Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Методы защиты информации

ОТЧЕТ

к лабораторной работе №3

на тему

**Асимметричная криптография.**

**Криптосистема Рабина.**

Выполнил Д.С. Шевцова

Проверил Е.А. Лещенко

Минск 2024

СОДЕРЖАНИЕ

[Введение 3](#_Toc158758843)

[1 Краткие теоретические сведения 4](#_Toc158758844)

[2 Результаты выполнения лабораторной работы](#_Toc158758845) 7

[Заключение](#_Toc158758846) 8

[Приложение А](#_Toc158758848) [(обязательное)](#_Toc158758849) [Листинг кода](#_Toc158758850) 9

## ВВЕДЕНИЕ

Целью данной лабораторной работы является реализация программных средств шифрования и дешифрования текстовых файлов при помощи криптосистемы Рабина.

Также необходимо добавить интерфейс командной строки, который позволит выбирать файл для шифрования/дешифрования, задавать ключи и режимы операций и предусмотреть обработку возможных ошибок (неверный формат файла, невалидный ключ, …).

## 1 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Криптосистема Рабина – криптографическая система с открытым ключом, безопасность которой обеспечивается сложностью поиска квадратных корней в кольце остатков по модулю составного числа. Безопасность системы, как и безопасность метода RSA, обусловлена сложностью разложения на множители больших чисел. Зашифрованное сообщение можно расшифровать 4 способами. Недостатком системы является необходимость выбора истинного сообщения из 4-х возможных.

Система Рабина, как и любая асимметричная криптосистема, использует открытый и закрытый ключи. Открытый ключ используется для шифрования сообщений и может быть опубликован для всеобщего обозрения. Закрытый ключ необходим для расшифровки и должен быть известен только получателям зашифрованных сообщений.

Процесс генерации ключей следующий: выбираются два случайных числа p и q с учётом следующих требований:

1 числа должны быть большими;

2 числа должны быть простыми;

3 должно выполняться условие: p ≡ q ≡ 3 mod 4.

Выполнение этих требований сильно ускоряет процедуру извлечения корней по модулю р и q, вычисляется число n = p · q, где число n – открытый ключ; числа p и q – закрытый.

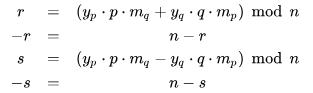
Исходное сообщение m (текст) шифруется с помощью открытого ключа – числа n по следующей формуле: c = m² mod n.

Благодаря использованию умножения по модулю скорость шифрования системы Рабина больше, чем скорость шифрования по методу RSA, даже если в последнем случае выбрать небольшое значение экспоненты.

Для расшифровки сообщения необходим закрытый ключ – числа p и q. Процесс расшифровки выглядит следующим образом:

1 сначала, используя алгоритм Евклида, из уравнения  находят числа y\_p и y\_q;

2 далее, используя китайскую теорему об остатках, вычисляют четыре числа:





Одно из этих чисел является истинным открытым текстом m.

# 2 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

Для шифрования и дешифрования файлом используется интерфейс командной строки, где указывается путь к файлу, также режим, а именно шифрования или дешифрования, длина ключа в битах. При дешифровании указываются также значения открытого и закрытых ключей. На рисунке 2.1 указана команда для шифрования, а также данные ключей. На рисунках 2.2 и 2.3 указаны команда для расшифрации и расшифрованные данные. Также предусмотрена обработка некорректных данных, таких как неверный формат файла, неверные ключи или отсутствие файлов.

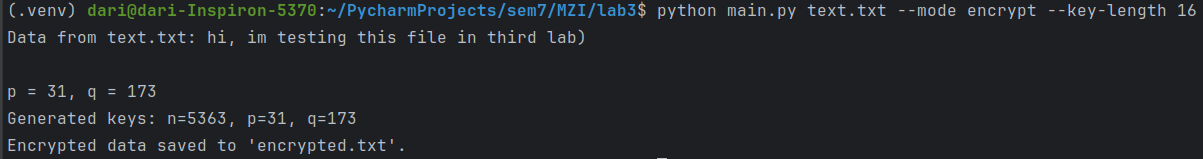


Рисунок 2.1 – Команда для шифрования и начальные данные



Рисунок 2.2 – Команда для дешифрования

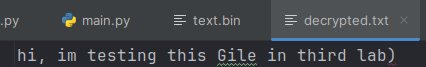


Рисунок 2.3 – Расшифрованный текст

## ЗАКЛЮЧЕНИЕ

В ходе выполнения данной лабораторной работы была написана реализация программных средств шифрования и дешифрования текстовых файлов при помощи криптосистемы Рабина.

Также был добавлен интерфейс командной строки, который позволяет выбирать файл для шифрования/дешифрования, задавать ключи и режимы операций и предусмотреть обработку возможных ошибок.

