International Journal on Emerging Technologies **3**(2): 38-51 (2012)
ISSN No. (Print) : 0975-8364
ISSN No. (Online) : 2249-3255

Implementation of Elliptic Curve Cryptography in 'C'

Kuldeep Bhardwaj and Sanjay Chaudhary
Department of Mathematics,
Dr. B. R. Ambedkar University, IBS, Khandari, Agra (UP)

(Received 15 Augest, 2012 Accepted 18 September, 2012)

ABSTRACT: Most of the hardware and software products and standards that use public key technique for encryption, decryption, authentication etc. are based on two major cryptosystem RSA and ElGamal cryptosystem. This paper involves the development of Elliptic Curve Cryptography for point doubling, point addition and scholar multiplication. We presented the timing details for different bit length for 10000 operations and compare the performance of ElGamal Encryption & Decryption over EC (Fp) and Fp at the different security levels. The implementation language is 'C' (LINUX based gcc compiler).

Keywords: Public key Cryptography, Elliptic Curve Cryptography, 'C' (LINUX based gcc compiler), GMP Library.

I. INTRODUCTION

Differ & Hellman [2] introduced the concept of Public Key Cryptography (PKC) in the year 1976, to solve the problem of key distribution associated with secret (symmetric) key cryptography. Here, encryption function is only one way function whose inverse is computationally infeasible to calculate unless some additional information is available. This additional information (called private key) serves as the decryption key and is not required for encryption of messages. The publicly available decryption key is used to encrypt the message. This key is associated with secret key but it is impractical to compute secret key from the knowledge of public key.

The one way function is based on sound mathematical foundations that are designed to make the problem hard for an intruder to get materialized and break into the system. The major approaches that since 1976 have withstood intruder attacks and provided are discrete logarithm problems as in El Gamal cryptosystem and the integer factorization problems as in RSA cryptosystem.

The recent developments in the field of factoring large integers and in finding discrete logarithms in large finite fields have resulted in increased key size but on the other hand this requires higher computational, communication cost. This extra cost has ramifications, especially for their commerce sites which conduct large number of source transactions. Keeping in view, a new public key technique ECC has shown its competency to challenge the existing cryptosystems. It is based on some very intricate mathematics involving elliptic curves in finite field.

Elliptic curves were first suggested in 1985 by N. Koblitz [5] and V. Miller [7] for implementing public key cryptosystems. They have recently been utilized in designing algorithms for primarily testing and also integer factorization. The main feature of ECC is that it relies on the difficulty of solving ECDLP (Elliptic Curve Discrete Log Problem) in the same way as RSA depends on the difficulty of factoring the product of two large primes. The best known method for solving ECDLP is fully exponential, whereas the number field sieve (for factoring) is sub-exponential. This allows ECC to use drastically smaller keys to provide equivalent security. Moreover, due of their rich structure one has more flexibility in choosing an elliptic curve than choosing a finite field.

II. HISTORY AND BASICS OF ELLIPTIC CURVES

The name "Elliptic curve" is based on ellipse. Elliptic curves were first discovered after 17th century in the form of Diophantine equation; $y^2 - x^3 = c$. Further, it is important to note that, though, it is easy to calculate the surface area of the ellipse; it is hard to calculate the circumference of the ellipse. The calculation can be reduced to an integral:

$$\int \frac{1}{\sqrt{x^3 + Ax + B}} dx \qquad ... (1)$$

This integral function which cannot be easy to solve, was the reason to consider the curve $y^2 = x^3 + ax + b$. This was done already during 18th century.

At the moment there are several definitions for an elliptic curve. However, the following definition is popular one:

Definition: An elliptic curve E, over the field K, is the set of points (x, y) $\in$ K $\times$ K that satisfy the equation

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6;\ a_i \in K \qquad \ldots(2)$$

together with an extra point at infinity (called "point at infinity") denoted by O, provided E is nonsingular i.e. $\forall$ P satisfying the equation of the curve, the partial derivatives $\frac{\partial F}{\partial x}, \frac{\partial F}{\partial y}$ are not both zero where F $y^2 + a_1xy + a_3y - x^3 - a_2x^2 - a_4x - a_6$. We denote this curve by E(K).

The definition given above is valid for any field but only for cryptographic applications, infinite fields are of little interests because of rounding up to figures and inaccuracy in problem. That is why it is a good idea to use finite fields. In rest of the paper we denote the finite field by GF (q), where the number of element in the field is $q = p^m$ with *p* prime and m being any positive integer. Thus the characteristic of the field is p. For practical applications two finite fields -- Prime fields F_p and Binary fields $F_2{}^m$ are of special interests.

A. Elliptic Curves over Prime Fields

As per the definition of elliptic curve which is given and described above; if we take the field K to be the finite field F_p where p > 3, and perform the following change of variables [8] as;

$$x \longrightarrow x - \frac{a_2}{3} \qquad \ldots(3)$$

$$y \longrightarrow y - \frac{a_1x + a_3}{2} \qquad \ldots(4)$$

We get much simpler form

$$y^2 = x^3 + ax + b \qquad \ldots(5)$$

Now, we consider the derivative w.r.t. x of the equation $y^2 = f(x)$ where $f(x) = x^3 + ax + b$, which is $f'(x) = 2y\frac{dy}{dx}$. The expression of $\frac{dy}{dx}$ is undefined iff $f'(x) = f(x) = y = 0$ for some (x,y). It means that (x, 0) is a singular point at which there is no definition for a real tangent value. Thus this condition is equivalent to the condition that f(x) has multiple root at the point x.

Note that, $= \left(\frac{a}{3}\right)^3 + \left(\frac{b}{2}\right)^2 = \frac{4a^3 + 27b^2}{4 \times 27}$ is the discriminent of the cubic polynomial f(x), f(x) has multiple root that is equivalent to $= 0$ and for the curve to be nonsingular we must have $4a^3 + 27b^2 \neq 0$.

