Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

КАФЕДРА ИНФОРМАТИКИ

Индивидуальная практическая работа № 2

Алгоритм шифрования RSA

Выполнил   
студент гр. 653502

Турцевич И.М.

Проверил

Артемьев В. С.

Минск, 2019

**Содержание**

[1. Цель работы 3](#_Toc26131161)

[2. Краткие теоретические сведения 4](#_Toc26131162)

[2.1 Алгоритм шифрования RSA 4](#_Toc26131163)

[2.2 Криптоанализ 5](#_Toc26131164)

[3. Примеры работы программы 6](#_Toc26131165)

[4. Выводы 7](#_Toc26131166)

[Приложение 1. Исходный код реализации алгоритма RSA 8](#_Toc26131167)

# **1. Цель работы**

Необходимо создать реализацию на языке программирования С++ алгоритма шифрования RSA.

# **2. Краткие теоретические сведения**

## **2.1 Алгоритм шифрования RSA**

Алгоритм, разработанный Ривестом, Шамиром и Адлеманом, использует выражения с экспонентами. Данные шифруются блоками, каждый блок рассматривается как число, меньшее некоторого числа n.

Шифрование и дешифрование имеют следующий вид для некоторого незашифрованного блока М и зашифрованного блока С.

С = Ме (mod n)

M = Cd (mod n) = (Me)d (mod n) = Med (mod n)

Как отправитель, так и получатель должны знать значение n. Отправитель знает значение е, получатель знает значение d. Таким образом, открытый ключ есть KU = {e, n} и закрытый ключ есть KR = {d, n}. При этом должны выполняться следующие условия:

1. Возможность найти е, d и n такие, что Med = M mod n для всех М < n.

2. Относительная легкость вычисления Ме и Сd для всех значений М < n.

3. Невозможность определить d, зная е и n.

Рассмотрим некоторые математические понятия, свойства и теоремы, которые позволят нам определить e, d и n.

1. Если (а · b) ≡ (a · c) mod n, то b ≡ c mod n, если а и n взаимнопростые, т.е gcd (a, n) = 1.

2. Обозначим Zp - все числа, взаимнопростые с p и меньшие p. Если p - простое, то Zp - это все остатки. Обозначим w-1 такое число, что w · w1 ≡ 1 mod p.

Тогда ∀ w ∈ Zp ∃ z: w · z ≡1 mod p

3. Определим функцию Эйлера следующим образом: Φ(n) - число положительных чисел, меньших n и взаимнопростых с n. Если p - простое, то Φ(р) = p-1. Если p и q - простые, то Φ(p · q) = (p-1) · (q-1).

4. Теорема Ферма. a n-1 ≡1 mod n, если n - простое.

5. Теорема Эйлера. a Φ(n) ≡1 mod n для всех взаимнопростых a и n.

Теперь рассмотрим все элементы алгоритма RSA (таблица 1).

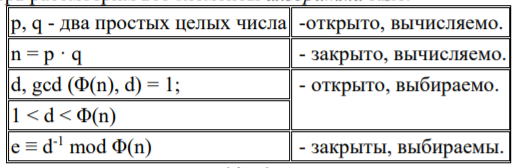


Таблица 1. Элементы алгоритма RSA

Закрытый ключ состоит из {d, n}, открытый ключ состоит из {e, n}. Предположим, что пользователь А опубликовал свой открытый ключ, и что пользователь B хочет послать пользователю А сообщение М. Тогда В вычисляет С = Ме (mod n) и передает С. При получении этого 58 зашифрованного текста пользователь А дешифрует вычислением М = Сd (mod n).

Суммируем алгоритм RSA:

Создание ключей

Выбрать простые р и q

Вычислить n = p · q

Выбрать d gcd (Φ(n), d) = 1; 1 < d < Φ(n)

Вычислить е е = d-1 mod Φ(n)

Открытый ключ KU = {e, n}

Закрытый ключ KR = {d, n}

Шифрование

Незашифрованный текст: М < n

Зашифрованный текст: С = Ме (mod n)

Дешифрование

Зашифрованный текст: С

Незашифрованный текст: М = Сd (mod n)

## **2.2 Криптоанализ**

Можно определить четыре возможных подхода для криптоанализа алгоритма RSA:

1. Лобовая атака: перебрать все возможные закрытые ключи.

