M_j's first.
2. Now we find out $[M_j]P$. We need the *DBNS* representation of M_j's which can be obtained by greedy algorithm using. In the greedy algorithm we use the table T^m for quick representation. Once we get the representation we can calculate $[M_j]P$ by looking at the precomputed points stored in T_{pr}.
3. Now to compute *[n] P*, we use the value $[M_j]P$ by using *w-doubling* and *w-tripling.*

3 Conclusion and Future Scope

The *DBNS* representation computed by conversion algorithms reported in literature is not suitable for our window based method. Hence we have proposed a new algorithm to convert a given integer to a suitable *DBNS* format. In Earlier proposed method the conversion scheme searching space is more for finding best approximation to the integer *n* .Also the maximum bounds of exponents of binary and ternary has been chosen suitably which are much reasonable. Hence the search space becomes smaller. The selection of length of window has also been made reasonably, which satisfy our validity criteria. Our conversion scheme is fast, as our searching task for the nearest *DBNS* integer is being done in a particular window, which is relatively small. The format obtained so far allow us to have less space to store some precomputed points which increase speed of calculation. The proposed algorithms can be extended to multi-base number system (*MBNS)*

Contents

Chapter 1

Introduction

Elliptic curve cryptography *(ECC)* has occupied the centre stage of public key cryptography research because of its relatively small key-length and enhanced theoretical robustness. The reason behind these is the fact that there is no known algorithm to solve elliptic curve discrete logarithm problem (*ECDLP)* even in sub exponential time. Hence it is believed that a 160-bit *ECC* key can provide the same level of security as an 80-bit symmetric – key for block ciphers or a 1024-bit RSA modulus. This is a major device with limited hardware resources as smart cards, cell phones or PDAs. The efficiency of an *ECC* implementation depends largely on the *scalar multiplication* computed. It is the computation of the point $mP = \sum_{i=1}^{m} P$, for a given point *P* on the curve and an integer *m*. Enormous efforts have been devoted to accelerate and secure this operation.

Among the various methods proposed for efficient and secure implementation of exponentiation, the window-based methods occupy a special place. For exponentiation of a group element in general exponentiations, the window based methods are the fastest. These methods require precomputations and may not be suitable for devices, where memory resources are very low.

The computation of scalar multiplication is also target of adversaries, who use side-channel information to attack cryptosystems. These attacks, instead of attacking the underlying hard problem of a cryptographic protocol, recreate vital information of the cryptosystem, by sampling and measuring side channel information like computation time, the power consumption or the electromagnetic radiation traces. These informations can reveal vital information to the attacker in a straight forward implementation. Many

proposal have been made in literature to prevent the attacker from obtaining any substantial data from the side-channel to endanger the security of a cryptosystem.

In the present work we propose a new window-based scalar multiplication algorithm where the scalar is represented in *double-base number system (DBNS).* DBNS is a number representation scheme, which uses 2 radii to represent integers. In this work we use 2 and 3 as radii. For scalar multiplication, inherent sparseness of this representation scheme leads to fewer point addition than the double-and-add methods. In this work we show that the terms of DBNS representation can be grouped together in small windows to reduce the number of addition further.

We use 2 and 3 as the base of the DBNS representation. That is because, in ECC, we have efficient formulas to compute doubling and tripling of elliptic curve points. Thus the exponent is represented as a sum of terms of the form $\pm 2^b 3^t$. In this work we have proposed algorithms to convert an integer in to a suitable DBNS representation. Though we use greedy algorithm, but it will be used for those integer which falls in a window. That will decrease search time up to great extent. We store some pre computed points which can be used while calculation of *[n] P*. We have proposed how to store those points efficiently. Finally for our representation of n we have proposed algorithm to compute *[n] P.* In general other bases may also be used.

Chapter 2

Elliptic Curve Cryptography in a Public-Key Cryptosystem

2.1 Elliptic Curves over Real Numbers

Elliptic curve over real number is defined as the set E of solutions $(x, y) \in R \times R$ to the equation

$$y^2 = x^3 + ax + b \qquad (1.1)$$

Where $a, b \in R$ are constants such that $4a^3 + 27b^2 \neq 0$. There is a special point O called *the point at infinity*.

The condition $4a^3 + 27b^2 \neq 0$ is necessary and sufficient to ensure that the equation has three distinct roots. Such elliptic curves are known as *non-singular* elliptic curves. If $4a^3 + 27b^2 = 0$ then the corresponding elliptic curve is called a *singular elliptic curve.*

We define a binary operation over a non-singular elliptic curve *E* which makes *E* into an abelian group. This operation is usually denoted by addition. The point at infinity, *O,* will the identity element, so $P+O = O+P = P$ for all $P \in E$.

There are three cases arise when we add two points $P, Q \in E$, where $P = (x_1, y_1)$ and $Q = (x_2, y_2)$.

1. $x_1 \neq x_2$
2. $x_1 = x_2$ and $y_1 = -y_2$
3. $x_1 = x_2$ and $y_1 = y_2$

For case 1, we define a line *L* through *P* and *Q*. *L* intersects *E* in the two points P and *Q*, and it is easy to see that *L* will intersect *E* in one further point, which we call R' .If we reflect R' in the *z*-axis, then we get a point which we name *R*. We define *P*+*Q* = *R*. We can find out the formula to compute *R*. We can write the equation of as $y = \lambda x + \upsilon$, where the slope of *L* is

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1},$$

and

$$\upsilon = y_1 - \lambda x_1 = y_2 - \lambda x_2 .$$

In order to find the intersection points of *E* and *L*, we substitute $y = \lambda x + \upsilon$ in to the equation for *E*, we get the following:

$$(\lambda x + \upsilon)^2 = x^3 + ax + b,$$

Which is the same as

$$x^3 - \lambda^2 x^2 + (a - 2\lambda\upsilon)x + b - \upsilon^2 = 0 \qquad (1.2)$$

The roots of the above equation give the *x*-co-ordinate of the intersection points of *E* and L. Since we have already two points *P* and *Q* .Hence x_1 and x_2 are the roots of equation (1.2).