B. Elliptic Curves over Binary Fields

Here we have two different equations; one is for super singular curve and other is for non-super singular curve and this is most complicated for Char F = 2. Without going into greater detail we are maintaining the following equations for the two curves [8]:

$y^2 + cy = x^3 + ax + b$ for super singular curve

$y^2 + xy = x^3 + ax^2 + b$ for non super singular curve

C. Addition and Multiplication Operation over Elliptic Curve Groups

As mentioned earlier the equation of the elliptic curve over prime field F_p (p > 3) is $y^2 = x^3 + ax + b \bmod p$, where $(4a^3 + 27b^2) \bmod p \neq 0$ and the set of elliptic curve points contains all the points $(x, y) \in F_p \times F_p$ satisfying this equation. In this section we will describe a rule called "chord and tangent rule" [9], for addition of two points on an elliptic curve E (F_p), which will give a third point which would lie on elliptic curve, so that, together with this addition operation, the set of elliptic curve points E(F_p) forms a group; with point of infinity O serving as identity element.

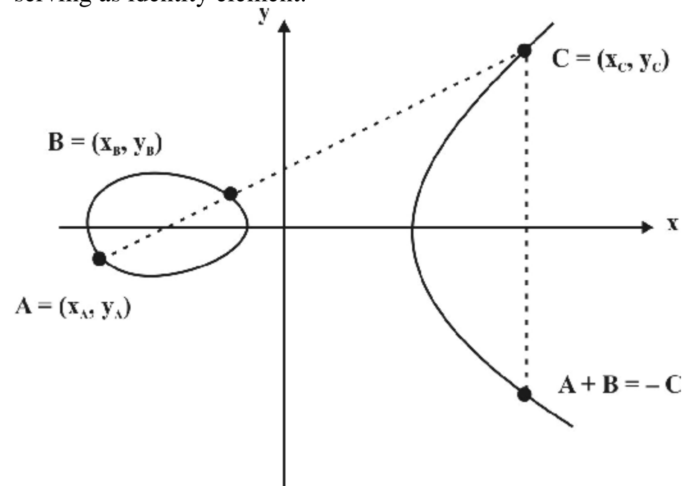

Fig. 1. Geometric Description of the addition of two distinct elliptic curve A + B = C.

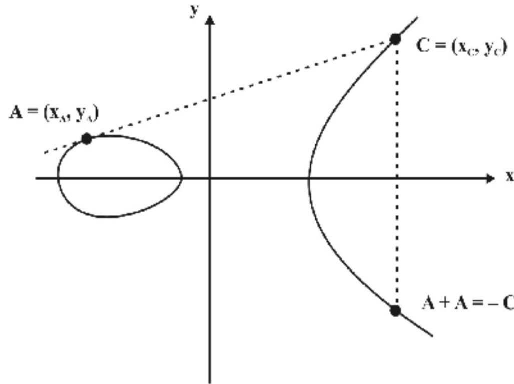

Fig. 2. Geometric Description of the addition of two distinct elliptic curve 2A = C.

For the sake of clarity, we first describe the operation geometrically. Consider the two distinct points A and B on an elliptic curve as shown in fig. 1. Then, to add A and B we draw a line through these points. If this line is not parallel to Y-axis, as the equation of elliptic curve is cubic, this line intersects the curve at exactly one more point. Then C = A + B is defined as the reflection of this point in X-axis. If the line is parallel to Y-axis we define C = O (point at infinity).

Hence, if A is any point on elliptic curve then to double the point to get C = 2A, we draw a tangent line at A, so that if this line is not parallel to Y-axis, it will intersect the curve at exactly one more point, then C = 2A is defined as the reflection of this point on X-axis and if the line is parallel to Y-axis then we define C = O (point at infinity).

We may sum up the mechanics of addition of two points on elliptic curve with following set of rules:

1. **Identity Element:** $A + O = O + A \ \forall \ A \in E(F_p)$ i. e. O is the identity element for the addition operation.
2. **Inverse Element:** If $A = (x_A, y_A)$ be a points on elliptic curve then negative (inverse) of point A is defined as $-A = (x_A, -y_A)$. If $B = -A$, then we define $A + B = O$.
3. **Point Addition:** Let $A = (x_A, y_A)$, $B = (x_B, y_B) \in E(F_p)$. If A ± B then A+B = C is as shown in Fig. 1. If $B = -A$ then we define $A + B = O$.
4. **Point Doubling:** Let $A = (x_A, y_A) \in E(F_p)$. If A – A then 2A = C is as shown in Fig. 2. If A = – A then we define 2A = O.

D. Analytical Explanation [6]

Hence, from the above geometrical explanation we may easily derive algebraic formulae for sum of two points and for doubling of two points.

D.1. Point Addition

Let us consider two distinct points $A = (x_A, y_A)$ and $B = (x_B, y_B)$ on the elliptic curve; with C = A + B; $C = (x_C, y_C)$. So, we have:

$$x_C = s^2 - x_A - x_B \pmod p \quad \ldots(6)$$

$$y_C = -y_A + s\,(x_A - x_C) \pmod p \quad \ldots(7)$$

$$s = (y_A - y_B)/(x_A - x_B) \pmod p \quad \ldots(8)$$

Where, s is the slope of the line through A and B.

Now, if B = – A i.e. $B = (x_A, -y_A \bmod p)$ then A + B = O where O is the point at infinity and if B = A i.e. A + B = 2A then following point doubling equations are used. Also A + B = B + A.

D.2. Point Doubling

Let us consider a point $A = (x_A, y_A)$ on elliptic curve, where y_A 0 with C = 2A; $C = (x_C, y_C)$. So we have:

$$x_C = s^2 - 2x_A \pmod p \quad \ldots(9)$$

$$y_C = -y_A + s(x_A - x_C) \pmod p \quad \ldots(10)$$

$$s = (3x_A^2 + a)/2y_A \pmod p \quad \ldots(11)$$

Here, s is the slope of tangent at point A and a is one of the parameters chosen with elliptic curve (remember the equation of elliptic curve which is $y^2 = x^3 + ax + b$) and if $y_A = 0$ then 2A = O, where O is the point at infinity.

Remark:

1. We define subtraction operation i.e. A – B to be A + (– B) which means addition of point A and inverse point of B.
2. One critical operation is scalar multiplication which is defined as; given an integer m and a point $A \in E(F_p)$ multiply P by scalar m (i.e. to find mA) which is the process of adding A to itself m times.

III. IMPLEMENTATION AND SIMULATION DESIGN

In this section we will provide details of the implementation and simulation design for the propose system with timing on INTEL (R) PENTIUM (R) DUAL CPU T2310 @ 1.46 GHZ 1.47 GHZ with RAM 1 GB processor. This processor is briefly described as 32 bit processor. The implementation language is 'C' (LINUX based gcc compiler). GMP library is used to support large integers and for scalar multiplication operation we have used "double and add method". The code for point doubling, point addition & scalar multiplication is in appendix A.

