National Institute of Technology, Tiruchirappalli

Department of Computer Applications

Information Security Lab

Lab 5

Submitted to:

Dr. Ghanshyam Bopche

Submitted by:

Akriti Upadhyay

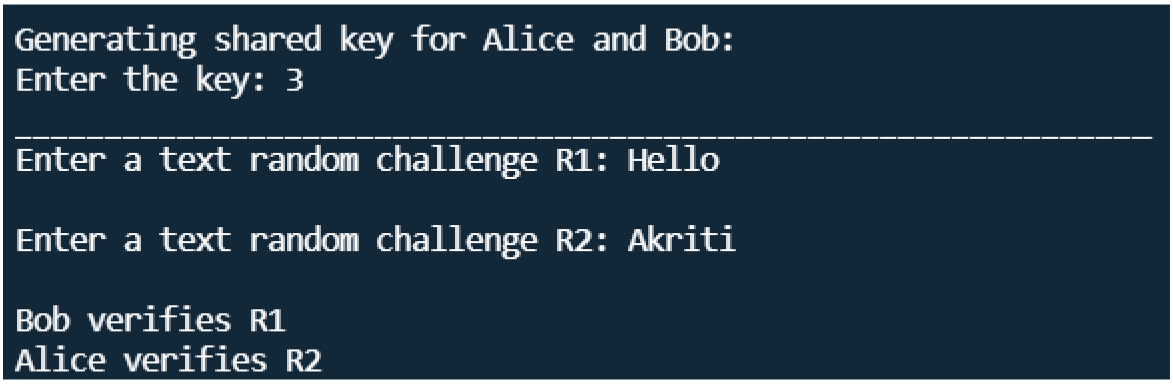
205120007

MCA – 2nd Year

Mutual Authentication and Digital Signature

1. Mutual Authentication using Public keys:

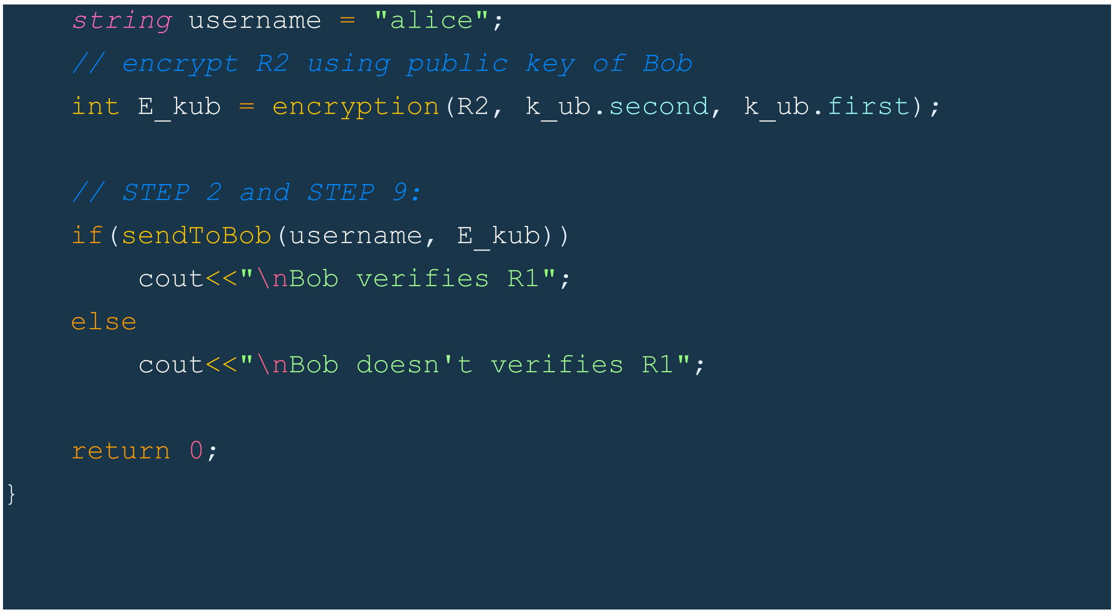
```
#include<iostream>
#include<string>
using namespace std;
// shared key
int k_ab;
// random challenges
string R1, R2;


// caesar-cipher encryption
string encryption(string msg){
    for(int i=0; i<msg.size(); i++){
        //encrypting lower case letters
        if(msg[i]>='a' && msg[i]<='z'){
            msg[i] = (((msg[i]-'a') + k_ab) % 26) + 'a';
        }
        //encrypting upper case letters
        else if(msg[i]>='A' && msg[i]<='Z'){
            msg[i] = (((msg[i]-'A') + k_ab) % 26) + 'A';
        }
    }
    return msg;
}


// caesar-cipher decryption
string decryption(string msg){
    for(int i=0; i<msg.size(); i++){
        //decrypting lower case letters
        if(msg[i]>='a' && msg[i]<='z'){
            msg[i] = ((((msg[i]-'a') - k_ab) + 26) % 26) + 'a';
        }
        //encrypting upper case letters
        else if(msg[i]>='A' && msg[i]<='Z'){
            msg[i] = ((((msg[i]-'A') - k_ab) + 26) % 26) + 'A';
        }
    }
    return msg;
}

void generateKey(){
    cout<<"\nEnter the key: ";
    cin>>k_ab;
}

// verifying username
bool verifyUser(string username){
    if(username == "alice")
        return true;