2. Разложить n на два простых сомножителя. Это даст возможность вычислить Φ(n) = (p-1) · (q-1) и d = e-1 (mod Φ(n)).

3. Определить Φ(n) непосредственно, без начального определения р и q. Это также даст возможность определить d = e-1 (mod Φ(n)).

4. Определить d непосредственно, без начального определения Φ(n). Защита от лобовой атаки для RSA и ему подобных алгоритмов приводится далее.

# **3. Примеры работы программы**

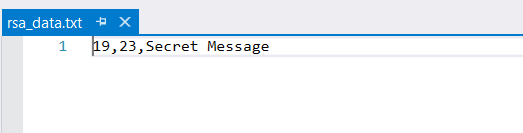


Рисунок 1. Содержимое файла rsa\_data.txt с данными для шифрования

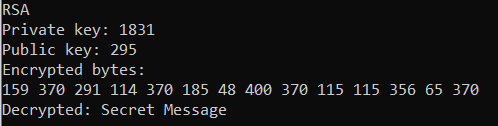


Рисунок 2. Пример вывода программы

# **4. Выводы**

В ходе данной работы был изучен шифрования RSA и была создана его программная реализация на языке программирования C++.

# **Приложение 1. Исходный код реализации алгоритма RSA**

**Файл des.cpp**

#include "rsa.h"

#define LIMIT 10000

int powerLog(int n, int p, int mod)

{

int result = 1;

for (; p; p >>= 1)

{

if (p & 1)

result = (1LL \* result \* n) % mod;

n = (1LL \* n \* n) % mod;

}

return result;

}

int gcd(int a, int b)

{

while (b)

{

int r = a % b;

a = b;

b = r;

}

return a;

}

int getCoprime(int n)

{

int generated = rand() % LIMIT;

while (gcd(n, generated) != 1)

generated = rand() % LIMIT;

return generated;

}

int isPrime(int num)

{

if (num == 2) return true;

if (num < 2 || num % 2 == 0) return false;

for (int i = 3; i < (int)pow(num, 0.5) + 2; i += 2) {

if (num % i == 0) return false;

}

return true;

}

pair<int, int> extendedEuclid(int a, int b) {

if (!b) {

return { 1, 0 };

}

pair<int, int> result = extendedEuclid(b, a % b);

return { result.second, result.first - (a / b) \* result.second };

}

int modularInverse(int n, int mod)

{

int inverse = extendedEuclid(n, mod).first;

while (inverse < 0)

inverse += mod;

return inverse;

}

pair<pair<int, int>, pair<int, int>> createKeys(int p, int q)

{

if (!(isPrime(p) && isPrime(q))) {

throw exception("Numbers must be prime!");

}

else if (p == q) {

throw exception("Numbers must be different!");

}

pair<pair<int, int>, pair<int, int>> result;

int n = p \* q;

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

int e = getCoprime(phi);

result.first = make\_pair(n, e);

int d = modularInverse(e, phi);

result.second = make\_pair(n, d);

return result;

}

int encryptByte(pair<int, int> key, int value)

{

return powerLog(value, key.second, key.first);

}

int decryptByte(pair<int, int> key, int value)

{

return powerLog(value, key.second, key.first);

}

vector<int> encryptRsa(pair<int, int> key, string text) {

vector<int> encBytes(text.length());

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

encBytes[i] = encryptByte(key, (int) text[i]);

}

return encBytes;

}

string decryptRsa(pair<int, int> key, vector<int> text) {

vector<char> decBytes(text.size());

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

decBytes[i] = (char)decryptByte(key, text[i]);

}

return string(decBytes.begin(), decBytes.end());

}

**Файл rsa.h**

#pragma once

#include <string>

#include <vector>

#include <iostream>

#include <sstream>

using namespace std;

int powerLog(int n, int p, int mod);

int gcd(int a, int b);

int getCoprime(int n);

int isPrime(int num);

pair<int, int> extendedEuclid(int a, int b);

int modularInverse(int n, int mod);

pair<pair<int, int>, pair<int, int>> createKeys(int, int);

int encryptByte(pair<int, int> key, int value);

int decryptByte(pair<int, int> key, int value);

vector<int> encryptRsa(pair<int, int> key, string text);

string decryptRsa(pair<int, int> key, vector<int> text);