EX-NO-11-ELLIPTIC-CURVE-CRYPTOGRAPHY-ECC

Aim:

To Implement ELLIPTIC CURVE CRYPTOGRAPHY(ECC)

ALGORITHM:

1. Elliptic Curve Cryptography (ECC) is a public-key cryptography technique based on the algebraic structure of elliptic curves over finite fields.

2. Initialization:

   - Select an elliptic curve equation (y^2 = x^3 + ax + b) with parameters (a) and (b), along with a large prime (p) (defining the finite field).
   - Choose a base point (G) on the curve, which will be used for generating public keys.

3. Key Generation:

   - Each party selects a private key (d) (a random integer).
   - Calculate the public key as (Q = d \times G) (using elliptic curve point multiplication).

4. Encryption and Decryption:

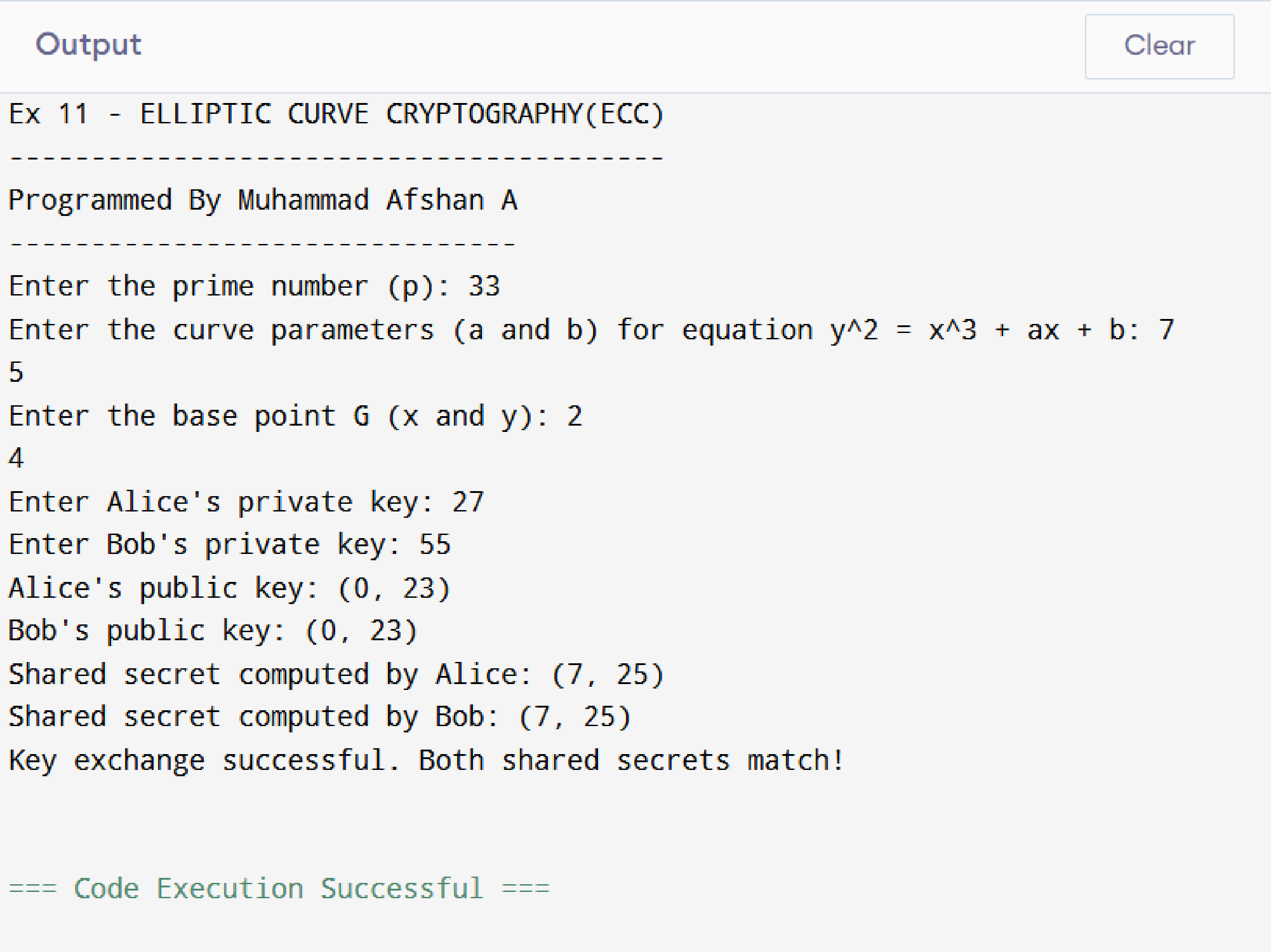
   - Encryption: The sender uses the recipient's public key and the base point (G) to encode the message.
   - Decryption: The recipient uses their private key to decode the message and retrieve the original plaintext.