The equation (1.2) is cubic in *x* and two of its roots namely x_1 and x_2 are real then third of its roots must be real, say x_3.The sum of the three roots must be the negative of the coefficient of the quadratic term, or λ^2 .There for

$$x_3 = \lambda^2 - x_1 - x_2 .$$

x_3 is the *x*-co-ordinate of the point R'. We will denote the *y*-co-ordinate of R' by $-y_3$, so the, so the *y*-co-ordinate of *R* will be y_3. Now the slope of line *L* can be determine by any two points on *L*. If we use the points (x_1, y_1) and $(x_3,-y_3)$ to compute the slope λ, we get

$$\lambda = \frac{-y_3 - y_1}{x_3 - x_1}$$

or,

$$y_3 = \lambda(x_1 - x_3) - y_1 .$$

Thus we have formula for *P*+*Q* in case 1: if $x_1 \neq x_2$, then $(x_1, y_1) + (x_2, y_2) = (x_3, y_3)$, where

$$x_3 = \lambda^2 - x_1 - x_2,$$
$$y_3 = \lambda(x_1 - x_3) - y_1, \text{ and}$$
$$\lambda = \frac{y_2 - y_1}{x_2 - x_2}$$

Case 2. where $x_1 = x_2$ and $y_1 = -y_2$ is simple: we define $(x, y) + (x,-y) = O$ for all $(x, y) \in E$ .There for*(x, y)* and *(x, -y)* are inverses with respect to the elliptic curve addition operation.

Case 3. In this case we are adding a point $P = (x, y)$ to itself. We assume that $y_1 \neq 0$, for then we would be in case 2. Here we compute slope with the help of calculus. The slope of L can be computed using implicit differentiation of the equation of *E:*

$$2y\frac{dy}{dx} = 3x^2 + a.$$

Substituting $x = x_1, y = y_1$, we have the slope of the tangent

$$\lambda = \frac{3{x_1}^2 + a}{2y_1}.$$

The rest of analysis in this case is the same as in case 1.The formula obtained is identical, except that λ is computed differently.

There are some properties of the addition operation as defined above are:

1. addition is closed on the set *E*
2. addition is commutative,
3. *O* is an identity with respect to addition, and
4. Every point on *E* has an inverse with respect to addition.

It is quite messy to prove that *(E +)* is associative by algebraic methods, and hence it is an abelian group.

2.2 Elliptic Curves Modulo a Prime

Let *p*>3 be prime. The *elliptic curve* $y^2 = x^3 + ax + b$ over Z_p is the set of solution $(x, y) \in Z_p \times Z_p$ to the congruence

$$y^2 \equiv x^2 + ax + b (\text{mod } p), \qquad (1.3)$$

where $a, b \in Z_p$ are constants such that $4a^2 + 27b^2 \not\equiv 0 (\text{mod } p)$,together with a special point *O* called *the point at infinity*.

The addition operation on *E* is defined as follows (where all arithmetic operations are performed in Z_p): Suppose

$$P = (x_1, y_1)$$

and

$$Q = (x_1, y_1)$$

are points on *E*. If $x_2 = x_1$ and $y_2 = -y_1$, then P+*Q* = *O:* otherwise *P*+*Q* = (x_3, y_3), where

$$x_3 - \lambda^2 - x_1 - x_2$$
$$y_3 = \lambda(x_1 - x_3) - y_1$$

and

$$\lambda = \begin{cases} (y_2 - y_1)(x_2 - x_1)^{-1}, \textit{if} \quad P \neq Q \\ (3x_1^{\,2} + a)(2y_1)^{-1}, \textit{if} \quad P = Q \end{cases}$$

Finally, define

$$P+O = O+P = P$$

for all $P \in E$.

The same formula can be used to define addition as we defined on the elliptic over reals. The pair *(E, +)* still forms an abelian group.

2.3 Discrete Logarithm Problem

Let *(G, .),* be a multiplicative group and an element $\alpha \in G$ having order *n*, and an element $\beta \in \langle \alpha \rangle$ .Then we define Discrete Logarithm Problem as to find the unique integer $a, 0 \le a \le n-1$, such that

$$\alpha^a = \beta \quad .$$

We will denote this integer *a* by $\log_\alpha \beta$.

The utility of the Discrete Logarithm problem in cryptographic setting is that finding discrete logarithm is (probably) difficult, but the inverse operation of exponentiation can be computed efficiently by using the square-multiply method

2.4 *ElGamal Public-key Cryptosystem in* Z_p^*

Let p be a prime such that the Discrete Logarithm problem in $(Z_p^*, \cdot)$ is infeasible, and let $\alpha \in Z_p^*$, be a primitive element. Let $\mathrm{P} = Z_p^*$, $C = Z_p^* \times Z_p^*$, and define

$$\mathrm{K} = \{(p, \alpha, a, \beta) : \beta \equiv \alpha^a \pmod{p}\}$$

The value p, α and β are the public key, and *a* is private key.

For $\mathrm{K} = (p, \alpha, a, \beta)$, and a (secret) random number $k \in Z_{p-1}$, define encryption function

$$e_{\mathrm{K}}(x, k) = (y_1, y_2),$$

where

$$y_1 = \alpha^k \bmod p$$

and

$$y_2 = x\beta^k \bmod p.$$

for $y_1, y_2 \in Z_p^*$, define decryption function

$$d_K(y_1, y_2) = y_2 (y_1^a)^{-1} \bmod p.$$

A necessary condition for the *ElGamal Cryptosystem* to be secure is that the Discrete Logarithm problem in Z_p^* is infeasible. This generally regarded as being the case if *p* is carefully chosen and α is a primitive element modulo *p*. In particular, there is no known polynomial-time algorithm for this version of the Discrete Logarithm problem.

