Journal of Theoretical and Applied Information Technology
15th February 2012. Vol. 36 No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

AN ELLIPTIC CURVE ARITHMETIC IN THE NORMAL BASIS OF GF(2^5) TO USE IN ECC

A.R.RISHIVARMAN B. PARTHASARATHY M.THIAGARAJAN
1. Assitant Professor, Mathematics,Dr.Pauls Engineering college,Villupuram, TN,India
2. Professor ,Mathematics, Mailam Engineering College,Villupuram,TN,India
3. Professor ,Mathematics ,SASTRA University,Tanjore,TN,India

Email: [1]rishi_130777@yahoo.co.in, [2]mlampmlc@gmail.com, [3]m_thiyagarajan@yahoo.com

ABSTRACT

Since the introduction of public-key cryptography by Diffe and Hellman in 1976, the potential for the use of the discrete logarithm problem in public-key cryptosystems has been recognized. Although the discrete logarithm problem as first employed by Diffe and Hellman was defined explicitly as the problem of finding logarithms with respect to a generator in the multiplicative group of the integers module a prime, this idea can be extended to arbitrary groups and in particular, to elliptic curve groups. The resulting public – key systems provide relatively small block size, high speed, and high security. This paper explores an efficient performance of GF(2^5) in the normal basis representation and an elliptic curve cryptosystems using a non-super singular curve over the field; so this scheme is of less computation cost which is valuable in applications with limited memory, communications bandwidth or computing power.

Key words: *Secret Sharing, Elliptic Curve Cryptography (ECC), GF(2^5), Irreducible Polynomial, ECDLP.*

1. INTRODUCTION:

There are three families of public–key algorithms that have considerable significance in current data security practice. They are integer factorization, discrete logarithm, and elliptic curve based schemes [2]-[3]. Integer factorization based schemes such as RSA [4] and discrete logarithm based schemes such as Diffe -Hellman [5] provide intuitive ways of implementation. However both methods admit of sub-exponential algorithm of cryptanalysis [7]. In this regard elliptic curve cryptography, first introduced Koblitz [2] and Miller [3] may be the most cryptographic method available [6]-[8]. The best current brute force algorithm for cryptanalysis of ECC require O($n^{1/2}$) steps where n is the order of the additive group. For example , using the best current brute force algorithms ECC with a key size of 173 bits provides the same level of cryptographic security as RSA with a key size of 1024 bits. This results in smaller system parameters band width savings, faster implementations and lower power consumptions. In addition, elliptic curve over finite fields offer an inexhaustible supply finite abelian groups, thus allowing more flexible fields selections than conventional discrete logarithm schemes. Because of these advantages ECC has attracted extensive attention in recent years [9]-[13]. This paper explores an efficient performance of GF(2^5) in the normal basis representation and an elliptic curve cryptosystems using a non-super singular curve E: $y^2 + xy = x^3 + x^2 + 1$ over the field GF(2^5) , an irreducible polynomial taken for construction of the field is $f(x) = x^5 + x^2 + 1$.

2. DATA TYPES AND CONVERSIONS:

The schemes specified in this document involve operations using several different data types. This section lists the different data types and describes how to convert one data type to another

.Five data types are employed in this document: three types associated with elliptic curve arithmetic -integers, field elements, and elliptic curve points - as well as octet strings which are used to communicate and store information, and bit strings which are used by some of the primitives. Throughout this document the above data types are

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

regarded as abstract data types consisting of distinct sets of elements - so that, for example, an octet string is regarded as distinct from a bit string. This formalism helps to clarify the requirements placed on implementations and helps avoid subtle coding errors. Frequently it is necessary to convert one of the data types into another - for example to represent an elliptic curve point as an octet string. The remainder of this section is devoted to describing how the necessary conversions should be performed.

2.1 BitString-to-OctetString Conversion

Bit strings should be converted to octet strings as described in this section. Informally the idea is to pad the bit string with 0's on the left to make its length a multiple of 8, then chop the result up into octets. Formally the conversion routine is specified as follows.

Input: A bit string *B* of length *blen* bits.

Output: An octet string *M* of length $mlen = \left\lceil \frac{blen}{8} \right\rceil$ octets.

Actions: Convert the bit string $B = B_0 B_1 \ldots B_{blen-1}$ to an octet string $M = M_0 M_1 \ldots M_{mlen-1}$ as follows:

1.For $0 \leq i \leq mlen - 1$, let:

$$M_i = B_{8i} B_{8i+1} \ldots B_{8i+7}$$

2. Let M_0 have its leftmost 8(*mlen*)-*blen* bits set to 0, and its rightmost (8-(8(*mlen*)-*blen*) bits set to $B_0 B_1 \ldots B_{8-8(mlen)+blen-1}$.

3. Output *M*.

2.2 OctetString-to-BitString Conversion

Octet strings should be converted to bit strings as described in this section. Informally the idea is simply to view the octet string as a bit string instead. Formally the conversion routine is specified as follows:

Input: An octet string *M* of length *mlen* octets.

Output: A bit string *B* of length *blen*= 8(*mlen*) bits.

Actions: Convert the octet string $M == M_0 M_1 \ldots M_{mlen-1}$ to a bit string $B = B_0 B_1 \ldots B_{blen-1}$ as follows:

1.For $0 \leq i \leq mlen - 1$, set:

$$B_{8i} B_{8i+1} B_{8i+7} = M_i$$

2. Output *B*.

2.3.EllipticCurvePoint-to-OctetString Conversion

Elliptic curve points should be converted to octet strings as described in this section. Informally, if point compression is being used, the idea is that the compressed *y*-coordinate is placed in the leftmost octet of the octet string along with an indication that point compression is on, and the *x*-coordinate is placed in the remainder of the octet string; otherwise if point compression is off, the leftmost octet indicates that point compression is off, and remainder of the octet string contains the *x*-coordinate followed by the *y*-coordinate. Formally the conversion routine is specified as follows:

Setup: Decide whether or not to represent points using point compression.

Input: A point *P* on an elliptic curve over F*q* defined by the field elements *a,b*.

Output: An octet string *M* of length *mlen* octets where *mlen*= 1 if $P = O$, *mlen*=[(log2 *q*) /8]+1 if $P \neq O$ and point compression is used, and *mlen* =2[(log$_2$ *q*)/8]+1 if $P \neq O$ and point compression is not used.

Actions: Convert *P* to an octet string $M = M_0 M_1 \ldots M_{mlen-1}$ as follows:

1. If P= *O*, output $M = 00_{16}$.

2. If $P = (x_P \; y_P) \neq O$ and point compression is being used, proceed as follows:

2.1. Convert the field element x_P to an octet string *X* of length[(log$_2$q / 8)] octets using the conversion routine specified in Section 2.3.5.

2.2. Derive from y_P a single bit $\bar{y}_P$ as follows (this allows the *y*-coordinate to be represented compactly using a single bit):

2.2.1. If q= *p* is an odd prime, set $\bar{y}_P = y_P \pmod 2$.

2.2.2. If $q = 2^m$ set $\bar{y}_P = 0$ *if* $x_P = 0$ otherwise compute

$z = z_{m-1} x^{m-1} + \cdots + z_1 x + z_0$ such that $z = y_P x_P^{-1}$ and set $\bar{y}_P = z_0$

2.3. If $\bar{y}_P = 0$ assign the value 02_{16} to the single octet *Y*. If $\bar{y}_P = 1$, assign the value 03_{16} to the single octet *Y*.

2.4. Output $M = Y \| X$.

3. If $P\ (x_P, y_P) \neq O$ and point compression is not being used, proceed as follows:

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

3.1. Convert the field element x_P to an octet string X of length $[(\log 2\ q)/8]$ octets using the conversion routine specified in Section 4.5.

