1. Министерство науки и высшего образования Российской Федерации
2. Санкт-Петербургский Политехнический Университет Петра Великого
3. —
4. **Институт Кибербезопасности и Защиты Информации**

**ЛАБОРАТОРНАЯ РАБОТА № 2**

1. **«Атаки на криптосистему RSA, использующие данные о показателях»**
2. по дисциплине «Криптографические методы защиты информации»
3. Выполнил
4. студент гр. 4851003/80802 Сошнев М.Д.

<*подпись*>

1. Проверил
2. преподаватель Ярмак А.В.

<*подпись*>

1. Санкт-Петербург
2. 2022

# Цель работы

Изучение уязвимостей криптосистемы RSA по отношению к атакам, использующим показатели, обладающие определенными свойствами, генерация параметров криптосистемы RSA.

# Задача

* Получить у преподавателя вариант задания.
* Разработать программу **П-1**, которая реализует разложение составного числа на множители по известным показателям RSA и вычисляет закрытый ключ другого пользователя в случае общего модуля . Программа **П-1** должна принимать на вход открытый и закрытый ключ RSA, а также открытый ключ RSA другого пользователя; на выходе – возвращать множители модуля RSA, закрытый ключ другого пользователя.
* Разработать программу **П-2**, которая реализует атаку Винера на криптосистему RSA. Программа **П-2** должна принимать на вход открытый ключ RSA: число и показатель ; на выходе – возвращать закрытый показатель .
* Разработать программу **П-3**, которая, в соответствии с вариантом задания, реализует одну из следующих атак:
* бесключевое дешифрование широковещательного сообщения в случае малого общего показателя . Для восстановления числа по китайской теореме об остатках и нахождения корня степени допускается использовать готовые процедуры. Тело сообщения – файл, зашифрованный алгоритмом AES-256 в режиме CBC. При длине исходного файла, не кратной 16 байтам, открытый текст дополняется символами с кодом 0x03 так, чтобы длина сообщения (файла) перед зашифрованием была кратна 16 байтам. Изначальная длина доступна в элементе данных «Длина сообщения» заголовка. Удаление выравнивания не обязательно. Программа **П-3** должна принимать на вход открытые ключи нескольких пользователей, а также шифртексты широковещательного сообщения; на выходе – возвращать восстановленный открытый текст;
* бесключевое дешифрование сообщения в случае малого порядка элемента в группе .. Программа **П-3** должна принимать на вход открытый ключ RSA, шифртекст; на выходе – возвращать восстановленный открытый текст.

Разработать программу **П-4**, которая осуществляет генерацию параметров криптосистемы RSA. Разработка процедур генерации случайных чисел и проверки числа на простоту не требуется. Генерация ключей должна обеспечивать выработку безопасных параметров криптосистемы с учетом произведенных атак. Программа **П-4** должна принимать на вход битовую длину модуля RSA; на выходе – возвращать параметры криптосистемы: . Выполнить анализ сгенерированных параметров с помощью инструмента CrypTool. Описание утилиты представлено в *Приложении Б*.

# Ход работы

Была разработана программа, реализующая 3 атаки на систему RSA и генерация оптимильных параметров для RSA. Использовался язык python и библиотека rsa.

## Атака по известным показателям

В начале программа генерирует две пары ключей (закрытый-открытый) при общем модуле n. Далее программа пытается взломать один из закрытых ключей по одному известному закрытому ключу. Поиск данного ключа сводится к нахождению корня из 1 в кольце Z/nZ. Продемонстрируем:

Рисунок 1 — атака по известным показателям

Стоит отметить что атака прошла успешна и ключ был скомпрометирован даже при большом размере ключа (2048 бит)

## Атака Винера

Данная атака пытается скомпрометировать закрытый ключ по известному открытому. При этом она имеет место быть только при малых показателях d. Продемонстрируем атаку для следующего ключа:

