Министерство науки и высшего образования РФ

Федеральное государственное автономное

образовательное учреждение высшего образования

«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

|  |
| --- |
| Институт космических и информационных технологий |
| институт |
| Программная инженерия |
| кафедра |

**ОТЧЕТ О ПРАКТИЧЕСКОЙ РАБОТЕ №2**

|  |
| --- |
| Основы криптографии с открытым ключом. Алгоритм RSA |
| тема |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Преподаватель | |  |  |  | Ю.В. Потылицина |
|  | |  | подпись, дата |  | инициалы, фамилия |
| Студент | КИ21-17/1Б, 032156940 |  |  |  | Н.А. Самарин |
|  | номер группы, зачётной книжки |  | подпись, дата |  | инициалы, фамилия |

Красноярск 2023

СОДЕРЖАНИЕ

[1 Цель 3](#_Toc147783990)

[2 Задание 3](#_Toc147783991)

[3 Теоретические сведения 3](#_Toc147783992)

[4 Практическая часть 6](#_Toc147783993)

[5 Вывод 11](#_Toc147783994)

# Цель

Ознакомиться с основами асимметричного шифрования, ознакомиться с элементами теории чисел, используемых в криптографии с открытым ключом, изучить особенности алгоритма с открытым ключом RSA, получить навыки разработки криптосистем с открытым ключом с использованием языка программирования высокого уровня.

# 2 Задание

Разработать и составить в виде блок-схемы, псевдокода, пошагово на естественном языке алгоритм RSA с числом N длиной 38 знаков. Убедиться в правильности составления алгоритма и затем на языке программирования составить программу, которая реализует данный алгоритм.

На ряде контрольных примеров (не менее 10) открытого текста, состоящего из различного количества символов, проверить правильность работы алгоритма.

Оценить криптостойкость своего варианта алгоритма RSA, а также сделать оценку производительности, разработанной программы.

# 3 Теоретические сведения

Алгоритм шифрования с открытым ключом RSA был предложен в 1977 году. Его название составлено из первых букв фамилий авторов: Р. Райвеста (R. Rivest), А. Шамира (A. Shamir) и Л. Адлемана (L. Adleman). Алгоритм RSA является очень популярным и широко применяемым асимметричным алгоритмом в криптографических системах. Алгоритм основан на использовании того факта, что задача разложения большого числа на простые сомножители является трудной. Криптографическая система RSA базируется на следующих двух фактах из теории чисел:

• задача проверки числа на простоту является сравнительно легкой;

• задача разложения чисел вида (где P и Q – простые числа) на множители является очень трудной, если мы знаем только N, а P и Q – большие числа (задача факторизации больших чисел). Алгоритм RSA представляет собой блочный алгоритм шифрования, где зашифрованные и незашифрованные данные должны быть представлены в виде целых чисел в диапазоне (0, N − 1) для некоторого N.

Первым этапом является генерация открытого и закрытого ключей. Для этого последовательно проделываются следующие действия:

1. Выбираются два больших и простых числа Р и Q.

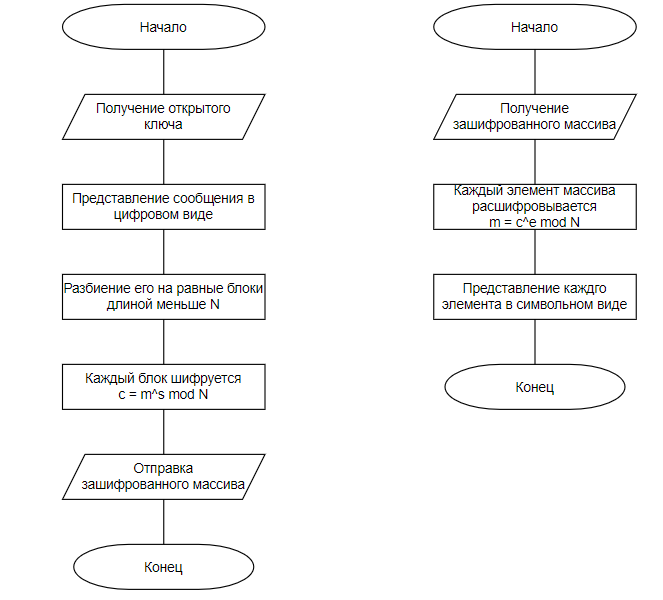
2. Вычисляется произведение .