3.2. Convert the field element y_P to an octet string Y of length $[(\log 2\ q)/8]$ octets using the conversion routine specified in Section 4.5.

3.3. Output $M=04_{16}||X||Y$.

2.4 OctetString-to-EllipticCurvePoint Conversion

Octet strings should be converted to elliptic curve points as described in this section. Informally the idea is that, if the octet string represents a compressed point, the compressed y-coordinate is recovered from the leftmost octet, the x-coordinate is recovered from the remainder of the octet string, and then the point compression process is reversed; otherwise the leftmost octet of the octet string is removed, the x-coordinate is recovered from the left half of the remaining octet string, and the y-coordinate is recovered from the right half of the remaining octet string. Formally the conversion routine is specified as follows:

Input: An elliptic curve over Fq defined by the field elements a,b , and an octet string M which is either the single octet 00_{16}, an octet string of length $mlen$=$[(\log 2\ q)/8]+1$, or an octet string of length $mlen$ =$2[(\log 2\ q)/8]+1$.

Output: An elliptic curve point P, or 'invalid'.

Actions: Convert M to an elliptic curve point P as follows:

1. If M =00_{16}, output P= O.

2. If M has length $[(\log 2\ q)/8]+1$ octets, proceed as follows:

2.1. Parse M=Y ||X as a single octet Y followed by $[(\log 2\ q)/8]$ octets X.

2.2. Convert X to a field element x_P of Fq using the conversion routine specified in Section 2.3.6.Output 'invalid' and stop if the routine outputs 'invalid'.

2.3. If Y= 02, set $\tilde{y}_P$ =0 and if Y =03, set $\tilde{y}_P$ = 1. Otherwise output 'invalid' and stop.

2.4. Derive from x_P and $\tilde{y}_P$ an elliptic curve point P=(x_P, y_P), where:

2.4.1. If q= p is an odd prime, compute the field element $\alpha = x_P^3 + a.x_P+b$ (mod p) and compute a square root β of α modulo p. Output 'invalid' and stop if there are no square roots of α modulo p, otherwise set $y_P=\beta$ if $\beta \equiv \tilde{y}_P \pmod 2$, and set $y_P = p-\beta$ if $\beta \not\equiv \tilde{y}_P \pmod 2$

2.4.2. If q= 2^m and x_P== 0, output $y_P=b^{2^{m-1}}$ in F_{2^m}

2.4.3. If q= 2^m and $x_P \neq 0$, compute the field element β x_P+a+bx_P^{-2} in F_{2^m}

and find an element $z = z_{m-1}x^{m-1} + \cdots + z_1x + z_0$ such that $z^2+z=\beta$ in F_{2^m}

Output 'invalid' and stop if no such z exists, otherwise set x_P = y_P z in F_{2^m} $z_0=\tilde{y}_P$and set y_P = $x_P(z+1)$ in F_{2^m} if $z_0 \not\equiv \tilde{y}_P$.

2.5. Output P=(x_P ,y_P).

3. If M has length $2[(\log_2 q)/8]+1$ octets, proceed as follows:

3.1. Parse $M = W \| X \| Y$ as a single octet W followed by $[(\log_2 q)/8]$ octets X followed by $[(\log_2 q)/8]$ octets Y.

3.2. Check that W= = 04_{16}. If $W \neq 04_{16}$, output 'invalid' and stop.

3.3. Convert X to a field element x_P of Fq using the conversion routine specified in Section 4.6.Output 'invalid' and stop if the routine outputs 'invalid'.

3.4. Convert Y to a field element y_P of Fq using the conversion routine specified in Section 4.6.Output 'invalid' and stop if the routine outputs 'invalid'.

3.5. Check that P=(x_P ,y_P) satisfies the defining equation of the elliptic curve.

3.6. Ou tput P=(x_P ,y_P).

2.5 FieldElement-to-OctetString Conversion

Field elements should be converted to octet strings as described in this section. Informally the idea is that, if the field is Fp, convert the integer to an octet string, and if the field is F_2^m, view the coefficients of the polynomial as a bit string with the highest degree term on the left and convert the bit string to an octet string. Formally the conversion routine is specified as follows:

Input: An element a of the field Fq.

Output: An octet string M of length $mlen$ =$[(\log_2 q)/8]$ octets.

Actions: Convert a to an octet string $M = M_0M_1 \ldots M_{mlen-1}$ as follows:

1. If q =p is an odd prime, then a is an integer in the interval[0,p-1]. Convert a to M using the conversion routine specified in Section 4.7. Output M.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

2. If $q=2^m$ then $a = a_{m-1}x^{m-1} + \cdots + a_1 x + a_0$ is a binary polynomial. Convert a to M as follows:

2.1. For $0 \leq i \leq mlen - 1$, let:

$M_i =$ $a_{7+8(mlen-1-i)}a_{6+8(mlen-1-i)} \cdots a_{8(mlen-1-i)}$

2.2. Let M_0 have its leftmost $8(mlen-)m$ bits set to 0, and its rightmost $8-8(mlen-m)$ bits set to $a_{m-1}a_{m-2} \cdots a_{8(mlen)-8}$.

2.3. Output M.

2.6 OctetString-to-FieldElement Conversion

Octet strings should be converted to field elements as described in this section. Informally the idea is that, if the field is Fp, convert the octet string to an integer, and if the field is F_{2^m}, use the bits of the octet string as the coefficients of the binary polynomial with the rightmost bit as the constant term. Formally the conversion routine is specified as follows:

Input: An indication of the field Fq used and an octet string M of length $mlen$ =[($\log_2 q$)/8] octets.

Output: An element a in Fq, or 'invalid'.

Actions: Convert $M = M_0 M_1 \ldots M_{mlen-1}$ with $M_i = M_i^7 M_i^6 \ldots M_i^1$ to a field element a as follows:

1. If $q = p$ is an odd prime, then a needs to be an integer in the interval [0,p-1][Convert M to an integer a using the conversion routine specified in Section4.8. Output 'invalid' and stop if a does not lie in the interval [0,p-1]], otherwise output a.

2. If $q = 2^m$, then a needs to be a binary polynomial of degree $m-1$ or less. Set the field element a to be $a = a_{m-1}x^{m-1} + \cdots + a_1 x + a_0$ with:

$a_i = M_{mlen-1-\lfloor i/8 \rfloor}^{7-i+8\lfloor i/8 \rfloor}$

Output 'invalid' and stop if the leftmost $8(mlen)-m$ bits of M_0 are not all 0, otherwise output a.

2.7 Integer-to-OctetString Conversion

Integers should be converted to octet strings as described in this section. Informally the idea is to represent the integer in binary then convert the resulting bit string to an octet string. Formally the conversion routine is specified as follows:

Input: A non-negative integer x together with the desired length $mlen$ of the octet string. It must be the case that: $2^{8(mlen)} > x$

Output: An octet string M of length $mlen$ octets.

Actions: Convert $x = x_{mlen}2^{8(mlen-1)} + x_{mlen-1}2^{8(mlen-2)} + \ldots + x_1 2^8 + x_0$ represented in base to an octet string $M = M_0 M_1 \ldots M_{mlen-1}$ as follows:

1. For $0 \leq i \leq mlen - 1$, set:
   $M_i = x_{mlen-1-i}$

2. Output M.

2.8 OctetString-to-Integer Conversion

Octet strings should be converted to integers as described in this section. Informally the idea is simply to view the octet string as the base 256 representation of the integer. Formally the conversion routine is specified as follows:

Input: An octet string M of length $mlen$ octets.