    return false;
}

string sendToBob(string E_R1){
    // STEP 5:
    // Bob decrypts encrypted R1 and compares its value with his own R1
    // D(E(R1)) = R1
    string D_E_R1 = decryption(E_R1);
    if(R1 == D_E_R1)
        cout<<"\nBob verifies R1";
    else
        cout<<"\nBob does not verify R1";

    // STEP 6:
    // Bob encrypts R2 using shared key and send to Alice
    string E_R2 = encryption(R2);
    return E_R2;
}

string sendToAlice(){
    // STEP 3:
    // Alice generates R2, which she'll send to Bob
    cout<<"\nEnter a text random challenge R2: ";
    cin>>R2;

    // STEP 4:
    // Alice encrypts R1 using shared key and send to Bob, along with
R2
    string E_R1 = encryption(R1);
    string E_R2 = sendToBob(E_R1);
    return E_R2;
}

// main
int main(){
    // generate shared key
    cout<<"\nGenerating shared key for Alice and Bob: ";
    generateKey();


    // STEP 1:
    string username = "alice";
    // if user is not verified => end the process
    if(!verifyUser(username))
        return 0;


    // STEP 2:
    // Bob generates R1 and send to Alice

cout<<"_______________________________________________________________"
;
    cout<<"\nEnter a text random challenge R1: ";
    cin>>R1;
    string E_R2 = sendToAlice();

    // STEP 7:
    // Alice decrypts encrypted R2 and compares its value with her own
R2
    // D(E(R2)) = R2
    string D_E_R2 = decryption(E_R2);
    if(R2 == D_E_R2)
        cout<<"\nAlice verifies R2";
    else
        cout<<"\nALice doesn't verify R2";

    return 0;
}
```