2.5 The Elliptic Curve Discrete Logarithm Problem

Let *E* be the curve over some finite field, F_q. Let *n* denotes the order of the group $E(F_q)$ and let *P* denotes an element of $E(F_q)$.The elliptic curve discrete logarithm problem (ECDLP) on *E* is, given $Q \in \langle P \rangle$, find the integer, *m*, such that

$$Q = [m]\ P.$$

Chapter 3

Point Arithmetic on Elliptic Curves

The basic building blocks of an elliptic cryptosystem over F_q are computation of the form

$$Q = [k]P = \underbrace{P + P + \cdots + P}_{k \quad times}$$

where P is curve point, and k is an arbitrary integer in the range $1 \le k < ord(P)$. The strength of the cryptosystem lies in the fact that given the curve, the point P (be it fixed or arbitrary) and [k] P, it is hard to recover k, which is *elliptic curve discrete logarithm problem* (*ECDLP).*

3.1 Point Addition

Depending on the characteristic of the underlying field the formulae for the group law take on different forms. We analyse the computational complexity of these formulae separately for characteristic $p > 3$, and for characteristic two.

3.1.1 Fields of Characteristic *p > 3.*

Affine Coordinates.

The point addition on an elliptic curve

$$E : Y^2 = X^3 + ax + b$$

with $a,b \in F_q$, $q = p^n$, p a prime greater than three. Let $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ be points in $E(F_q)$ given in affine coordinates, and where some convention is used to represent *the point at infinity, O.* Assume $P_1, P_2 \neq O$ and $P_1 \neq -P_2$, conditions that are all easily checked. The sum $P_3 = (x_3, y_3) = P_1 + P_2$ can be computed as follows.

If $P_1 \neq P_2$,

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1}$$
$$x_3 = \lambda^2 - x_1 - x_2,$$
$$y_3 = (x_1 - x_3)\lambda - y_1$$

If $P_1 = P_2$

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

When $P_1 \neq P_2$, the computation requires one field inversion and three field multiplications. We will denote this computational cost by 1*I*+3*M*, where *I* and *M* denote, respectively, the cost of field inversion and multiplication. Squaring is counted as regular multiplications. When $P_1 = P_2$, the cost of the point doubling is *I + 4M*. we neglect the cost of field addition, as well as the cost of multiplication by small constants.

Projective coordinates:

In the case where field inversions are significantly more expensive than multiplications, it is efficient to implement *projective* coordinates. A projective point (*X, Y, Z)* on the curve satisfies the homogeneous Weierstrass equation

$$Y^2Z = X^3 + aXZ^2 + bZ^3,$$

and when $Z \neq 0$, it corresponds to the affine point *(X/Z, Y/Z).* There are other projective representations. We will prefer a *weighted projective* representation which is also referred as *jacobian* representation. A triplet *(X, Y, Z)* corresponds to the affine coordinates $(X / Z^2, Y / Z^3)$ whenever $Z \neq 0$. This equivalent to using a weighted projective curve equation of the form

$$Y^2 = X^3 + aXZ^4 + bZ^6$$

The point at infinity *O* is represented by any triplet $(\gamma^2, \gamma^3, 0), \quad \gamma \in F_q^*$.

Conversion from affine to projective coordinates is trivial, while conversion in other direction costs 1*I* + 4*M*. Inversion is costless operation however it increases the number of multiplications, so the appropriateness of using projective coordinates is strongly determined by the ration *I*: M.

Let $P_1 = (X_1, Y_1, Z_1)$ and $P_2 = (X_2, Y_2, Z_2)$, and their sum be $P_3 = (X_3, Y_3, Z_3)$ in projective coordinates. We assume that $P_1, P_2 \neq O$ and that $P_1 \neq \pm P_2$.According to *IEEE P1363* draft standard, the total cost for general point addition comes out to be *16M* and the point doubling computation costs *10M.* This can be reduced to *8M* when $a = -3$ The following table gives the summaries

Table 1 :Cost of point addition, characteristic p>3

Operation	*Coordinates*	
	Affine	***Projective***
General addition	1*I* +3M	16M
Doubling(arbitrary a)	1*I* +4M	10M
Doubling(a=-3)	1*I* +4M	8M

3.1.2 Fields of characteristic two.

Affine Coordinates.

The point addition on an elliptic curve

$$E : Y^2 + XY = X^3 + a_2 X^2 + a_6$$

with $a_2, a_6 \in F_q$, $q = 2^n$, $a_6 \neq 0$. Let $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ be points in affine coordinates, where some convention is used to represent *the point at infinity O.* Assume $P_1, P_2 \neq O$ and $P_1 \neq -P_2$. The sum $P_3 = (x_3, y_3) = P_1 + P_2$ is computed as follows. If $P_1 \neq P_2$,

$$\lambda = \frac{y_1 + y_2}{x_1 + x_2},$$
$$x_3 = \lambda^2 + \lambda + x_1 + x_2 + a_2,$$
$$y_3 = (x_1 + x_3)\lambda + x_3 + y_1 .$$

If $P_1 = P_2$

$$\lambda = \frac{y_1}{x_1} + x_1,$$
$$x_3 = \lambda^2 + \lambda + a_2,$$
$$y_3 = (x_1 + x_3)\lambda + x_3 + y_1 .$$

In either case, the computation requires on field inversion, two field multiplications, and in squaring, or 1*I* + 2*M* +1*S*. In the case of characteristic two, the cost of squaring operation, denoted by *S*, is much lower than that of general multiplication. Therefore, squarings are counted separately, and in fact, we will later on neglect their cost completely.

Projective coordinates.

