WINDOW BASED SCALAR MULTIPLICATION ON ELLIPTIC CURVE USING MULTI-BASE NUMBER SYSTEM

A dissertation submitted in partial fulfillment of the requirements for the M. Tech(cs) degree of the Indian Statistical Institute

By

Sumit Kumar Pandey

Under the supervision of

Prof. Rana Barua

Indian Statistical Institute
203, B.T. Road, Kolkata - 108

Contents

Chapter 1

Introduction

Elliptic curve cryptography has a wide application in public key cryptography and has received a lot of attention because of its small key size (the equivalent key sizes for ECC are 173 and 313 bits as compared to the key sizes 1024 and 4096 bits for RSA) and increased theoretical robustness (there is no subexponential algorithm to solve elliptic curve discrete logarithm problem, ECDLP). The efficiency of an ECC mainly depends upon the scalar multiplication, i.e., the computation of the point $[n]P = P + ... + P$ (n times), for a given point on an elliptic curve E. An extensive amount of research has been done and being done to efficiently compute and accelarate and secure the scalar multiplication.

Several representations of the scalar n (binary, ternary, non-adjacent form (NAF), window methods (w-NAF)...) and various efficient methods for point addition ($P+Q$, $[2]P$, $[2]P \pm Q$, $[2^w]P$) have been proposed in both affine and projective coordinates. In recent years, a new representation scheme using Double-base number system (DBNS) and Multi-base number system (MBNS) has gained much popularity due to shorter length representation and sparseness. Introduction of new point additions like $[3]P$, $[3]P \pm Q$, $[3^w]P$, $[5]P$ have given new dimensions to calculate scalar multiplication and its results are overwhelming.

In this report, we propose a new window based scalar multiplication algorithm which has advantage over earlier proposed methods that it requires to search for a better window length for bases than searching for maximum bound on bases, which results a smaller size of static table and much faster search. Although it keeps a table of relatively large size of precomputed points, it has overall less storage requirement. Besides it, computation of scalar multiplication using this method has shown an almost equal complexity as earlier proposed methods.

Chapter 2

Elliptic Curve Cryptography

2.1 What is Elliptic Curve?

Elliptic curves are described by the set of solutions to certain equations in two variables. Elliptic curves defined modulo a prime p are of central importance in public-key cryptography. We begin by looking briefly at elliptic curves defined over the real numbers.

2.1.1 Elliptic Curves over the Reals

Definition : Let $a, b \in \mathbb{R}$ be constants such that $4a^3 + 27b^2 \neq 0$. A *non-singular elliptic curve* is the set E of solutions $(x, y) \in \mathbb{R} \times \mathbb{R}$ to the equation

$$y^2 = x^3 + ax + b \tag{2.1}$$

(known as **Weierstrass equation**) together with special point O called the point at *point at infinity*

If the roots of the cubic are r_1, r_2, r_3, then it can be shown that the discriminant of the cubic is

$$((r_1 - r_2)(r_2 - r_3)(r_3 - r_1))^2 = -(4A^3 + 27B^2) \tag{2.2}$$

The condition $4a^3+27b^2 \neq 0$ is both necessary and sufficient condition to ensure that the equation $x^3+ax+b = 0$ has three distinct roots (which may be real or complex). If $4a^3+27b^2 = 0$, then the corresponding curve is called a *singular elliptic curve*.

In order to have little more flexibility, we also allow somewhat more general equations of the form

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6, \tag{2.3}$$

where $a_1, ..., a_6$ are constants. This more general form (**generalized Weierstrass equation**) is useful when working with fields of characteristic 2 and characteristic 3. If the characteristic of the field is not 2, then we can divide by 2 and complete the square:

$$\left(y + \frac{a_1x}{2} + \frac{a_3}{2}\right)^2 = x^3 + \left(a_2 + \frac{{a_1}^2}{4}\right)x^2 + a_4x + \left(\frac{{a_3}^2}{4} + a_6\right), \tag{2.4}$$

which can be written as

$${y_1}^2 = x^3 + {a_2}'x^2 + {a_4}'x + {a_6}', \tag{2.5}$$

with $y_1 = y + a_1x/2 + a_3/2$ and with some constants ${a_2}', {a_4}', {a_6}'$. If the characteristic is also not 3, then we can let $x_1 = x + {a_2}'/3$ and obtain

$${y_1}^2 = {x_1}^3 + Ax_1 + B, \tag{2.6}$$

for some constants A, B.

2.1.2 Group Law

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

Suppose $P, Q \in E$, where $P = (x_1, y_1)$ and $Q = (x_2, y_2)$. We consider three cases:

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

In case 1, we define L to be the line 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 x-axis, then we get a point which we name R. We define $P + Q = R$.

Let's work out an algebraic formula to compute R. First, the equation of L is $y = \lambda x + y$, where the slope of L is

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

and

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

In order to find the points in $E \cap L$, we substitute $y = \lambda x + \nu$ into the equation for E, obtaining the following:

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

which is same as

$$x^3 - \lambda^2 x^2 + (a - 2\lambda\nu)x + b - \nu^2 = 0. \tag{2.7}$$

The roots of equation (2.7) are the x-co-ordinates of the points in $E \cap L$. We already know two points in $E \cap L$, namely, P and Q. Hence x_1 and x_2 are two roots of equation (2.7).

Since equation (2.7) is a cubic equation over the reals having two real roots, the third root, say x_3, must also be real. The sum of the three roots must be the negative of the coefficient of the quadratic term, or λ^2. Therefore

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

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 y-co-ordinate of R will be y_3. An easy way to compute y_3 is to use the fact that the slope of L, namely λ, is determined by any two points on L. If we use the points (x_1, y_1) and $(x_3, -y_3)$ to compute this slope, we get

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

or

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

Therefore we have derived a 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

$$x^3 = \lambda^2 - x_1 - x_2,$$
$$y_3 = \lambda(x_1 - x_3) - y_1,$$

and

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

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

Case 3 remains to be considered. Here we are adding a point $P = (x_1, y_1)$ to itself. We can assume that $y_1 \neq 0$, for then we would be in case 2. case 3 is handled much like Case 1, except that we define L to be the tangent to E at the point P. A little of claculus makes the computation quite simple. 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 see that the slope of the tangent is

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

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

At this point, it can be shown that the addition of ponts on an elliptic curve E satisfies the following properties:

1. *(commutativity)* $P_1 + P_2 = P_2 + P_1$ for all P_1, P_2 on E.

2. *(existence of identity)* $P + O = P$ for all points P on E.

3. *(existence of inverses)* given P on E, there exists P' on E with $P + P' = O$. This point P' will be denoted by $-P$.

4. *(associativity)* $(P_1 + P_2) + P_3 = P_1 + (P_2 + P_3)$ for all P_1, P_2, P_3 on E.

In other words, the points on E form an additive abelian group with O as the identity element.

2.1.3 Elliptic Curves over Fields

In **Weierstrass equation** for an elliptic curve, we specified that a, b, x and y belong to real numbers $\mathbb{R}$, but usually they are taken to elements os a field, for example, the real numbers $\mathbb{R}$, the complex numbers $\mathbb{C}$, the rational numbers $\mathbb{Q}$, one of the finite fields $\mathbb{F}_p$ for a prime p, or one of the finite fields $\mathbb{F}_q$, where $q = p^k$ with $k \geq 1$. If K is a field with $a, b \in K$, then we say that E is defined over K. In this report, E and K will implicitly assumed to denote an elliptic curve and a field over which E is defined.

If we want to consider points with coordinates in some field $L \supseteq K$, we write $E(L)$. Hence,

Definition : Elliptic curve over field L is defined as

$$E(L) = \{O\} \cup \{(x, y) \in L \times L | y^2 = x^3 + ax + b\}$$

where O is the *point at infinity.*

The addition operation on E is defined as follows: Suppose

$$P = (x_1, y_1)$$

and,

$$Q = (x_2, y_2)$$

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

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

Finally, define

$$P + 0 = O + P = P$$

for all $P \in E$.

Although the addition of points on an elliptic curve over $\mathbb{F}_p$ or $\mathbb{F}_q$, where $q = p^k$ and $k \geq 1$, does not have nice geometric interpretation that it does on an elliptic curve over the reals, the same formula can be used to define addition, and the resulting pair $(E, +)$ still forms an abelian group.

2.2 Elliptic Curves in Cryptography

In this section, we'll discuss some cryptosystem based on eliptic curves, especially on the discrete logarithm problem for elliptic curves. The reason for using elliptic curves in cryptography is that it provies security equivalent to classical system while using fewer bits. For example, it is estimated that a key size of 4096 bits for RSA gives the same level of security as 313 bits in an elliptic curve system. This means that implementations of elliptic curve cryptosystem require smaller chip size, less power consumtion etc. Though certain procedures, such as signature verifications, were slightly faster for RSA, the elliptic curve methodssuch as ECC-DSA clearly offer great increases in speed in many situations.