3. Вычисляется значение функции Эйлера .

4. Случайным образом выбирается число s < d и взаимно простое с d.

5. Вычисляется число e, такое, что

После всех этих операций, числа s и N будут открытым ключом пользователя, а e и N – закрытым ключом.

Рисунок 1 – Блок-схема алгоритма RSA

Рассмотрим алгоритм RSA на естественном языке. Абонент А представляет своё сообщение в цифровом виде (используя, например, одну из кодировок символов) и разбивает его на блоки m (длина блока меньше N). Соответственно, после шифрования, зашифрованное сообщение будет состоять из блоков c.

Далее абонент А шифрует каждый блок своего сообщения по формуле . После этого зашифрованное сообщение пересылается по открытой линии.

Абонент Б, получивший зашифрованное сообщение, расшифровывает все блоки полученного сообщения по формуле .. В случае, если все зашифрованные блоки были переданы без ошибок (случайных или преднамеренных) все расшифрованные блоки будут точно такими же, как и исходящие от пользователя А.

Алгоритм RSA на псевдокоде представлен в листинге 1

Листинг 1 – Псевдокод RSA

функция шифрования(сообщение, s, N, m):

блоки = разделить\_сообщение\_на\_блоки(сообщение, m)

зашифрованные\_блоки = []

для каждого блока в блоках:

зашифрованный\_блок = возведение\_в\_степень\_по\_модулю(блок, s, N)

добавить зашифрованный\_блок в зашифрованные\_блоки

вернуть зашифрованные\_блоки

функция дешифрования(зашифрованные\_блоки, e, N):

дешифрованные\_блоки = []

для каждого блока в зашифрованных\_блоках:

дешифрованный\_блок = возведение\_в\_степень\_по\_модулю(блок, e, N)

добавить дешифрованный\_блок в дешифрованные\_блоки

вернуть дешифрованные\_блоки

сообщение = "Сообщение что я хочу зашифровать"

s = открытый\_ключ

e = закрытый\_ключ

N = модуль

m = размер\_блока

зашифрованное\_сообщение = шифрование(сообщение, s, N, m)

расшифрованное\_сообщение = дешифрование(зашифрованное\_сообщение, e, N)

функция разделить\_сообщение\_на\_блоки(сообщение, m):

// Разбить сообщение на блоки длины m

// Вернуть список блоков

блоки = []

для каждого i от 0 до длины(сообщение) с шагом m:

блок = подстрока(сообщение, i, i + m)

добавить блок в блоки

вернуть блоки

# 4 Практическая часть

Алгоритм был реализован в виде программы на языке python. Для определения границ символов в блоке перед блоком записываются длины цифровых кодов входящих в него. Также свободной место в последнем блоке заполняется пробелами.

Оценим криптостойкость. Из-за длины N равной 38 десятичным знакам каждый блок имеет длину максимум 4 (в программе берется максимальная длина для кодировки Unicode (38 // 8)). Из-за этого данную версию шифра можно взломать если зашифровать открытым ключом все комбинации из 4 символов. После замены шифрованных блоков известными комбинациями, мы сможем прочитать текст, хотя в нем, вероятно, будут нерасшифрованные блоки. Если взять набор символов, включающий в себя: русский алфавит + английский + цифры + знаки (суммарно около 150 символов), то число возможных комбинаций составит приблизительно . Хотя это число довольно велико, но очевидно, что проще выбрать этот способ вместо нахождения простых делителей числа с 38 десятичными знаками. Для повышения криптостойкости следует увеличить длину блоков (для этого необходимо увеличить длину N).