Output: An integer x.

Actions: Convert $M = M_0 M_1 \ldots M_{mlen-1}$ to an integer x as follows:

1. View Mi as an integer in the range [0,255] and set:
   $x = \sum_{i=0}^{mlen-1} 2^{8(mlen-1-i)} M_i$

2. Output x.

2.9 FieldElement-to-Integer Conversion

Field elements should be converted to integers as described in this section. Informally the idea is that,if the field is F p no conversion is required, and if the field is F_{2^m} Ffirst convert the binary polynomial to an octet string then convert the octet string to an integer. Formally the conversion routine is specified as follows:

Input: An element a of the field Fq.

Output: An integer x.

Actions: Convert the field element a to an integer x as follows:

1. If $q = p$ is an odd prime, then a must be an integer in the interval[[0,P− 1]]. Output $x = a$.

2. If $q = 2^m$ then a must be a binary polynomial of degree $m-1$,i.e.

$a = a_{m-1}x^{m-1} + \cdots + a_1 x + a_0$, Set:

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

$$x = \sum_{i=0}^{m-1} 2^i a_i$$

Output x.

3. GOLOIS FIELD ARITHMETIC: GF(2^5)

GF(2^5) can be viewed as a vector space of dimension 5 over the field GF(2). There are several bases known for GF(2^5) . The most common bases are polynomial bases and nonmal bases . with a polynomial basis , the field elements are represented by binary polynomials modulo an irreducible binary polynomial of degree 5 . given an irreducible polynomial

$$p(x) = x^5 + x^2 + 1$$

An element $A \in GF(2^5)$ is represented either as $A(\alpha) = \sum_{i=0}^{4} a_i \alpha^i$ or as $(a_4 a_3 a_2 a_1 a_0)$, where $a_i \in GF(2)$ and α , the root of $p(x)$ [28]-[29]. Here the basis is { 1, $\alpha^1 \alpha^2 \alpha^3 \alpha^4$}. It has been proved that there always exists a normal basis for the given finite field GF(2^5) which is of the form N={$\beta, \beta^2 \beta^4 ,.., \beta^{2^4}$} where β is aroot of the irreducible polynomial $p(x) = x^5 + x^2 + 1$ over GF(2) and elements of the set are linearly independent . we say that β generates the normal basis N , or β is a normal elements of $GF(2^5)$. βwill equal to $\alpha^i$for some i. Then every element $A \in GF(2^5)$is represented as

$$A(\beta) = \sum_{i=0}^{4} a_i \beta^{2^i},$$

Where $a_i \in$ GF(2). A field element can thus be represented in a bit vector of length 5. Hence we have;

S.NO	BIT STRING	POLYNOMIAL FORM	NORMAL FORM
1	00000	0	β^{32}
2	00001	1	β^{31}
3	00010	$a_1\alpha^1$	β^1
4	00100	$a_2\alpha^2$	β^2
5	01000	$a_3\alpha^3$	β^3
6	10000	$a_4\alpha^4$	β^4
7	00011	$a_2\alpha^1 + 1$	β^{18}
8	00101	$a_2\alpha^2 + 1$	β^5
9	01001	$a_3\alpha^3 + 1$	β^{29}
10	10001	$a_4\alpha^4 + 1$	β^{10}
11	11000	$a_4\alpha^4 + a_3\alpha^3$	β^{11}
12	10100	$a_4\alpha^4 + a_2\alpha^2$	β^7
13	10010	$a_4\alpha^4 + a_1\alpha^1$	β^{20}
14	00110	$a_2\alpha^2 + a_1\alpha^1$	β^{23}
15	01100	$a_3\alpha^3 + a_2\alpha^2$	β^{30}
16	01010	$a_3\alpha^3 + a_1\alpha^1$	β^6
17	00111	$a_2\alpha^2 + a_1\alpha^1 + 1$	β^{11}
18	01011	$a_3\alpha^3 + a_1\alpha^1 + 1$	β^{17}
19	10011	$a_4\alpha^4 + a_1\alpha^1 + 1$	β^{17}
20	11100	$a_4\alpha^4 + a_3\alpha^3 + a_2\alpha^2$	β^{12}
21	11010	$a_4\alpha^4 + a_3\alpha^3 + a_1\alpha^1$	β^9
22	01110	$a_3\alpha^3 + a_2\alpha^2 + a_1\alpha^1$	β^{12}
23	10110	$a_4\alpha^4 + a_2\alpha^2 + a_1\alpha^1$	β^{28}
24	11001	$a_4\alpha^4 + a_3\alpha^3 + 1$	β^{25}
25	10101	$a_4\alpha^4 + a_2\alpha^2 + 1$	β^{22}
26	01101	$a_3\alpha^3 + a_2\alpha^2 + 1$	β^8
27	01111	$a_3\alpha^3 + a_2\alpha^2 + a_1\alpha^1 + 1$	β^{23}
28	10111	$a_4\alpha^4 + a_2\alpha^2 + a_1\alpha^1 + 1$	β^{28}
29	11011	$a_4\alpha^4 + a_3\alpha^3 + a_1\alpha^1 + 1$	β^{18}
30	11101	$a_4\alpha^4 + a_3\alpha^3 + a_2\alpha^2 + 1$	β^{14}
31	11110	$a_4\alpha^4 + a_3\alpha^3 + a_2\alpha^2 + a_1$	β^{24}
32	11111	$a_4\alpha^4 + a_3\alpha^3 + a_2\alpha^2 + a_1$	β^{13}

The following properties of a Golois field with normal basis are useful in application:

1. For any element $A \in GF(2^5)$
   $1 = A + A^2 + A^4 + .. + A^{2^4}$.
   This implies that normal basis representation of 1 is (11111).
2. For any element A $\in GF(2^5)$

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

$A^2 = \sum_{i=0}^{4} a_i \beta^{2i+1} = \sum_{i=0}^{4} a_{i-1} \beta^{2i} = (a_3 a_2 a_1 a_0 a_4)$

.

$GF(2^5)$ addition:

$(a_4,a_3,a_2,a_1, a_0) \pm (b_4,b_3,b_2,b_1, b_0) = (c_4,c_3,c_2,c_1,c_0)$, where $c_i = a_i \oplus b_i$ overGF(2)

Note that in GF(2^5), since $(a_4,a_3,a_2,a_1,a_0) + (a_4,a_3,a_2,a_1,a_0) = (0,0,0,0,0)$, each element $(a_4,a_3,a_2,a_1,a_0)$is its own additive inverse. Addition and Subtraction can be implemented efficiently as component wise exclusive OR in the NB representation.

$GF(2^5)$ squaring

By the second property of normal basis ,squaring of an element a in NB representation is a cyclic shift operation . So the hardware implementation of squaring operation $GF(2^5)$ multiplication

Let A and B be two arbitrary elements in $GF(2^5)$ in a NB representation and C= A.B be the product of A and B. we denote $A = \sum_{i=0}^{4} a_i \beta^{2^i}$ as a vector A=(a_0,a_1,a_2,a_3,a_4) , $B = \sum_{i=0}^{4} b_i \beta^{2^i}$ as a vector B = (b_0,b_1,b_2,b_3,b_4) where C=

(c_0,c_1,c_2,c_3,c_4) , then the last term c_4 of C is a logic function of the components of A and B, that is, c_4=

$f(a_0,a_1,a_2,a_3,a_4; b_0,b_1,b_2,b_3,b_4)$.