2.2.1 The Discrete Logarithm Problem

Let p be a prime and let a, b be integers that are nonzero mod p. suppose we know that there exists an integer k such that

$$a^k \equiv b(modp)$$

The classical **discrete logarithm problem** is to find k. Since $k + (p - 1)$ is also a solution, the answer k should be rregarded as being defined $modp - 1$, or mod a divisor d of $p - 1$ if $a^d \equiv 1(modp)$.

More generally, let G be any group, written multiplicately for the moment, and let $a, b \in G$. Suppose we know that $a^k \equiv b$ for some integer k. In this context, the discrete logarithm problem is to find k. For example, G could be the multiplicative group ${\mathbb{F}_q}^\times$ of a finite field. Also G could be $E(\mathbb{F}_q)$ for some elliptic curves, in which case a and b are points on E and we are trying to find an integer k with $ka = b$.

2.2.2 Public Key Cryptography

Public key cryptography, also known as **asymmetric cryptography**, is a form of cryptography in which a user has a pair of cryptographic keys - **a public key** and a **private key**. The private key is kept secret, while the public key may be widely distributed. The keys are related mathematically, but the private key cannot be practically derived from the public key. A message encrypted with the public key can be decrypted only with the corresponding private key.

Conversely, Secret key cryptography, also known as **symmetric cryptography** uses a single secret key for both encryption and decryption.

The two main branches of public key cryptography are:

1. Public key encryption a message encrypted with a recipient's public key cannot be decrypted by anyone except the recipient possessing the corresponding private key. This is used to ensure confidentiality.

2. Digital signatures a message signed with a sender's private key can be verified by anyone who has access to the sender's public key, thereby proving that the sender signed it and that the message has not been tampered with. This is used to ensure authenticity.

Modern cryptography, as applied in the commercial world, is concerned with a number of problems. The most important of these are:

1. **Confidentiality:** A message sent from sender to receiver cannot be read by anyone else.

2. **Autenticity:** Receiver knows that only sender could have sent the message he/she has just received.

3. **Integrity:** Receiver knows that the message from sender has not been tampered with in transit.

4. **Non-repudation:** It is impossible for sender to turn around later and say he/she did not send the message.

It is common in literature to introduce public key techniques in the area of confidentiality protection. Public key techniques are, however, usually infeasible to use directly in the context, being orders of magnitude slower than symmetric techniques. Their use in confidentiality is often limited to the transmission of symmetric cipher keys. On the other hand *digital signatures*, which give the user the authentication, integrity and non-repudiation properties required in electronic commerce, seem to require the use of public key cryptography.

2.2.3 Cryptography Based on Groups

In this section, some of the standard protocols of public key cryptography are surveyed. The protocols discussed here only require the use of a finite abelian group G, of order $\#G$, which is assumed to be cyclic. The group of interest in this work is the *additive* group of points on an elliptic curve. However, it is convenient for the remainder of this section to assume the group is *multiplicative*, with generator g, and the order $\#G$, is a prime. If this not the case, we can always take a prime order subgroup of G as our group, with no loss of security.

Diffie-Hellman key exchange.

Sender and receiver wish to agree on a secret random element in the group, which could be of use as a key for a higher speed symmetric algorithm like the *Data Encryption Standard*(DES). They wish to make this agreement over an insecure channel, without having exchanged any information previously. The only public items, which can be shared amongst a group of users, are the group G and an element $g \in G$ of large known order.

1. Sender generates a random integer $x_A \in \{1, ..., \#G - 1\}$. He/She sends to receiver the element.

$$g^{x_A}.$$

2. Receiver generates a random integer $x_B \in 1, ..., \#G - 1$. He/She sends to receiver the element.

$$g^{x_B}.$$

3. Receiver can then compute

$$g^{x_A x_B} = (g_{x_B})^{x_A}.$$

4. Likewise, receiver can then compute

$$g^{x_A x_B} = (g_{x_A})^{x_B}.$$

The only information that eavesdropper knows is G, g, g^{x_A} and g^{x_B}. If eavesdropper can recover $g^{x_A x_B}$ from this data then he/she is said to have solved a $Diffie-Hellmanproblem$(DHP). It is easy to see that if eavesdropper can find discrete logarithms in G then he/she can solve the DHP.

ElGamal encryption.

Sender wishes to send a message to receiver. His/Her message, m, is assumed to be encoded as an element in the group. Receiver has a public key consisting of g and $h = g^x$, where x is the private key.

1. Sender generates a random integer $k \in \{1, ..., \#G - 1\}$ and computes

$$a = g^k, b = h^k m.$$

2. Sender sends the cipher text (a, b) to receiver.

3. Receiver can recover the message from the equation

$$ba^{-x} = h^k m g^{-kx} = g^{xk-xk} m = m.$$

ElGamal digital signature.

Here, Receiver wants to sign a message $m \in (\mathbb{Z}/(\#G)\mathbb{Z})$. He/She can use the same public and private key pair, h and x, as he/she used for the encryption scheme. We will need a bijection f from G to $\mathbb{Z}/(\#G)\mathbb{Z})$.

1. Sender generates a random integer $k \in 1, ..., \#G - 1$, and computes

$$a = g^k.$$

2. Sender computes a solution, $b \in \mathbb{Z}/(\#G)\mathbb{Z}$, to the congruence

$$m \equiv xf(a) + bk(mod \#G).$$

3. Sender sends the signature, (a, b), and the message, m, to receiver.

4. Receiver verifies the signature by checking that the following equation holds:

$$h^{f(a)}a^b = g^{xf(a)+kb} = g^m.$$

Digital Signature Algorithm.

A version of *Digital Signature Algorithm*(DSA), is the basis of the Digital Signature standard. The signature procedure is almost identical to the ElGamal scheme abve. Sender wants to sign a message $m \in \mathbb{Z}/(\#G)\mathbb{Z}$. He/She uses the same public and private key pair h and x as before, and both he/she and receiver use a common bijective mapping, f, from G to $\mathbb{Z}/(\#G)\mathbb{Z}$.

1. Sender generates a random integer $k \in \{1, ..., \#G - 1\}$, and computes

$$a = g^k.$$

2. He/She computes the solutiion, b, to the congruence

$$m \equiv -xf(a) + kb(mod \#G).$$

3. He/She sends the signature, (a, b), and the message, m, to receiver.

4. Receiver computes

$$u = mb^{-1}(mod \#G), v = f(a)b^{-1}(mod \#G).$$

5. He/She then computes

$$w = g^u h^v.$$

and verifies that

$$\begin{aligned} w &= g^u h^v = g^{mb^{-1}} g^{vx} = g^{mb^{-1}+xf(a)b^{-1}} \\ &= g^{(m+xf(a))b^{-1}} = g^{kbb^{-1}} = g^k \\ &= a. \end{aligned}$$

2.3 Point additions in elliptic curves

The efficiency of an ECC mainly depends upon the scalar multiplication, i.e., the computation of the point $[n]P = P + ... + P$ (n times), for a given point on an elliptic curve E. Several representations of the scalar n (binary, ternary, non-adjacent form (NAF), window methods (w-NAF)...) has been proposed earlier[1]. In recent years, a new representation scheme using Double-base number system (DBNS)[5] and Multi-base number system (MBNS)[6] has gained much popularity due to shorter length representation and sparseness. Introduction of algorithms of new point additions, $3P$ in both affine [2] and jacobian coordinates [**?**], $3^w P$ in jacobian coordinates [5], $5P$ in both affine and jacobian coordinates [6], $2^w P$ in both affine [4] and jacobian coordinates [7] and mixed addition $m - (P + Q)$ in projective coordinates [3] has made computaion much faster. Table 2.3 summarizes the cost of operation required in different point addition algorithm along with their references.

Operation	affine		jacobian	
	proposed	cost	proposed	cost
$[2]P$	-	$1[I] + 2[M]$	-	$6[S] + 4[M]$
$P + Q$	-	$1[I] + 2[M]$	-	$4[S] + 12[M]$
$m - (P + Q)$	-	-	[3]	$3[S] + 8[M]$
$[2^w]P$	[4]	$1[I] + (4w - 2)[M]$	[7]	$(4w + 2)[S] + 4w[M]$
$[3]P$	[2]	$1[I] + 7[M]$	[5]	$6[S] + 10[M]$
$[3^w]P$	-	-	[5]	$(4w + 2)[S] + (11w - 1)[M]$
$[5]P$	[6]	$1[I] + 13[M]$	[6]	$9[S] + 15[M]$

Chapter 3

Scalar multiplication using Multi-base number system

In this chapter, we propose an efficient and secure point multiplicaton algorithm based on multi-base chains. This is achieved by taking advantage of the sparseness and the ternary nature of the so-called multi-base number system (MBNS). The speed-ups are the results of fewer point additions and improved formulae for point triplings and quintuplings in both even and odd characteristic.

3.1 Multi-base Number System

Let k be an integer and let $B = \{b_1, ..., b_l\}$ be a set of small integers. A representation of k as a sum of powers of elements of B ($\sum_{j=1}^{m} s_j b_1^{e_{j1}} ... b_l^{e_{jl}}$, where s_j is a sign) is called a multibase representation of n using the base B. The integer m is the length of the representation. In the current chapter we are particularly interested in multibase representation with $B = \{2, 3, 5\}$. The multibase representations are short and highly redundant. The number of representations of n grows very fast in the number of base elements. This is clearly evident from Table 3.1. The multibase representations are very sparse too. One can represent a 160 bit integer using around 23 terms using $B = \{2, 3\}$ and around 15 terms using $B = \{2, 3, 5\}$.