Here, we have use following structures

```
struct Elliptic_Curve
{
        mpz_t a;
        mpz_t b;
        mpz_t p;
}
```

```
struct Point
        {

        mpz_t x;

        mpz_t y;
}
```

In this first need is used to represent parameters a, b, p for the elliptic curve $y^2 = x^3 + ax + b$ over F_p and then after second requirement is used to represent point (x, y) on elliptic curve. EC, which is a global variable of type struct Elliptic_Curve. We may choose any point (X, Y) s.t $Y^3 \quad X^3 + aX + b$ ((X, Y) that does not lie on elliptic curve) to represent point at infinity. In our implementation we have used (0, 0) for this which if it does not lie on chosen elliptic curve.

The timing details for different bit length are presented in table 1 for 10000 operations.

Table 1: Timing details for different bit length.

Size of p (bits)	Time (Second)
50	0.13
100	0.09
150	0.10
200	0.13
250	0.16
300	0.20
350	0.23
400	0.28

IV. DOMAIN PARAMETERS FOR ELLIPTIC CURVE OVER THE FIELD F_p

The domain parameters for elliptic curve over F_p are p, a, b, G, n. p prime numbers are defined for finite field F_p. A and B are the parameters defining the curve $y^2 = x^3 + ax + b \pmod p$. G is the generator point (x_G, y_G), and is a point on elliptic curve for cryptographic operations and n is the order of the elliptic curve.

Generating a Secure curve Parameters: Hence due to the threat of special attack, it is essential to obtain the parameters of the elliptic curve that meet following requirements [11]:

- The order of elliptic curve must be prime or nearly prime.
- It should have high embedding degree so, their MOV attack [1] is not applicable.

Paulo S. L. M. Barreto and Michael Naehrig [10] proposed a technique to construct elliptic curve of prime order and embedding degree 12. The algorithm takes an input as the approximate size m of the curve (in bits) and out puts the parameters p, n, b, y such that curve $y^2 = x^3 + b$ has order n over F_p and the point G = (1, y) is a generator of the curve. We start with smallest value x of $2^{m/4}$ such that $\lceil \log_2 p(-x) \rceil = m$ where $p(x) = 36x^4 + 36x^3 + 24x^2 + 6x + 1$. So that by using a loop and iterating positive and negative values of x, it is not difficult to find a desired size of p and n that are both prime.

V. IMPLEMENTATION DETAILS AND TIMINGS

Appendix B shows the code in '*C*' that can be used to find primes p, n, b, y. We have used GMP library to support large integers. Here also the two structures explained earlier are used. We have used algorithm as explained in [4] to determine the square root modulo p. We start with a value $x \approx 2^{m/4}$ and then iterate positive and negative values of *x* for finding *x,* s.t. $p(-x) = m$. Here we have used $p(x_1) < m$, $p(x_2) > m$ root of p(x) will be near to $\frac{x_1+x_2}{2}$ to get desired elliptic curve and if one does not find it, we may change the value of RANGE which will decrease the starting value of x so that one will have greater range for x to iterate. Table 2 shows the timing for different size of p.

Table 2: Timing details for different size of p.

Size of p (bits)	Time (Second)
≈ 100	0.04
≈ 200	0.64
≈ 300	2.86
≈ 400	3.68
≈ 500	14.01

VI. ElGamal ENCRYPTION IMPLEMENTATION AND PERFORMANCE

In this section we make an attempt to compare the performance of ElGamal [11, 3] Encryption and Decryption over EC (F_p) (elliptic curve over F_p) and over F_p (finite field) at the different security levels. Table 3 shows the timings on INTEL (R) PENTIUM (R) DUAL CPU T2310 @ 1.46 GHZ 1.47 GHZ with RAM 1 GB processor. Again we used the 'C' language for it implementation in Appendix C.

Table 3.

ElGamal Key Size over Z_p	Timings for Encryption & Decryption	ElGamal Key Size over EC	Timings for Encryption & Decryption
1024	12.95×10^{-3}	160	6.09×10^{-3}
2048	76.83×10^{-3}	224	10.98×10^{-3}
3072	235.99×10^{-3}	256	15.46×10^{-3}
7680	3622.5×10^{-3}	384	35.12×10^{-3}
15360	217.65×10^{-3}	521	$66.21\ 10^{-3}$

VII. CONCLUSION

We have briefly described ECC which offers more security with per bit increase in key size than other existing public key techniques. So that, due to much smaller key sizes involved, ECC provides faster implementation. We have also taken the reader through the processes of implementation of operations on elliptic curves, Generation of secure elliptic curve domain parameters and also these elliptic curves are friendly pairing. We have also provided an overview and comparison of ElGlamal Encryption over EC, group EC (F_p) & over finite field F_p at different security levels. The codes given in this paper can be used as basic guide in the implementation of other elliptic curve cryptosystems.