Since squaring in NB representation is a cyclic shift operation, we have $C^2=A^2.B^2$ or equivalently

$(c_4 ,c_0,c_1,c_2,c_3) = (a_4 ,a_0,a_1,a_2,a_3). (b_4 ,b_0,b_1,b_2,b_3)$.
Hence, the last component c_3 can be obtained by the same function f that is, $c_3 = f(a_4 ,a_0,a_1,a_2,a_3; b_4 ,b_0,b_1,b_2,b_3)$. By squaring C repeatedly , we get

$c_4 = f(a_0,a_1,a_2,a_3,a_4; b_0,b_1,b_2,b_3,b_4)$

$c_3 = f(a_4 ,a_0,a_1,a_2,a_3; b_4 ,b_0,b_1,b_2,b_3)$

:

$C_0 = f(a_1,a_2,a_3, a_4, a_0; b_1,b_2,b_3, b_4, b_0)$

By the above equation define the Massey-Omura multiplier in normal basis representation. In the multiplier, the same logic function f for computing the last component of c_4 of the product'c'

Can be used to get the remaining components c_3, c_2, c_1, c_0 of the product sequentially. In parallel architechture, we can use 5 identical logic function f for calculating all components of the products of the product simultaneously .the product of A and B in the field $GF(2^5)$is

$C = A*B = c_0\beta^1 + c_1\beta^2 + c_2\beta^4 + c_3\beta^8 + c_4\beta^{16}$

$= (a_0\beta^1 + a_1\beta^2 + a_2\beta^4 + a_3\beta^8 + a_4\beta^{16}) \times (b_0\beta^1 + b_1\beta^2 + b_2\beta^4 + b_3\beta^8 + b_4\beta^{16})$

Thus, we can get $c_k = \sum_{i=0}^{4}\sum_{j=0}^{4} \mu_{ij}^{(k)} a_i b_j$, $0 \le k \le 5$.

The 5×5 matrices $\mu^{(k)}(0 \le k \le 5)$ whose elements $\mu_{ij}^{(k)}$, $0 \le i,j \le 5$ can be obtained if we know the transformation between the elements of the PB and the elements of NB ,that is, the normal basis representation of the elements of the PB.

For a normal basis there always exist a multiplication table T(corresponding to the irreducible polynomial),which is given by $\beta \begin{bmatrix} \beta \\ \beta^2 \\ \vdots \\ \beta^{2^4} \end{bmatrix} = T \begin{bmatrix} \beta \\ \beta^2 \\ \vdots \\ \beta^{2^4} \end{bmatrix}$.

Corresponding to aT matrix , there always exists a matrix $\mu^{(k)}$for any $c_k$of the product c , for the given irreducible polynomial which defines the normal basis in $GF(2^5)$. After the multiplication table T is obtained , the matrix $\mu^{(k)}$can be calculated according to the above method.

$GF(2^5)$ invertion

We know from Femat's theorem that for any non-zero elements $\beta, \beta^{2^5-1} = 1$, that is, $\beta^{-1} = \beta^{2^5-2}$.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

$2^5 - 2 = 2^1 + 2^2 + 2^3 + 2^4$. Hence ,

$\beta^{-1} = \beta^2 * \beta^4 * \beta^8 * \beta^{16}$. That is an inversion requires 4squaring 3 multiplication. This could be reduced further by alternative methods[12],[13].

4. ELLIPTIC CURVE GROUP OPERATION:

Elliptic group operations includes point negation, point subtraction, point doubling, and scalar multiplication.

Let GF(2^5) be a characteristic 2 finite field. Then a (non-super singular) elliptic curve E(GF(2^5)) over GF(2^5) defined by E: $y^2 + xy = x^3 + ax^2 + b$ consists of the set of solutions or points P=(x,y) for x,y ∈ GF(2^5) .

E: $y^2 + xy = x^3 + ax^2 + b$ *in* GF(2^5)

together with an extra point *O* called the point at infinity.

The number of points on E(GF(2^5)) is denoted by # E(GF(2^5)). The Hasse Theorem states that:

$2^5 + 1 - 2\sqrt{2^5} \leq \#\ E(GF(2^m)) \leq 2^5 + 1 + 2\sqrt{2^5}$.

It is again possible to define an addition rule to add points on *E* as the addition rule is specified as follows:

1. Rule to add the point at infinty to itself:
   $O + O = O$

2. Rule to add the point at infinity to any other point:
   $(x,y) + O = O + (x,y) = (x,y)$ *for all* $(x,y) \in$ GF(2^5)

3. Rule to add two points with the same x–coordinates when the points are either distinct or have *x*-coordinates 0:

   $(x,y) + (x,x + y) = O$ *for all* $(x,y) \in$ GF(2^5)

4. Rule to add two points with different *x*-coordinates: Let $(x_1,y_1) \in$ GF(2^5) and $(x_2,y_2) \in$ GF(2^5) be two points such that $x_1 \neq x_2$. Then $(x_1,y_1) + (x_2,y_2) = (x_3,y_3)$, where:

   $x_3 = \lambda^2 + \lambda + x_1 + x_2 + a$ in GF(2^5), $y_3 = \lambda.(x_1 + x_3) + x_3 + y_1$ in GF(2^5), and $\lambda \equiv \dfrac{y_1 + y_2}{x_1 + x_2}$ in GF(2^5).

5. Rule to add a point to itself (double a point): Let $(x_1,y_1) \in$ GF(2^5) be a point with $x_1 \neq 0$. Then $(x_1,y_1) + (x_1,y_1) = (x_3,y_3)$,where:

   $x_3 = \lambda^2 + \lambda + a$ in GF(2^5) , $y_3 = x_1^2 + (\lambda + 1)x_3$ in GF(2^5) , and $\lambda = x_1 + \dfrac{y_1}{x_1}$ in GF(2^5).

The set of points on *E*(GF(2^5)) forms an abelian group under this addition rule. Notice that the addition rule can always be computed efficiently using simple field arithmetic. Cryptographic schemes based on ECC rely on scalar multiplication of elliptic curve points. As before given an integer *k* and a point *P* ∈ GF(2^5) , scalar multiplication is the process of adding *P to itself* k times. The result of this scalar multiplication is denoted 'kP'.

5. SCALAR MULTIPLICATION :

In the scalar multiplication just we used the binary method and generated a cyclic group. We propose a non-super singular elliptic curve $y^2 + xy = x^3 + x^2 + 1$ defined over GF(2^5). Let the primitive polynomial chosen for constructing the finite field be x^5+x^2+1. For example Assume that :P = (x_1 , y_1) and Q = (x_2 , y_2) are two distinct points on the elliptic curve E: $y^2 + xy = x^3 + x^2 + 1$ ----(1) . when Q ≠ -P, the operation P + Q=(x_3 , y_3) can then be derived as shown in the following ,in which part (a) corresponds to the point addition and part (b) for the point doubling operation.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

$(a) P \neq Q\ (x_1 \neq x_2\ \&\ y_1 \neq y_2)$

$$\lambda = \frac{y_1 + y_2}{x_1 + x_2} \quad (2)$$

$$x_3 = \lambda^2 + \lambda + x_1 + x_2 + a_2 \quad (3)$$

$$y_3 = \lambda(x_1 + x_3) + x_3 + y_1 \quad (4)$$

$(b) P = Q\ (x_1 = x_2\ \&\ y_1 = y_2)$

$$\lambda = x_1 + \frac{y_1}{x_1} \quad (5)$$

$$x_3 = \lambda^2 + \lambda + a_2 \quad (6)$$

$$y_3 = \lambda(x_1 + x_3) + x_3 + y_1 \quad (7)$$

Since the field multiplication and field inversion /division are much more complicated than the field addition, we can ignore the effect of field addition operation and conclude the following observations:

(i) the value of λ can be computed by first finding the inverse of $1/(x_1 + x_2)$ $or 1/x_1$ and then multiplying the denominator $(y_1 + y_2) or\ y_1$ .we can also get λ by directly employing the field division operation[8][12].