In this chapter, by a multibase representation of n, we mean a representation of the form

$$n = \sum_{i=1}^{m} s_i 2^{b_i} 3^{t_i} 5^{q_i}, \text{ with } s_i \in \{-1, 1\}, \text{ and } b_i, t_i, q_i \geq 0 \tag{3.1}$$

A general multibase representation, although very short, is not suitable for a scalar multipli-

Table 3.1: number of multibase representations of small numbers using various bases.

n	$B = \{2,3\}$	$B = \{2,5\}$	$B = \{2,3,5\}$	$B = \{2,3,5,7\}$
10	5	3	8	10
20	12	5	32	48
50	72	18	489	1266
100	402	55	8425	43777
150	1296	119	63446	586862
200	3027	223	316557	4827147
300	11820	569	4016749	142196718

cation algorithm. So, we are interested in a special representation with restricted exponents.
Definition. A multibase representation $n = \sum_i 2^{b_i} 3^{t_i} 5^{q_i}$ using the bases $B = \{2,3,5\}$ is called a step multibase representation (SMBR) if the exponents $\{b_i\}, \{t_i\}$ and $\{q_i\}$ form three seperate monotonic sequences.

An integer n has several SMBR, the simplest one being the binary representation. If n is represented in SMBR, then we can write using Horner's rule and an addition chain for sclar multiplication can easily be developed.

Some approaches and modifications have been proposed [6] to yield a better and efficient computation of scalar multiplication, however, there are some drawbacks of earlier methods of scalar multiplication.

1. Large Search Space : Large value of maximum bounds on exponents of $2, 3$ and 5 makes the search space too large in conversion from integer to multi-base chain.

2. Large Table Size : There is a static table which keeps $max_2 \times max_3 \times max_5$ entries. For $n = 160$ bit integer, there will be almost $O(10,000)$ entries.

3. Monotonicity in SMBR: Monotonicity puts an unwanted restriction on exponents of bases. Sometimes, a better representation can be found without taking consideration of monotonicity. For example,

$$
\begin{aligned}
159 &= 150 + 10 - 1 \\
&= 2^1 3^1 5^2 + 2^1 3^0 5^1 - 2^0 3^0 5^0
\end{aligned}
$$

but,

$$\begin{aligned} 159 &= 150 + 9 \\ &= 2^1 3^1 5^2 + 2^0 3^2 5^0 \end{aligned}$$

To overcome above problems partially, we propose an alternative method, *window-based method*, for scalar multiplication.

3.2 Proposed Window-based method for scalar multiplication

In this section, We will focus on (a) bounds on maximum exponents of bases, namely 2, 3 and 5, and (b) criteria for suitable window length which gives less computation and less memory size.

3.2.1 Maximum bounds

In earlier proposed methods, search space was too large to find a representation of an integer to multibase-chain. For example, for 160-bit integer, maximum exponent for 2, 3 and 5 is 160, 103 and 69 respectively. Although it gives a better candidate for the nearest integer to n, but at the sake of large searching. There are other bounds also proposed which are significantly much less and work better, but all these are hueristic. We propose a reasonable bound, say max_2, max_3 and max_5 for 2, 3 and 5 respectively. let n be an r-bit integer, then maximum value of n will be $2^{r+1} - 1$. So,

$$\lfloor 2^{max_2} 3^{max_3} 5^{max_5} \rfloor \geq 2^{r+1}, \text{ assuming } 2^{r+1} - 1 \approx 2^{r+1}$$

or,

$$max_2 + max_3 \log_2 3 + max_5 \log_2 5 \geq r + 1 \tag{3.2}$$

We need to find the smallest value of $max_2 + max_3\log_2 3 + max_5 \log_2 5$ which satisfies equation (3.2). This reduces the search space extensively.

3.2.2 Window selection

Reducing the maximum bound is yet not sufficient. We break the entire range into ρ parts, called *window*. Let *window length* for base 2, 3 and 5 be w_2, w_3 and w_5 respectively. Then,

$$max_2 = \rho w_2 \quad (3.3)$$

$$max_3 = \rho w_3 \quad (3.4)$$

$$max_5 = \rho w_5 \quad (3.5)$$

Substituting equation (3.3), (3.4) and (3.5) in equation (3.2), we get

$$\rho(w_2 + w_3\log_2 3 + w_5\log_2 5) \geq r + 1 \quad (3.6)$$

Equation (3.6) diverts the search criteria from maximum bound to no. of partitions and window length w_2, w_3 and w_5.

Algorithm 1: To compute no. of partitons for a given window
Input : window lengths w_2, w_3, w_5 for 2, 3 and 5 resp. and bit length r. **Output :** no. of partitions, ρ. 1: $s \leftarrow \lfloor w_2 + w_3\log_2 3 + w_5\log_2 5 \rfloor$. 2: $\rho \leftarrow (r+1)/s$. 3: **return** ρ.

3.3 Representation of n.

Before proceeding to the modified greedy algorithm for representation of n, we need to observe that how n looks?

Lemma 3.3.1. *Let $1 \leq n < 2^{w_2}3^{w_3}5^{w_5}$ and the nearest integer to n be $2^b3^t5^q$, where $0 \leq b \leq w_2$, $0 \leq t \leq w_3$, $0 \leq q \leq w_5$, then $k = |n - 2^b3^t5^q| < n$.*

Proof. Case 1: If $2^b3^t5^q = 2^03^05^0 = 1$, then $k = |n - 1| < n$.
Case 2: Let $k = |n - 2^b3^t5^q| \geq n$. Take $k' = |n - 1| < n$, then $2^b3^t5^q$ won't be the nearest integer to n, a contradiction. □

Corollary 3.3.2. *Every integer $0 \leq n < 2^{w_2}3^{w_3}5^{w_5}$ can be represented as $\sum_j s_j 2^{b_j}3^{t_j}5^{q_j}$, where $s_j \in \{-1, 0, 1\}$ and $0 \leq b_j \leq w_2$, $0 \leq t_j \leq w_3$, $0 \leq q_j \leq w_5$.*

Proof. Case 1: If $n = 0$, put $j = 1$ and $s_j = 0$.
Case 2: If $1 \leq n < 2^{w_2}3^{w_3}5^{w_5}$, then by lemma 3.3.1, there exists an integer $2^b 3^t 5^q$ s.t.

$$k = |n - 2^b 3^t 5^q| < n - 1, \text{ where } 0 \leq b \leq w_2, 0 \leq t \leq w_3, \text{ and } 0 \leq q \leq w_5.$$

Put $s = 1$, if $n - 2^b 3^t 5^q \geq 0$ else $s = -1$, if $n - 2^b 3^t 5^q < 0$. Apply same procedure for k till we get 0 (Note that least value for $2^b 3^t 5^q$ is 1.) Hence, the corollary. □

It may be easily observed that there can be at most 2 integers which are nearest to some integer n, say $n - d$ and $n + d$. In that case we will choose the nearest integer which is smaller than n.

Lemma 3.3.3. *Every integer* $0 \leq n < 2^{\rho w_2}3^{\rho w_3}5^{\rho w_5}$ *can be uniquely represented as* $n = (2^{w_2}3^{w_3}5^{w_5})^{\rho-1}M_{\rho-1} + (2^{w_2}3^{w_3}5^{w_5})^{\rho-2}M_{\rho-2} + ... + M_0$ *s.t.* $0 \leq M_{\rho-1}, M_{\rho-2}, ..., M_0 < 2^{w_2}3^{w_3}5^{w_5}$.