Appendix A

```
// Point at Infinity is Denoted by (0,0)
#include<stdio.h>
#include<stdlib.h>
#include<gmp.h>

    struct Elliptic_Curve
    {
        mpz_t a;
        mpz_t b;
        mpz_t p;
    };
    struct Point
    {
        mpz_t x;
        mpz_t y;
    };
    struct Elliptic_Curve EC;
main()
{
    int choice;
    mpz_init(EC.a); mpz_init(EC.b);  mpz_init(EC.p);
    Select_EC();
    printf("\n Enter Choice of Operation\n");
    printf("\n Enter 1 For Point Addition Operation\n");
    printf("\n Enter 2 For Scalar Multiplication
    Operation\n");
    scanf("%d",&choice);
    struct Point P,R;
    mpz_init(P.x);    mpz_init(P.y);
    mpz_init_set_ui(R.x,0);    mpz_init_set_ui(R.y,0);
    printf("\n Enter Points P(x,y) and/or Q(x,y) to be
    Added\n");
    gmp_scanf("%Zd",&P.x);
    gmp_scanf("%Zd",&P.y);

    if(choice==1)
    {
    struct Point Q;
    mpz_init(Q.x);    mpz_init(Q.y);
    gmp_scanf("%Zd",&Q.x);
    gmp_scanf("%Zd",&Q.y);
    Point_Addition(P,Q,&R);
    }
    else
    {
    printf("\n Enter m to Find mP\n");
    mpz_t m;
    mpz_init(m);
    gmp_scanf("%Zd",&m);
    Scalar_Multiplication(P,&R,m);
    }
    gmp_printf("\n Resultant Point is %Zd,
    %Zd",R.x,R.y);
}
    Select_EC()
{
    printf("\n Enter Elliptic Curve Parameters i.e. a,b and
    p");
    gmp_scanf("%Zd",&EC.a);
    gmp_scanf("%Zd",&EC.b);
    gmp_scanf("%Zd",&EC.p);
}
Point_Addition(struct Point P,struct Point Q,struct Point
*R)
{
    mpz_mod(P.x,P.x,EC.p);
    mpz_mod(P.y,P.y,EC.p);
    mpz_mod(Q.x,Q.x,EC.p);
    mpz_mod(Q.y,Q.y,EC.p);
    mpz_t temp,slope;
    mpz_init(temp);
    mpz_init_set_ui(slope,0);
        if(mpz_cmp_ui(P.x,0)==0 &&
    mpz_cmp_ui(P.y,0)==0)
```

```
    { mpz_set(R->x,Q.x); mpz_set(R->y,Q.y); return;}
    if(mpz_cmp_ui(Q.x,0)==0 &&
    mpz_cmp_ui(Q.y,0)==0)
        { mpz_set(R->x,P.x); mpz_set(R->y,P.y);
    return;}
    if(mpz_cmp_ui(Q.y,0)!=0)
    {mpz_sub(temp,EC.p,Q.y);mpz_mod(temp,temp,EC
    .p);}
    else
    mpz_set_ui(temp,0);
//  gmp_printf("\n temp=%Zd\n",temp);
    if(mpz_cmp(P.y,temp)==0 &&
    mpz_cmp(P.x,Q.x)==0)
    { mpz_set_ui(R->x,0); mpz_set_ui(R->y,0); return;}
    if(mpz_cmp(P.x,Q.x)==0 &&
    mpz_cmp(P.y,Q.y)==0)
    {
    Point_Doubling(P,R);
    return;
    }
    else
{
        mpz_sub(temp,P.x,Q.x);
        mpz_mod(temp,temp,EC.p);
        mpz_invert(temp,temp,EC.p);
        mpz_sub(slope,P.y,Q.y);
        mpz_mul(slope,slope,temp);
        mpz_mod(slope,slope,EC.p);
        mpz_mul(R->x,slope,slope);
        mpz_sub(R->x,R->x,P.x);
        mpz_sub(R->x,R->x,Q.x);
        mpz_mod(R->x,R->x,EC.p);
        mpz_sub(temp,P.x,R->x);
        mpz_mul(R->y,slope,temp);
        mpz_sub(R->y,R->y,P.y);
        mpz_mod(R->y,R->y,EC.p);
        return;
    }
}

Point_Doubling(struct Point P,struct Point *R)
{
    mpz_t slope,temp;
    mpz_init(temp);
    mpz_init(slope);
    if(mpz_cmp_ui(P.y,0)!=0)
    {
        mpz_mul_ui(temp,P.y,2);
        mpz_invert(temp,temp,EC.p);
        mpz_mul(slope,P.x,P.x);
        mpz_mul_ui(slope,slope,3);
        mpz_add(slope,slope,EC.a);
        mpz_mul(slope,slope,temp);
        mpz_mod(slope,slope,EC.p);
        mpz_mul(R->x,slope,slope);
        mpz_sub(R->x,R->x,P.x);
        mpz_sub(R->x,R->x,P.x);
        mpz_mod(R->x,R->x,EC.p);
        mpz_sub(temp,P.x,R->x);
        mpz_mul(R->y,slope,temp);
        mpz_sub(R->y,R->y,P.y);
        mpz_mod(R->y,R->y,EC.p);
    }
    else
    {
        mpz_set_ui(R->x,0);
        mpz_set_ui(R->y,0);
    }
}

Scalar_Multiplication(struct Point P,struct Point *R,mpz_t
m)
{
    struct Point Q,T;
    mpz_init(Q.x);   mpz_init(Q.y);
    mpz_init(T.x);   mpz_init(T.y);
    long no_of_bits,loop;

    no_of_bits=mpz_sizeinbase(m,2);
    mpz_set_ui(R->x,0);mpz_set_ui(R->y,0);

    if(mpz_cmp_ui(m,0)==0)
    return;
    mpz_set(Q.x,P.x);
    mpz_set(Q.y,P.y);
    if(mpz_tstbit(m,0)==1)
    {mpz_set(R->x,P.x);mpz_set(R->y,P.y);}

    for(loop=1;loop<no_of_bits;loop++)
    {
        mpz_set_ui(T.x,0);
        mpz_set_ui(T.y,0);
        Point_Doubling(Q,&T);
        gmp_printf("\n  %Zd %Zd %Zd %Zd "
                ,Q.x,Q.y,T.x,T.y);
        mpz_set(Q.x,T.x);
        mpz_set(Q.y,T.y);
        mpz_set(T.x,R->x);
        mpz_set(T.y,R->y);
        if(mpz_tstbit(m,loop))
        Point_Addition(T,Q,R);
    }
}
```