(ii) whether (a) or (b), P+Q will be computed in a sequential manner based on data dependent relationship among $\lambda, x_3, and\ y_3$. And, as seen from Eqs. (2) and (5),the computation of λ in the next iteration can start before the completion of y_3 in the current time, if the inverse operation is used instead of employing the division operation.

(iii) Equations (6) can be viewed as the degenerate case of Eq.(3) as $x_1+x_2=0$ over GF(2^5) when P=Q; therefore, Eqs.(3)and (6) can share the same hardware implementing $x_3 = \lambda^2 + \lambda + x_1 + x_2 + a$

In essence, the elliptic curve cryptographic scheme requires the scalar multiplication defined as follows.

$$Q = kP = P + P + \ldots + P \text{ , k times}$$

where P denotes a point on the elliptic curve and k is a random integer with $1 \le k \le order(P) \approx 2^m - 1$. Algorithm-1 gives the well-known double and add algorithm, also referred to as the binary method, to compute kP assuming that Q is initialized as an infinite point denoted by the symbol O.

Algorithm. ; scalar multiplication-binary method

Input: k and P

Output: Q = kP

/*convert the integer k into the binary representation */

$$Q = k.P; k = (k_{t-1}, k_{t-2}, \ldots, k_0); k_i \in \{0,1\}$$

1. Q=O;
2. for i from t-1 down to 0 do
3. Q=Q+Q;
4. if k_i =1, then Q=Q+P;
5. return Q:

Double –and-add algorithm.

As seen from algorithm 1, the expected number of point additions is approximately 0.5t and the number of doubles is exactly t. The expected number of point additions is actually equivalent to the average number of nonzero coefficients of k. hence the cyclic group generated is;

Using the algorithms above a elliptic curve points can be constructed from the curve E: $y^2 + xy = x^3 + x^2 + 1$ .Consider the base point , (12,11),means $(\alpha^{12}, \alpha^{11})$where α is the root of polynomial x^5+x^2+1. Order of the base point is 11. Hence it is generated 110 random out put points for different seed value, that is, (12,11),(12,29),(24,22),(17,13),(3,15),(6,21),(17,23), (6,30),(17,13),(17,23),(3,26),(17,23),(24,27),(12,29),(24,27),(24,22),(24,27),(17,23),(3,26),(6,21)……..

6. ELLIPTIC CURVE DOMAIN PARAMETERS OVER $GF(2^5)$

Elliptic curve domain parameters over $GF(2^5)$ are a sextuple:

$$T = (m, f(x), a, b, G, n, h)$$

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

consisting of an integer m specifying the finite field $GF(2^m)$ an irreducible binary polynomial $f(x)$ of degree m specifying the representation of $GF(2^m)$, two elements $a,b \in GF(2^m)$ specifying the elliptic curve $E(GF(2^m))$ defined by the equation:

E: $y^2 + xy = x^3 + x^2 + 1$, over $GF(2^m)$

a base point $G = (x_G, y_G)$ on $E(GF(2^m))$, a prime n which is the order of G, and an integer h which is the

cofactor $h = \#E(GF(2^m))/n$;that is,

Where m=5

$f(x)=x^5+x^2+1$(irreducible)

a=1

b=1($\neq 0$)

n=11

G=(12,11)

h=110/11=10

Elliptic curve domain parameters over $GF(2^m)$. precisely specify an elliptic curve and base point. This is

necessary to precisely define public-key cryptographic schemes based on ECC.

Section 6.1.2.1 describes how to generate elliptic curve domain parameters over (2^m) , and Section 6.1.2.2 describes how to validate elliptic curve domain parameters over $GF(2^m)$.

6.1.2.1 Elliptic Curve Domain Parameters over (2^m) . Generation Primitive

Elliptic curve domain parameters over (2^m) . should be generated as follows:

Input: The approximate security level in bits required from the elliptic curve domain parameters .

Output: Elliptic curve domain parameters over (2^m) .

$$T = (m, f(x), a, b, G, n, h)$$

such that taking logarithms on the associated elliptic curve is believed to require approximately 2^t operations.

Actions: Generate elliptic curve domain parameters over $GF(2^m)$ as follows:

1. Selecting a binary irreducible polynomial $f(x)$ of degree m from to determine the representation of (2^m) .
2. Selecting elements $a,b \in GF(2^m)$ to determine the elliptic curve $E(GF(2^m))$ defined by the equation:

$$E : y^2+xy = x^3+ax^2+b \text{ in } GF(2^m).$$

a base point $G = (x_G, y_G)$ on $E(GF(2^m))$, a prime n which is the order of G, and an integer h which is the cofactor $h = \#E(GF(2^m))/n$, subject to the following constraints:

- $b \neq 0$ in $GF(2^m)$
- $\#E(GF(2^m)) \neq 2^m$.
- $2^{mB} \neq 1 \pmod n$ for any $1 \leq B < 20$.
- $h \leq 4$.

3. Output $T = (m, f(x), a, b, G, n, h)$.

This primitive also allows any of the known curve selection methods to be used. However to foster interoperability it is strongly recommended that implementers use one of the recommended elliptic curve domain parameters over $GF(2^m)$.

6.1.2.2 Validation of Elliptic Curve Domain Parameters over (2^m) .

Frequently it is either necessary or desirable for an entity using elliptic curve domain parameters over $GF(2^m)$ to receive an assurance that the parameters are valid—that is that they satisfy the arithmetic requirements of elliptic curve domain parameters—either to prevent malicious insertion of insecure parameters, or to detect inadvertent coding or transmission errors.

There are four acceptable methods for an entity U to receive an assurance that elliptic curve domain parameters over $GF(2^m)$ are valid. Only one of the methods must be supplied, although in many cases greater security may be obtained by carrying out more than one of the methods.

The four acceptable methods are:

1. U performs validation of the elliptic curve domain parameters over (2^m) . itself using the validation primitive described in Section 6.1.2.2.1.

2. U generates the elliptic curve domain parameters over $GF(2^m)$ itself using a trusted system using the primitive specified in Section 6.1.2.1.

3. U receives assurance in an authentic manner that a party trusted with respect to U's use of the elliptic curve domain parameters over $GF(2^m)$. has performed validation of the parameters using the

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

validation primitive described in Section 6.1.1.2.1.

4. *U* receives assurance in an authentic manner that a party trusted with respect to *U*'s use of the elliptic curve domain parameters over $GF(2^m)$. generated the parameters using a trusted system using the primitive specified in Section 6.1.2.1.

6.1.2.2.1 Elliptic Curve Domain Parameters over $GF(2^m)$. Validation Primitive

The elliptic curve domain parameters over (2^m) . validation primitive should be used to check elliptic curve domain parameters over $GF(2^m)$. are valid as follows:

Input: Elliptic curve domain parameters over (2^m) .

$$T = (m, f(x), a, b, G, n, h)$$

Output: An indication of whether the elliptic curve domain parameters are valid or not — either 'valid' or 'invalid'.

Actions: Validate the elliptic curve domain parameters over $GF(2^m)$ as follows:

1. Check that $f(x)$ is a binary irreducible polynomial of degree m .
2. Check that a, b, x_G, and y_G are binary polynomials of degree m-1 or less.
3. Check that $b \neq 0$ in (2^m) .
4. Check that $y_G^2 + x_G y_G = x_G^3 + a x_G^2 + b$ in $GF(2^m)$.
5. Check that n is prime.
6. Check that $h \leq 4$, and that $h = \lfloor (\sqrt{2^m} + 1)^2 / n \rfloor$.
7. Check that $nG = O$.
8. Check that $2^{mB} \not\equiv 1 \pmod{n}$ for any $1 \leq B < 20$, and that $nh \neq 2^m$.
9. If any of the checks fail, output 'invalid', otherwise output 'valid'.