Proof. We divide the proof into two parts, (a) existence and (b) uniqueness.
(a) Existence: Let $n = M_{\rho-1}(2^{w_2}3^{w_3}5^{w_5})^{\rho-1} + R_{\rho-1}$, where $0 \leq R_{\rho-1} < (2^{w_2}3^{w_3}5^{w_5})^{\rho-1}$. $M_{\rho-1}$ should be strictly less than $2^{w_2}3^{w_3}5^{w_5}$, i.e., $0 \leq M_{\rho-1} < 2^{w_2}3^{w_3}5^{w_5}$, otherwise $n \geq (2^{w_2}3^{w_3}5^{w_5})^{\rho}$. Similarly

$$\begin{aligned}
R_{\rho-1} &= M_{\rho-2}(2^{w_2}3^{w_3}5^{w_5})^{\rho-2} + R_{\rho-2}, \text{ s.t. } 0 \leq R_{\rho-2} < (2^{w_2}3^{w_3}5^{w_5})^{\rho-2}, 0 \leq M_{\rho-2} < 2^{w_2}3^{w_3}5^{w_5} \\
&\vdots \\
R_2 &= M_1(2^{w_2}3^{w_3}5^{w_5})^1 + R_1, \text{ s.t. } 0 \leq R_1 < (2^{w_2}3^{w_3}5^{w_5}), 0 \leq M_1 < 2^{w_2}3^{w_3}5^{w_5}. \\
R_1 &= M_0, \text{ s.t. } , 0 \leq M_0 < 2^{w_2}3^{w_3}5^{w_5}.
\end{aligned}$$

After summing up, we get the desired representation.
(b) Uniqueness: Let $n = (2^{w_2}3^{w_3}5^{w_5})^{\rho-1}M_{\rho-1} + (2^{w_2}3^{w_3}5^{w_5})^{\rho-2}M_{\rho-2} + ... + M_0$ and $n = (2^{w_2}3^{w_3}5^{w_5})^{\rho-1}M'_{\rho-1} + (2^{w_2}3^{w_3}5^{w_5})^{\rho-2}M'_{\rho-2} + ... + M'_0$ be two different representations of n, i.e., there exist at least one $M_i \neq M'_i$ for some $0 \leq i \leq \rho - 1$. Thus,

$$(2^{w_2}3^{w_3}5^{w_5})^{\rho-1}(M_{\rho-1} - M'_{\rho-1}) + ... + (M_0 - M'_0) = 0$$

which implies that $2^{w_2}3^{w_3}5^{w_5}$ is the root of the equation

$$(M_{\rho-1} - M'_{\rho-1})X^{\rho-1} + (M_{\rho-2} - M'_{\rho-2})X^{\rho-2} + ... + (M_0 - M'_0) = 0 \tag{3.7}$$

Let l be the degree of the polynomial $(M_{\rho-1} - M'_{\rho-1})X^{\rho-1} + (M_{\rho-2} - M'_{\rho-2})X^{\rho-2} + ... + (M_0 -$

M_0') , i.e, $(M_i - M_i') = 0$ for all $i > l$. The integral roots of the above equation (3.7) are in the form of $\pm$ *some factor of* $(M_0 - M_0')$ if $(M_0 - M_0') \neq 0$, but $-2^{w_2}3^{w_3}5^{w_5} < (M_0 - M_0') < 2^{w_2}3^{w_3}5^{w_5}$, so $2^{w_2}3^{w_3}5^{w_5}$ can't be the root of equation (3.7). If $(M_0 - M_0') = 0$, then equation (3.7) reduces to

$$(M_{\rho-1} - M_{\rho-1}')X^{\rho-2} + (M_{\rho-2} - M_{\rho-2}')X^{\rho-3} + ... + (M_1 - M_1') = 0 \qquad (3.8)$$

Applying same reasoning as above, we get that $M_j = M_j'$ for all $0 \leq j \leq \rho - 1$. Hence the lemma. □

We can verify Lemma (3.3.3) alternatively; there are $2^{w_2}3^{w_3}5^{w_5}$ choices for each M_j where $0 \leq j \leq (\rho - 1)$. So, there will be $(2^{w_2}3^{w_3}5^{w_5})^\rho$ different combinations, yielding numbers from 0 to $(2^{w_2}3^{w_3}5^{w_5})^\rho - 1$.

Lemma (3.3.3) gives a nice representation of n and suggests that finding $M_{\rho-1}, M_{\rho-2}, ..., M_0$ is sufficient to represent n. Since $0 \leq M_j < 2^{w_2}3^{w_3}5^{w_5}$ for all $0 \leq j \leq \rho - 1$, only search in a given window will give the desired representation. Consequently, search will be much faster and static table size will be much smaller.

3.4 Average number of inverse, square and multiplication

Let $0 \leq m < 2^{w_2}3^{w_3}5^{w_5}$ and t be the average number of terms for representing m. Then for $0 \leq n < 2^{\rho w_2}3^{\rho w_3}5^{\rho w_5}$,

$$n = 2^{w_2}3^{w_3}5^{w_5}(...(2^{w_2}3^{w_3}5^{w_5}(M_{\rho-1}) + M_{\rho-2}) + ...) + M_0 \ (\textit{Horner's rule})$$

there will be an average of $(\rho - 1)w_2$ doublings, $(\rho - 1)w_3$ triplings, $(\rho - 1)w_5$ quintuplings and $\rho t - 1$ additions.

If there are do_i, tr_i, qu_i, ad_i no. of inverses, do_m, tr_m, qu_m, ad_m no. of multiplications and do_s, tr_s, qu_s, ad_s no. of squares in doubling, tripling, quintupling and addition respectively, then

1. Average no. of inverses $(A_i) = (\rho - 1)\{w_2(do_i) + w_3(tr_i) + w_5(qu_i)\} + (\rho t - 1)(ad_i)$

2. Average no. of squares $(A_s) = (\rho - 1)\{w_2(do_s) + w_3(tr_s) + w_5(qu_s)\} + (\rho t - 1)(ad_s)$

3. Average no. of multiplications $(A_m) = (\rho - 1)\{w_2(do_m) + w_3(tr_m) + w_5(qu_m)\} + (\rho t - 1)(ad_m)$

If all values of $M_j s$ are stored, i.e, from 1 to $2^{w_2}3^{w_3}5^{w_5} - 1$, then we need less computation since we can save computation for calculating $M_j P$. In that case, there will be $(\rho - 1)w_2$ doublings, $(\rho - 1)w_3$ triplings, $(\rho - 1)w_5$ multiplications and $(\rho - 1)$ additions. The probability of having non-zero M_j is $(2^{w_2}3^{w_3}5^{w_5} - 1)/2^{w_2}3^{w_3}5^{w_5}$. Hence

1. Average no. of inverses $(A_i) =$

   $(\rho - 1)\{w_2(do_i) + w_3(tr_i) + w_5(qu_i) + ((2^{w_2}3^{w_3}5^{w_5} - 1)/2^{w_2}3^{w_3}5^{w_5})(ad_i)\}$

2. Average no. of squares $(A_s) =$

   $(\rho - 1)\{w_2(do_s) + w_3(tr_s) + w_5(qu_s) + ((2^{w_2}3^{w_3}5^{w_5} - 1)/2^{w_2}3^{w_3}5^{w_5})(ad_s)\}$

3. Average no. of multiplications $(A_m) =$

   $(\rho - 1)\{w_2(do_m) + w_3(tr_m) + w_5(qu_m) + ((2^{w_2}3^{w_3}5^{w_5} - 1)/2^{w_2}3^{w_3}5^{w_5})(ad_m)\}$

Table (3.2) shows the average number of terms and partitons for different values of $w2, w3, w5$.

3.5 To find $[n]P$.

In the proposed window based scalar multiplication algorithm, we need to form two tables, namely T^P and T^P_{pr}. Table T^P contains the values of different $2^b3^t5^q$, $0 \leq b \leq w_2, 0 \leq t \leq w_3$ and $0 \leq q \leq w_5$ such that $T^P(b,t,q) = 2^b3^t5^q$. On the other hand $T^P_{pr}(b,t,q) = [2^b3^t5^q]P$, where P is the point on an elliptic curve E. Although we will store much precomputed points than any earlier proposed method for scalar multipication, the total storage size will be much smaller. In earlier proposed methods, we need to form a static table, i.e, T^P whose size were $max_2 \times max_3 \times max_5$ that will be almost equal to $O(\rho^3 w_2 w_3 w_5)$ but in proposed window based method, total storage size for both tables T^P and T^P_{pr} will be $2(w_2+1)(w_3+1)(w_5+1)$, which is almost $1/(\rho)^3$ times less than tables formed by earlier proposed methods.

To reduce computation of scalar multipication, we can use another table by help of $T^P_{pr}\prime$. We can form a table which contains all prcomputed points within window. There will be $2^{w_2}3^{w_3}5^{w_5} - 1$ no. of points from $[2^0 3^0 5^0]P$ to $[2^{w_2}3^{w_3}5^{w_5} - 1]P$; there is no need to compute $[2^{w_2}3^{w_3}5^{w_5}]P$, but it was already precomputed (while forming T^P_{pr}), hence we can assume that $T^P_{pr}\prime$ contains $2^{w_2}3^{w_3}5^{w_5}$ no. of precomputed points. We form $T^P_{pr}\prime$ with the help of T^P_{pr} in the same way as it is done to calculate an MBNS representation of some number m using greedy approach. It requires addition of points only.

Computation of precomputed points in T_{pr}^{P} may be higher (depending upon window length), but it may be reduced if $2^b3^t5^q$ could be computed recursively. It is easily noticable that (a) $[2^{b+1}3^t5^q]P = [2[2^b3^t5^q]]P$, (b) $[2^b3^{t+1}5^q]P = [3[2^b3^t5^q]]P$ and (c) $[2^b3^t5^{q+1}]P = [5[2^b3^t5^q]]P$. Algorithm 2 uses recursive method to form T_{pr}^{P}.

After having both tables T^P and T_{pr}^{P} or T^P and $T_{pr}^{P}\prime$, we can calculate $[n]P$ using following steps:

1. First, we calculate M_js, where $n = \sum_{j=1}^{\rho} M_{\rho-j}(2^{w_2}3^{w_3}5^{w_5})^{\rho-j}$ and ρ is the number of partitons. (Algorithm 4).

