Program: 8 **Implementation of RSA Algorithm**

Date:

**AIM**

**ALGORITHM**

**CODE**

#include <iostream>

#include <vector>

#include <cmath>

#include <random>

using namespace std;

long gcd(long long a, long long b) {

    while (b != 0) {

        long long temp = b;

        b = a % b;

        a = temp;

    }

    return a;

}

int isPrime(int n) {

    int is\_prime = 1;

    int i;

    if (n == 0 || n == 1)

        is\_prime = 1;

    for (i = 2; i <= n/2; ++i) {

        if (n % i == 0) {

            is\_prime = 0;

            break;

        }

    }

    return is\_prime;

}

int getNextPrime(int n) {

    while (1) {

        n = n+1;

        if (isPrime(n)) {

            return n;

        }

    }

}

void primeFactorization(int num, vector<int> &factors) {

    int cur\_prime = 2;

    while (num != 1) {

        while (num % cur\_prime != 0) {

            cur\_prime = getNextPrime(cur\_prime);

        }

        num = num / cur\_prime;

        factors.push\_back(cur\_prime);

        cur\_prime = 2;

    }

}

int powerMod(long long x, int y, int z) {

    int hasAddition = 0;

    int initialX = x;

    vector<int> factorsY;

    primeFactorization(y, factorsY);

    if (factorsY.size() == 1) {

        hasAddition = 1;

        factorsY.clear();

        primeFactorization(y-1, factorsY);

    }

    int i=0;

    while (i < factorsY.size()) {

        if (factorsY[i] > 2) {

            x = powerMod(x, factorsY[i], z);

        } else {

            x = pow(x, factorsY[i]);

        }

        if (x >= z) {

            x = x % z;

        }

        i++;

    }

    if (hasAddition) {

        x = x \* initialX;

    }

    return x % z;

}

void printVector(vector<int> vec, int as\_char) {

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

        if (as\_char) {

            cout << (char) vec[i];

        } else {

            cout << vec[i] << " ";

        }

    }

    cout << endl;

}

class RSA {

    public:

    int p, q;

    long long n, phi, e, d;

    RSA(int p, int q) {

        this->p = p;

        this->q = q;

        this->generateKeys();

    }

    void generateKeys() {

        this->n = this->p \* this->q;

        this->phi = (this->p - 1) \* (this->q - 1);

        vector<int> e\_list;

        for (long long i=2; i<this->phi; i++) {

            if (gcd(this->phi, i) == 1) {

                this->e = i;

                e\_list.push\_back(i);

            }

        }

        std::random\_device rd;

        std::mt19937 gen(rd());

        std::uniform\_int\_distribution<> distr(0, e\_list.size()-1);

        this->e = e\_list[distr(gen)];

        for (long long i=1; i<INT\_MAX; i++) {

            if ((i \* this->e) % this->phi == 1) {

                this->d = i;

                break;

            }

        }

        cout << "p: " << this->p << endl;

        cout << "q: " << this->q << endl;

        cout << "n: " << this->n << endl;

        cout << "phi: " << this->phi << endl;

        cout << "e: " << this->e << endl;

        cout << "d: " << this->d << endl << endl;

    }

    void encrypt(string plaintext, vector<int> &cipher) {

        int l = plaintext.length();

        for (int i=0; i<l; i++) {

            cipher.push\_back(powerMod(plaintext[i], this->e, this->n));

        }

    }

    void decrypt(vector<int> cipher, vector<int> &plaintext) {

        int l = cipher.size();

        for (int i=0; i<l; i++) {

            plaintext.push\_back(powerMod(cipher[i], this->d, this->n));

        }

    }

};

int main() {

    RSA rsa(379, 449);

    string text = "Shazin 1029";

    cout << "Text: " << text << endl;

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

        cout << (int) text[i] << " ";

    }

    cout << endl << endl;

    vector<int> cipher;

    rsa.encrypt(text, cipher);

    cout << "Cipher: " << endl;

    printVector(cipher, 0);

    cout << endl;

    vector<int> plaintext;

    rsa.decrypt(cipher, plaintext);

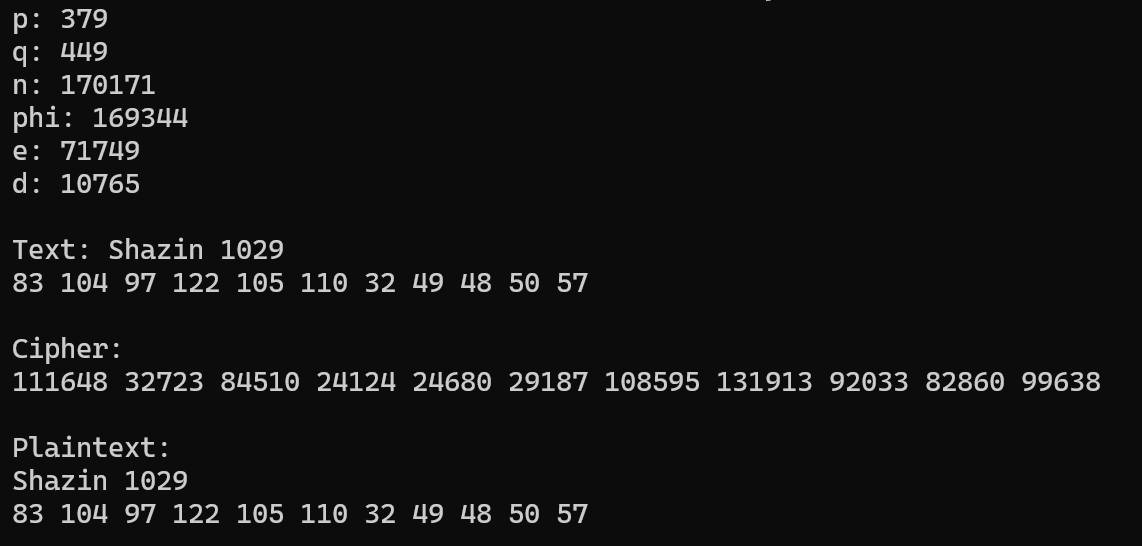
    cout << "Plaintext: " << endl;

    printVector(plaintext, 1);

    printVector(plaintext, 0);

    return 0;}

**OUTPUT**

****

**RESULT**

Thus, the program to implement encryption and decryption using columnar transposition cipher.