Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

КАФЕДРА ИНФОРМАТИКИ

Отчёт по лабораторной работе №3

По дисциплине «Методы защиты информации»

По теме «Ассиметричная криптография. RSA»

Выполнил:

Студент гр. 653501

Никитинская А. С.

Проверил:

Артемьев В. С.

Минск 2019

Содержание

[1. Постановка задачи 3](#_Toc26714875)

[2.Краткие теоретические сведения 4](#_Toc26714876)

[3. Блок-схема алгоритма 6](#_Toc26714877)

[4. Вывод 7](#_Toc26714878)

[Приложение 1. Исходный код программы 8](#_Toc26714879)

[Приложение 2. Скриншот работы программы 11](#_Toc26714880)

## 1. Постановка задачи

1) Изучить теоретические сведения.

2) Реализовать программные средства шифрования и дешифрования текстовых файлов при помощи алгоритма RSA.

## Краткие теоретические сведения

RSA относится к ассиметричным алгоритмам шифрования: если для шифрования используется открытый ключ, то для дешифрования используется закрытый, и наоборот. Первое свойство позволяет кому угодно зашифровать сообщение открытым ключом в адрес владельца закрытого ключа и тем самым обеспечить его конфиденциальность. Второе свойство позволяет владельцу ключа зашифровать хэш сообщения закрытым ключом, чтобы кто угодно мог дешифровать зашифрованный хэш, сравнить его с хэшем сообщения и определить, было ли сообщение модифицровано.

На данный момент асимметричное шифрование на основе открытого ключа RSA (расшифровывается, как Rivest, Shamir and Aldeman - создатели алгоритма) использует большинство продуктов на рынке информационной безопасности.

Его криптостойкость основывается на сложности разложения на множители больших чисел, а именно - на исключительной трудности задачи определить секретный ключ на основании открытого, так как для этого потребуется решить задачу о существовании делителей целого числа. Наиболее криптостойкие системы используют 1024-битовые и большие числа.

Рассмотрим алгоритм RSA с практической точки зрения.

Для начала необходимо сгенерировать открытый и секретные ключи:

* Возьмем два больших простых числа p and q.
* Определим n, как результат умножения p on q (n= p\*q).
* Выберем случайное число, которое назовем d. Это число должно быть взаимно простым (не иметь ни одного общего делителя, кроме 1) с результатом умножения (p-1)\*(q-1).
* Определим такое число е, для которого является истинным следующее соотношение (e\*d) mod ((p-1)\*(q-1))=1.
* Hазовем открытым ключем числа e и n, а секретным - d и n.

Для того, чтобы зашифровать данные по открытому ключу {e,n}, необходимо следующее:

* разбить шифруемый текст на блоки, каждый из которых может быть представлен в виде числа M(i)=0,1,2..., n-1( т.е. только до n-1).
* зашифровать текст, рассматриваемый как последовательность чисел M(i) по формуле C(i)=(M(I)^e)mod n.

Чтобы расшифровать эти данные, используя секретный ключ {d,n}, необходимо выполнить следующие вычисления: M(i) = (C(i)^d) mod n. В результате будет получено множество чисел M(i), которые представляют собой исходный текст.

Стойкость алгоритма основывается на сложности вычисления обратной функции к функции шифрования



Для вычисления {\displaystyle m}m по известным {\displaystyle c,e,n}c, e, n нужно найти такой {\displaystyle d}d, чтобы



То есть



Вычисление [обратного элемента](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%80%D0%B0%D1%82%D0%BD%D1%8B%D0%B9_%D1%8D%D0%BB%D0%B5%D0%BC%D0%B5%D0%BD%D1%82) по модулю не является сложной задачей, однако злоумышленнику неизвестно значение {\displaystyle \varphi (n)}phi(n). Для вычисления [функции Эйлера](https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F_%D0%AD%D0%B9%D0%BB%D0%B5%D1%80%D0%B0) от известного числа {\displaystyle n}n необходимо знать разложение этого числа на простые множители. Нахождение таких множителей и является сложной задачей, а знание этих множителей — *«потайной дверцей»* ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) [*backdoor*](https://ru.wikipedia.org/wiki/Backdoor)), которая используется для вычисления {\displaystyle d}d владельцем ключа. Существует множество алгоритмов для нахождения простых сомножителей, так называемой [*факторизации*](https://ru.wikipedia.org/wiki/%D0%A4%D0%B0%D0%BA%D1%82%D0%BE%D1%80%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F_%D1%86%D0%B5%D0%BB%D1%8B%D1%85_%D1%87%D0%B8%D1%81%D0%B5%D0%BB), самый быстрый из которых на сегодняшний день — [общий метод решета числового поля](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%89%D0%B8%D0%B9_%D0%BC%D0%B5%D1%82%D0%BE%D0%B4_%D1%80%D0%B5%D1%88%D0%B5%D1%82%D0%B0_%D1%87%D0%B8%D1%81%D0%BB%D0%BE%D0%B2%D0%BE%D0%B3%D0%BE_%D0%BF%D0%BE%D0%BB%D1%8F), скорость которого для k-битного целого числа составляет