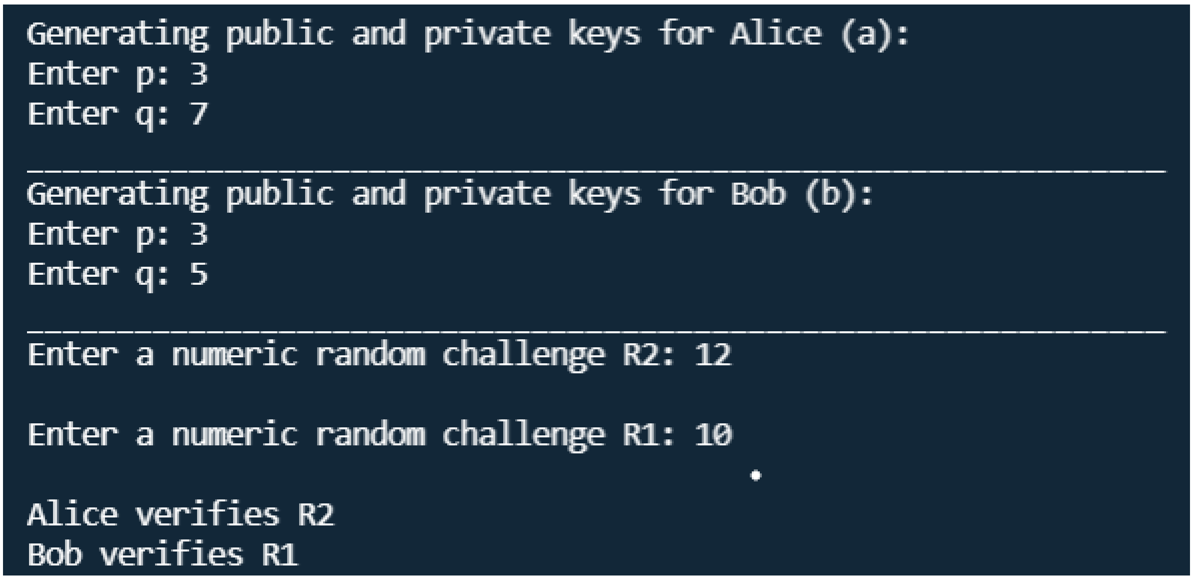
```
Generating shared key for Alice and Bob:
Enter the key: 3

---------------------------------------------------------------
Enter a text random challenge R1: Hello

Enter a text random challenge R2: Akriti

Bob verifies R1
Alice verifies R2
```

2. Mutual Authentication using Shared key:

```
#include<iostream>
#include<vector>
using namespace std;
// keys for alice and bob
pair<int, int> k_ra, k_ua;
pair<int, int> k_rb, k_ub;
// random challenges
// R1 = 12;
// R2 = 11;
int R1, R2;

// function to find GCD using Euclid algorithm
int gcd(int a, int b)
{
    if (b == 0)
        return a;
    return gcd(b, a % b);
}

// function to find modular multiplicative inverse
int euclideanAlgo(int z, int e1)
{
    int x1 = 1, x2 = 0, x3 = z;
    int y1 = 0, y2 = 1, y3 = e1;
    int q;
    vector<int> temp(3);
    while (y3 != 1)
    {
        q = x3 / y3;
        temp[0] = y1;
        temp[1] = y2;
        temp[2] = y3;
        y1 = x1 - y1 * q;
        y2 = x2 - y2 * q;
        y3 = x3 - y3 * q;
        x1 = temp[0];
        x2 = temp[1];
        x3 = temp[2];
    }
    if (y2 > 0)
        return y2;
    else
        return y2 + z;
}

// binary exponentiation function
int binpow(int a, int b, int m)
{
    a %= m;
    int res = 1;
    while (b > 0)
    {
        if (b & 1)
            res = res * a % m;
        a = a * a % m;
        b >>= 1;
    }
    return res;
}

// function for encryption
int encryption(int ptext, int e, int n)
{
    int cipher;
    cipher = binpow(ptext, e, n);
    return cipher;
}

// function for decryption
int decryption(int cipher, int d, int n)
{
    int ptext;
    ptext = binpow(cipher, d, n);
    return ptext;
}

// generating keys using RSA Algo
void rsa(pair<int, int> &k_ra, pair<int, int> &k_ua){
    int p, q;
    cout<<"\nEnter p: ";
    cin >> p;
    cout<<"Enter q: ";
    cin>>q;

    int n = p * q;
    int z = (p - 1) * (q - 1);

    int e;
    for (int i = 2; i < z; i++){
        if (gcd(i, z) == 1){
            e = i;
            break;
        }
    }
    int d = euclideanAlgo(z, e);
    k_ra = {n, e};
    k_ua = {n, d};
}

int sendToAlice(int E_kua, int D_krb){
    // STEP 6:
    if(R2 == D_krb)
        cout<<"\nAlice verifies R2";
    else{
        cout<<"\nAlice doesn't verifies R2";
        return 0;
    }
    // STEP 7:
    // decrypting using private key of Bob
    // D_kra(E_kua(R1)) = R1
    int D_kra = decryption(E_kua, k_ra.second, k_ra.first);
    return D_kra;
}

bool sendToBob(string username, int E_kub){
    if(username == "alice"){
        // STEP 3:
        // decrypting using private key of Bob
        // D_krb(E_kru(R2)) = R2
        int D_krb = decryption(E_kub, k_rb.second, k_rb.first);

        // STEP 4:
        // input R1
        cout<<"\nEnter a numeric random challenge R1: ";
        cin>>R1;

        // encrypt R1 using public key of Alice
        int E_kua = encryption(R1, k_ua.second, k_ua.first);

        // STEP 5 and STEP 8:
        int D_kra = sendToAlice(E_kua, D_krb);
        if(D_kra == R1)
            return true;
    }
    return false;
}

// main
int main(){
    // keys for alice
    cout<<"\nGenerating public and private keys for Alice (a): ";
    rsa(k_ra, k_ua);

    // keys for bob

cout<<"________________________________________________________________";
    cout<<"\nGenerating public and private keys for Bob (b): ";
    rsa(k_rb, k_ub);


    // STEP 1:
    // R2

cout<<"________________________________________________________________";
    cout<<"\nEnter a numeric random challenge R2: ";
    cin>>R2;
```