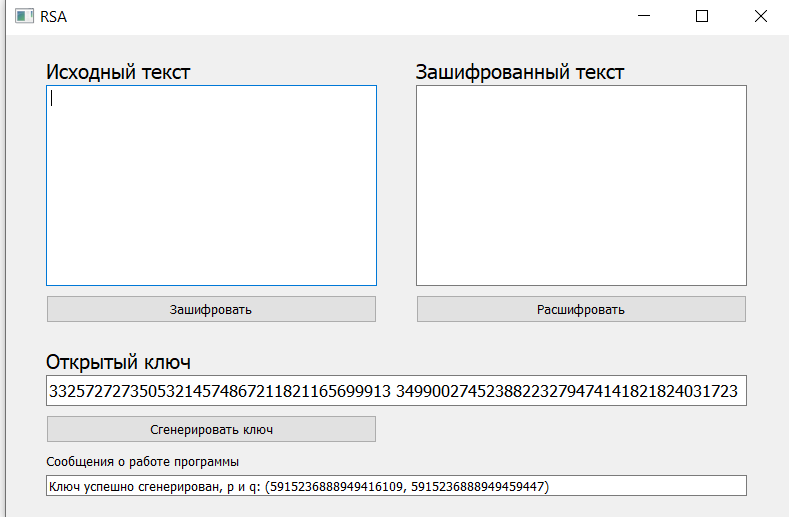
Оценим производительность. Программа использует встроенную в Python оптимизированную функцию возведения в степень, что позволяет практически мгновенно шифровать и расшифровывать любые тексты. Генерация нового ключа происходит за миллисекунды, однако из-за того, что числа P и Q подбирается случайным образом и их произведение не всегда имеет 38 знаков, это может занять некоторое время (в среднем 0.4 секунды).

Код алгоритма RSA представлен в листинге 2

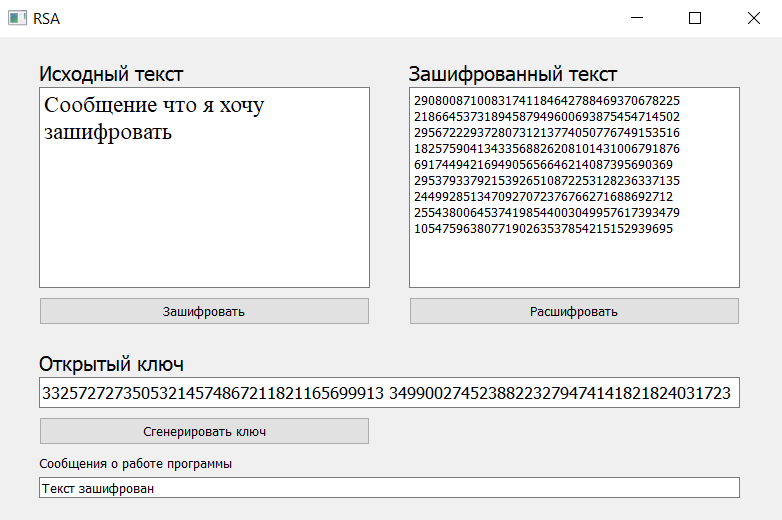
Листинг 2 – Код программы, реализующей шифр RSA