Steps 1 and 8 above excludes the known weak classes of curves which are susceptible to either the Menezes-Okamoto-Vanstone attack, or the Frey-Ruck attack, or the Semaev-Smart-Satoh-Araki attack,or to attacks based on the Weil descent. If the elliptic curve domain parameters have been generated verifiably at random using SHA-1 as described in ANSI X9.62 [24], it may also be checked that a and b have been correctly derived from the random seed.

6.2 Elliptic Curve Key Pairs

All the public-key cryptographic schemes described in this document use key pairs known as elliptic curve key pairs.

Given some elliptic curve domain parameters $T = (p,a,b,G,n,h)$ or $(m,f(x),a,b,G,n,h)$, an elliptic curve key pair (d,Q) associated with T consists of an elliptic curve secret key d which is an integer in the interval $[1,n-1]$, and an elliptic curve public key $Q = (x_Q y_Q)$ which is the point $Q = dG$.

Section 4.2.1 describes how to generate elliptic curve key pairs, Section 4.2.2 describes how to validate elliptic curve public keys.

6.2.1 Elliptic Curve Key Pair Generation Primitive

Elliptic curve key pairs should be generated as follows:

Input: Valid elliptic curve domain parameters $T = (p,a,b,G,n,h)$ or $(m,f(x),a,b,G,n,h)$.

Output: An elliptic curve key pair (d,Q) associated with T.

Actions: Generate an elliptic curve key pair as follows:

1. Randomly or pseudo randomly select an integer d in the interval $[1,n-1]$.
2. Calculate $Q = dG$.
3. Output (d,Q).

6.2.2 Validation of Elliptic Curve Public Keys

Frequently it is either necessary or desirable for an entity using an elliptic curve public key to receive an assurance that the public key is valid — that is that it satisfies the arithmetic requirements of an elliptic curve public key — either to prevent malicious insertion of an invalid public key to enable attacks like small subgroup attacks, or to detect inadvertent coding or transmission errors.

There are four acceptable methods for an entity *U* to receive an assurance that an elliptic curve public key is valid. Only one of the methods must be supplied, although in many cases greater security may be obtained by carrying out more than one of the methods.

The four acceptable methods are:

1. *U* performs validation of the elliptic curve public key itself using the public key validation primitive described in Section 6.2.2.1.

2. *U* generates the elliptic curve public key itself using a trusted system.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

3. *U* receives assurance in an authentic manner that a party trusted with respect to *U*'s use of the elliptic curve public key has performed validation of the public key using the public key validation primitive described in Section 6.2.2.1.

4. *U* receives assurance in an authentic manner that a party trusted with respect to *U*'s use of the elliptic curve public key generated the public key using a trusted system.

Usually when*U* accepts another party's assurance that an elliptic curve public key is valid, the other party is a CA who validated the public key during the certification process. Occasionally *U* may also receive assurance from another party other than a CA. For example, in the Station-to-Station protocol described in ANSI X9.63 [4], *U* receives an ephemeral public key from *V*. *V* is trusted with respect to *U*'s use of the public key because *U* is attempting to establish a key with *V* and *U* only combines the public key with its own ephemeral key pair. It is therefore acceptable in this circumstance for *U* to accept assurance from *V* that the public key is valid because the public key is received in a signed message.

6.2.2.1 Elliptic Curve Public Key Validation Primitive

The elliptic curve public key validation primitive should be used to check an elliptic curve public key is valid as follows:

Input: Valid elliptic curve domain parameters $T = (p,a,b,G,n,h)$ or $(m, f(x),a,b,G,n,h)$, and an elliptic curve public key $Q = (x_Q, y_Q)$ associated with *T*.

Output: An indication of whether the elliptic curve public key is valid or not—either 'valid' or 'invalid'.

Actions: Validate the elliptic curve public key as follows:

1. Check that $Q \neq O$.
2. If *T* represents elliptic curve domain parameters over F*p*, check that x_Q and y_Q are integers in the range $[1,p-1]$, and that:
$$y_Q^2 \equiv x_Q^3+ax_Q+b \pmod p:$$
3. If *T* represents elliptic curve domain parameters over F_{2^m}, check that x_Q and y_Q are binary polynomials of degree at most $m-1$, and that:
$$y_Q^2+x_Qy_Q = x_Q^3+ax_Q^2+b \text{ in } (2^m),$$
4. Check that $n\,Q = O$.
5. If any of the checks fail, output 'invalid', otherwise output 'valid'.

In the above routine, steps 1, 2, and 3 check that *Q* is a point on *E* other than the point at infinity, and step 4 checks that *Q* is a scalar multiple of *G*.

7. ENCRYPTION SCHEMES

This section specifies the public-key encryption schemes based on ECC supported in this document.Public-key encryption schemes are designed to be used by two entities — a sender *U* and a recipient *V* —when *U* wants to send a message *M* to *V* confidentially, and *V* wants to recover *M*.

Here public-key encryption schemes are described in terms of an encryption operation, a decryption operation, and associated setup and key deployment procedures. *U* and *V* should use the scheme as follows when they want to communicate. First *U* and *V* should use the setup procedure to establish which options to use the scheme with, then *V* should use the key deployment procedure to select a key pair and*U* should obtain *V*'s public key—*U* will use *V*'s public key to control the encryption procedure, and *V* will use its key pair to control the decryption operation. Then each time*U* wants to send a message *M* to *V*, *U* should apply the encryption operation to *M* under *V*'s public key to compute an encryption or ciphertext *C* of *M*, and convey *C* to *V*. Finally when*V* receives *C*,V should apply the decryption operation to *C* under its key pair to recover the message *M*.

Loosely speaking public-key encryption schemes are designed so that it is hard for an adversary who does not possess *V*'s secret key to recover information about messages (other than their length) from their ciphertexts. Thus the schemes provide data confidentiality.

The public-key encryption schemes specified in this section may be used to encrypt messages of any kind. They may be used to transport keying data from *U* to *V*, or to encrypt information data directly. This flexibility allows the schemes to be applied in a broad range of cryptographic systems. Nonetheless it is envisioned that the majority of applications will apply the schemes for key transport, and subsequently use the transported key in conjunction with a symmetric bulk encryption scheme to encrypt information data. This is the traditional usage for public-key encryption schemes.

The only public-key encryption scheme supported at this time is the Elliptic Curve Integrated

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Encryption Scheme (ECIES). ECIES is specified in Section7.1.

7.1 Elliptic Curve Integrated Encryption Scheme

The Elliptic Curve Integrated Encryption Scheme (ECIES) is a public-key encryption scheme based on ECC. It is designed to be semantically secure in the presence of an adversary capable of launching chosen-plaintext and chosen-ciphertext attacks.

(Note that the Elliptic Curve Integrated Encryption Scheme has a complex naming history. It is occasionally known instead as the Elliptic Curve Augmented Encryption Scheme or simply the Elliptic Curve Encryption Scheme.)

The setup procedure for ECIES is specified in Section 7.1.1, the key deployment procedure is specified in Section 7.1.2, the encryption operation is specified in Section 7.1.3, and the decryption operation is specified in Section 7.1.4.