# **ПРИЛОЖЕНИЕ А**

## (обязательное)

## Листинг кода

lab3.py

import random

from sympy import isprime, mod\_inverse

def mod\_sqrt(a, p):

assert isprime(p), "p should be simple"

# Если p ≡ 3 (mod 4), можно использовать упрощенный случай

if p % 4 == 3:

return pow(a, (p + 1) // 4, p)

# Алгоритм Тонелли-Шенкса для общего случая

def legendre\_symbol(a, p):

return pow(a, (p - 1) // 2, p)

def find\_non\_residue(p):

for i in range(2, p):

if legendre\_symbol(i, p) == p - 1:

return i

return None

# Находим квадратичный невычет z

z = find\_non\_residue(p)

q = p - 1

s = 0

while q % 2 == 0:

q //= 2

s += 1

# Инициализация переменных

m = s

c = pow(z, q, p)

t = pow(a, q, p)

r = pow(a, (q + 1) // 2, p)

while t != 1:

t2i = t

i = 0

for i in range(1, m):

t2i = pow(t2i, 2, p)

if t2i == 1:

break

b = pow(c, 1 << (m - i - 1), p)

m = i

c = pow(b, 2, p)

t = (t \* c) % p

r = (r \* b) % p

return r

# Генерация простых чисел

def generate\_large\_prime(bits):

while True:

candidate = random.getrandbits(bits)

if isprime(candidate):

return candidate

# Генерация ключей

def make\_key\_pair(length):

if length < 4:

raise ValueError('Key pair length must be greater than 4')

# Генерация двух больших простых чисел

p = generate\_large\_prime(length // 2)

q = generate\_large\_prime(length // 2)

n = p \* q

print("p = {}, q = {}".format(p, q))

return n, p, q # Возвращаем открытый ключ n и закрытые ключи p, q

# Шифрование

def encrypt(data, n):

encrypted\_data = []

for char in data:

m = ord(char) # Преобразуем символ в число

c = pow(m, 2, n) # m^2 % n

encrypted\_data.append(c)

return encrypted\_data

def decrypt(encrypted\_data, p, q):

n = p \* q

decrypted\_message = []

for c in encrypted\_data:

if c < 0 or c >= n:

raise ValueError(f"Encrypted value {c} is out of range for modulus n={n}")

# Найти квадратные корни по модулям p и q

roots\_p = mod\_sqrt(c, p)

roots\_q = mod\_sqrt(c, q)

if roots\_p is None or roots\_q is None:

raise ValueError(f"Cannot find sqrt for {c}")

# Применяем китайскую теорему об остатках (CRT)

possible\_solutions = []

for r\_p in [roots\_p, -roots\_p % p]:

for r\_q in [roots\_q, -roots\_q % q]:

m = (r\_p \* q \* mod\_inverse(q, p) + r\_q \* p \* mod\_inverse(p, q)) % n

possible\_solutions.append(m)

# Для простоты берем первое решение, которое является символом ASCII

for solution in possible\_solutions:

if 32 <= solution <= 126: # Диапазон печатных символов ASCII

decrypted\_message.append(chr(solution))

break

else:

# Если не нашли подходящего символа, просто добавляем знак вопроса

decrypted\_message.append('?')

return ''.join(decrypted\_message)

# Запись в файл

def write\_to\_file(file\_name, data):

with open(file\_name, "w") as file:

file.write(str(data))

# Чтение из файла

def read\_from\_file(file\_name):

with open(file\_name, "r") as file:

return file.read()

main.py

import argparse

import os

from lab3 import \*

def validate\_file(file\_name):

"""Проверяет существование и формат файла (только .txt)."""

if not os.path.exists(file\_name):

raise FileNotFoundError(f"Файл {file\_name} не найден.")

if not os.path.isfile(file\_name):

raise ValueError(f"{file\_name} не является файлом.")

if not file\_name.endswith(".txt"):

raise ValueError(f"Файл {file\_name} не является .txt файлом. Можно шифровать только .txt файлы.")

def validate\_keys(p, q, n):

"""Проверяет корректность переданных ключей."""

if p \* q != n:

raise ValueError(f"Неверные ключи: p={p}, q={q} не дают n={n}.")

def main():

parser = argparse.ArgumentParser(description="Шифрование/дешифрование данных.")

parser.add\_argument("file", help="Имя файла для шифрования или дешифрования.")

parser.add\_argument("--mode", choices=["encrypt", "decrypt"], required=True, help="Режим работы: encrypt или decrypt.")

parser.add\_argument("--key-length", type=int, help="Длина ключа (только для шифрования).")

parser.add\_argument("--n", type=int, help="Параметр n (для дешифрования).")

parser.add\_argument("--p", type=int, help="Простое число p (для дешифрования).")

parser.add\_argument("--q", type=int, help="Простое число q (для дешифрования).")

args = parser.parse\_args()

try:

# Проверка на существование и формат файла

validate\_file(args.file)

if args.mode == "encrypt":

# Генерация ключей для шифрования

if args.key\_length is None:

raise ValueError("Для шифрования необходимо указать длину ключа через --key-length.")

data = read\_from\_file(args.file)

print(f"Data from {args.file}: {data}\n")

# Генерация ключей

n, p, q = make\_key\_pair(args.key\_length)

print(f"Generated keys: n={n}, p={p}, q={q}")

# Шифрование данных

encrypted\_data = encrypt(data, n)

write\_to\_file("encrypted.txt", encrypted\_data)

print("Encrypted data saved to 'encrypted.txt'.")

elif args.mode == "decrypt":

# Проверка на наличие ключей для дешифрования

if None in [args.n, args.p, args.q]:

raise ValueError("Для дешифрования необходимо указать ключи через --n, --p, --q.")

# Проверка ключей

validate\_keys(args.p, args.q, args.n)

encrypted\_data = eval(read\_from\_file(args.file))

decrypted\_data = decrypt(encrypted\_data, args.p, args.q)

write\_to\_file("decrypted.txt", decrypted\_data)

print("Decrypted data saved to 'decrypted.txt'.")

except FileNotFoundError as fnf\_error:

print(f"Ошибка: {fnf\_error}")

except ValueError as ve:

print(f"Ошибка: {ve}")

except Exception as e:

print(f"Непредвиденная ошибка: {e}")

if \_\_name\_\_ == '\_\_main\_\_':

main()