**Assignment no 7: Implementation of RSA Encryption/Decryption**

**2020BTECS00085**

**MANJIRI CHANDURE**

**BATCH : B5**

**RSA algorithm** is an asymmetric cryptography algorithm. Asymmetric actually means that it works on two different keys i.e. **Public Key** and **Private Key.** As the name describes that the Public Key is given to everyone and the Private key is kept private.

**An example of asymmetric cryptography:**

1. A client (for example browser) sends its public key to the server and requests some data.
2. The server encrypts the data using the client’s public key and sends the encrypted data.
3. The client receives this data and decrypts it.

**Steps in RSA Algorithm**

**Key Generation**

You need to generate public and private keys before running the functions to generate your ciphertext and plaintext. They use certain variables and parameters, all of which are explained below:

• Choose two large prime numbers (p and q)

• Calculate n = p\*q and z = (p-1)(q-1)

• Choose a number e where 1 < e < z

• Calculate d = e-1mod(p-1)(q-1)

• You can bundle private key pair as (n,d)

• You can bundle public key pair as (n,e)

**Encryption/Decryption Function**

Once you generate the keys, you pass the parameters to the functions that calculate your ciphertext and plaintext using the respective key.

• If the plaintext is m, ciphertext = m^e mod n.

• If the ciphertext is c, plaintext = c^d mod n

**Code in cpp:**

#include <bits/stdc++.h>

using namespace std;

set<int>

    prime; // a set will be the collection of prime numbers,

        // where we can select random primes p and q

int public\_key;

int private\_key;

int n;

// we will run the function only once to fill the set of

// prime numbers

void primefiller()

{

    // method used to fill the primes set is seive of

    // eratosthenes(a method to collect prime numbers)

    vector<bool> seive(250, true);

    seive[0] = false;

    seive[1] = false;

    for (int i = 2; i < 250; i++) {

        for (int j = i \* 2; j < 250; j += i) {

            seive[j] = false;

        }

    } // filling the prime numbers

    for (int i = 0; i < seive.size(); i++) {

        if (seive[i])

            prime.insert(i);

    }

}

// picking a random prime number and erasing that prime

// number from list because p!=q

int pickrandomprime()

{

    int k = rand() % prime.size();

    auto it = prime.begin();

    while (k--)

        it++;

    int ret = \*it;

    prime.erase(it);

    return ret;

}

void setkeys()

{

    int prime1 = pickrandomprime(); // first prime number

    int prime2 = pickrandomprime(); // second prime number

    // to check the prime numbers selected

    // cout<<prime1<<" "<<prime2<<endl;

    n = prime1 \* prime2;

    int fi = (prime1 - 1) \* (prime2 - 1);

    int e = 2;

    while (1) {

        if (\_\_gcd(e, fi) == 1)

            break;

        e++;

    } // d = (k\*Φ(n) + 1) / e for some integer k

    public\_key = e;

    int d = 2;

    while (1) {

        if ((d \* e) % fi == 1)

            break;

        d++;

    }

    private\_key = d;

}

// to encrypt the given number

long long int encrypt(double message)

{

    int e = public\_key;

    long long int encrpyted\_text = 1;

    while (e--) {

        encrpyted\_text \*= message;

        encrpyted\_text %= n;

    }

    return encrpyted\_text;

}

// to decrypt the given number

long long int decrypt(int encrpyted\_text)

{

    int d = private\_key;

    long long int decrypted = 1;

    while (d--) {

        decrypted \*= encrpyted\_text;

        decrypted %= n;

    }

    return decrypted;

}

// first converting each character to its ASCII value and

// then encoding it then decoding the number to get the

// ASCII and converting it to character

vector<int> encoder(string message)

{

    vector<int> form;

    // calling the encrypting function in encoding function

    for (auto& letter : message)

        form.push\_back(encrypt((int)letter));

    return form;

}

string decoder(vector<int> encoded)

{

    string s;

    // calling the decrypting function decoding function

    for (auto& num : encoded)

        s += decrypt(num);

    return s;

}

int main()

{

    primefiller();

    setkeys();

    string message;

    // uncomment below for manual input

    cout<<"enter the message\n";

    getline(cin,message);

    // calling the encoding function

    vector<int> coded = encoder(message);

    cout << "Initial message:\n" << message;

    cout << "\n\nThe encoded message(encrypted by public "

            "key)\n";

    for (auto& p : coded)

        cout << p;

    cout << "\n\nThe decoded message(decrypted by private "