As in the case of characteristic *p >3,* we will use weighted projective coordinates where a projective point (X, Y, Z), $Z \neq 0$, maps to affine point $(X / Z^2, Y / Z^3)$. This corresponds to using weighted projective curve equation of the form

$$Y^2 + XYZ = X^3 + aX^2Z^2 + a_6Z^6.$$

Conversion from projective to affine coordinates costs, in this case $1I$ +3M+1S. According to *IEEE **[P1363]*** draft standard, the total cost for general point addition comes out to be 15M + 5S. This is reduced to 14M + 4S when $a_2 = 0$.The point doubling computation costs 5M +5S. Since squaring is much faster than general multiplication in characteristic two, point doubling in projective coordinates is close to three times as fast as general point addition.

Table 2: Cost of point addition, characteristic two

Operation	*Coordinates*	
	Affine	***Projective***
General addition $(a_2 \neq 0)$	1I +2M+1S	15M+5S
General addition ($a_2 = 0$)	1I +2M+1S	14M + 4S
Doubling	1I +2M+1S	5M +5S

3.2 New Point Arithmetic Results

There are several efficient algorithms to compute Addition *(ADD)*, Doubling *(DBL)*, Tripling *(TPL)*, *w*-Doubling *(w-DBL)*, Doubling and addition (*DA)*, Tripling and Addition(*TA)*, *Mixed*-addition(m*ADD)*, window-Doubling(*w-DBL)*, window-Tripling (*w-TPl)* in affine and jocobian coordinates. The complexity of their calculations will be used in our proposed work. Some of them are listed below with appropriate references.

Table 3: Cost of various Elliptic Curve group operations.

Operations	Output	For $E(F_{2^m})$		For $E(F_p)$	
		proposed	Cost	proposed	Cost
$DBL(P)$	$2P$	-	$1[i]+2[m]$	-	$6[s]+4[m]$
$ADD(P,Q)$	$P+Q$	-	$1[i]+2[m]$	-	$4[s]+12[m]$
$mADD(P,Q)$	–	-	–	[4]	$3[s]+8m]$
$w-DBL(P)$	$2^w P$	[5]	$1[i]+(4w-2)[m]$	[7]	$4w[m]+(4w+2)[s]$
$DA(P,Q)$	$2P \pm Q$	[6]	$1[i]+9[m]$	-	–
$TPL(P)$	$3P$	[6]	$1[i]+7[m]$	[8]	$10[m]+6[s]$
$w-TPL$	$3^w P$	-	–	[8]	$(4w+2)[m]+(11w-1)[s]$
$TA(P,Q)$	$3P+Q$	[6]	$2[i]+9[m]$	-	–

The costs for curves over binary fields $(E(F_{2^m}))$ are in affine coordinates. Those for curves over prime fields $(E(F_p))$ are in Jcobian coordinates.

Chapter 4

Scalar Multiplication Using Double-Base Number System

All elliptic curve discrete logarithm problems (ECDLP) based cryptographic primitives, like encryption, decryption, Signature generation and verification; need the operation of scalar multiplication. Given an integer *k* and an elliptic curve point *P*, it is the operation of computing *kP*. Efficiency of the scalar multiplication depends largely upon efficiency of the algorithms used for group arithmetic and representation of the scalar. In this chapter, we propose an efficient and secure scalar multiplication algorithm based on double-base chains. We use new window method to find double-base representation.

4.1 Double-Base number System

The Double-base number system (DBNS)[9] is a representation scheme in which every positive integer *k* is represented as the sum or difference of {2, 3}-integers (i.e., numbers of the form $2^b 3^t$) as

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

This number representation is highly redundant and most of these representations are useless. Suitably chosen representation gives better result. So we are interested in a special representation with restricted exponents. The most important theoretical result about the double-base number system is the following theorem which is proved in [10]

Theorem 1. *Every positive integer k can be represented as the sum of at most* $O\left(\frac{\log k}{\log \log k}\right)$ *{2, 3}-integers.*

Table 4: number of double-base representation of small numbers

N	*B = {2, 3}*
10	5
20	12
50	72
100	402
150	1296
200	3096
300	11820

We can see that for small number such a large number of double-base representations exist. Several methods have been proposed to get suitable representation of *n* .one of them is *greedy algorithm*, but the searching space for best approximation is too large. A modification of the greedy algorithm is also being in practice, in which the search space for best approximation is reduced to the size of the window. The liberty to choose exponents of *2* and *3* in particular window gives a short representation of a number *n*. The main advantage of doing this is keeping window size small our search for best approximation becomes fast. But instead of using modified greedy algorithm for finding a double-base representation we have proposed new method to find out the *DBNS* representation. Here is greedy algorithm.

Algorithm1. Greedy Algorithm for conversion into DBNS

Input: *k* a positive integer; max_2, max_3, *>0,* the largest allowed binary, ternary exponents and the array T[0…max_2; 0……max_3]

Output: The sequence $(s_i, b_i, t_i)_{i>0}$ such that $k = \sum_{i=1}^{m} s_i 2^{b_i} 3^{t_i}$,

1. s $\leftarrow 1$
2. **while k >o do**
3. for(b=0 to max_2,t=0 to max_3)
   z = **T[b,t]**, the best approximation of *k*
4. **print(s,b,t)**
5. $max_2 \leftarrow b$, $max_3 \leftarrow t$,
6. **if(*k* < *z)* then**
7. $s \leftarrow -s$
8. $k \leftarrow |k - z|$

4.2 Proposed Window–based method for scalar multiplication

This section is divided in two parts: i) finding a representation of *n*, using windows of *2* and *3* and ii) computing scalar multiplication using the obtained representation.

4.2.1 Finding representation of *n*

As *DBNS* is two dimensional representations, the windows are two dimensional too. Let $\max_2, \max_3$ be the maximum powers of *2* and *3* to occur in the *DBNS* representation. For example, for 160 bit-integer the maximum value of max_2 and max_3 can be 160 and 103 respectively. But looking at greedy algorithm this will give large search space. The selection of bounds of $\max_2, \max_3$ and the dimension of windows effects the *DBNS* representation greatly, so appropriate selection of $\max_2, \max_3$ and window size is needed. Thus the task is divided into two parts: i) finding appropriate maximum bounds and ii) finding appropriate window dimension.