Система RSA используется для защиты программного обеспечения и в схемах [цифровой подписи](https://ru.wikipedia.org/wiki/%D0%AD%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%86%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D0%B0%D1%8F_%D0%BF%D0%BE%D0%B4%D0%BF%D0%B8%D1%81%D1%8C).

Также она используется в открытой системе шифрования [PGP](https://ru.wikipedia.org/wiki/PGP) и иных системах шифрования (к примеру, DarkCryptTC и формат xdc) в сочетании с [симметричными алгоритмами](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BC%D0%BC%D0%B5%D1%82%D1%80%D0%B8%D1%87%D0%BD%D1%8B%D0%B9_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D1%88%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F).

Из-за низкой скорости шифрования (около 30 кбит/с при 512-битном ключе на процессоре 2 ГГц), сообщения обычно шифруют с помощью более производительных симметричных алгоритмов со случайным *сеансовым ключом* (например, [AES](https://ru.wikipedia.org/wiki/Advanced_Encryption_Standard), [IDEA](https://ru.wikipedia.org/wiki/IDEA), [Serpent](https://ru.wikipedia.org/wiki/Serpent" \o "Serpent), [Twofish](https://ru.wikipedia.org/wiki/Twofish" \o "Twofish)), а с помощью RSA шифруют лишь этот ключ, таким образом реализуется [гибридная криптосистема](https://ru.wikipedia.org/wiki/%D0%93%D0%B8%D0%B1%D1%80%D0%B8%D0%B4%D0%BD%D0%B0%D1%8F_%D0%BA%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0). Такой механизм имеет потенциальные уязвимости ввиду необходимости использовать [криптографически стойкий генератор псевдослучайных чисел](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8_%D1%81%D1%82%D0%BE%D0%B9%D0%BA%D0%B8%D0%B9_%D0%B3%D0%B5%D0%BD%D0%B5%D1%80%D0%B0%D1%82%D0%BE%D1%80_%D0%BF%D1%81%D0%B5%D0%B2%D0%B4%D0%BE%D1%81%D0%BB%D1%83%D1%87%D0%B0%D0%B9%D0%BD%D1%8B%D1%85_%D1%87%D0%B8%D1%81%D0%B5%D0%BB" \o "Криптографически стойкий генератор псевдослучайных чисел) для формирования случайного сеансового ключа симметричного шифрования.

## 3. Блок-схема алгоритма

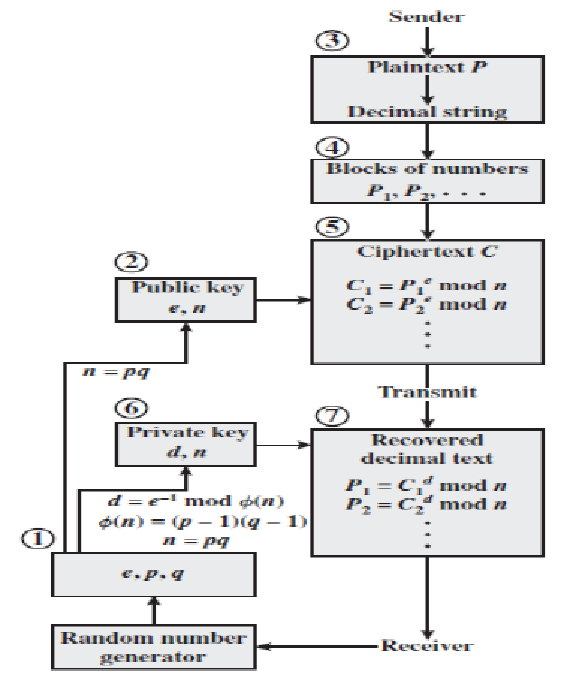
****

Рисунок 3.1 Блок-схема алгоритма RSA

## 4. Вывод

На данный момент асимметричное шифрование на основе открытого ключа RSA (расшифровывается, как Rivest, Shamir and Aldeman - создатели алгоритма) использует большинство продуктов на рынке информационной безопасности.

