C&NS Lab Assignment 12

Onkar Santosh Gavali (2019BTECS00037)

Batch B2

# Index

RSA Factorization challenge

* Explain the RSA Factorization challenge.
* Implement theRSA Factorization challenge using any programming language.

# 

# 

# RSA Factorization challenge

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 :

A client (for example browser) sends its public key to the server and requests some data.

The server encrypts the data using the client’s public key and sends the encrypted data.

The client receives this data and decrypts it.

Since this is asymmetric, nobody else except the browser can decrypt the data even if a third party has the public key of the browser.

The idea! The idea of RSA is based on the fact that it is difficult to factorize a large integer. The public key consists of two numbers where one number is a multiplication of two large prime numbers. And private key is also derived from the same two prime numbers. So if somebody can factorize the large number, the private key is compromised. Therefore encryption strength totally lies on the key size and if we double or triple the key size, the strength of encryption increases exponentially. RSA keys can be typically 1024 or 2048 bits long, but experts believe that 1024-bit keys could be broken in the near future. But till now it seems to be an infeasible task.

**Generating Public Key :**

**Select two prime no's. Suppose P = 53 and Q = 59.**

**Now First part of the Public key : n = P\*Q = 3127.**

**We also need a small exponent say e :**

**But e Must be**

**An integer.**

**Not be a factor of n.**

**1 < e < Φ(n)**

**Our Public Key is made of n and e**

**Generating Private Key :**

**We need to calculate Φ(n) :**

**Such that Φ(n) = (P-1)(Q-1)**

**so, Φ(n) = 3016**

**Now calculate Private Key, d :**

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

**For k = 2, value of d is 2011.**

**Now we are ready with our – Public Key ( n = 3127 and e = 3) and Private Key(d = 2011) Now we will encrypt “HI” :**

**Convert letters to numbers : H = 8 and I = 9**

**Thus Encrypted Data c = 89e mod n.**

**Thus our Encrypted Data comes out to be 1394**

**Now we will decrypt 1394 :**

**Decrypted Data = cd mod n.**

**Thus our Encrypted Data comes out to be 89**

**8 = H and I = 9 i.e. "HI".**

# Code

#include <bits/stdc++.h>

#define N 1000

using namespace std;

long long power(long long a, long long b, long long mod){

long long result = 1;

while(b > 0){

// check if the last bit is odd

if(b&1)

result = (result\*a)%mod;

a = (a\*a)%mod;

// b /= 2

b >>= 1;

}

return result;

}

long long convertToASCII(string letter){

long long ans = 0;

string str;

if(letter.length() > 9){

cout<<"provide input with 3 letters";

return 0;

}

for (int i = 0; i < letter.length(); i++)

{

int x = letter[i];

str = str + to\_string(x);

}

return (long long)stoll(str);

}

long long gcdExtended(long long a, long long b, long long \*x, long long \*y)

{

// cout << a << " " << b << " "<< " " << \*x << " " << \*y << "\n";

// Base Case

if (b == 0)

{

return \*x;

}

long long q = a / b;

long long x1 = \*y;

long long y1 = \*x - q \* (\*y);

long long t1 = gcdExtended(b, a % b, &x1, &y1);

// cout << a << " " << \*x << "\n";

if (\*x == 0 && t1 < 0)

return a + t1;

else

return t1;

// return gcd;

}

void SieveOfEratosthenes(int n, vector<int> &primes) {

bool prime[n + 1];

memset(prime, true, sizeof(prime));

for(int p = 2; p \* p <= n; p++) {

if (prime[p]) {

for (int i = p \* p; i <= n; i += p)

prime[i] = false;

}

}

for (int p = 2; p <= n; p++)

if (prime[p]){

primes.push\_back(p);

}

}

int main() {

char patternChar = '-';

char resetChar = ' ';

int lineWidth = 90;

int initialWidth = 50;

cout << setfill(patternChar) << setw(lineWidth) << patternChar << endl;

cout << setfill(resetChar);

cout << setw(initialWidth) << "RSA Algorithm" << endl;

cout << setfill(patternChar) << setw(lineWidth) << patternChar << endl;

cout << setfill(resetChar);

vector<int> primes;

// generating primes between 1 and N;

SieveOfEratosthenes(N, primes);

srand(time(0));

// choose any two primes randomly

int p, q;

int primesSize = primes.size();

int rand = std::rand();

p = primes[(rand % primesSize)];

do{

rand = std::rand();

q = primes[(rand % primesSize)];

}while(p == q);

cout << "\nRandomly selected primes\n" << endl;

cout << "p: " << p << endl;

cout << "q: " << q << endl;

// calculate the value of n

long long n = p\*1LL\*q;

cout << "n = p\*q" << endl;

cout << "n = " << n << endl;

// calculate the value of phi

long long phi = (p-1)\*1LL\*(q-1);

cout << "\nValue of phi(n): " << phi << endl;

// generating all the co-primes between 2 and phi👎

// acquire prime (a) such that a\*a < phi value

// store them

vector<int> primeList;

for(size\_t i = 0; i < primes.size(); i++){

if(primes[i]\*1LL\*primes[i] <= phi){

primeList.push\_back(primes[i]);