2. Now, we find out the $[M_j]P$, (Algorithm 5). To obtain this we first need to represent M_j in MBNS which can be done by using greedy approach, (Algorithm 3). In Algorithm 3 we use table T^P by which we can get MBNS representation of M_j quickly. After getting the representation, we can evalute $[M_j]P$ by looking at the precomputed points stored in table T_{pr}^{P} and adding them.

3. After getting the value of all $[M_j]P$, we can evaluate $[n]P$ by quintupling, tripling, doubling and addition of points, (Algorithm 6).

3.6 Cost of operations in computation of precomputed points

Computation of $[2^b3^t5^q]P$ has been suggested to do recursively i.e., (a) $[2^{b+1}3^t5^q]P = [2[2^b3^t5^q]]P$, (b) $[2^b3^{t+1}5^q]P = [3[2^b3^t5^q]]P$ and (c) $[2^b3^t5^{q+1}]P = [5[2^b3^t5^q]]P$. There will be w_5 quintuplings, $w_3(w_5+1)$ triplings and $w_2(w_3+1)(w_5+1)$ doublings in formation of T_{pr}^{P} (Algorithm 2), hence

1. Total no. of inverse $(T_{T_{pr}^{P}i}) = w_5(qu_i) + w_3(w_5+1)(tr_i) + w_2(w_3+1)(w_5+1)(do_i)$.

2. Total no. of square $(T_{T_{pr}^{P}s}) = w_5(qu_s) + w_3(w_5+1)(tr_s) + w_2(w_3+1)(w_5+1)(do_s)$.

3. Total no. of mult. $(T_{T_{pr}^{P}m}) = w_5(qu_m) + w_3(w_5+1)(tr_m) + w_2(w_3+1)(w_5+1)(do_m)$.

We form $T_{pr}^{P}\prime$ with the help of T_{pr}^{P} in the same way as it is done to calculate an MBNS representation of some number m using greedy approach. It requires addition of points only.

Hence total no additons will be (total no. of points in a window) $\times$ (average no. of terms in the representation using T^P_{pr}). Since calculation of $[m]P$ using T^P_{pr} needs to find a representaion which is same as finding an MBNS representation of m, average no. of terms will be t. So, total cost of formation of table $T^P_{pr}\prime$ will be sum of total cost of formation of T^P_{pr} and total cost of additions. Hence,

1. Total no. of inverse $(T'_{T^P_{pr}i}) = T_{T^P_{pr}i} + 2^{w_2}3^{w_5}5^{w_5} \times t(ad_i)$.
2. Total no. of square $(T'_{T^P_{pr}s}) = T_{T^P_{pr}s} + 2^{w_2}3^{w_5}5^{w_5} \times t(ad_s)$.
3. Total no. of mult $(T'_{T^P_{pr}m}) = T_{T^P_{pr}m} + 2^{w_2}3^{w_5}5^{w_5} \times t(ad_m)$.

3.7 Comparison

Let us compare the performance of the proposed window based scalar multiplication scheme to other schemes in literature. We have compared results with [6]. In [6], authors have proposed an efficient method to calculate $[5]P$ and they used it to develop a different scalar multiplication algorithm using MBNS with bases 2, 3 and 5. They compared their results with several other methods and their result was better than that of other proposed methods.

We have computed cost of $[n]P$ using existing algorithm for $[2^w]P$ ([4]), $[3]P$ ([2]) and $[5]P$ ([6]) in affine coordinates for curves over characterisic 2. Since square is almost free in affine coordinates, we have not taken the cost of squaring in this coordinate system. We have done numerical tests on window length (0,0,0) to (5,3,2). Our proposed method for 160-bit integer with $(w_2 + 1) \times (w_3 + 1) \times (w_5 + 1)$ precomputed points requires almost 1513 multiplications for window length $(w_2, w_3, w_5) = (5, 0, 0)$ with 6 precomputed points, whereas same with $2^{w_2}3^{w_3}5^{w_5}$ precomputed points requires almost 1142 multiplications with 32 precomputed points as compared to the best result obatined in [6] with 1469 multiplications and 5 precomputed points (taking $[I]/[M] = 8$). Table 3.3 and Table 3.4 shows cost of inverse, multiplication and their equivalent multplication cost for different window lengths with $(w_2 + 1) \times (w_3 + 1) \times (w_5 + 1)$ and $2^{w_2}3^{w_3}5^{w_5}$ no. of precomputed points respectively.

For curves over prime characteristic not equal to 2 , we have used algorithm for computing $[2^w]P$ ([7]), $[3^w]P$ ([5]), $[5]P$ ([6]) and mixed addition ([3]) in jacobian coordinates. We have done numerical tests on window length from (0,0,0) to (5,3,2). Our proposed method for 160-bit integer with $(w_2 + 1) \times (w_3 + 1) \times (w_5 + 1)$ precomputed points requires almost 1748 multiplications for window length $(w_2, w_3, w_5) = (3, 3, 2)$ with 48 precomputed points, whereas same with $2^{w_2}3^{w_3}5^{w_5}$ precomputed points requires almost 1454 multiplications with

5400 precomputed points as compared to the best result obatined in [6] with almost 1502 multiplications with 5 precomputed points (taking $[S]/[M] = 0.8$). Table 3.5 and Table 3.6 shows cost of inverse, multiplication and their equivalent multplication cost for different window lengths with $(w_2+1)\times(w_3+1)\times(w_5+1)$ and $2^{w_2}3^{w_3}5^{w_5}$ no. of precomputed points respectively.

Algorithm 2: Table construction for precomputed points.

Input : window length w_2, w_3, w_5 for 2, 3 and 5 resp. and a point P on an elliptic curve E.

Output : An array $T_{pr}^{P}(i,j,k)$ such that $T_{pr}^{P}(i,j,k) = [2^i 3^j 5^k]P$ where $0 \le i \le w_2$, $0 \le j \le w_3$ and $0 \le k \le w_5$.

$T_{pr}^{P}[0,0,0] = P$
$i \leftarrow 0$
$j \leftarrow 0$
$k \leftarrow 0$
while $k < w_5$ **do**
    $T_{pr}^{P}(i,j,k+1) = [5]T_{pr}^{P}(i,j,k)$
    $k \leftarrow k+1$
$i \leftarrow 0$
$j \leftarrow 0$
$k \leftarrow 0$
while $k < w_5 + 1$ **do**
    while $j < w_3$ **do**
        $T_{pr}^{P}(i,j+1,k) = [3](T_{pr}^{P} i,j,k)$
        $j \leftarrow j+1$
    $k \leftarrow k+1$
$i \leftarrow 0$
$j \leftarrow 0$
$k \leftarrow 0$
while $k < w_5 + 1$ **do**
    while $j < w_3 + 1$ **do**
        while $i < w_2$ **do**
            $T_{pr}^{P}(i+1,j,k) = [2]T_{pr}^{P}(i,j,k)$
            $i \leftarrow i+1$
        $j \leftarrow j+1$
    $k \leftarrow k+1$
return T_{pr}^{P}

Algorithm 3: Conversion to MBNS

Input : m, an integer, such that $0 \le m < 2^{w_2}3^{w_3}5^{w_5}$ for a given window length w_1, w_2, w_3 for 2, 3 and 5 and T^P

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

1: $i \leftarrow 1$
2: $s_i \leftarrow 1$
3: **while** $m > 0$ **do**
4: define $X = 2^{b_i}3^{t_i}5^{q_i}$, the best approximation of m with $0 \le b_i \le w_2$, $0 \le t_i \le w_3$ and $0 \le q_i \le w_5$. if there are two choices, choose nearest integer smaller to m.
5: let $A[i] \leftarrow (s_i, b_i, t_i, q_i)$
6: **if** $m < X$ **then**
7: $s_{i+1} \leftarrow -s_i$.
8: $m \leftarrow |m - X|$.
9: $i \leftarrow i + 1$.
10: **return** A.

Algorithm 4: To find $M_j's$.

Input : an integer n such that $0 \le n < (2^{w_2}3^{w_3}5^{w_5})^{\rho}$ for a given window length w_2, w_3, w_5 for 2, 3 and 5 resp. and no. of partition ρ.

Output : a seq. of $(M_i)_{i>0}$ such that $n = \sum_{i=1}^{\rho-1} M_{\rho-i}(2^{w_2}3^{w_3}5^{w_5})^{\rho-i}$, where $0 \le M_i < 2^{w_2}3^{w_3}5^{w_5}$ for all $0 \le i < \rho - 1$.

1: $i \leftarrow 1$
2: $R \leftarrow n$
3: $X \leftarrow (2^{w_2}3^{w_3}5^{w_5})^{\rho-1}$
4: **while** $i \le \rho$ **do**
5: $M_{\rho-i} \leftarrow \lfloor R/X \rfloor$
6: $R' \leftarrow R - M_{\rho-i}X$
7: $X \leftarrow X/2^{w_2}3^{w_3}5^{w_5}$
8: $R \leftarrow R'$
9: $i \leftarrow i + 1$
10: $A[\rho - i] \leftarrow M_{\rho-i}$
11: **return** A

Algorithm 5: calculation of $[m]P$

Input : an integer m such that $0 \leq m < 2^{w_2}3^{w_3}5^{w_5}$, a point P on an elliptic curve E and T_{pr}^{P}.