Его криптостойкость основывается на сложности разложения на множители больших чисел, а именно - на исключительной трудности задачи определить секретный ключ на основании открытого, так как для этого потребуется решить задачу о существовании делителей целого числа. Наиболее криптостойкие системы используют 1024-битовые и большие числа.

## Приложение 1. Исходный код программы

class Rsa {

private:

string password;

string text;

static long int calculate\_e(long int t) {

long int e;

for (e = 2; e < t; e++) {

if (Utils::greatest\_common\_divisor(e, t) == 1) {

return e;

}

}

return -1;

}

static long int calculate\_d(long int e, long int t) {

long int d;

long int k = 1;

while (true) {

k = k + t;

if (k % e == 0) {

d = (k / e);

return d;

}

}

}

public:

Rsa(string password, string text) {

this->password = std::move(password);

this->text = std::move(text);

}

static vector<vector<int>> generate\_keys(int p, int q) {

if (!Utils::is\_prime(p) || !Utils::is\_prime(q)) {

throw InvalidRsaPrime();

}

else if (p == q) {

throw InvalidRsaParams();

}

auto n = p \* q;

auto t = (p - 1) \* (q - 1);

auto e = Rsa::calculate\_e(t);

auto d = Rsa::calculate\_d(e, t);

auto result = vector<vector<int>>();

result.push\_back({ n, e });

result.push\_back({ n, d });

return result;

}

static long int encrypt(long int c, long int e, long int n) {

long int current = c - 97;

long int result = 1;

for (auto i = 0; i < e; i++) {

result = result \* current;

result = result % n;

}

return result;

}

static long int decrypt(long int c, long int d, long int n) {

long int current = c;

long int result = 1;

for (auto j = 0; j < d; j++)

{

result = result \* current;

result = result % n;

}

return result + 97;

}

string run\_encode(int p, int q, vector<int>& key) {

auto keys = Rsa::generate\_keys(p, q);

auto public\_key = keys[0];

auto private\_key = keys[1];

key = Utils::vector\_copy<int>(private\_key);

string result = "";

for (auto i = 0; i < this->text.size(); i++) {

result = result + (char)Rsa::encrypt(this->text[i], public\_key[1], public\_key[0]);

}

return result;

}

string run\_decode(vector<int> private\_key) {

string result = "";

for (auto i = 0; i < this->text.size(); i++) {

result = result + (char)Rsa::decrypt(this->text[i], private\_key[1], private\_key[0]);

}

return result;

}

static string encrypt(string password, string text, int p, int q, vector<int>& key) {

return Rsa(password, text).run\_encode(p, q, key);

}

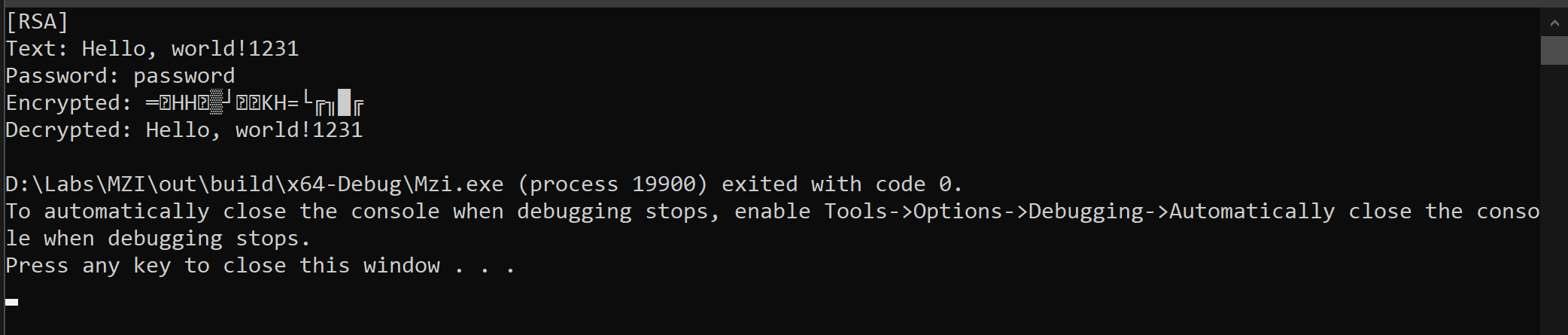
static string decrypt(string password, string text, vector<int> key) {

return Rsa(password, text).run\_decode(key);

}

};

## Приложение 2. Скриншот работы программы



Скриншот работы алгоритма RSA