Finding appropriate maximum bounds

As we have seen in greedy algorithm the searching depends largely on the maximum bounds max_2, and $max3$.We can find the suitable bounds of binary and ternary exponents. The method is heuristic, but the bounds are significantly much less and work better. Let *n* an *r*-bit integer, then maximum value of *n* will be $2^{r+1}-1$. So,

$$\lfloor 2^{\max_2} 3^{\max_3} \rfloor \geq 2^{r+1},$$

Assuming $2^{r+1}-1 \approx 2^{r+1}$ or, taking logarithm both side

$$\max_2 + \max_3 \log_2 3 \geq r+1 \qquad (3.1)$$

The smallest value of $\max_2$ *and* $\max_3$ which satisfy equation (3.1) will give the bounds of binary and ternary exponents.

Window selection

After getting the maximum bounds for binary and ternary exponents, the window is selected in such a way that each window should be of same length. For that we break entire range (bounds) in ρ parts. Each part will be called a window. Let w_2, *and* w_3 are the window length of base 2 and 3.

$$\max_2 = \rho w_2 \tag{3.2}$$

$$\max_3 = \rho w_3 \tag{3.3}$$

Substituting equation (3.2) and (3.3) in equation (3.1), we get

$$\rho(w_2 + w_3 \log_2 3) \geq r + 1 \tag{3.4}$$

The equation (3.1) suggests that given a number of *r-bit* size, we can approximate the value of max_2 and max_3. With the help of equation (3.2) or (3.3) we can find number of partitions for different value of w_2 and w_3. Equation (3.4) is validity criteria for the values of windows. Thus we can find pairs of valid window values which satisfy (3.4). For different value of w_2 and w_3 we have find out the number of partitions. For *r = 160* bit there is a table which shows the number of partitions for different number w_2 and w_3.

Now we finally move on, how the number *n* will be represented with proposed window double base number system.

Let w_2 and w_3 be the window size for binary and ternary exponents.

Preposition 1*: Let m be a number such that* $1 \leq m \leq 2^{w_2} 3^{w_3}$ *and the best approximation of m be* $2^b 3^t$, *where* $0 \leq b \leq w_2$,$0 \leq t \leq w_3$ *then* $k = \left|m - 2^b 3^t\right| < m$

***Proof*:** Case 1: If b = *t =0* then $|m-1| < m$.

Case2: Let $k = \left|m - 2^b 3^t\right| \geq m$. Take $k' = |m-1| < m$, then $2^b 3^t$ will not be the integer to *m* , a contradiction.

Preposition 2: *Let w_2 and w_3 be the window size and m be a positive integer such that $0 < m < 2^{w_2}3^{w_3}$ can be represented as $\sum_j s_j 2^{b_j}3^{t_j}$ where $s_j \in \{-1, 0, 1\}$ and $0 \le b_j \le w_2$, $0 \le t_j \le w_3$*

***Proof*:** Case 1: If *m = 0* , put *j = 1* and $s_j = 0$.

Case 2: If $1 \le m \le 2^{w_2}3^{w_3}$ then by Preposition 1, there exists an integer 2^b3^t such that

$k = |m - 2^b3^t| < m - 1$, where $0 \le b \le w_2$, $0 \le t \le w_3$

Put *s* = 1, if $(m - 2^b3^t) \ge 0$ else $s = -1$, if $(n - 2^b3^t) < 0$. Since k has been decreased, we apply the same procedure on *k* till *k* becomes 0.

Preposition 3: *Every integer $0 \le n < 2^{\rho w_2}3^{\rho w_3}$ can uniquely be represented as $n = (2^{w_2}3^{w_3})^{\rho-1}M_{\rho-1} + (2^{w_2}3^{w_3})^{\rho-2}M_{\rho-2} + \ldots\ldots + M_0$ such that $0 \le M_{\rho-1}, M_{\rho-2}, \ldots, M_0 < 2^{w_2}3^{w_3}$.*

Proof: We first show that this kind of representation exists. Let $n = M_{\rho-1}(2^{w_2}3^{w_3})^{\rho-1} + R_{\rho-1}$, where $0 \le R_{\rho-1} < (2^{w_2}3^{w_3})^{\rho-1}$. $M_{\rho-1}$ should be strictly less than $2^{w_2}3^{w_3}$ i.e. $0 \le M_{\rho-1} < 2^{w_2}3^{w_3}$, otherwise $n \ge (2^{w_2}3^{w_3})^{\rho}$. Similarly

$$R_{\rho-1} = M_{\rho-2}(2^{w_2}3^{w_3})^{\rho-2} + R_{\rho-2}, \quad \textit{where} \quad 0 \le R_{\rho-2} < (2^{w_2}3^{w_3})^{\rho-2}$$

$$\text{and } 0 \le M_{\rho-2} < 2^{w_2}3^{w_3}$$

$$\vdots$$

$$\vdots$$

$$R_2 = M_1(2^{w_2}3^{w_3})^1 + R_1, \quad \textit{where } 0 \le R_1 < (2^{w_2}3^{w_3})$$

$$\text{and } 0 \le M_1 < 2^{w_2}3^{w_3}.$$

$$R_1 = M_0, \quad \textit{where} \quad 0 \le M_0 < 2^{w_2}3^{w_3}.$$

Thus we get the desired representation.