import random  
from sympy import primerange  
def generate\_primes(bits):  
 while True:  
 low\_border = random.randint(10 \*\* (bits // 2 - 1), 10 \*\* (bits // 2))  
 high\_border = low\_border + random.randint(10\*\*4, 10\*\*5)  
 primes = list(primerange(low\_border, high\_border))  
 index1 = random.randint(0, len(primes) - 1)  
 index2 = random.randint(0, len(primes) - 1)  
 p = primes[index1]  
 q = primes[index2]  
 print("primes generating try", p, q, p\*q)  
 if 10 \*\* (bits-1) <= p \* q <= 10 \*\* (bits):  
 return p, q  
def gcd(a, b):  
 *"""Функция для нахождения наибольшего общего делителя"""* while b:  
 a, b = b, a % b  
 return a  
def mod\_inverse(a, m):  
 *"""Обобщенный алгоритм Евклида"""* m0, x0, x1 = m, 0, 1  
 while a > 1:  
 q = a // m  
 m, a = a % m, m  
 x0, x1 = x1 - q \* x0, x0  
 return x1 + m0 if x1 < 0 else x1  
def generate\_params\_and\_keys(bits):  
 *"""Генерация параметров и ключей"""* p, q = generate\_primes(bits)  
 print("primes generated")  
 n = p \* q  
 phi = (p - 1) \* (q - 1)  
 s = random.randint(2, phi - 1)  
 while gcd(s, phi) != 1:  
 s = random.randint(2, phi - 1)  
 e = mod\_inverse(s, phi)  
 print("keys generated")  
 return ((p, q), (s, n), (e, n))  
def encrypt\_block(public\_key, block):  
 s, n = public\_key  
 cipher = ''.join([str(len(str(ord(char)))) for char in block])  
 for char in block:  
 cipher += str(ord(char))  
 return str(pow(int(cipher), s, n))  
def decrypt\_block(private\_key, block):  
 e, n = private\_key  
 try:  
 plain = str(pow(int(block), e, n))  
 except OverflowError:  
 return -1  
 return plain  
def rsa\_encrypt\_text(public\_key, text, block\_size):  
 encrypted\_text = []  
 if k\_space := (len(text) % block\_size):  
 text += (block\_size - k\_space) \* " "  
 text += str(k\_space) \* block\_size  
 print(text)  
 for i in range(0, len(text), block\_size):  
 block = text[i:i + block\_size]  
 encrypted\_text.append(encrypt\_block(public\_key, block))  
 return encrypted\_text  
def rsa\_decrypt\_text(private\_key, text, block\_size):  
 decrypt\_text = ""  
 for block in text:  
 block\_str = decrypt\_block(private\_key, block)  
 if block\_str == -1:  
 return -1  
 sizes = [int(block\_str[i]) for i in range(block\_size)]  
 curr = block\_size  
 for size in sizes:  
 try:  
 decrypt\_text += chr(int(block\_str[curr:curr+size]))  
 except ValueError:  
 return -1  
 curr += size  
 padding = (block\_size - int(decrypt\_text[-1])) % block\_size  
 decrypt\_text = decrypt\_text[:-(block\_size+padding)]  
 return decrypt\_text

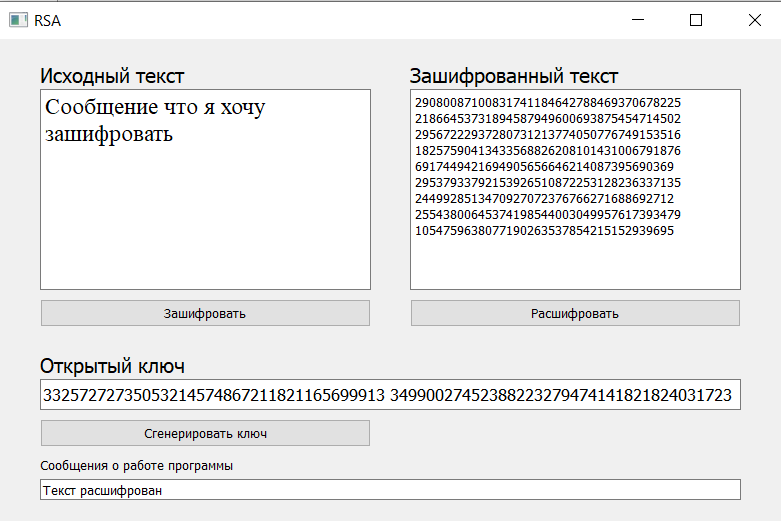
На рисунке 2 показана генерация ключей.