Appendix B

```
// does not starts with smallest value of x as
// suggested in algorithm
#include<stdio.h>
#include<stdlib.h>
#include<gmp.h>
#include<math.h>
#include<time.h>
# define RANGE 1000
    struct Elliptic_Curve
    {
        mpz_t a;
        mpz_t b;
        mpz_t p;
    };
    struct Point
    {
        mpz_t x;
        mpz_t y;
    };
    struct Elliptic_Curve EC;
main()
{
    clock_t time1,time2;
    gmp_randstate_t state;
    gmp_randinit_default(state);
    int m;long i;int ret;
    mpz_t exp;
    mpz_init(exp);
    struct Point B,R;


    mpz_init(EC.p);
    mpz_init(EC.b);
    mpz_init(EC.a);
    mpz_init(B.x);
    mpz_init(B.y);
    mpz_init(R.x);
    mpz_init(R.y);
    mpz_t y,p,t,n;
    mpz_init(y);
    mpz_init(p);
    mpz_init(t);
    mpz_init(n);

    mpz_t array[5];
    for(i=0;i<5;i++)
    mpz_init(array[i]);

    int P[5]={1,6,24,36,36};
    scanf("%d",&m);
    mpz_t x,two;
    mpz_init(x);
    mpz_init_set_ui(two,2);
    time1=clock();
    mpz_pow_ui(x,two,m/4);
    mpz_t negx,temp;
    mpz_init(negx);mpz_init(temp);
    mpz_neg(negx,x);
    evaluate(P,4,negx,temp,array);
    i=1;
    mpz_t x1;mpz_init(x1);
    mpz_t x2;mpz_init(x2);
    mpz_pow_ui(x1,two,m/4);
    while(mpz_sizeinbase(temp,2)>=m)
    {
        mpz_pow_ui(x2,two,m/4-i);
        i++;
        mpz_neg(negx,x2);
        evaluate(P,4,negx,temp,array);
    }
    while(mpz_sizeinbase(temp,2)!=m)
    {
        mpz_add(x,x2,x1);
        mpz_div_ui(x,x,2);
//      gmp_printf("\n m=%d temp=%d temp=%Zd \n",
        m,mpz_sizeinbase(temp,2),temp);
        mpz_neg(negx,x);
        evaluate(P,4,negx,temp,array);
        if(mpz_sizeinbase(temp,2)<m)
        mpz_set(x2,x);
        else
        mpz_set(x1,x);
    }
    int ctr=0;
    while(mpz_sizeinbase(temp,2)==m)
    {
        ctr++;
        mpz_sub_ui(x,x,1);
        mpz_neg(negx,x);


        evaluate(P,4,negx,temp,array);
        if(ctr>RANGE)
        break;
    }
    mpz_add_ui(x,x,1);
    mpz_neg(negx,x);
    evaluate(P,4,negx,temp,array);
    fflush(stdout);
    ctr=0;
    while(1)
    {
        ctr++;
        mpz_mul(t,x,x);
        mpz_mul_ui(t,t,6);
        mpz_add_ui(t,t,1);
        evaluate(P,4,negx,p,array);
        mpz_add_ui(n,p,1);
        mpz_sub(n,n,t);
        if(mpz_probab_prime_p(p,1000) &&
           mpz_probab_prime_p(n,1000))
        break;
```

```
            evaluate_neg(p,array);
            mpz_add_ui(n,p,1);
            mpz_sub(n,n,t);
            if(mpz_probab_prime_p(p,1000) &&
               mpz_probab_prime_p(n,1000))
            break;
            mpz_add_ui(x,x,1);
            mpz_neg(negx,x);
        }
        mpz_set(EC.p,p);
        mpz_set_ui(EC.a,0);
        mpz_set_ui(EC.b,0);
        do
        {
            do
            {
            mpz_add_ui(EC.b,EC.b,1);
            mpz_add_ui(temp,EC.b,1);
            }
            while(mpz_legendre(temp,EC.p)!=1);
            ret=root(temp,EC.p,&y,1);
//          gmp_printf("\n temp=%Zd y=%Zd
            EC.b=%Zd\n", temp,y,EC.b);
            mpz_set_ui(B.x,1);
            fflush(stdout);
            mpz_set(B.y,y);
            mpz_set_ui(R.x,1);
            mpz_set(R.y,y);
            mpz_set(exp,n);
            Scalar_Multiplication(B,&R,exp);
            if(mpz_cmp_ui(R.x,0)==0 &&
               mpz_cmp_ui(R.y,0)==0)
            ret=1;
            else
            ret=0;
        }
        while(ret!=1);



        time2=clock();
        gmp_printf("\n p=%Zd t=%Zd n=%Zd B.x=%Zd
        B.y=%Zd y=%Zd b=%Zd\n",EC.p,t,n,B.x,B.y,y,EC.b);
        printf("\n Time Taken is %f \n",(float) (time2-
        time1)/CLOCKS_PER_SEC);
        FILE *fp;
        fp=fopen("Domain_Parameters","w");
        gmp_fprintf(fp,"%Zd\n",EC.p);
        gmp_fprintf(fp,"0\n");
        gmp_fprintf(fp,"%Zd\n",EC.b);
        gmp_fprintf(fp,"%Zd\n",B.x);
        gmp_fprintf(fp,"%Zd\n",B.y);
        gmp_fprintf(fp,"%Zd\n",n);
        fclose(fp);
}
Point_Addition(struct Point P,struct Point Q,
struct Point *R)
{
    mpz_t temp,slope;
    mpz_init(temp);
    mpz_init(slope);
    if(mpz_cmp_ui(P.x,0)==0 && mpz_cmp_ui(P.y,0)==0)
    { mpz_set(R->x,Q.x); mpz_set(R->y,Q.y);
    mpz_clear(temp);mpz_clear(slope);return;}
    if(mpz_cmp_ui(Q.x,0)==0 && mpz_cmp_ui(Q.y,0)==0)
    { mpz_set(R->x,P.x); mpz_set(R->y,P.y);
    mpz_clear(temp);mpz_clear(slope);return;}
    if(mpz_cmp_ui(Q.y,0)!=0)
    {mpz_sub(temp,EC.p,Q.y);mpz_mod(temp,temp,
    EC.p);}
    else
    mpz_set_ui(temp,0);
    if(mpz_cmp(P.y,temp)==0 &&
    mpz_cmp(P.x,Q.x)==0)
    { mpz_set_ui(R->x,0); mpz_set_ui(R->y,0);
    mpz_clear(temp);mpz_clear(slope);return;}
    if(mpz_cmp(P.x,Q.x)==0 && mpz_cmp(P.y,Q.y)==0)

    {Point_Doubling(P,R);mpz_clear(temp);mpz_clear(sl
    ope);return;}
    else
    {
        mpz_sub(temp,P.x,Q.x);
        mpz_invert(temp,temp,EC.p);
        mpz_sub(slope,P.y,Q.y);
        mpz_mul(slope,slope,temp);
        mpz_mod(slope,slope,EC.p);
        mpz_mul(R->x,slope,slope);
        mpz_sub(R->x,R->x,P.x);
        mpz_sub(R->x,R->x,Q.x);
        mpz_mod(R->x,R->x,EC.p);
        mpz_sub(temp,P.x,R->x);
        mpz_mul(R->y,slope,temp);
        mpz_sub(R->y,R->y,P.y);
        mpz_mod(R->y,R->y,EC.p);
        mpz_clear(temp);



        mpz_clear(slope);
        return;
    }
}
Point_Doubling(struct Point P,struct Point *R)
{
    mpz_t slope,temp;
    mpz_init(temp);
    mpz_init(slope);
    if(mpz_cmp_ui(P.y,0)!=0)
    {
        mpz_mul_ui(temp,P.y,2);
```

```
        mpz_invert(temp,temp,EC.p);
        mpz_mul(slope,P.x,P.x);
        mpz_mul_ui(slope,slope,3);
        mpz_add(slope,slope,EC.a);
        mpz_mul(slope,slope,temp);
        mpz_mod(slope,slope,EC.p);
        mpz_mul(R->x,slope,slope);
        mpz_sub(R->x,R->x,P.x);
        mpz_sub(R->x,R->x,P.x);
        mpz_mod(R->x,R->x,EC.p);
        mpz_sub(temp,P.x,R->x);
        mpz_mul(R->y,slope,temp);
        mpz_sub(R->y,R->y,P.y);
        mpz_mod(R->y,R->y,EC.p);
    }
    else
    {
        mpz_set_ui(R->x,0);
        mpz_set_ui(R->y,0);
    }
    mpz_clear(temp);
    mpz_clear(slope);
}

Scalar_Multiplication(struct Point P,struct Point
*R,mpz_t  m)
{
    struct Point Q,T;
    mpz_init(Q.x);  mpz_init(Q.y);
    mpz_init(T.x);  mpz_init(T.y);
    long no_of_bits,loop;
    no_of_bits=mpz_sizeinbase(m,2);
    mpz_set_ui(R->x,0);mpz_set_ui(R->y,0);

    if(mpz_cmp_ui(m,0)==0)
    return;

    mpz_set(Q.x,P.x);
    mpz_set(Q.y,P.y);
    if(mpz_tstbit(m,0)==1)
    {mpz_set(R->x,P.x);mpz_set(R->y,P.y);}
    for(loop=1;loop<no_of_bits;loop++)
    {
        mpz_set_ui(T.x,0);
        mpz_set_ui(T.y,0);



        Point_Doubling(Q,&T);
        mpz_set(Q.x,T.x);
        mpz_set(Q.y,T.y);
        mpz_set(T.x,R->x);
        mpz_set(T.y,R->y);
        if(mpz_tstbit(m,loop))
        Point_Addition(T,Q,R);
    }
}
root(mpz_t x,mpz_t p,mpz_t *s,int n)
{
    int i, ret;
    for(i=1;i<=n;i++)
    {
    if(mpz_cmp_ui(p,2)==0)
    ret=tworoot(x,p,s,i);
    else
    ret=sqroot(x,p,s,i);
    }
    return(ret);
}
sqroot(mpz_t r,mpz_t p,mpz_t *s,int n)
{
    int i,ret;
    mpz_t f,pn,op,po,pn1,ch,x;
    mpz_init(f);
    mpz_init(x);
    mpz_init(ch);
    mpz_init(pn);
    mpz_init(op);
    mpz_init(po);
    mpz_init(pn1);
    mpz_pow_ui(pn,p,n);
    mpz_mod(x,r,pn);
    if(mpz_legendre(x,p)==-1)
