Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра автоматизированных систем управления (АСУ)

АСИММЕТРИЧНЫЕ АЛГОРИТМЫ ШИФРОВАНИЯ

Отчет по лабораторной работе №3

По дисциплине

«Информационная безопасность»

Студент гр. 431-3

\_\_\_\_\_\_\_\_\_\_\_\_\_ Е.П. Бекиш

(подпись)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(дата)

Руководитель:

Ассистент кафедры АСУ

\_\_\_\_\_\_\_\_\_\_\_\_\_ Я.В. Яблонский

(подпись)

Томск 2024

Оглавление

[1 Цель работы 3](#_Toc179631872)

[2 Задание на лабораторную работу 4](#_Toc179631873)

[3 Описание алгоритма шифрования 5](#_Toc179631874)

[3.1 Алгоритм создания согласованной пары 5](#_Toc179631875)

[3.2 Шифрование и расшифрование 6](#_Toc179631876)

[4 Листинг программы 7](#_Toc179631877)

[5 Примеры работы программы 12](#_Toc179631878)

[6 Вывод 13](#_Toc179631879)

# Цель работы

Познакомиться и научиться работать с асимметричными алгоритмами шифрования.

# Задание на лабораторную работу

Задание по варианту №4: пользуясь алгоритмом RSA с параметрами p=3823, q=2269, e=11, напишите программу, которая позволит зашифровать произвольный открытый текст, предварительно закодировав его согласно прилагаемым таблицам 1, 2, 3 и расшифровать его. Зашифрованный текст должен сохраняться в файле для пересылки своему другу. При написании программы используйте алгоритм быстрого возведения в степень и алгоритмы Евклида.

Таблица 2.1 — Кодировка русского алфавита

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| А | Б | В | Г | Д | Е | Ж | З | И | Й | К | Л | М | Н | О | П |
| 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 |
| Р | С | Т | У | Ф | Х | Ц | Ч | Ш | Щ | Ъ | Ы | Ь | Э | Ю | Я |
| 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 |

Таблица 2.2 — Кодировка латинского алфавита

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A | B | C | D | E | F | G | H | I | J | K | L | M |
| 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 | 51 | 52 | 53 | 54 |
| N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
| 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 | 64 | 65 | 66 | 67 |

Таблица 2.3 — Дополнительные символы

|  |  |  |
| --- | --- | --- |
| Пробел | Запятая | Точка |
| 68 | 69 | 70 |

# Описание алгоритма шифрования

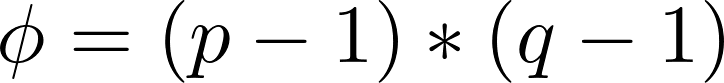
RSA (Rivest, Shamir, Adleman) — это ассиметричный криптографический алгоритм с открытым ключом, основывающийся на вычислительной сложности задачи факторизации больших полупростых чисел. Криптосистема RSA стала первой системой, пригодной как для шифрования, так и для создания цифровой подписи.

В криптографической системе с открытым ключом каждый участник располагает как открытым ключом, так и закрытым ключом. В криптографической системе RSA каждый ключ состоит из пары целых чисел. Каждый участник создаёт свой открытый и закрытый ключ самостоятельно. Закрытый ключ каждый из них держит в секрете, а открытые ключи можно сообщать кому угодно или даже публиковать их. Открытый и закрытый ключи каждого участника обмена сообщениями в криптосистеме RSA образуют «согласованную пару» в том смысле, что они являются взаимно обратными.

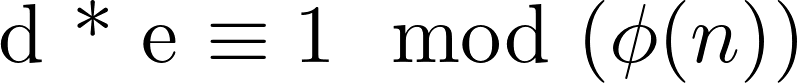
## **3.1 Алгоритм создания согласованной пары**

RSA-ключи генерируются следующим образом:

* Выбираются два различных случайных простых числа p и q заданного размера. Согласно варианту, эти значения уже даны: p=3823, q=2269.
* Вычисляется их произведение n=p\*q, которое называется модулем.
* Вычисляется значение функции Эйлера от числа n по формуле 3.1:

 (3.1)