}

}

// find the factors of unique prime factors of phu value

vector<int> phiPrimeList;

for(size\_t i =0; i < primeList.size(); i++){

if(phi > primeList[i] && (phi % primeList[i] == 0)){

phiPrimeList.push\_back(primeList[i]);

while(phi % primeList[i] == 0){

phi /= primeList[i];

}

}

}

if(phi > 1){

phiPrimeList.push\_back(phi);

}

// reassining the value of phi

phi = (p-1)\*1LL\*(q-1);

long long sizeRestriction = 1e6;

sizeRestriction = min(sizeRestriction, phi);

// note : We are restricting the random coPrime upto 1e6

vector<int> coPrimesOfPhi;

vector<bool> phiVec(sizeRestriction, true);

phiVec[0] = phiVec[1] = false;

for(auto prime : phiPrimeList){

for(int i = prime; i < sizeRestriction; i += prime){

phiVec[i] = false;

}

}

for(size\_t i = 0; i < phiVec.size(); i++){

if(phiVec[i])

coPrimesOfPhi.push\_back(i);

}

// cout << "Co-Primes between [2,maxLimit of restriction) are as follows: " << endl;

// for(size\_t i = 0; i < coPrimesOfPhi.size(); i++){

// cout << coPrimesOfPhi[i] << " ";

// }

// cout << endl;

// avoiding selecting the first or any specific number of coprime which occured

rand = std::rand();

int e = coPrimesOfPhi[rand%coPrimesOfPhi.size()];

cout << "The ramdomly selected value of e is: " << e << endl;

long long x,y;

x=0;

y=1;

int d = gcdExtended(phi, e, &x, &y);

cout << "The value of d for selected e is: " << d << endl;

// message to be encrypted

string str;

cin>>str;

long long msg = convertToASCII(str);

cout<<"The ASCII of the message is "<<msg<<"\n";

msg = msg % n;

cout<<"The input message is (taking mod) "<<msg<<"\n";

long long c = power(msg,e,n);

cout<<"The ciphered text is power(msg,e,n): "<<c<<"\n";

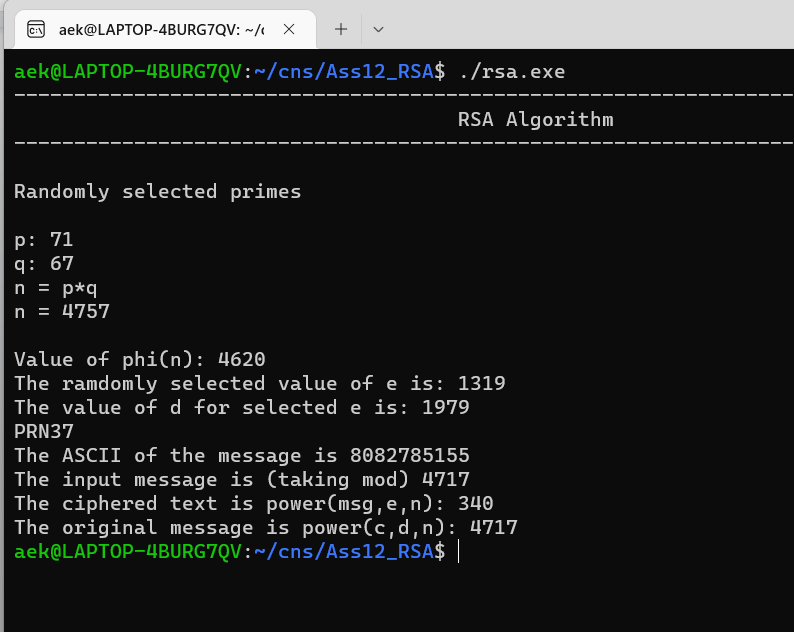
long long org = power(c,d,n);

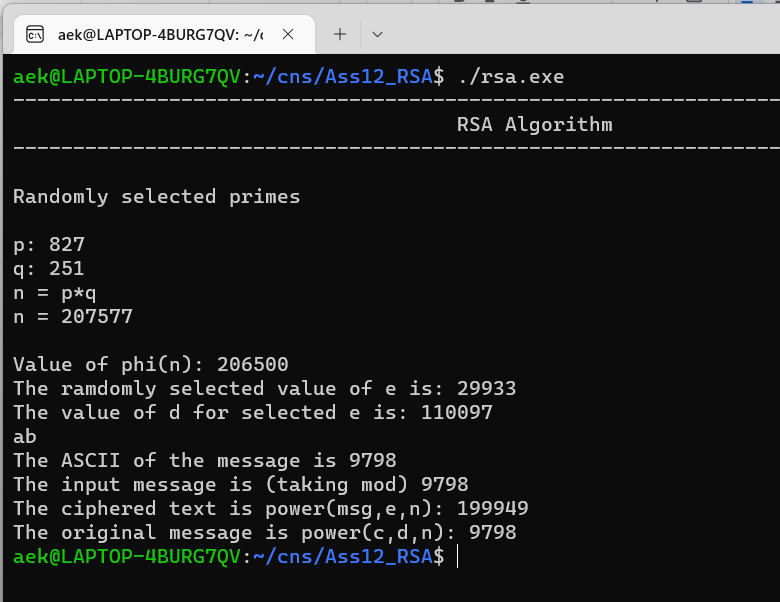
cout<<"The original message is power(c,d,n): "<<org<<"\n";

return 0;

}

# Output





# 