    return(0);
    if(n==1)
    {
        mpz_sub_ui(ch,p,3);
        if(mpz_divisible_ui_p(ch,4)!=0)
        case1(x,p,s);
        else
        {
                mpz_sub_ui(ch,p,5);
                if(mpz_divisible_ui_p(ch,8)!=0)
                {
                        ret=case2(x,p,s);
                        if(ret!=1)
                        squareroot(x,p,s);
                }
                else
                squareroot(x,p,s);
        }
    }
    else
    {
                mpz_pow_ui(pn,p,n-1);



                mpz_mul_ui(f,s[0],2);
                mpz_invert(f,f,pn);
```

```
                mpz_mul(op,s[0],s[0]);
                mpz_sub(po,x,op);
                mpz_cdiv_q(po,po,pn);
                mpz_mul(po,po,f);
                mpz_mod(po,po,p);
                mpz_mul(op,po,pn);
                mpz_add(s[0],s[0],op);
                mpz_pow_ui(pn1,p,n);
                mpz_mod(s[0],s[0],pn1);
    }
    mpz_clear(f);
    mpz_clear(x);
    mpz_clear(ch);
    mpz_clear(pn);
    mpz_clear(op);
    mpz_clear(po);
    mpz_clear(pn1);
    return(1);
}
squareroot(mpz_t a,mpz_t p,mpz_t *s)
{
    mpz_t i,r,j,e,d,c,b,w,t;
    mpz_init(r);
    mpz_init(i);
    mpz_init(j);
    mpz_init(e);
    mpz_init(d);
    mpz_init(b);
    mpz_init(w);
    mpz_init(t);
    mpz_init(c);
    mpz_set_ui(b,1);
    while(mpz_legendre(b,p)!=-1)
    mpz_add_ui(b,b,1);
    mpz_sub_ui(t,p,1);
    mpz_set_ui(w,0);
    mpz_mod_ui(j,t,2);
    while(mpz_cmp_ui(j,0) ==0)
    {
        mpz_add_ui(w,w,1);
        mpz_div_ui(t,t,2);
        mpz_mod_ui(j,t,2);
    }
    mpz_invert(i,a,p);
    mpz_powm(c,b,t,p);
    mpz_add_ui(t,t,1);
    mpz_cdiv_q_ui(t,t,2);
    mpz_powm(r,a,t,p);
    while(mpz_cmp(j,w)<0)
    {
        mpz_sub(e,w,j);
        mpz_sub_ui(e,e,1);
        mpz_set_ui(t,2);
        mpz_powm(e,t,e,p);
        mpz_mul(b,r,r);
        mpz_mul(d,b,i);
        mpz_powm(d,d,e,p);
        mpz_add_ui(d,d,1);
        if(mpz_divisible_p(d,p)!=0)
        {
            mpz_mul(r,r,c);
            mpz_mod(r,r,p);
        }
        mpz_mul(c,c,c);
        mpz_mod(c,c,p);
        mpz_add_ui(j,j,1);
    }
    mpz_set(s[0],r);
    mpz_clear(r);
    mpz_clear(i);
    mpz_clear(j);
    mpz_clear(e);
    mpz_clear(d);
    mpz_clear(b);
    mpz_clear(w);
    mpz_clear(t);
    mpz_clear(c);
    return;
}
case1(mpz_t a,mpz_t p,mpz_t *s)
{
    mpz_t x,r;
    mpz_init(r);
    mpz_init(x);
    mpz_add_ui(x,p,1);
    mpz_div_ui(x,x,4);
    mpz_powm(r,a,x,p);
    mpz_set(s[0],r);
    mpz_clear(r);
    mpz_clear(x);
    return;
}
case2(mpz_t a,mpz_t p,mpz_t *s)
{
mpz_t x,b,c,r,d;
mpz_init(r);
mpz_init(b);
mpz_init(c);
mpz_init(x);
mpz_init(d);
mpz_sub_ui(x,p,1);
mpz_div_ui(x,x,4);
mpz_powm(d,a,x,p);
if(mpz_cmp_ui(d,1)==0)
{
        mpz_add_ui(x,p,3);
        mpz_div_ui(x,x,8);
        mpz_powm(r,a,x,p);
        mpz_set(s[0],r);
```

```
mpz_clear(r);
mpz_clear(b);
mpz_clear(c);



mpz_clear(x);
mpz_clear(d);
return(1);
}
mpz_sub_ui(x,p,1);
if(mpz_cmp(d,x)==0)
{
        mpz_mul_ui(b,a,4);
        mpz_sub_ui(x,p,5);
        mpz_div_ui(x,x,8);
        mpz_powm(b,b,x,p);
        mpz_mul_ui(c,a,2);
        mpz_mul(r,c,b);
        mpz_mod(r,r,p);
        mpz_set(s[0],r);
mpz_clear(r);
mpz_clear(b);
mpz_clear(c);
mpz_clear(x);
mpz_clear(d);
return(1);
}
mpz_clear(r);
mpz_clear(b);
mpz_clear(c);
mpz_clear(x);
mpz_clear(d);

return(0);
}
tworoot(mpz_t x,mpz_t p,mpz_t *s,int n)
{
    mpz_t pn;
    mpz_t pr,pr1;
    mpz_init(pn);
    mpz_init(pr1);
    mpz_init(pr);
    mpz_pow_ui(pn,p,n);
    if(n==1 || n==2 || n==3)
    {
        if(n==1)
        {
                mpz_mod_ui(pr1,x,2);
                if(mpz_cmp_ui(pr1,0)==0)
                return(0);
                else
                {
                        mpz_set_ui(s[0],1);
                        mpz_clear(pn);
                        mpz_clear(pr1);
                        mpz_clear(pr);
                        return(1);
                }
        }
        if(n==2)
        {
                mpz_mod_ui(pr1,x,2);


                if(mpz_cmp_ui(pr1,0)==0)
                {
                        mpz_clear(pn);
                        mpz_clear(pr1);
                        mpz_clear(pr);
                        return(0);
                }
                else
                {
                        mpz_sub_ui(pr1,x,1);
                        mpz_mod_ui(pr1,x,4);
                        if(mpz_cmp_ui(pr1,0)==0)
                        {
                           mpz_set_ui(s[0],1);
                           {
                                mpz_clear(pn);
                                mpz_clear(pr1);
                                mpz_clear(pr);
                                return(1);
                           }
                        }
                        else
                        {
                                mpz_clear(pn);
                                mpz_clear(pr1);
                                mpz_clear(pr);
                                return(0);
                        }
                }

        }
        if(n==3)
        {
                mpz_sub_ui(pr1,x,1);
                mpz_mod_ui(pr1,pr1,8);
                if(mpz_cmp_ui(pr1,0)!=0)
                {
                        mpz_clear(pn);
                        mpz_clear(pr1);
                        mpz_clear(pr);
                        return(0);
                }
                else
                {
                        mpz_set_ui(s[0],1);
                        mpz_clear(pn);
```

```
                    mpz_clear(pr1);
                    mpz_clear(pr);
                    return(1);
                }
            }
        }
        else
        {
                mpz_sub_ui(pr1,x,1);
                mpz_mod_ui(pr1,pr1,8);
                if(mpz_cmp_ui(pr1,0)!=0)
                {
                    mpz_clear(pn);
                    mpz_clear(pr1);
                    mpz_clear(pr);
                    return(0);
                }
        mpz_pow_ui(pr,s[0],3);
        mpz_sub_ui(pr1,x,2);
        mpz_mul(pr1,pr1,s[0]);
        mpz_sub(s[0],pr,pr1);
        mpz_div_ui(s[0],s[0],2);
        mpz_mod(s[0],s[0],pn);
        mpz_clear(pn);
        mpz_clear(pr1);
        mpz_clear(pr);
        return(1);
        }
}
evaluate(int *Poly1, int Degree_K, mpz_t val,mpz_t
        eval,mpz_t *array)
{
    int i;
    mpz_t tt;
    mpz_init_set_ui(tt,1);

    mpz_set_ui(eval,Poly1[0]);
    mpz_set_ui(array[0],1);
    for(i=1;i<=Degree_K;i++)
    {
        mpz_mul(tt,tt,val);
        mpz_mul_ui(array[i],tt,Poly1[i]);
        mpz_add(eval,eval,array[i]);
    }
    mpz_clear(tt);
    return;
}
evaluate_neg(mpz_t eval,mpz_t *array)
{
    mpz_set_ui(eval,0);
    int i;
    for(i=0;i<5;i++)
    {
        if(i%2==0)
        mpz_add(eval,eval,array[i]);
        else
        mpz_sub(eval,eval,array[i]);
    }
}
evaluate_dec(mpz_t eval,mpz_t *array)
{
    mpz_t temp;mpz_init(temp);
    mpz_add_ui(eval,array[0],102);
    mpz_mul_ui(temp,array[1],51);
    mpz_add(eval,eval,temp);
    mpz_mul_ui(temp,array[2],29);
    mpz_div_ui(temp,temp,2);
    mpz_add(eval,eval,temp);

    mpz_mul_ui(temp,array[3],5);
    mpz_add(eval,eval,temp);
    mpz_add(eval,eval,array[4]);
    mpz_clear(temp);
}
```