7.1.1 Scheme Setup

U and *V* should perform the following setup procedure to prepare to use ECIES:

1. *V* should establish which of the key derivation functions to use, and select any options involved in the operation of the key derivation function. Let *KDF* denote the key derivation function chosen. (In this edition the only possibility is ANSI-X9.63-KDF with the option SHA-1[27].)

2. *V* should establish which of the MAC schemes (*Loosely speaking, MAC schemes are designed so that it is hard for an adversary to forge valid message and tag pairs so that the schemes provide data origin authentication and data integrity.The list of supported MAC schemes at this time is: HMAC–SHA-1–160 with 20 octet or 160 bit keys HMAC–SHA-1–80 with 20 octet or 160 bit keys Both these MAC schemes are specified in IETF RFC 2104 [21] and ANSI X9.71 [26] based on the hash function SHA-1 specified in FIPS 180-1 [22]*) to use, and select any options involved in the operation of the MAC scheme. Let *MAC* denote the MAC scheme chosen, *mackeylen* denote the length in octets of the keys used by *MAC*, and *maclen* denote the length in octets of tags produced by *MAC*.

3. *V* should establish which of the symmetric encryption schemes to use,and select any options involved in the operation of the encryption scheme. Let *ENC* denote the encryption scheme chosen, and *enckeylen* denote the length in octets of the keys used by *ENC*.

4. *V* should establish whether to use the 'standard' elliptic curve Diffie-Hellman primitive , or the elliptic curve cofactor Diffie-Hellman primitive .

5. *V* should establish elliptic curve domain parameters $T = (p,a,b,G,n,h)$ or $(m, f(x),a,b,G,n,h)$ at the desired security level. The elliptic curve domain parameters *T* should be generated using the primitive specified in Section 6.1.1.1 or the primitive specified in Section6.1.2.1. *V* should receive an assurance that the elliptic curve domain parameters *T* are valid using one of the methods specified in Section 6.1.1.2 or Section 6.1.2.2.

6. *U* should obtain in an authentic manner the selections made by *V* — the key derivation function *KDF*, the MAC scheme *MAC*, the symmetric encryption scheme *ENC*, the elliptic curve domain parameters *T*, and an indication whether to use the 'standard' elliptic curve Diffie-Hellman primitive or the cofactor Diffie-Hellman[27]. *U* should also receive an assurance that the elliptic curve domain parameters *T* are valid using one of the methods specified in Section 6.1.1.2 or Section 6.1.2.2.

7. *U* should establish whether or not to represent elliptic curve points using point compression.

7.1.2 Key Deployment

U and *V* should perform the following key deployment procedure to prepare to use ECIES:
1. *V* should establish an elliptic curve key pair (d_V, Q_V) associated with the elliptic curve domain parameters *T* established during the setup procedure. The key pair should be generated using the primitive specified in Section 6.2.1.

2. *U* should obtain in an authentic manner the elliptic curve public key Q_V selected by *V*. If the 'standard' elliptic curve Diffie-Hellman primitive is being used,*U* should receive an assurance that Q_V is valid using one of the methods specified in Section 6.2.2, and if the elliptic curve cofactor Diffie-Hellman primitive is being used, *U* should receive an assurance that Q_V is at least partially valid using one of the methods specified in Section 6.2.2 or Section 6.2.3.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

7.1.3 Encryption Operation

U should encrypt messages using ECIES using the keys and parameters established during the setup procedure and the key deployment procedure as follows:

Input: The input to the encryption operation is:
1. An octet string *M* which is the message to be encrypted.
2. (Optional) Two octet strings *SharedInf o*1 and *SharedInf o*2 which consist of some data shared by *U* and *V*.

Output: An octet string *C* which is an encryption of *M*, or 'invalid'.
Actions: Encrypt *M* as follows:

1. Select an ephemeral elliptic curve key pair (k,R) with $R=(x_R,y_R)$ associated with the elliptic curve domain parameters *T* established during the setup procedure. Generate the key pair using the key pair generation primitive specified in Section 6.2.1.

2. Convert *R* to an octet string *R* using the conversion routine specified in Section4.3. Decide whether or not to represent *R* using point compression according to the convention established during the setup procedure.

3. Use one of the Diffie-Hellman primitives specified [27] to derive a shared secret field element $z \in F_q$ from the ephemeral secret key *k* and *V*'s public key Q_V obtained during the key deployment procedure. If the Diffie-Hellman primitive outputs 'invalid', output 'invalid' and stop.
Decide whether to use the 'standard' elliptic curve Diffie-Hellman primitive or the elliptic curve cofactor Diffie-Hellman primitive[53] according to the convention established during the setup procedure.

4. Convert $z \in F_q$ to an octet string *Z* using the conversion routine specified in Section 4.5.

5. Use the key derivation function *KDF* established during the setup procedure to generate keying data *K* of length *enckeylen+mackeylen* octets from *Z* and [*SharedInf o*1]. If the key derivation function outputs 'invalid', output 'invalid' and stop.

6. Parse the leftmost *enckeylen* octets of *K* as an encryption key *EK* and the rightmost *mackeylen* octets of *K* as a MAC key *MK*.

7. Use the encryption operation of the symmetric encryption scheme *ENC* established during the setup procedure to encrypt *M* under *EK* as ciphertext *EM*. If the encryption scheme outputs 'invalid',
output 'invalid' and stop.

8. Use the tagging operation of the MAC scheme *MAC* established during the setup procedure to compute the tag *D* on $EM||[SharedInf\ o2]$ under *MK*. If the MAC scheme outputs 'invalid', output 'invalid' and stop.

9. Output $C = R||EM||D$.

7.1.4 Decryption Operation

V should decrypt ciphertext using ECIES using the keys and parameters established during the setup procedure and the key deployment procedure as follows:
Input: The input to the decryption operation is:
1. An octet string *C* which is the ciphertext.
2. (Optional) Two octet strings *SharedInf o*1 and *SharedInf o*2 which consist of some data shared by *U* and *V*.
Output: An octet string *M* which is the decryption of *C*, or 'invalid'.
Actions: Decrypt *C* as follows:

1. If the leftmost octet of *C* is 02_{16} or 03_{16}, parse the leftmost $\lceil \log_2 q \rceil + 1$ octets of *C* as an octet string *R*, the rightmost *maclen* octets of *C* as an octet string *D*, and the remaining octets of *C* as an octet string *EM*. If the leftmost octet of *C* is 04_{16}, parse the leftmost $2\lceil \log_2 q \rceil + 1$ octets of *C* as an octet string *R*, the rightmost *maclen* octets of *C* as an octet string *D*, and the remaining octets of *C* as an octet string *EM*. If the leftmost octet of *C* is not 02_{16}, 03_{16}, or 04_{16}, output 'invalid' and stop.

2. Convert the octet string *R* to an elliptic curve point $R = (x_R,y_R)$ associated with the elliptic curve domain parameters *T* established during the setup procedure using the conversion routine specified in Section4.4. If the conversion routine outputs 'invalid', output 'invalid' and stop.

3. If the 'standard' elliptic curve Diffie-Hellman primitive is being used[27], receive an assurance that *R* is a valid elliptic curve public key using one of the methods specified in Section 6.2.2. If the

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

elliptic curve cofactor Diffie-Hellman primitive is being used, receive an assurance that *R* is at least a partially valid elliptic curve public key using one of the methods specified in Section 6.2.2 Section 6.2.3. If an appropriate assurance is not obtained, output 'invalid' and stop.

4. Use one of the Diffie-Hellman primitives to derive a shared secret field element $z \in Fq$ from *V*'s secret key d_V established during the key deployment procedure and the public key *R*. If the Diffie-Hellman primitive outputs 'invalid', output 'invalid' and stop. Decide whether to use the 'standard' elliptic curve Diffie-Hellman primitive or the elliptic curve cofactor Diffie-Hellman primitive [27]according to the convention established during the setup procedure.