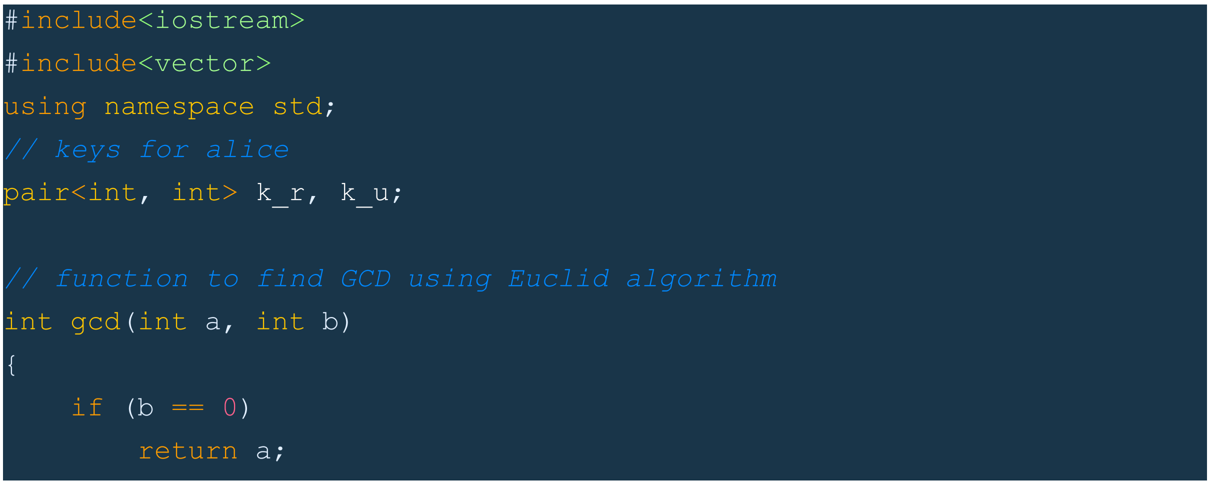


```
    string username = "alice";
    // encrypt R2 using public key of Bob
    int E_kub = encryption(R2, k_ub.second, k_ub.first);

    // STEP 2 and STEP 9:
    if(sendToBob(username, E_kub))
        cout<<"\nBob verifies R1";
    else
        cout<<"\nBob doesn't verifies R1";

    return 0;
}
```