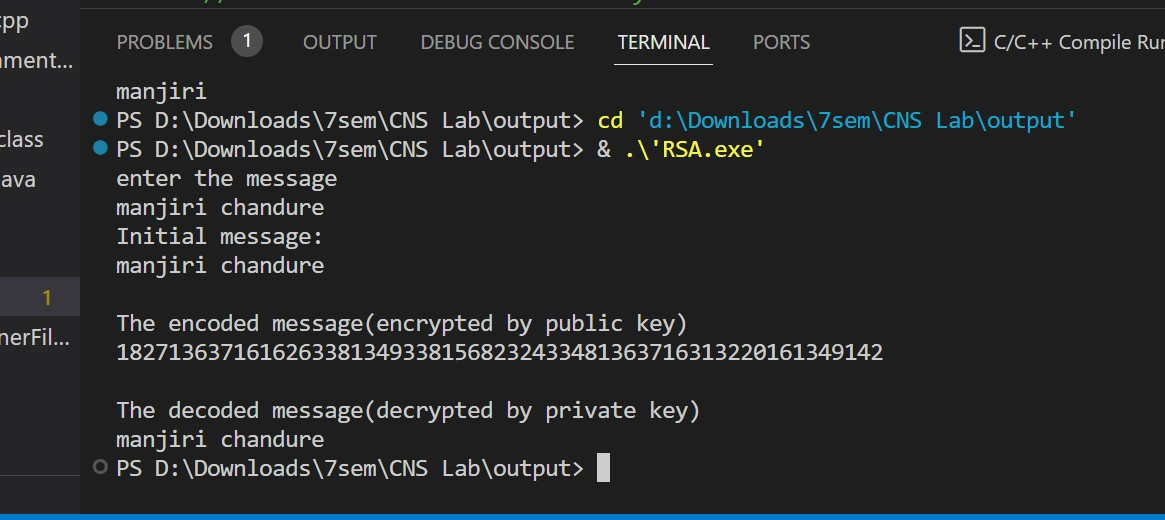
            "key)\n";

    cout << decoder(coded) << endl;

    return 0;

}

**Output:**



**Advantages:**

* **Security:**RSA algorithm is considered to be very secure and is widely used for secure data transmission.
* **Public-key cryptography:**RSA algorithm is a public-key cryptography algorithm, which means that it uses two different keys for encryption and decryption. The public key is used to encrypt the data, while the private key is used to decrypt the data.
* **Key exchange:**RSA algorithm can be used for secure key exchange, which means that two parties can exchange a secret key without actually sending the key over the network.
* **Digital signatures:**RSA algorithm can be used for digital signatures, which means that a sender can sign a message using their private key, and the receiver can verify the signature using the sender’s public key.
* **Speed:** The RSA technique is suited for usage in real-time applications since it is quite quick and effective.
* **Widely used:** Online banking, e-commerce, and secure communications are just a few fields and applications where the RSA algorithm is extensively developed.

**Disadvantages:**

* **Slow processing speed:**RSA algorithm is slower than other encryption algorithms, especially when dealing with large amounts of data.
* **Large key size:**RSA algorithm requires large key sizes to be secure, which means that it requires more computational resources and storage space.
* **Vulnerability to side-channel attacks:** RSA algorithm is vulnerable to side-channel attacks, which means an attacker can use information leaked through side channels such as power consumption, electromagnetic radiation, and timing analysis to extract the private key.
* **Limited use in some applications:** RSA algorithm is not suitable for some applications, such as those that require constant encryption and decryption of large amounts of data, due to its slow processing speed.
* **Complexity:** The RSA algorithm is a sophisticated mathematical technique that some individuals may find challenging to comprehend and use.
* **Key Management:** The secure administration of the private key is necessary for the RSA algorithm, although in some cases this can be difficult.

**2) Prime Factorization:**

it's the process of breaking down a composite number into a series of prime numbers that, when multiplied together, result in the original number.

import math

import time

def getFactors(A,n):

    print("Factorizing.....")

    end = (int)(math.sqrt(A)) + 1

    start\_time = time.time()

    for i in range(2, end):

        if A % i == 0:

            print(f"Factors --> {i} , {A // i}")

            break

    end\_time = time.time()

    print("Number of Digits: ", n)

    print("Execution Time: ", end\_time - start\_time)

    print("")

semiprimes = 100000000088787877679

digits = 20

getFactors(semiprimes,digits)

**Output:**