Appendix C

```
// Point at Infinity is Denoted by (0,0)
#include<stdio.h>
#include<stdlib.h>
#include<gmp.h>
#include<time.h>

main()
{
    clock_t time1,time2;double Total_Time;
    int choice;
    mpz_t p,g;
    mpz_init(p);
    mpz_init(g);

    FILE *fileptr;
    fileptr=fopen("Domain_Parameters_Zp","r");
    gmp_fscanf(fileptr,"%Zd",&p);
    gmp_fscanf(fileptr,"%Zd",&g);
    fclose(fileptr);
//  printf("\n Enter Your Choice\n");
//  printf("\n Enter 1 to Encrypt\n");
//  printf("\n Enter 2 to Decrypt\n");
//  scanf("%d",&choice);
    long Iterations,i;
    printf("\n Enter Number Of Iterations\n");
    scanf("%ld",&Iterations);
    time1=clock();
    for(i=0;i<Iterations;i++)
    {
//  if(choice==1)
    Encrypt(p,g);
//  else
```

```
     Decrypt(p,g);
     }
     time2=clock();
     Total_Time=(double)(time2-time1)
     /CLOCKS_PER_SEC;
     printf("\n Total Time=%lf\n",Total_Time);
}

Encrypt(mpz_t p,mpz_t g)
{
          mpz_t Randomization_Param,p_minus_2,.
          Public_Key;
          mpz_init_set_ui(Randomization_Param,0);
          mpz_init(p_minus_2);
          mpz_init(Public_Key);
          mpz_sub_ui(p_minus_2,p,2);
          mpz_t C1,C2;



          mpz_init_set_ui(C1,0);
          mpz_init_set_ui(C2,0);

     mpz_t M;
     mpz_init(M);

     FILE *fileptr;char File_Name[20];
//   printf("\n Enter File Name Where Embedded
     Message is Stored\n");
//   scanf("%s",&File_Name);
//   fileptr=fopen(File_Name,"r");
     fileptr=fopen("Message","r");
     gmp_fscanf(fileptr,"%Zd",&M);
     fclose(fileptr);
//   gmp_printf("\n M=%Zd  \n",M);
     gmp_randstate_t state;
     gmp_randinit_default(state);
     gmp_randseed_ui(state,random());
//   gmp_printf("\np_minus_2 = %Zd \n",p_minus_2);
     while(mpz_cmp_ui(Randomization_Param,0)==0)
     mpz_urandomm(Randomization_Param,
     state,p_minus_2);
//   gmp_printf("\nRandomization_Parm =
     %Zd\n",Randomization_Param);
     mpz_powm(C1,g,Randomization_Param,p);
//   printf("\n Enter File Name Where Public Key is Stored\n");
//   scanf("%s",&File_Name);
//   fileptr=fopen(File_Name,"r");
     fileptr=fopen("richa","r");
     gmp_fscanf(fileptr,"%Zd",&Public_Key);
     fclose(fileptr);
     mpz_powm(C2,Public_Key,
     Randomization_Param,p);
     mpz_mul(C2,C2,M);
     mpz_mod(C2,C2,p);
     fileptr=fopen("encryption","w");
     gmp_fprintf(fileptr,"%Zd ",C1);
     gmp_fprintf(fileptr,"%Zd ",C2);
     fclose(fileptr);
}

Decrypt(mpz_t p,mpz_t g)
{
     mpz_t Secret_Key;
     mpz_init(Secret_Key);
     FILE *fileptr;char File_Name[20];
//   printf("\n Enter File Name Where Secret Key is Stored\n");
//   scanf("%s",&File_Name);
//   fileptr=fopen(File_Name,"r");
     fileptr=fopen("richa_sec","r");
     gmp_fscanf(fileptr,"%Zd",&Secret_Key);
     fclose(fileptr);
//   gmp_scanf("%Zd",&Secret_Key);

     mpz_t M,C1,C2,I;
     mpz_init_set_ui(M,0);
     mpz_init_set_ui(C1,0);
     mpz_init_set_ui(C2,0);
     mpz_init_set_ui(I,0);
     mpz_sub_ui(I,p,1);
     mpz_sub(I,I,Secret_Key);
//   printf("\n Enter File Name Where Encrypted
     Message is Stored\n");
//   scanf("%s",&File_Name);
//   fileptr=fopen(File_Name,"r");
     fileptr=fopen("encryption","r");
     gmp_fscanf(fileptr,"%Zd",&C1);
     gmp_fscanf(fileptr,"%Zd",&C2);
     fclose(fileptr);

     mpz_powm(M,C1,I,p);
     mpz_mul(M,M,C2);
     mpz_mod(M,M,p);
//   gmp_printf("\n M=%Zd \n",M);

}
```