5.Convert $z \in Fq$ to an octet string *Z* using the conversion routine specified in Section 4.5.

6. Use the key derivation function *KDF* established during the setup procedure to generate keying data *K* of length *enckeylen+mackeylen* octets from *Z* and [*SharedIn f o*1]. If the key derivation function outputs 'invalid', output 'invalid' and stop.

7. Parse the leftmost *enckeylen* octets of *K* as an encryption key *EK* and the rightmost *mackeylen* octets of *K* as a MAC key *MK*.

8. Use the tag checking operation of the MAC scheme *MAC* established during the setup procedure to check that *D* is the tag on $EM \| [SharedIn\ f\ o2]$ under *MK*. If the MAC scheme outputs 'invalid', output 'invalid' and stop.

9. Use the decryption operation of the symmetric encryption scheme *ENC* established during the setup procedure to decrypt *EM* under *EK* as *M*. If the encryption scheme outputs 'invalid', output 'invalid' and stop.

10. Output *M*.

Note that when implementing the above decryption operation on a constrained device, it may be desirable to perform the symmetric decryption operation (step 9) before the tag checking operation (step 8). This variant is allowed. However implementers of this variant should guard against the possibility that the output of step 9 is available to attackers regardless of the output of step 8.

8. CONCLUTION:

Elliptic curve over finite field are being extensively used in the design of public-key cryptographic schemes. Due to the emerging market of electronic commerce public-key cryptosystem gain more and more attention. Unlike for military purposes there is a need of flexible user groups. By this article we contributed a secure cryptosystem choosing the non-super-singular-curve
E: $y^6 + xy = x^3 + x^6 + 1$ over $GF(2^5)$ as per the guidelines given the ear liar works.

9. LIMITATION AND FUTURE WORK:

In our article, we tried our level best to implement The Elliptic curve cryptography in the finite field GF(2^5) . We plan to generalize the results ,that is , implementation in the binary field GF(2^ n). Also the speed of the computations can be compared between normal and polynomial basis..

REFERENCES :

[1] E.R.Berlekamp, *Algebraic Coding Theory*, NY, McGraw-Hill, 1968
[2] I.Blake, G.Seroussi, and N.Smart, Elliptic Curves in Cryptography, *Cambridge University Press*,1999.
[3] P.Buhler,H.W.Lenstra, and C.Pomerance. The development of the number field sieve, volume 1554 of *lecture Notes in Computer Science, Springer-Verlag,* 1994.
[4] W.Diffie and M.E.Hellman, "New directions in cryptography," *IEEE Trans. On Information Theory,* IT-22:pp.644-654,1976.
[5] D.M.Gordon, "A Survey of fast exponention methods," *J.Algorithms,*27,1998,pp.129-146.
[6] J.Guajardo and C.Paar,"Efficient Algorithms for Elliptic Curve Cryptosystems," *Advances in Cryptology-CYYPTO,97,LNCS 1462*,pp.342-356.
[7] Y.Han,P.leong,P.Tan,a nd J.Zhang, "Fast Algorithmsfor Elliptic Curve Cryptosystems over Binary Finite Field," *Advances in Cryptology-CRYPTO'99 LNCS 1716*,pp 75-85.
[8] T.Itoh and S.Tsujii, "A fast Algorithm for Computing Multiplication inverses in GF(2^m) Using Normal Bases," *Information & Computation, 78*,1988,pp.171-177.
[9] T.Kobayashi,H.Morita,K.Kobayashi,and F.Hoshino, "Fast Elliptic Curve Algorithm

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Combining Frobenius map and Table referenced to Adapt to higher Characteristic," *Advances in Cryptology-CRYPTO'99,LNCS 1592*,pp.176-189.

[10] N.Koblitz, A Course in Number Theory and Cryptography, 2nd Eds., *Springer-Verlag,* 1994.B.Schneier,"*Applied Cryptography,"2nd ed.,J.Wiley & Sons,Inc.*,1996.

[11] N.Koblitz, Introduction to Elliptic Curves and Modular Forms, *2nd Ed., Spinger-Verlag,*1993.

[12] N.Koblitz, Elliptic Curve Cryptosystems, *Math.Compu.*Vol.48,No.177, Jan,1987,pp 203-209.

[13] ZhiLi, JohnHiggins, Mark Element "Perfomance of Finite Field Arithmatic in an Elliptic Curve Crypto Systems" *IEEE Transactions*, pp 249-256, 0-7695-1315-8/01,2001.

[14] D.E.Knuth, Seminumerical Algorithms, MA, Addison-Wesley,1981.

[15] D.F.Lawden, *Elliptic Functions and Applications, Springer-Verlag,*1989.

[16] 16]A.Menezes. T. Okamoto, and S.Vanstone, "Reducing Elliptic Curve Logorithms to Logorithms in a Finite Fiels," *IEEE Transactions on Information Theory,* 39(5): 1639-1646, September, 1993.

[17] F. Morain, J. Olives, Speeding up the computations on an elliptic curve using addition-subtraction chains, *RAIRO Theoretical Information and Applications* 24 (1990) 531-543.

[18] R.L.Rivest, A.Shamir, and L.M.Adleman,"A Method for obtaining Digital Signatures and Public-Key Cryptosystems," *Communications of the ACM,* v.21,n.2, Feb 1978,pp.120-126.

[19] K.Sakurao and H.Shizuya, "A Structural Comparison of the Computational Difficulty of Breaking Discrete log Cryptosystems," *J.Cryptology*(1988) 11: 29-43.

[20] A.Schroeppel, H.Orman, S.O.Malley, and O.Spatschek,"Fast key Exchange with Elliptic Curve Systems," *Advances in Cryptology-CRYPTO'95,LNCS 963,* Springer-Verlag, 1995, pp.43-56.

[21] H. Krawcyzk, M. Bellare, and R. Canetti, HMAC: Keyed Hashing for Message Authentication. *Internet Engineering Task Force, Internet* RFC 2104, 1997. Available from: http://www.ietf.org/

[22] FIPS 180-1. Secure Hash Standard, *Federal Information Processing Standards Publication 180-1*,1995. Available from: http://csrc.nist.gov/

[23] ANSI X9.52-1998: *Triple Data Encryption Algorithm Modes of Operation*. American Bankers Association,1998.

[24] ANSI X9.62-1998: *Public Key Cryptography for the Financial Services Industry: the Elliptic Curve Digital Signature Algorithm (ECDSA)*. American Bankers Association, 1999.

[25] ANSI X9.63-199x: Public Key Cryptography for the Financial Services Industry: Key Agreement and Key Transport Using Elliptic Curve Cryptography. October, 1999. Working Draft.

[26] ANSI X9.71-199x: Keyed Hash Message Authentication Code. March, 1998. Working Draft.

[27] W. Diffie and M. Hellman. New directions in cryptography. IEEE Transactions on Information Theory, IT-22(6): 644–654, November 1976.

[28] A.R.Rishivarman, B. Parthasarathy, and M.Thiyagarajan, : An e_cient perfor-

mance of GF(2^5) arithmatic in an elliptic curve cryptosystem, *International Journal of Computing and Applications,* 4(2), 111-116 (2009).

[29] A.R.Rishivarman, B. Parthasarathy, and M.Thiyagarajan,: A Montgomery repre- sentation of elements in GF(25) for e_cient arithmetic to use in ECC. *InternationalJournal of Advanced Networking and Applications*, 1(5), 323-326 (2010).