* Выбирается целое число e, взаимно простое со значением функции Эйлера, которое называется открытой экспонентой. Обычно в качестве e берут простые числа, содержащие небольшое количество единичных бит в двоичной записи, так как в этом случае время, необходимое для шифрования с использованием быстрого возведения в степень, будет меньше. Слишком малые значения e потенциально могут ослабить безопасность схемы RSA. Однако, по варианту данное значение уже задано: e=11.
* Вычисляется число d, мультипликативно обратное к числу e по модулю функции Эйлера. Число d называется секретной экспонентой. Для реализации этой задачи внутри метода использовался расширенный алгоритм Евклида, суть которого заключается в том, что, помимо вычисления непосредственно НОД, находятся коэффициенты Безу. В результате мультипликативное обратное с переданными аргументами e и phi возвращает значение (x mod phi + phi) % phi в том случае, если НОД = 1. Определение мультипликативного обратного в модульной арифметрике представлено в формуле 3.2.

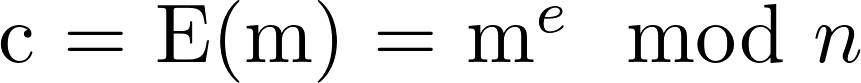
 (3.2)

* Пара (e, n) публикуется в качестве открытого ключа RSA.
* Пара (d, n) играет роль закрытого ключа RSA и держится в секрете.

## **3.2 Шифрование и расшифрование**

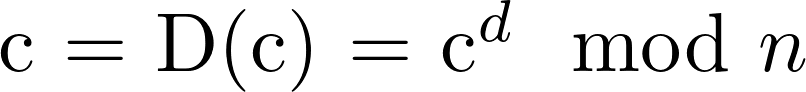
Алгоритм шифрования заключается в следующем:

* Берётся открытый ключ получателя (e, n).
* Берется открытый текст m.
* Сообщение *m* шифруется с использованием открытого ключа получателя по формуле 3.3. При этом используется алгоритм быстрого возведения в степень по модулю в варианте «справа-налево».

 (3.3)

Алгоритм расшифрования заключается в следующем:

* Принимается зашифрованное сообщение *c.*
* Получатель берёт закрытый ключ (d, n).
* К зашифрованному сообщению *c* применяется закрытый ключ для расшифрования сообщения по формуле 3.4.

 (3.4)

# Листинг программы

Листинг файла main.py

# Взять два простых числа

p: int = 3823

q: int = 2269

# Выбрать экспоненту

e: int = 11

# Тестовое сообщение из Wikipedia

m: int = 111111

# Вычисление произведения

def n(p: int, q: int) -> int:

return p \* q

# Вычисление произведения функции Эйлера

def eiler(p: int, q: int) -> int:

return (p - 1) \* (q - 1)

#Вычисление секретной экспоненты

def secret\_e(e: int, eiler: int) -> float:

return (2 \* eiler + 1) / e

#Публикация открытого ключа

def open\_key(eiler: int, n: int) -> tuple[int, int]:

return eiler, n

#Сохранение закрытого ключа

def close\_key(secret\_e: int, n: int) -> tuple[int, int]:

return secret\_e, n

def fast\_pow(m, e, n):

tmp, \_m, \_e, \_n = 1, m, e, n

while (\_e != 0):

if (\_e % 2 == 1):

tmp \*= \_m

\_e >>= 1

\_m \*= \_m

tmp %= \_n

return tmp

def \_\_open\_key(a, b):

if not b:

return (1, 0, a)

y, x, g = \_\_open\_key(b, a % b)

return (x, y - (a // b) \* x, g)

#Шифрование сообщения

def encrypted(m: int | bytes, e: int, n: int) -> int | list[int]:

#if e <= 32:

# return (m \*\* e) % n

if isinstance(m, bytes):

\_bytearray: list[bytes] = []

for byte in m:

\_bytearray += [encrypted(byte, e, n)]

return \_bytearray

#tmp: int = 1

#for \_ in range(e):

# tmp \*= m

# tmp %= n

return fast\_pow(m, e, n)

#Расшифрование сообщения

def decrypted(m: int, secret\_e: int, n: int) -> int | list[bytes]:

#if secret\_e <= 32:

# return (m \*\* secret\_e) % n

if isinstance(m, list) or isinstance(m, bytes):

\_bytearray: list[bytes] = []

for byte in m:

\_bytearray += [decrypted(byte, secret\_e, n)]

return \_bytearray

#tmp: int = 1

#for \_ in range(secret\_e):