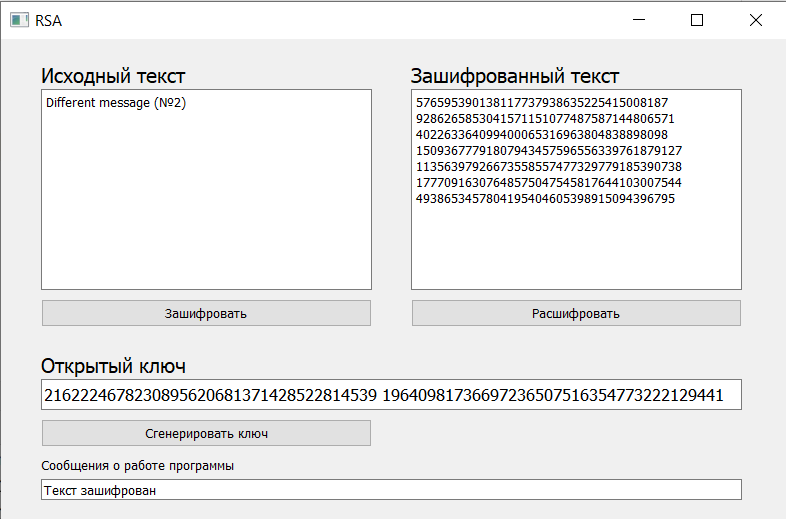
 Рисунок 2 – Пример генерации ключей RSA

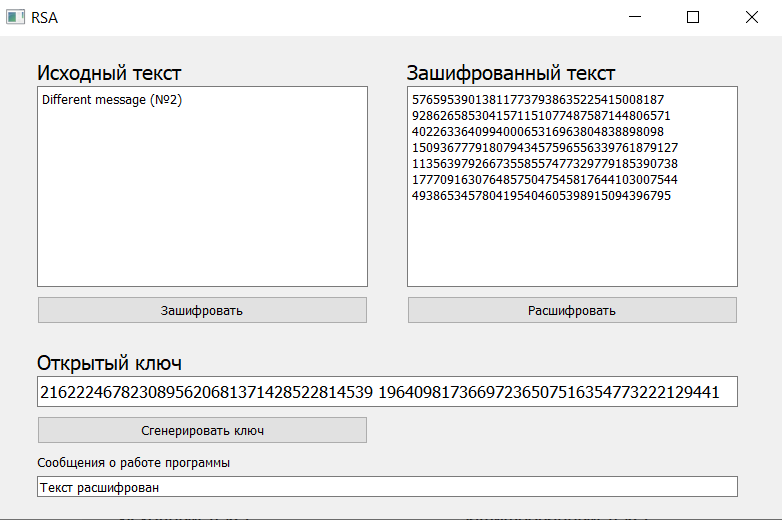
На рисунке 3 зашифровали «Сообщение что я хочу зашифровать», используя открытый ключ «33257272735053214574867211821165699913 34990027452388223279474141821824031723».

 Рисунок 3 – Пример шифрования RSA

На рисунке 4 расшифровали зашифрованный ранее текст.

 Рисунок 4 – Пример дешифрования RSA

 Рисунок 5 – Пример шифрования RSA №2

 Рисунок 6 – Пример дешифрования RSA №2

На рисунке 7 показана попытка расшифровать текст ключом, который не создавался в данной программе.

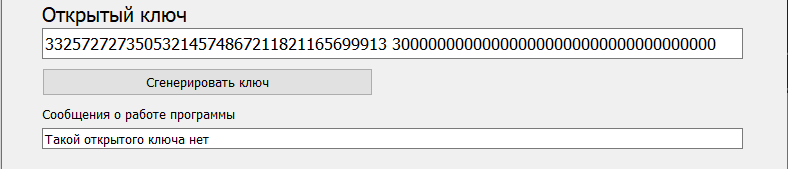


Рисунок 7 – Ввод некорректного ключа

# 5 Вывод

Анализ криптостойкости показал, что разработанную вариацию шифра возможно взломать, хотя это займет некоторое время. Причиной уязвимости является малая длина числа N. Для файлов взлом будет осуществляться через поиск простых чисел.

Анализ производительности не показал каких-то серьезных проблем, программа работает достаточно быстро.

Изучил основы ассиметричного шифрования, вспомнил линейную алгебру, с помощью теоретических знаний о простых шифрах была разработана программа, реализующая шифр RSA.