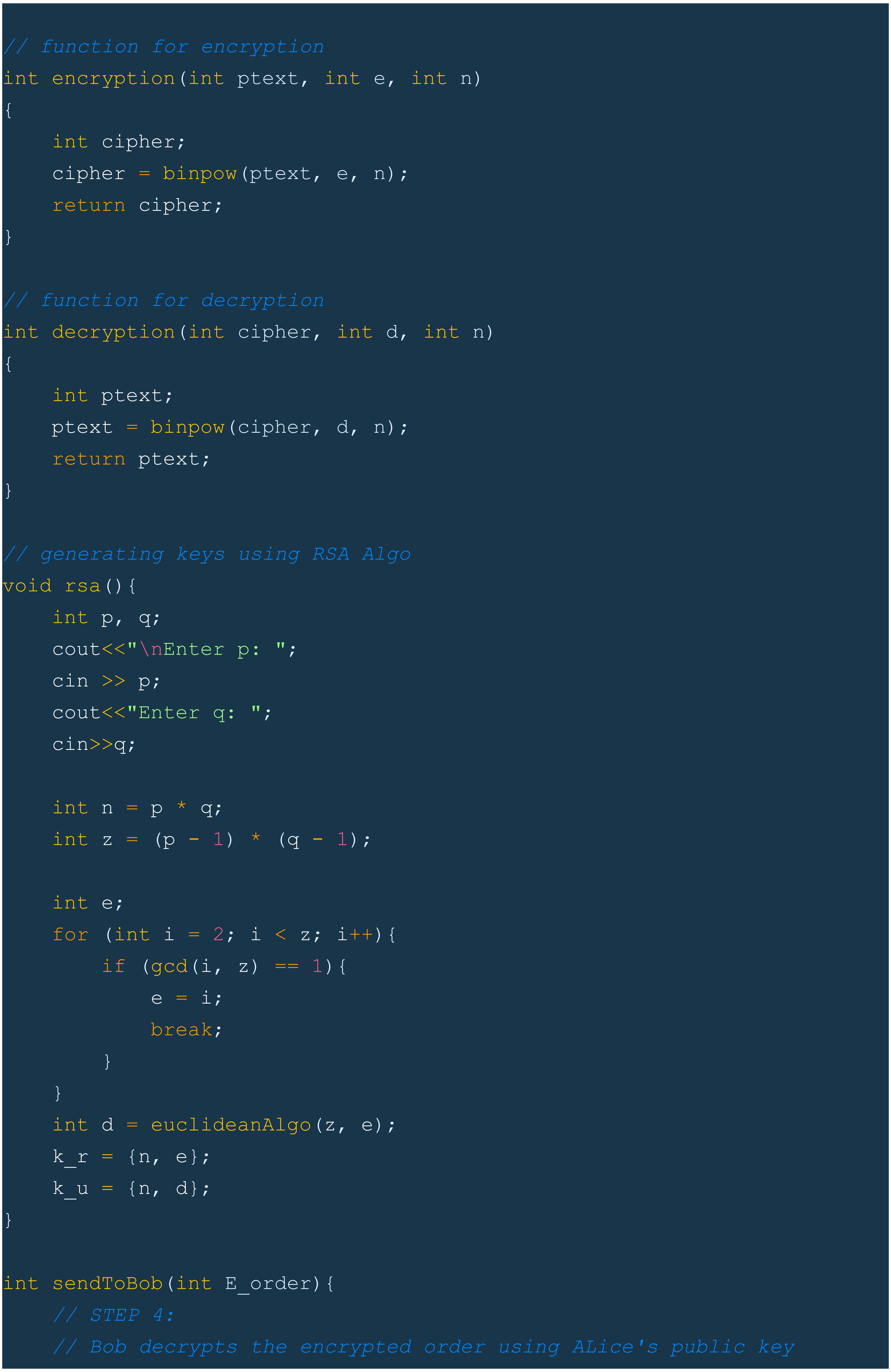


```
Generating public and private keys for Alice (a):
Enter p: 3
Enter q: 7

____________________________________________________________________
Generating public and private keys for Bob (b):
Enter p: 3
Enter q: 5

____________________________________________________________________
Enter a numeric random challenge R2: 12

Enter a numeric random challenge R1: 10

Alice verifies R2
Bob verifies R1
```