Now we show that the is unique representation of *n.*

Let $n = (2^{w_2}3^{w_3})^{\rho-1} M_{\rho-1} + (2^{w_2}3^{w_3})^{\rho-2} M_{\rho-2} + \ldots\ldots + M_0$ and

$n = (2^{w_2}3^{w_3})^{\rho-1} M'_{\rho-1} + (2^{w_2}3^{w_3})^{\rho-2} M'_{\rho-2} + \ldots\ldots + M'_0$ be two different representations of n .i.e. there exist at least one $M_i \neq M'_i$ for some $0 \leq i < \rho - 1$. Thus ,

$$(2^{w_2}3^{w_3})^{\rho-1}(M_{\rho-1} - M'_{\rho-1}) + (2^{w_2}3^{w_3})^{\rho-2}(M_{\rho-2} - M'_{\rho-2}) + \ldots + (M_0 - M'_0) = 0$$

This shows that $2^{w_2}3^{w_3}$ is the root of equation

$$(M_{\rho-1} - M'_{\rho-1})X^{\rho-1} + (M_{\rho-2} - M'_{\rho-2})X^{\rho-2} + \ldots\ldots(M_0 - M'_0) = 0 \qquad (3.5)$$

Let f be the degree of polynomial

$$(M_{\rho-1} - M'_{\rho-1})X^{\rho-1} + (M_{\rho-2} - M'_{\rho-2})X^{\rho-2} + \ldots\ldots(M_0 - M'_0)$$

i.e. $(M_i - M'_i) = 0$ for all *i* > *f*. The integral roots of the above equation (3.5) are in the form of $\pm$ *some factor of* $(M_0 - M'_0)$ if $(M_0 - M'_0) \neq 0$, but $-2^{w_2}3^{w_3} < (M_0 - M'_0) < 2^{w_2}3^{w_3}$, so $2^{w_2}3^{w_3}$ can not be the root of equation (3.5). If $(M_i - M'_i) = 0$, then equation (3.5) reduces to

$$(M_{\rho-1} - M'_{\rho-1})X^{\rho-2} + (M_{\rho-2} - M'_{\rho-2})X^{\rho-3} + \ldots\ldots(M_1 - M'_1) = 0 \qquad (3.6)$$

Applying same method as above, we will get $M_j = M'_j$ for all $0 \leq j \leq \rho - 1$. Hence proved.

Preposition 3 gives a suitable representation of *n*. It suggests that finding the representation of $M_{\rho-1}, M_{\rho-2}, \ldots M_0$ in window size w_2 *and* w_3 in double base is sufficient to represent *n*. Now finding the representation of M_i's in double base number system in the window size w_2 *and* w_3 is easy. We can apply greedy algorithm which has been given in Algorithm 1. Just we need to search the best approximation in a window w_2 and w_3. The only change that we need to make is to change max_2 to w_2 and max_3 to w_3, and the static storage space is changed to T [$0 \ldots w_2$; $0 \ldots\ldots w_3$]. Finding double base representation in window (which relatively very small compare to maximum bound) becomes faster and static table size will be much smaller.

4.3 Calculation of complexities

We will calculate the complexity of average number of inverse, square and multiplication applied in calculating scalar multiplication. We have ρ partition of maximum bounds of each binary and ternary exponent. By preposition 3 we get that any integer *n* can have at most ρ terms in our proposed representation scheme. Suppose on average there are *t* number of terms needed to represent M_i 's in a given window.

Then for $0 \le n < 2^{\rho w_2} 3^{\rho w_3}$,

$$n = 2^{w_2} 3^{w_2} (\ldots (2^{w_2} 3^{w_3} (M_{\rho-1}) + M_{\rho-2}) + \ldots) + M_0 \text{ (Horner's rule)} \qquad (3.7)$$

As we can see that for calculating *nP*, there will $(\rho-1)w_2$ doublings, $(\rho-1)w_3$ triplings and $\rho t - 1$ addition needed.

Let there are d_i, t_i, ad_i no. of inverses, d_m, t_m, ad_m no. of multiplications and d_s, t_s, ad_s no. of square needed in doubling, tripling and addition respectively. Then for calculating *nP* the total number of inverses, multiplication, square and addition required are:

Average number of inverses = $(\rho-1)\{w_2 d_i + w_3 t_i\} + (\rho t - 1)(ad_i)$

Average number of Squares= $(\rho-1)\{w_2 d_s + w_3 t_s\} + (\rho t - 1)(ad_s)$

Average number of multiplication= $(\rho-1)\{w_2 d_m + w_3 t_m\} + (\rho t - 1)(ad_m)$.

Here we assume that there are *t* terms on an average present in M_j's, which cause and addition overhead of $\rho t - 1$ additions. If we store all values of M_j'*s* which ranges from 1 to $2^{w_2} 3^{w_3} - 1$ in a table say T^m, then we can save computation for calculating $M_j P$. Now the probability of having non-zero M_j is $(2^{w_2} 3^{w_3} - 1) / 2^{w_2} 3^{w_3}$ .Thus we need the following costs:

Average no. of inverses = $(\rho-1)\{w_2 d_i + w_3 t_i\} + ((2^{w_2} 3^{w_3} - 1) / 2^{w_2} 3^{w_3})(ad_i)$

Average no. of squares = $(\rho-1)\{w_2 d_s + w_3 t_s\} + ((2^{w_2} 3^{w_3} - 1) / 2^{w_2} 3^{w_3})(ad_s)$

Average no. of Multiplication = $(\rho-1)\{w_2 d_m + w_3 t_m\} + ((2^{w_2} 3^{w_3} - 1) / 2^{w_2} 3^{w_3})(ad_m)$

4.4 Finding Scalar multiplication $[n]P$

After finding the representation of *n* in the form of equation (3.7), we will choose some dimension of window. We have static storage table T^m where all the value of $m = 2^b 3^t$, $0 \le b \le w_2, 0 \le t \le w_3$ such that $T^m(b,t) = 2^b 3^t$. Cleary the size of table will be *O(w₂*w₃)*. We will also need to calculate $M_j P$. We can use the static table T^m to calculate $M_j P$ and store in a table say T_{pr}. This table will be of same size as that of T^m. The entries of the table T_{pr} will be like $T_{pr}(b,q) = [2^b 3^t]P$, where *P* is a point on an elliptic curve *E* (algorithm 2). Clearly we are storing more pre computed points, but it will save the computation up to a great extent. The total storage size will be *O(w₂*w₃)*, which is quite less, since we will have small window dimension compare to maximum bounds of the exponents of binary and ternary. The computation of precomputed points in T_{pr} may becomes higher, but it can be reduced if we use recursive computation, as : $[2^{b+1}3^t]P = [2[2^b 3^t]]P$ and $[2^b 3^{t+1}]P = [3[2^b 3^t]]P$. Algorithm 2 gives method to form T_{pr}.