5. Security: ECC's security relies on the Elliptic Curve Discrete Logarithm Problem (ECDLP), making it highly secure with shorter key lengths compared to traditional methods like RSA.

Program:

```
#include <stdio.h>

// Define a structure for points on the elliptic curve
typedef struct {
    long long int x, y;
} Point;

// Function to compute modular inverse using Extended Euclidean Algorithm
long long int modInverse(long long int a, long long int m) {
    long long int m0 = m, t, q;
    long long int x0 = 0, x1 = 1;
    if (m == 1) return 0;

    while (a > 1) {
        q = a / m;
        t = m;
        m = a % m;
        a = t;
        t = x0;
        x0 = x1 - q * x0;
        x1 = t;
    }

    if (x1 < 0) x1 += m0;
    return x1;
}

// Function for point addition on the elliptic curve
Point pointAddition(Point P, Point Q, long long int a, long long int p) {
    Point R;
    long long int lambda;

    if (P.x == Q.x && P.y == Q.y) {
        lambda = (3 * P.x * P.x + a) * modInverse(2 * P.y, p) % p;
    } else {
        lambda = (Q.y - P.y) * modInverse(Q.x - P.x, p) % p;
    }

    R.x = (lambda * lambda - P.x - Q.x) % p;
    R.y = (lambda * (P.x - R.x) - P.y) % p;

    R.x = (R.x + p) % p;
    R.y = (R.y + p) % p;

    return R;
}

// Function for scalar multiplication (repeated addition)
Point scalarMultiplication(Point P, long long int k, long long int a, long long int p) {
    Point result = P;
    k--; // Because we start with the base point

    while (k > 0) {
        result = pointAddition(result, P, a, p);
        k--;
    }

    return result;
}

// Main function
int main() {
    printf("Ex 11 - ELLIPTIC CURVE CRYPTOGRAPHY(ECC)\n");
    printf("----------------------------------------\n");
    printf("Programmed By Muhammad Afshan A\n");
    printf("-------------------------------\n");

    long long int p, a, b, privateA, privateB;
    Point G, publicA, publicB, sharedSecretA, sharedSecretB;

    printf("Enter the prime number (p): ");
    scanf("%lld", &p);
    printf("Enter the curve parameters (a and b) for equation y^2 = x^3 + ax + b: ");
    scanf("%lld %lld", &a, &b);

    printf("Enter the base point G (x and y): ");
    scanf("%lld %lld", &G.x, &G.y);

    printf("Enter Alice's private key: ");
    scanf("%lld", &privateA);

    printf("Enter Bob's private key: ");
    scanf("%lld", &privateB);

    publicA = scalarMultiplication(G, privateA, a, p); // Alice's public key
    publicB = scalarMultiplication(G, privateB, a, p); // Bob's public key

    printf("Alice's public key: (%lld, %lld)\n", publicA.x, publicA.y);
    printf("Bob's public key: (%lld, %lld)\n", publicB.x, publicB.y);

    sharedSecretA = scalarMultiplication(publicB, privateA, a, p); // Alice's shared secret
    sharedSecretB = scalarMultiplication(publicA, privateB, a, p); // Bob's shared secret

    printf("Shared secret computed by Alice: (%lld, %lld)\n", sharedSecretA.x, sharedSecretA
    printf("Shared secret computed by Bob: (%lld, %lld)\n", sharedSecretB.x, sharedSecretB.y

    if (sharedSecretA.x == sharedSecretB.x && sharedSecretA.y == sharedSecretB.y) {
        printf("Key exchange successful. Both shared secrets match!\n");
    } else {
        printf("Key exchange failed. Shared secrets do not match.\n");
    }

    return 0;
}
```

Output:



```
Ex 11 - ELLIPTIC CURVE CRYPTOGRAPHY(ECC)
----------------------------------------
Programmed By Muhammad Afshan A
-------------------------------
Enter the prime number (p): 33
Enter the curve parameters (a and b) for equation y^2 = x^3 + ax + b: 7
5
Enter the base point G (x and y): 2
4
Enter Alice's private key: 27
Enter Bob's private key: 55
Alice's public key: (0, 23)
Bob's public key: (0, 23)
Shared secret computed by Alice: (7, 25)
Shared secret computed by Bob: (7, 25)
Key exchange successful. Both shared secrets match!


=== Code Execution Successful ===
```

Result:

The program is executed successfully