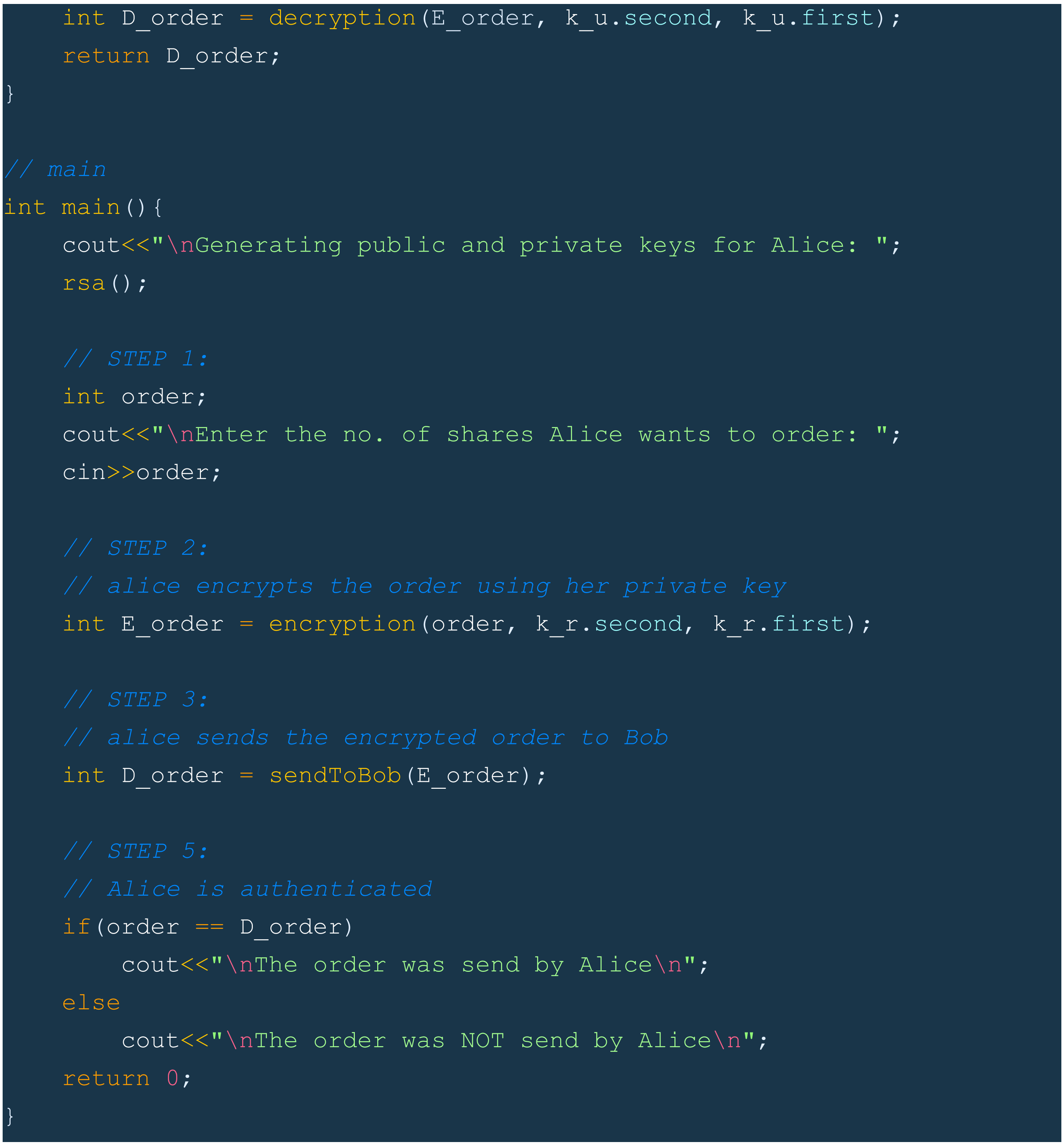
3. Digital Signature:



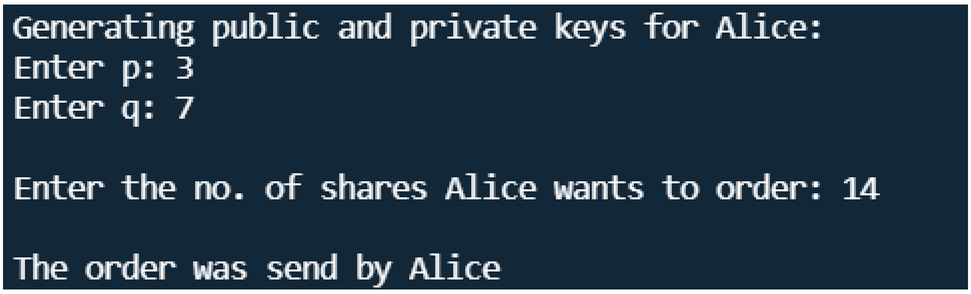
```
#include<iostream>
#include<vector>
using namespace std;
// keys for alice
pair<int, int> k_r, k_u;

// function to find GCD using Euclid algorithm
int gcd(int a, int b)
{
    if (b == 0)
        return a;
    return gcd(b, a % b);
}

// function to find modular multiplicative inverse
int euclideanAlgo(int z, int e1)
{
    int x1 = 1, x2 = 0, x3 = z;
    int y1 = 0, y2 = 1, y3 = e1;
    int q;
    vector<int> temp(3);
    while (y3 != 1)
    {
        q = x3 / y3;
        temp[0] = y1;
        temp[1] = y2;
        temp[2] = y3;
        y1 = x1 - y1 * q;
        y2 = x2 - y2 * q;
        y3 = x3 - y3 * q;
        x1 = temp[0];
        x2 = temp[1];
        x3 = temp[2];
    }
    if (y2 > 0)
        return y2;
    else
        return y2 + z;
}

// binary exponentiation function
int binpow(int a, int b, int m)
{
    a %= m;
    int res = 1;
    while (b > 0)
    {
        if (b & 1)
            res = res * a % m;
        a = a * a % m;
        b >>= 1;
    }
    return res;
}
```



```
// function for encryption
int encryption(int ptext, int e, int n)
{
    int cipher;
    cipher = binpow(ptext, e, n);
    return cipher;
}

// function for decryption
int decryption(int cipher, int d, int n)
{
    int ptext;
    ptext = binpow(cipher, d, n);
    return ptext;
}

// generating keys using RSA Algo
void rsa(){
    int p, q;
    cout<<"\nEnter p: ";
    cin >> p;
    cout<<"Enter q: ";
    cin>>q;

    int n = p * q;
    int z = (p - 1) * (q - 1);

    int e;
    for (int i = 2; i < z; i++){
        if (gcd(i, z) == 1){
            e = i;
            break;
        }
    }
    int d = euclideanAlgo(z, e);
    k_r = {n, e};
    k_u = {n, d};
}

int sendToBob(int E_order){
    // STEP 4:
    // Bob decrypts the encrypted order using ALice's public key
```



```
    int D_order = decryption(E_order, k_u.second, k_u.first);
    return D_order;
}

// main
int main(){
    cout<<"\nGenerating public and private keys for Alice: ";
    rsa();

    // STEP 1:
    int order;
    cout<<"\nEnter the no. of shares Alice wants to order: ";
    cin>>order;

    // STEP 2:
    // alice encrypts the order using her private key
    int E_order = encryption(order, k_r.second, k_r.first);

    // STEP 3:
    // alice sends the encrypted order to Bob
    int D_order = sendToBob(E_order);

    // STEP 5:
    // Alice is authenticated
    if(order == D_order)
        cout<<"\nThe order was send by Alice\n";
    else
        cout<<"\nThe order was NOT send by Alice\n";
    return 0;
}
```



```
Generating public and private keys for Alice:
Enter p: 3
Enter q: 7

Enter the no. of shares Alice wants to order: 14

The order was send by Alice
```

X-X-X-X