# tmp \*= m

# tmp %= n

return fast\_pow(m, secret\_e, n)

def get\_key(eiler: int, n: int) -> tuple[tuple[int, int], int]:

e: int = 2

while True:

\_secret\_e: int = secret\_e(e, eiler)

\_close\_key: tuple[int, int] = close\_key(\_secret\_e, n)

if \_close\_key[0] == int(\_close\_key[0]):

return (int(\_close\_key[0]), \_close\_key[1]), e

e += 1

#def get\_text() -> str:

# return open('file.txt').read()

def get\_text\_byte() -> str:

return open('file.txt', 'rb').read()

#text: str = get\_text()

text\_byte: bytes = get\_text\_byte()

\_n: int = n(p, q)

\_eiler: int = eiler(p, q)

\_secret\_e: int = secret\_e(e, \_eiler)

\_open\_key: tuple[int, int] = open\_key(e, \_n)

\_close\_key, \_\_open\_key = get\_key(\_eiler, \_n)

cipher\_message: list[int | bytes] = encrypted(text\_byte, \*open\_key(\_\_open\_key, \_n))

open('cipher\_text.txt', 'wb').write(bytes([\_int % 256 for \_int in cipher\_message]))

originall\_message: list[bytes] = decrypted(cipher\_message, \*\_close\_key)

open('originall\_text.txt', 'wb').write(bytes(originall\_message))

#print(

# f"""

# {text\_byte=}

# {p=}

# {q=}

# {e=}

# {\_n=}

# {\_eiler=}

# {\_secret\_e=}

# {\_open\_key=}

# {\_close\_key=}

# {\_\_open\_key}

# {m=}

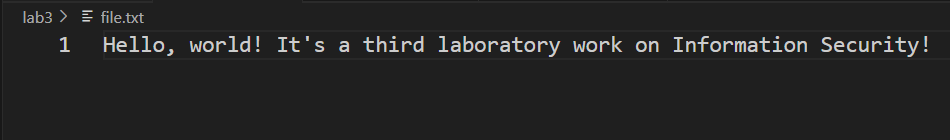
# {cipher\_message=}

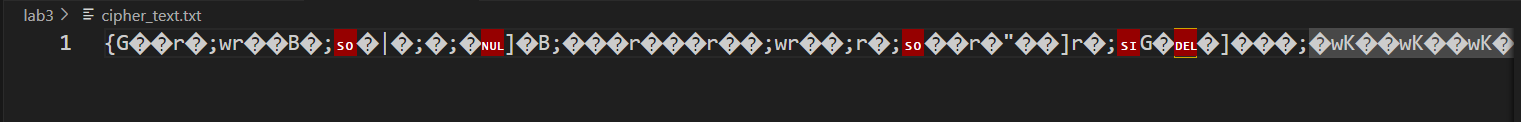
# {originall\_message=}

# """

#)

# Примеры работы программы

Далее подадим на вход в файл .txt два сообщения для шифрования и расшифрования. Результат работы программы можно увидеть на рисунках 5.1 – 5.6.

Рисунок 5.1 – Первое сообщение

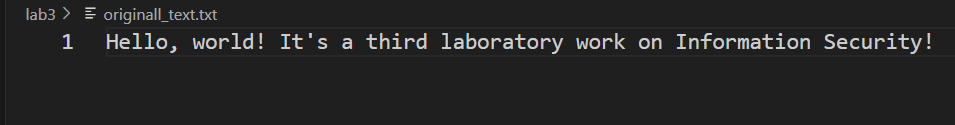
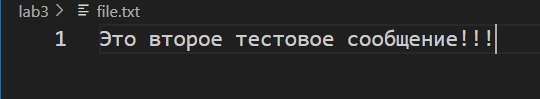
Рисунок 5.2 – Шифрование первого сообщения

Рисунок 5.3 – Расшифрование первого сообщения

Рисунок 5.4 – Второе сообщение

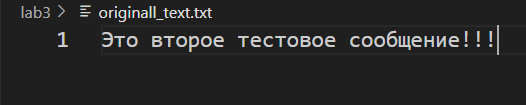
Рисунок 5.5 – Шифрование второго сообщения

Рисунок 5.6 – Расшифрование второго сообщения

# Вывод

В результате выполнения лабораторной работы я познакомился и научился работать с ассиметричными алгоритмами шифрования.