LAB – 9

Name : Shubham Pareshbhai Shingala

Roll no. : CE146

College ID: 19CEUOS159

Aim: Write a program to implement Elliptical Curve Cryptography.

- Key Generation
- Encryption
- Decryption

❖ **Program :** Elliptical Curve Cryptography

➢ **Source Code:**

```
#include <bits/stdc++.h>
#define ll long long
#define v vector<ll>
#define loop(var, s, n) for (ll var = s; var < n; var++)
#define rloop(var, s, n) for (ll var = s; var >= n; var--)
#define pb push_back
using namespace std;

ll a, b, p, n;

class Point
{
public:
    ll x, y;
    Point(ll X, ll Y)
    {
        x = X;
        y = Y;
    }

    void print()
```

```
    {
        cout << "(" << x << ',' << y << ")\n";
    }
};

ll squareMultiply(ll base, ll exp, ll mod) // base^exp(% mod)
{
    ll z = 1;
    while (exp > 0)
    {
        if (exp % 2 == 1)
            z = (z * base) % mod;
        exp = exp / 2;
        base = (base * base) % mod;
    }
    return z;
}

bool isPrime(ll n)
{
    // Corner cases
    if (n <= 1)
        return false;
    if (n <= 3)
        return true;

    // This is checked so that we can skip
    // middle five numbers in below loop
    if (n % 2 == 0 || n % 3 == 0)
        return false;

    for (int i = 5; i * i <= n; i += 6)
        if (n % i == 0 || n % (i + 2) == 0)
            return false;

    return true;
}

ll mod(ll a, ll b)
{
    ll mode = a % b;
    if (mode < 0)
        mode += b;
    return mode;
}
```

```
ll multiplicativeInverse(ll a, ll b)
{
    ll q, r, t, t1 = 0, t2 = 1, r1 = b, r2 = a;
    while (r2 > 0)
    {
        q = r1 / r2;
        r = r1 - q * r2;
        r1 = r2;
        r2 = r;
        t = t1 - q * t2;
        t1 = t2;
        t2 = t;
    }
    if (r1 == 1)
    {
        if (t1 < 0)
            t1 += b;
        return t1;
    }
    else
        return -1;
}

ll randomNumberInRange(ll n, ll m) // m not included and n included
{
    srand(time(0));
    ll random = n + rand() % (m - n - 1);
    return random;
}

Point operator+(Point p1, Point p2)
{
    Point p3(0, 0);
    int x1 = p1.x, x2 = p2.x, y1 = p1.y, y2 = p2.y;
    int lamda;
    if (x1 != x2 && y1 != y2)
    {
        int dx = x2 - x1, dy = y2 - y1;
        if (dx < 0)
        {
            dy = (-1) * dy;
            dx = (-1) * dx;
        }
```

```
        lamda = mod(dy * multiplicativeInverse(dx, p), p); ///((y2-
y1)/(x2-x1))mode p
    }
    else if (x1 == x2 && y1 == y2)
    {
        lamda = mod((3 * x1 * x1 + a) * multiplicativeInverse(2 * y1,
p), p);
    }
    p3.x = mod((lamda * lamda - x1 - x2), p); // x3=(lamda^2 - x1 -
x2)mode p
    p3.y = mod(lamda * (x1 - p3.x) - y1, p);  // y3=(lamda(x1-x3) -
y1)mode p
    return p3;
}

Point operator*(Point p1, int n)
{
    Point ans(p1.x, p1.y);
    n--;
    while (n--)
        ans = ans + p1;
    return ans;
}

bool isPointOnCurve(Point p, vector<Point> points)
{
    loop(i, 0, points.size())
    {
        if (p.x == points[i].x && p.y == points[i].y)
            return true;
    }
    return false;
}

vector<Point> pointGeneration()
{
    vector<Point> points;
    loop(x, 0, p)
    {
        ll y_square = mod((x * x * x) + (a * x) + b, p);
        ll r = squareMultiply(y_square, (p - 1) / 2, p);
        if (r == 1)
        {
            ll y = sqrt(y_square);
            while (y * y != y_square)
```

```
            {
                y_square += p;
                y = sqrt(y_square);
            }
            ll y1 = mod(-y, p);
            points.pb(Point(x, y));
            points.pb(Point(x, y1));
        }
        else if (r == 0)
            points.pb(Point(x, 0));
    }
    return points;
}

vector<Point> keyGeneration(int &d)
{
    vector<Point> points = pointGeneration(), e;
    n = points.size();
    int index = randomNumberInRange(0, n);
    Point e1 = points[index];
    // Point e1(1,26);
    d = randomNumberInRange(1, 5);
    // d = 4;
    Point e2 = e1 * d;
    while (!isPointOnCurve(e2, points))
    {
        index = randomNumberInRange(0, n);
        e1 = points[index];
        e2 = e1 * d;
    }
    e.pb(e1);
    e.pb(e2);
    return e;
}

vector<Point> encrypt(Point e1, Point e2, Point m)
{
    vector<Point> c;
    int r = randomNumberInRange(1, 5);
    // int r = 1;
    cout << "r = " << r << endl;
    Point c1 = e1 * r;
    Point c2 = m + e2 * r;
    c.pb(c1);
    c.pb(c2);
```

```
    return c;
}

Point decrypt(Point c1, Point c2, int d)
{
    Point t = c1 * d;
    return (c2 + Point(t.x, (-1) * t.y)); // c2 + inverse of t
}

int main()
{
    cout << "Enter a and b :";
    cin >> a >> b;

    while (1)
    {
        cout << "Enter prime number: ";
        cin >> p;
        if (!isPrime(p))
            cout << p << " is not a prime number so , ";
        else
            break;
    }

    int d;
    vector<Point> e = keyGeneration(d);
    Point e1 = e[0], e2 = e[1];
    cout << "e1 = ";
    e1.print();
    cout << "e2 = ";
    e2.print();
    cout << "d = " << d;
    int x, y;

    cout << "\nEnter the message : ";
    cin>>x>>y;
    Point m(x, y);

    cout << "Message = ";
    m.print();
    vector<Point> c = encrypt(e1, e2, m);
    Point c1 = c[0], c2 = c[1];
    cout << "c1 = ";
    c1.print();
    cout << "c2 = ";
```

```
    c2.print();
    Point msg = decrypt(c1, c2, d);
    cout << "Decrypt Message = ";
    msg.print();
}
```

- **Test Case – 1:**

```
d:\Semaster6\NIS\Labs\Lab9>ECC.exe
Enter a and b :1 1
Enter prime number: 13
e1 = (11,11)
e2 = (10,7)
d = 3
Enter the message : 9 7
Message = (9,7)
r = 2
c1 = (4,2)
c2 = (8,5)
Decrypt Message = (9,7)
```

Test Case – 2:

```
d:\Semaster6\NIS\Labs\Lab9>ECC.exe
Enter a and b :43 62
Enter prime number: 113
e1 = (0,66)
e2 = (75,100)
d = 3
Enter the message : 42 95
Message = (42,95)
r = 2
c1 = (7,49)
c2 = (23,73)
Decrypt Message = (42,95)
```

- **Test Case – 3:**

```
d:\Semaster6\NIS\Labs\Lab9>ECC.exe
Enter a and b :34 31
Enter prime number: 52
52 is not a prime number so , Enter prime number: 85
85 is not a prime number so , Enter prime number: 29
e1 = (5,6)
e2 = (5,6)
d = 1
Enter the message : 22 13
Message = (22,13)
r = 3
c1 = (21,1)
c2 = (14,25)
Decrypt Message = (22,13)
```