Now we are approaching towards calculation of *[n] P*. In algorithm 5 we have the given pseudo code to obtain this task. We use the following steps :

1. To represent n we calculate M_j's first. Algorithm 3 tells how to calculate.
2. Now we find out $[M_j]P$. We need the *DBNS* representation of M_j's which can be obtained by greedy algorithm using algorithm 1. In the greedy algorithm we use the table T^m for quick representation. Once we get the representation we can calculate $[M_j]P$ by looking at the precomputed points stored in T_{pr}. Pseudo code is given in algorithm 4.
3. Now to compute *[n] P*, we use the value $[M_j]P$ by using *w-doubling* and *w-tripling.* The pseudo code is given in algorithm 5

Algorithm2. Generating static table for precomputed points(T_{pr})

Input: dimension of window w2, w_3 for 2 and 3 respectively and a point P on an elliptic Curve E.

Output: An array $T_{pr}(i, j,)$ such that $T_{pr}(i, j) = [2^i 3^j]P$ where $0 \le i \le w_2, 0 \le j \le w_3$

1. $T_{pr}[0,0] = P$
2. $i \leftarrow 0$
3. $j \leftarrow 0$
4. **while** $j < w_3$ **do**
5. $T_{pr}[i, j+1] = 3[T_{pr}[i, j]]$
6. $j \leftarrow j+1$
7. $i \leftarrow 0$
8. $j \leftarrow 0$
9. **while** $j < w_3 + 1$ **do**
10. **while** $i < w_2$ **do**
11. $T_{pr}[i+1, j] = 2[T_{pr}[i, j]]$
12. $i \leftarrow i+1$
13. $j \leftarrow j+1$
14. **return** T_{pr}

Algorithm 3. To find M_j's

Input: the integer number *n* such that $0 \le n < (2^{w_2}3^{w_3})^{\rho}$ for a given window length w_2, w_3 for 2 and 3 respectively and number of partition ρ

Output: a sequence of $(M_j')_{j>0}$ such that $n = \sum_{j=1}^{\rho-1} M_{\rho-j}(2^{w_2}3^{w_3})^{\rho-j}$, where $0 \le M_j < 2^{w_2}3^{w_3}$ for all $0 \le j < \rho - 1$

1. $j \leftarrow 1$
2. $R \leftarrow n$
3 $X \leftarrow (2^{w_2}3^{w_3})^{\rho-1}$
4. **while** $j \le \rho$ **do**
5. $M_{\rho-1} \leftarrow \left\lfloor \frac{R}{X} \right\rfloor$
6. $R' \leftarrow R - M_{\rho-j}X$
7. $X \leftarrow \frac{X}{2^{w_2}3^{w_3}}$
8. $R \leftarrow R'$
9. $j \leftarrow j + 1$
10. $A[\rho - j] \leftarrow M_{\rho-j}$
11. **return A.**

Algorithm4. Calculation of [*m*] *P*

Input: an integer *m* such that $0 \le m < 2^{w_2}3^{w_3}$, a point *P* on an elliptic curve *E* and T_{pr}

Output: [*m*] *P*

1. $A \leftarrow$ **Algorithm 3**(m, w_2, w_3)
2. $L \leftarrow length(A)$
3. $P \leftarrow O$(point at infinity on elliptic curve *E*)
4. $i \leftarrow 1$
5. **while** $i \le L$ **do**
6. $(s_i, b_i, t_i) \leftarrow A[i]$
7. $P \leftarrow P + s_i T_{pr}(b_i, t_j)$
8. $i \leftarrow i + 1$
9. **return *P***

Algorithm5. Calculation of [*n*] *P*

Input: an integer *n* such that $0 \le n < (2^{w_2}3^{w_3})^{\rho}$, a point *P* on an elliptic curve *E*, no. of partition ρ and T_{pr}.

Output: [*n*]*P*

1. $A \leftarrow$ **Algorithm** $3(n, w_2, w_3, \rho)$
2. $P \leftarrow O$ (point at infinity on elliptic curve *E*)
3. $i \leftarrow 1$
4. **while** $i \le (\rho - 1)$ **do**
5. $Q \leftarrow$ **Algorithm 4** $(A, \rho - 1, w_2, w_3, P, T_{pr})$
6. $P \leftarrow P + Q$
7. $P \leftarrow [3^{w_3}]P$
8. $P \leftarrow [2^{w_2}]P$
9. $i \leftarrow i + 1$
10. **return *P***

Table 5: Average number of terms in a given window and partition for different values of w_2 ,w_3.

w_2	w_3	Average no. terms\	no. of partitions
0	0	0.000000	∞
0	1	1.000000	102
0	2	1.777778	51
0	3	2.481482	34
1	0	0.500000	161
1	1	1.166667	63
1	2	1.722222	39
1	3	2.240741	28
2	0	1.000000	81
2	1	1.416667	45
2	2	1.888889	32
2	3	2.268518	24
3	0	1.375000	54
3	1	1.708333	36
3	2	2.097222	27
3	3	2.458333	21
4	0	1.750000	41
4	1	1.979167	29
4	2	2.333333	23
4	3	2.668982	19
5	0	2.093750	33
5	1	2.218750	25
5	2	2.552083	20
5	3	2.878472	17