Output : $[m]P$.

$A \leftarrow$ **Algorithm 3**(m, w_2, w_3, w_5)
$L \leftarrow length(A)$
$P \leftarrow O$ (point at infinity on elliptic curve E)
$i \leftarrow 1$
while $i \leq L$ **do**
$(s_i, b_i, p_i, q_i) \leftarrow A[i]$
$P \leftarrow P + s_i T_{pr}^{P}(b_i, p_i, q_i)$
$i \leftarrow i + 1$
return P

Algorithm 6: calculation of $[n]P$

Input : an integer n such that $0 \leq n < (2^{w_2}3^{w_3}5^{w_5})^{\rho}$, a point P on an elliptic curve E, no. of partitions ρ and T_{pr}^{P}.

Output : $[n]P$

$A \leftarrow$ **Algorithm 4**(n, w_2, w_3, w_5, ρ)
$P \leftarrow O$ (point at infinity on elliptic curve E)
$i \leftarrow 1$
while $i \leq (\rho - 1)$ **do**
$Q \leftarrow$ **Algorithm 5**$(A[\rho - i], w_2, w_3, w_5, P, T_{pr}^{P})$
$P \leftarrow P + Q$
$P \leftarrow [5^{w_5}]P$
$P \leftarrow [3^{w_3}]P$
$P \leftarrow [2^{w_2}]P$
$i \leftarrow i + 1$
return P

Table 3.2: Average number of terms in a given window and partions for different values of w_2, w_3, w_5.

w_2	w_3	w_5	Average no. of terms	no. of partitions
0	0	0	0.000000	∞
0	0	1	1.600000	70
0	0	2	2.880000	35
0	1	0	1.000000	102
0	1	1	1.933333	42
0	1	2	2.760000	26
0	2	0	1.777778	51
0	2	1	2.488889	30
0	2	2	3.164444	21
0	3	0	2.481482	34
0	3	1	2.940741	23
0	3	2	3.583704	18
1	0	0	0.500000	161
1	0	1	1.500000	49
1	0	2	2.340000	29
1	1	0	1.166667	63
1	1	1	1.833333	33
1	1	2	2.486667	23
1	2	0	1.722222	39
1	2	1	2.233333	25
1	2	2	2.775556	19
1	3	0	2.240741	28
1	3	1	2.640741	20
1	3	2	3.123704	16
2	0	0	1.000000	81
2	0	1	1.750000	38
2	0	2	2.460000	25
2	1	0	1.416667	45
2	1	1	1.983333	28
2	1	2	2.536667	20
2	2	0	1.888889	32
2	2	1	2.294445	22
2	2	2	2.786667	17
2	3	0	2.268518	24
2	3	1	2.642593	18
2	3	2	3.084074	15

w_2	w_3	w_5	Average no. of terms	no. of partitions
3	0	0	1.375000	54
3	0	1	2.025000	31
3	0	2	2.675000	22
3	1	0	1.708333	36
3	1	1	2.175000	24
3	1	2	2.651667	18
3	2	0	2.097222	27
3	2	1	2.452778	19
3	2	2	2.893889	15
3	3	0	2.458333	21
3	3	1	2.758333	16
3	3	2	3.167963	13
4	0	0	1.750000	41
4	0	1	2.300000	26
4	0	2	2.905000	19
4	1	0	1.979167	29
4	1	1	2.354167	21
4	1	2	2.842500	16
4	2	0	2.333333	23
4	2	1	2.620833	17
4	2	2	3.043333	14
4	3	0	2.668982	19
4	3	1	2.915278	15
4	3	2	3.297500	13
5	0	0	2.093750	33
5	0	1	2.568750	22
5	0	2	3.137500	17
5	1	0	2.218750	25
5	1	1	2.568750	19
5	1	2	3.030417	15
5	2	0	2.552083	20
5	2	1	2.811805	16
5	2	2	3.207083	13
5	3	0	2.878472	17
5	3	1	3.077546	14
5	3	2	3.439306	12

Table 3.3: Costs of elliptic curve scalar multiplication for 160-bit multipliers using affine coordinates ($\mathbb{F}_{2^m} - cost$) taking $(w_2+1)(w_3+1)(w_5+1)$ number of storage points. $[I]/[M] = 8$(assuming square is free).

w_2	w_3	w_5	# storage	inverse $[I]$	multiplication $[M]$	$\approx [M]$
0	0	0	0	-	-	-
0	0	1	2	180.000000	1119.000000	2559.000000
0	0	2	3	167.800003	1083.599976	2426.000000
0	1	0	2	202.000000	909.000000	2525.000000
0	1	1	4	162.199982	980.399963	2277.999756
0	1	2	6	145.759995	966.520020	2132.600098
0	2	0	3	189.666687	879.333374	2396.666992
0	2	1	6	160.666672	930.333313	2215.666748
0	2	2	9	145.453323	930.906616	2094.533203
0	3	0	4	182.370392	859.740784	2318.703857
0	3	1	8	154.637039	881.274109	2118.370361
0	3	2	12	148.506668	926.013367	2114.066650
1	0	0	2	239.500000	479.000000	2395.000000
1	0	1	4	168.500000	865.000000	2213.000000
1	0	2	6	150.860001	917.719971	2124.600098
1	1	0	4	196.500015	703.000061	2275.000244
1	1	1	8	155.499985	823.000000	2067.000000
1	1	2	12	144.193344	882.386658	2035.933350
1	2	0	6	180.166656	740.333313	2181.666504
1	2	1	12	150.833328	805.666626	2012.333252
1	2	2	18	141.735565	859.471130	1993.355713
1	3	0	8	169.740753	744.481506	2102.407471
1	3	1	16	146.814819	787.629639	1962.148193
1	3	2	24	138.979263	832.958496	1944.792603
2	0	0	3	160.000000	640.000000	1920.000000
2	0	1	6	139.500000	834.000000	1950.000000
2	0	2	9	132.500000	889.000000	1949.000000
2	1	0	6	150.750015	697.500000	1903.500122
2	1	1	12	135.533325	811.066650	1895.333252
2	1	2	18	125.733345	840.466675	1846.333496
2	2	0	9	152.444443	738.888916	1958.444458
2	2	1	18	133.477783	791.955566	1859.777832
2	2	2	27	126.373344	828.746704	1839.733398
2	3	0	12	145.444427	727.888855	1891.444336
2	3	1	24	131.566666	773.133362	1825.666748
2	3	2	36	129.261108	832.522217	1866.611084

w_2	w_3	w_5	# storage	inverse $[I]$	multiplication $[M]$	$\approx [M]$
3	0	0	4	126.250000	676.500000	1686.500000
3	0	1	8	121.775002	813.549988	1787.750000
3	0	2	12	120.849998	871.700012	1838.500000
3	1	0	8	130.499985	716.000000	1759.999878
3	1	1	16	120.199997	792.400024	1754.000000
3	1	2	24	114.730011	824.460022	1742.300049
3	2	0	12	133.625000	735.250000	1804.250000
3	2	1	24	117.602783	757.205566	1698.027832
3	2	2	36	112.408333	784.816650	1684.083252
3	3	0	16	130.625000	721.250000	1766.250000
3	3	1	32	118.133331	746.266663	1691.333252
3	3	2	48	112.183517	764.367065	1661.835205
4	0	0	5	110.750000	701.500000	1587.500000
4	0	1	10	108.799995	792.599976	1663.000000
4	0	2	15	108.195000	828.390015	1693.949951
4	1	0	10	112.395844	700.791687	1599.958496
4	1	1	20	108.437508	776.875000	1644.375000
4	1	2	30	104.479996	793.960022	1629.800049
4	2	0	15	118.666656	721.333313	1670.666504
4	2	1	30	107.554161	743.108337	1603.541626
4	2	2	45	106.606659	785.213318	1638.066650
4	3	0	20	121.710655	729.421326	1703.106567
4	3	1	40	112.729172	757.458313	1659.291748
4	3	2	60	113.867500	815.734985	1726.675049
5	0	0	6	100.093750	712.187500	1512.937500
5	0	1	12	97.512497	762.025024	1542.125000
5	0	2	18	100.337502	808.674988	1611.375000
5	1	0	12	102.468750	708.937500	1528.687500
5	1	1	24	101.806252	779.612488	1594.062500
5	1	2	36	100.456253	802.912537	1606.562500
5	2	0	18	107.041656	708.083313	1564.416504
5	2	1	36	103.988876	762.977783	1594.888794
5	2	2	54	100.692078	777.384155	1582.920776
5	3	0	24	111.934029	719.868042	1615.340332
5	3	1	48	107.085640	760.171265	1616.856445
5	3	2	72	106.271675	795.543335	1645.716797