REFERENCES

[1] Rong-Jaye Chen, The MOV Attack, ECC 2008 available at http://www.cs.nctu.edu.tw/~rjchen/ECC2009/19_MOVattack.pdf.

[2] W. Diffie and M. E. Hallman. New Directions in Cryptography. *IEEE Transactions on Information Theory,* IT-22:644-654, 1976 available at http://securespeech.cs.cmu.edu/reports/DiffieHellman.pdf.

[3] ElGamal Encryption available at http://en.wikipedia.org/wiki/ElGamal_encryption

[4] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. Handbook of Applied

Cryptography. CRC Press. October 16, 1996. ISBN: 0849385237.

[5] N., Koblitz, 1987. Elliptic curve cryptography. *Mathematics of Computation*, **48**: 203-209.

[6] Anoop MS. Elliptic Curve Cryptography-An Implementation Guide available at http://www.tataelxsi.com/whitepapers/ECC_Tut_v1_0.pdf?pdf_id=public_key_TEL.pdf .

[7] S., Miller, 1986. Use of Elliptic Curves in Cryptography. In CRYPTO ’85, pp: 417-426.

[8] Elisabeth Oswald, Introduction to Elliptic Curve Cryptography.

[9] Kefa Rabah. Theory and Implementation of Elliptic Curve Cryptography**.** *Journal of Applied Sciences* **5**(4): 604-633, 2005. available at http://docsdrive.com/pdfs/ansinet/jas/2005/604-633.pdf

[10] Paulo S.L. M. Barreto and Michael Naehrig. Pairing-Friendly Elliptic Curves of Prime Order. Springer-Verlag, pp 319—331, 2006.

[11] Nagoya Torrid & Kazuhiro Yokoyama, ‘Elliptic curve cryptosystem’. *FUJITSU Sci. Tech. J.,* **36**(2), pp. 140-146, December 2000.