Table 7: Cost of elliptic curve scalar multiplication for 160-bit integer using affine coordinates (F_{2^m} -cost) taking ($w_2+1)(w_3+1)$ number of storage points. $[i]/[m]=8$

W_2	W_3	# storage	Inverse [I]	Multiplication [M]	$\approx M$
4	2	15	118.666665	721.33333	1670.67[m]
3	3	16	130.625000	721.25000	1766.25[m]
3	2	12	133.625000	735.25000	1804.25[m]
5	3	24	111.934029	719.86804	1615.34[m]

Table 8: Cost of elliptic curve scalar multiplication for 160-bit integer using affine coordinates (F_{2^m} -cost) taking $2^{w_2}3^{w_3}-1$ number of storage points. $[i]/[m]=8$

W_2	W_3	# storage	Inverse [I]	Multiplication [M]	$\approx M$
4	2	143	87.8472	659.6944	1362.47[m]
3	2	71	103.6388	625.2777	1504.48[m]
3	3	215	99.9074	659.8148	1459.07[m]
5	3	863	79.9814	655.9629	1295.81[m]

Table 9: Cost of elliptic curve scalar multiplication for 160-bit integer using <u>Jacobian</u> coordinates (F_{2^m} -cost) taking ($w_2+1)(w_3+1)$ number of storage points.
$[s]/[m]=0\cdot 8$

W_2	W_3	# storage	Square[S]	Multiplication [M]	$\approx M$
4	2	15	774.000	1235.3332	1854.57[m]
3	2	11	790.875	1303.0000	1935.69[m]
3	3	15	711.8750	1285.0000	1854.50[m]
5	3	71	719.8020	1215.4721	1791.31[m]

Table 10: Cost of elliptic curve scalar multiplication for 160-bit integer using Jacobian coordinates (F_{2^m} -cost) taking $2^{w_2}3^{w_3}-1$ number of storage points. $[s]/[m]=0\cdot 8$

W_2	W_3	# storage	Square [S]	Multiplication [M]	$\approx M$
4	2	143	681.5416	988.7777	1534.01[m]
3	2	71	700.9166	1063.1110	1623.84[m]
3	3	215	619.7222	1039.2592	1535.07[m]
5	3	863	623.9445	959..8518	1459.00[m]

Comparison:

In earlier proposed method [8] using double- chains the cost of complexity for computing scalar multiplication for 160- bit integer is $\approx$ 1863[m] . The coordinate used here is affine. The table7 shows our better result for the same coordinate system. We store very less number of precomputed points. Also the searching space for the best approximation to the given scalar is more in the earlier proposed method, where as the same task has been done in less amount of time ,because of suitably chosen maximum bound, and window size.

Chapter 5

Discussion and Conclusion

In the present work, we have presented a window based method for computing *ECC* scalar multiplication using double-base number system *DBNS* representation of the scalar. The *DBNS* representation computed by conversion algorithms reported in literature is not suitable for our window based method. Hence we have proposed a new algorithm to convert a given integer to a suitable *DBNS* format. In Earlier proposed method [11] the conversion scheme searching space is more for finding best approximation to the integer n .Also the maximum bounds of exponents of binary and ternary has been chosen suitably which are much reasonable. Hence the search space becomes smaller. The selection of length of window has also been made reasonably, which satisfy our validity criteria. Our conversion scheme is fast, as our searching task for the nearest *DBNS* integer is being done in a particular window, which is relatively small. The format obtained so far allow us to have less space to store some precomputed points which increase speed of calculation. The proposed algorithms can be extended to multi-base number system (*MBNS)*

Bibliography

1. Douglas R. Stinson. *Cryptography theory and practice* .CRC press 2002

2. Andreas Enge, *Elliptic curve and their application to cryptography an Introduction.* Kluwer Academic Publishers 2001.

3. I. F. Blake, G. Seroussi, N.P. Smart, Elliptic curves in cryptography, Cambridge University press, 1999.

4. H. Cohen, A. Miyaji, and T.Ono. *Efficient Elliptic Curve Exponentation Using Mixed coordinates,* In ASIACRYPT'98, LNCS 1514, pp. 51-65, Springer-Verlag, 1998.

5. R.Dahab and J.Lopez, An Improvement of Guajardo-Paar Method for Multiplication on non supersingular elliptic Curves. Inproceedings of The XVIII International Conference of the Chiliean Computer Science Society (SCCC'98), IEEE CS Press, November 12-14, Antofagasta, Chile, pp.91-95, 1998.

6. M.Ciet, K. Lauter, M.Joye and P.L. Montgomary Trading inversion for multiplications in elliptic curve cryptography In *Design, codes and cryptography, 32(2):189-206,2006.*

7. K.Itoh, M.Takenaka, N. Torii, S. Temna, and Y. Kurihara. Fast implementation of public –key cryptography on a DSP TMS320C6201. In C. K. Koc and C.Paar, editers, *Cryptographic hardware and Embedded Systems –CHES '99, Volume1717 of lecture notes in computer science, pages 61-72. Springer-Verlag , 1999.*

8. V.Dimitrov, L. Imbert, and P.K.Mishra, Efficient and Secure Elliptic Curve Point Multiplication Using Double Base Chain. In B. Roy Ed., *Asiacrypt 2005, volume 3788 of lecture notes in computer science, Pages 59-79.* Springer-Verlag, 2005

9. V.S. Dimitrov,G.A. Julient, and W.C.Miller. Theory and applications of the double-base number system. *IEEE Transaction on computers* 48(10): 1098-1106, Oct. 1999.

10. V.S. Dimitrov, G.A.Jullian, W.C Miller. An algorithm for modular exponentiation, *information Processing Letters,* 66(3):155-159, may 1998.

11 P.K.Mishra, Window based Elliptic curve Scalar Multiplication Using Double Base Number Representation.

[**P1363]** **IEEE P1363/D3** (Draft version 3). Standard specification for public key crypto graphy. May 1998.