Table 3.4: Costs of elliptic curve scalar multiplication for 160-bit multipliers using affine coordinates ($\mathbb{F}_{2^m} - cost$) taking $2^{w_2}3^{w_3}5^{w_5}$ number of storage points. $[I]/[M] = 8$(assuming square is free).

w_2	w_3	w_5	# storage	inverse $[I]$	multiplication $[M]$	$\approx [M]$
0	0	0	0	-	-	-
0	0	1	5	124.199997	1007.400024	2001.000000
0	0	2	25	100.639999	949.280029	1754.400024
0	1	0	3	168.333328	841.666687	2188.333252
0	1	1	15	120.266670	896.533325	1858.666748
0	1	2	75	99.666664	874.333313	1671.666626
0	2	0	9	144.444443	788.888916	1944.444458
0	2	1	45	115.355553	839.711121	1762.555542
0	2	2	225	99.911110	839.822205	1639.111084
0	3	0	27	130.777771	756.555542	1802.777710
0	3	1	135	109.837036	791.674072	1670.370361
0	3	2	675	101.974815	832.949646	1648.748169
1	0	0	2	240.000000	480.000000	2400.000000
1	0	1	10	139.199997	806.400024	1920.000000
1	0	2	50	111.440002	838.880005	1730.400024
1	1	0	6	175.666672	661.333313	2066.666748
1	1	1	30	126.933334	765.866638	1781.333252
1	1	2	150	109.853333	813.706665	1692.533325
1	2	0	18	149.888885	679.777771	1878.888916
1	2	1	90	119.733330	743.466675	1701.333252
1	2	2	450	107.959999	791.919983	1655.599976
1	3	0	54	134.500000	674.000000	1750.000000
1	3	1	270	113.929626	721.859253	1633.296265
1	3	2	1350	104.988892	764.977783	1604.888916
2	0	0	4	140.000000	600.000000	1720.000000
2	0	1	20	109.150002	773.299988	1646.500000
2	0	2	100	95.760002	815.520020	1581.600098
2	1	0	12	128.333328	652.666687	1679.333252
2	1	1	60	107.550003	755.099976	1615.500000
2	1	2	300	94.936668	778.873352	1538.366699
2	2	0	36	123.138885	680.277771	1665.388916
2	2	1	180	104.883331	734.766663	1573.833252
2	2	2	900	95.982224	767.964417	1535.822266
2	3	0	108	114.787041	666.574097	1584.870361
2	3	1	540	101.968521	713.937012	1529.685181
2	3	2	2700	97.994812	769.989624	1553.948120

w_2	w_3	w_5	# storage	inverse [I]	multiplication [M]	≈ [M]
3	0	0	8	99.375000	622.750000	1417.750000
3	0	1	40	89.250000	748.500000	1462.500000
3	0	2	200	83.894997	797.789978	1468.949951
3	1	0	24	103.541664	662.083313	1490.416626
3	1	1	120	91.808334	735.616638	1470.083252
3	1	2	600	84.971664	764.943359	1444.716675
3	2	0	72	103.638885	675.277771	1504.388916
3	2	1	360	89.949997	701.900024	1421.500000
3	2	2	1800	83.992226	727.984436	1399.922241
3	3	0	216	99.907410	659.814819	1459.074097
3	3	1	1080	89.986115	689.972229	1409.861084
3	3	2	5400	83.997780	707.995544	1379.977783
4	0	0	16	77.500000	635.000000	1255.000000
4	0	1	80	74.687500	724.375000	1321.875000
4	0	2	400	71.955002	755.909973	1331.550049
4	1	0	48	83.416664	642.833313	1310.166626
4	1	1	240	79.916664	719.833313	1359.166626
4	1	2	1200	74.987503	734.974976	1334.875000
4	2	0	144	87.847221	659.694458	1362.472168
4	2	1	720	79.977776	687.955566	1327.777832
4	2	2	3600	77.996391	727.992798	1351.963867
4	3	0	432	89.958336	665.916687	1385.583374
4	3	1	2160	83.993515	699.987061	1371.935181
4	3	2	10800	83.998886	755.997803	1427.988892
5	0	0	32	63.000000	638.000000	1142.000000
5	0	1	160	62.868752	692.737488	1195.687500
5	0	2	800	63.980000	735.960022	1247.800049
5	1	0	96	71.750000	647.500000	1221.500000
5	1	1	480	71.962502	719.924988	1295.625000
5	1	2	2400	69.994164	741.988342	1301.941650
5	2	0	288	75.934029	645.868042	1253.340332
5	2	1	1440	74.989586	704.979187	1304.895874
5	2	2	7200	71.998337	719.996643	1295.983398
5	3	0	864	79.981483	655.962952	1295.814819
5	3	1	4320	77.996994	701.993958	1325.969971
5	3	2	21600	76.999489	736.998962	1352.994873

Table 3.5: Costs of elliptic curve scalar multiplication for 160-bit multipliers using jacobian coordinates ($\mathbb{F}_{p^m}-cost$) taking $(w_2+1)(w_3+1)(w_5+1)$ number of storage points. $[S]/[M] = 0.8$

w_2	w_3	w_5	# storage	square $[S]$	multiplication $[M]$	$\approx [M]$
0	0	0	0	-	-	-
0	0	1	2	954.000000	1923.000000	2686.199951
0	0	2	3	911.400024	1818.400024	2547.520020
0	1	0	2	909.000000	1818.000000	2545.199951
0	1	1	4	855.599976	1666.599854	2351.079834
0	1	2	6	812.280029	1566.079956	2215.904053
0	2	0	3	769.000061	1767.333496	2382.533447
0	2	1	6	772.000000	1633.333374	2250.933350
0	2	2	9	756.359985	1543.626587	2148.714600
0	3	0	4	712.111145	1722.963135	2292.652100
0	3	1	8	705.911133	1567.096313	2131.825195
0	3	2	12	734.520020	1562.053345	2149.669434
1	0	0	2	1198.500000	1276.000000	2234.800049
1	0	1	4	937.500000	1492.000000	2242.000000
1	0	2	6	872.580017	1486.880005	2184.944092
1	1	0	4	961.500061	1448.000122	2217.200195
1	1	1	8	850.499939	1403.999878	2084.399902
1	1	2	12	828.580017	1417.546753	2080.410889
1	2	0	6	806.500000	1479.333252	2124.533203
1	2	1	12	764.500000	1398.666626	2010.266602
1	2	2	18	767.206726	1403.884521	2017.649902
1	3	0	8	725.222229	1465.926025	2046.103760
1	3	1	16	706.444458	1383.518555	1948.674072
1	3	2	24	716.937805	1381.834106	1955.384399
2	0	0	3	1040.000000	1280.000000	2112.000000
2	0	1	6	899.500000	1375.000000	2094.600098
2	0	2	9	853.500000	1396.000000	2078.800049
2	1	0	6	892.250061	1294.000122	2007.800171
2	1	1	12	838.599976	1327.266602	1998.146606
2	1	2	18	795.200012	1309.866699	1946.026733
2	2	0	9	798.333313	1374.555542	2013.222168
2	2	1	18	757.433350	1319.822266	1925.768921
2	2	2	27	747.119995	1314.986694	1912.682739
2	3	0	12	712.333313	1347.555420	1917.422119
2	3	1	24	700.700012	1307.533325	1868.093384
2	3	2	36	723.783325	1342.088867	1921.115479

w_2	w_3	w_5	# storage	square [S]	multiplication [M]	$\approx$ [M]
3	0	0	4	961.750000	1222.000000	1991.400024
3	0	1	8	875.325012	1304.200073	2004.460083
3	0	2	12	845.549988	1344.800049	2021.239990
3	1	0	8	881.499939	1253.999878	1959.199829
3	1	1	16	820.599976	1260.599976	1917.079956
3	1	2	24	786.190002	1257.840088	1886.792114
3	2	0	12	790.875000	1303.000000	1935.699951
3	2	1	24	730.808350	1228.822266	1813.468994
3	2	2	36	715.224976	1221.266724	1793.446655
3	3	0	16	711.875000	1285.000000	1854.500000
3	3	1	32	684.399963	1230.066650	1777.586670
3	3	2	48	672.550537	1209.468140	1747.508545
4	0	0	5	932.250000	1206.000000	1951.800049
4	0	1	10	851.400024	1245.400024	1926.520020
4	0	2	15	810.585022	1261.559937	1910.027954
4	1	0	10	841.187500	1179.166748	1852.116699
4	1	1	20	805.312500	1207.500000	1851.750000
4	1	2	30	763.440002	1195.839966	1806.591919
4	2	0	15	774.000000	1235.333252	1854.533203
4	2	1	30	722.662476	1180.433228	1758.563232
4	2	2	45	722.820007	1203.853271	1782.109253
4	3	0	20	725.131958	1261.685303	1841.790894
4	3	1	40	702.187500	1223.833374	1785.583374
4	3	2	60	725.602478	1270.939941	1851.421875
5	0	0	6	908.281250	1184.750000	1911.375000
5	0	1	12	817.537476	1179.099976	1833.130005
5	0	2	18	797.012512	1218.699951	1856.309937
5	1	0	12	835.406250	1155.750000	1824.074951
5	1	1	24	809.418762	1192.449951	1839.984985
5	1	2	36	777.368774	1195.650024	1817.545044
5	2	0	18	758.125000	1179.333252	1785.833252
5	2	1	36	746.966614	1191.911011	1789.484253
5	2	2	54	722.076233	1177.536621	1755.197632
5	3	0	24	719.802063	1215.472168	1791.313843
5	3	1	48	711.256897	1207.685181	1776.690674
5	3	2	72	714.815002	1224.173340	1796.025391

Table 3.6: Costs of elliptic curve scalar multiplication for 160-bit multipliers using jacobian coordinates ($\mathbb{F}_{p^m} - cost$) taking $2^{w_2}3^{w_3}5^{w_5}$ number of storage points. $[S]/[M] = 0.8$

w_2	w_3	w_5	# storage	square $[S]$	multiplication $[M]$	$\approx [M]$
0	0	0	0	-	-	-
0	0	1	5	786.599976	1476.599976	2105.879883
0	0	2	25	709.919983	1281.119995	1849.056030
0	1	0	3	808.000000	1548.666626	2195.066650
0	1	1	15	729.799988	1331.133301	1914.973267
0	1	2	75	674.000000	1197.333374	1736.533325
0	2	0	9	633.333313	1405.555542	1912.222168
0	2	1	45	636.066650	1270.844482	1779.697754
0	2	2	225	619.733337	1179.288940	1675.075562
0	3	0	27	557.333313	1310.222168	1756.088867
0	3	1	135	571.511108	1208.696289	1665.905151
0	3	2	675	594.924438	1189.798462	1665.738037
1	0	0	2	1200.000000	1280.000000	2240.000000
1	0	1	10	849.599976	1257.599976	1937.279907
1	0	2	50	754.320007	1171.520020	1774.976074
1	1	0	6	899.000000	1281.333374	2000.533325
1	1	1	30	764.799988	1175.466675	1787.306641
1	1	2	150	725.559998	1142.826660	1723.274658
1	2	0	18	715.666687	1237.111084	1809.644409
1	2	1	90	671.200012	1149.866699	1686.826660
1	2	2	450	665.880005	1133.680054	1666.384033
1	3	0	54	619.500000	1184.000000	1679.599976
1	3	1	270	607.788879	1120.437012	1606.668091
1	3	2	1350	614.966675	1109.911133	1601.884521
2	0	0	4	980.000000	1120.000000	1904.000000
2	0	1	20	808.450012	1132.199951	1778.959961
2	0	2	100	743.280029	1102.079956	1696.703979
2	1	0	12	825.000000	1114.666626	1774.666626
2	1	1	60	754.650024	1103.400024	1707.119995
2	1	2	300	702.809998	1063.493286	1625.741333
2	2	0	36	710.416687	1140.111084	1708.444458
2	2	1	180	671.650024	1091.066650	1628.386719
2	2	2	900	655.946655	1071.857788	1596.615112
2	3	0	108	620.361084	1102.296265	1598.585083
2	3	1	540	611.905579	1070.748169	1560.272583
2	3	2	2700	629.984436	1091.958496	1595.946045

w_2	w_3	w_5	# storage	square $[S]$	multiplication $[M]$	$\approx [M]$
3	0	0	8	881.125000	1007.000000	1711.900024
3	0	1	40	777.750000	1044.000000	1666.199951
3	0	2	200	734.684998	1049.160034	1636.908081
3	1	0	24	800.625000	1038.333374	1678.833374
3	1	1	120	735.424988	1033.466675	1621.806641
3	1	2	600	696.914978	1019.773315	1577.305298
3	2	0	72	700.916687	1063.111084	1623.844482
3	2	1	360	647.849976	1007.599976	1525.880005
3	2	2	1800	629.976685	993.937805	1497.919189
3	3	0	216	619.722229	1039.259277	1535.037109
3	3	1	1080	599.958313	1004.888916	1484.855591
3	3	2	5400	587.993347	983.982239	1454.376953
4	0	0	16	832.500000	940.000000	1606.000000
4	0	1	80	749.062500	972.500000	1571.750000
4	0	2	400	701.864990	971.640015	1533.131958
4	1	0	48	754.250000	947.333313	1550.733276
4	1	1	240	719.750000	979.333313	1555.133301
4	1	2	1200	674.962524	959.900024	1499.869995
4	2	0	144	681.541687	988.777771	1534.011108
4	2	1	720	639.933350	959.822205	1471.768921
4	2	2	3600	636.989197	974.971130	1484.562500
4	3	0	432	629.875000	1007.666687	1511.566650
4	3	1	2160	615.980530	993.948120	1486.732544
4	3	2	10800	635.996643	1031.991089	1540.788452
5	0	0	32	797.000000	888.000000	1525.599976
5	0	1	160	713.606262	901.950012	1472.835083
5	0	2	800	687.940002	927.840027	1478.192017
5	1	0	96	743.250000	910.000000	1504.599976
5	1	1	480	719.887512	953.700012	1529.609985
5	1	2	2400	685.982483	951.953308	1500.739258
5	2	0	288	664.802063	930.472229	1462.313843
5	2	1	1440	659.968750	959.916687	1487.891724
5	2	2	7200	635.994995	947.986694	1456.782715
5	3	0	864	623.944458	959.851868	1459.007446
5	3	1	4320	623.990967	974.975952	1474.168701
5	3	2	21600	626.998474	989.995911	1491.594727

Chapter 4

Conclusion and Future scope

In this report, we have presented a new method called window based scalar multiplication method for computing scalar multiplication using MBNS representation of the scalar. We are using greedy algorithm to find an MBNS representation of a scalar m, but there is a slight modification from previous algorithms; it is sufficient to find the representation in a window due to suggested representation of n which results in a much smaller static table size. If some precomputed points are allowed to store then the complexity turns out to be almost equal to earlier proposed best methods, and more storage of precomputed points give better than all.

If we look at table 3.3 and table 3.4, we conclude that single base representaion performs better (at $(w_2, w_3, w_5) = (5, 0, 0)$) than DBNS and MBNS, while table 3.5and table 3.6 gives better computation results (at $(w_2, w_3, w_5) = (3, 3, 2)$)in MBNS. The reason is obvious; existing method for efficient calculation of $[2^w]P$ in affine cordinates gives better result in single base representation, i.e., with base 2, while in jacobian coordinates, efficient algoritm for computing $[2^w]P$, $[3^w]P$ and $[5]P$ gives better result. It clearly suggests that if there be an efficient algorithm for computing $[2^w]P$, $[3^w]P$ and $[5^w]P$ in both affine and projective coordinates, compuational complexity will be much reduced in scalar multiplication.

Calculation of average no. of inverse, square and multiplication requires ρ (no. of partitions), t (average no. of terms using MBNS for a given m lying in a window) and window length (w_2, w_3, w_5). For a given r no of bits, equation (3.6) gives a relation between ρ and w_2, w_3, w_3, but there doesn't exist any mathematical way which gives a perfect relation between ρ and t or w_2, w_3, w_5 and t. Any such relation will help in finding an optimal window length which takes less computation and requires less storage of precomputed points.

Bibliography

[1] Ian Blake, Gadiel Seroussi and Nigel Smart. Elliptic Curves in Cryptography, *London Mathematical Society Lecture Note Series 265*, pages 57-78, Cambridge University Press, 1999.

[2] M. Ciet, K. Lauter, M. Joye and P. L. Montgomery. Trading inversions for multiplications in elliptic curve cryptography, In *Designs, Codes and Cryptography*, 39(2):189-206, 2006.

[3] H. Cohen, A. Miya ji, and T. Ono. *Efficient Elliptic Curve Exponentiation Using Mixed coordinates*, In ASIACRYPT98, LNCS 1514, pp. 51-65, Springer-Verlag, 1998.

[4] R. Dahab and J. Lopez, An Improvement of Guajardo-Paar Method for Multiplication on non-supersingular Elliptic Curves. In Proceedings of the XVIII International Conference of the Chilean Computer Science Society (SCCC98), IEEE CS Press, November 12-14, Antofagasta, Chile, pp.91-95, 1998.

[5] 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 5979. Springer-Verlag, 2005.

[6] Pradeep Kumar Mishra and Vassil Dimitrov. Efficient Quintuple Formulas for Elliptic Curves and Efficient Scalar Multiplication Using Multibase Number Representation, 20070410:061728, 2007.

[7] K. Itoh, M. Takenaka, N. Torii, S. Temma, and Y. Kurihara. Fast implementation of public-key cryptography on a DSP TMS320C6201. In C. K. Ko and C. Paar, editors, *Cryptographic Hardware and Embedded Systems CHES 99*, volume 1717 of *Lecture Notes in Computer Science*, pages 6172. Springer-Verlag, 1999.

[8] Lawrence C. Washington. *ELLIPTIC CURVES Number theory and Cryptography*, Chapman and Hall/CRC, 2003.

[9] Douglas R. Stinson. *CRYPTOGRAPHY Theory and Practice*, Chapman and Hall/CRC, second edition, 2002.