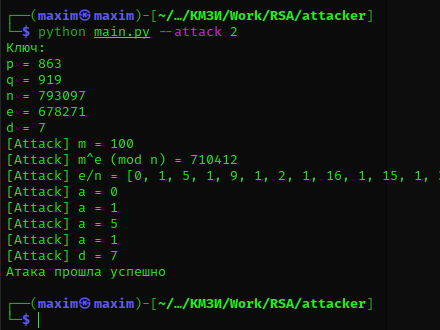
p = 863

q = 919

n = 793097

e = 678271

d = 7

Рисунок 2 — атака Винера

## Атака «бесключевое дешифрование»

Данная атака предполагает, что у параметра e малый порядок в кольце Z/nZ. Нарушитель пытается скомпрометировать исходное сообщение по шифртексту и открытому ключу. Программа написана таким образом, что нарушитель делает определенное число итераций и если за это время не удалось посчитать исходное сообщение, то считается, что атака закончилась неудачей. Продемонстрируем атаку для следующего ключа:

p = 863

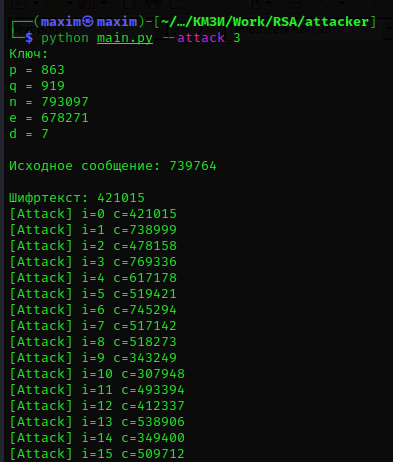
q = 919

n = 793097

e = 678271

d = 7

и исходного сообщения m = 739764

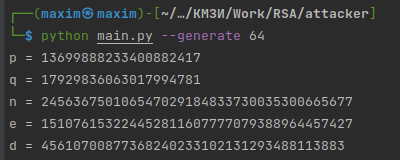
Рисунок 3, 4 — атака «бесключевое дешифрование»

## Генерация параметров для RSA

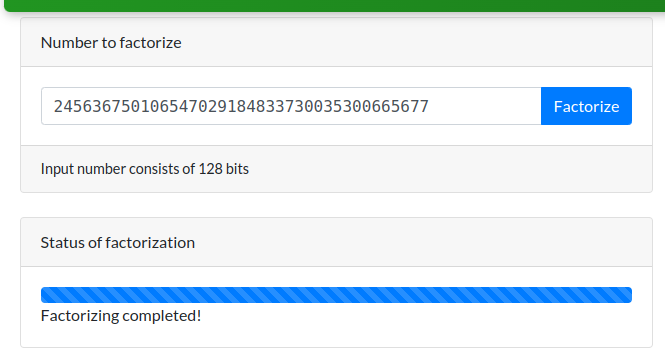
Программа работает следующим образом:

1. В зависимости от размерности, необходимой пользователю определяет промежуток для p и q
2. Рандомно генерирует p, пока оно не будет простым
3. Рандомно генерирует q, пока оно не будет простым
4. Считает n=pq и f(n)=(p-1)(q-1)
5. Рандомно генерирует e, пока оно не будет взаимно-простым с f(n)
6. Считает d, как e-1(mod f(n))

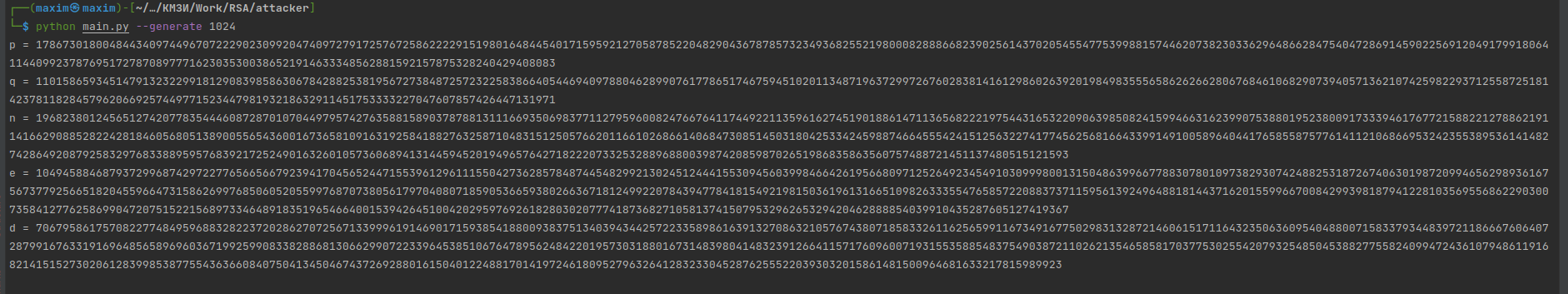
Пример работы программы для размерности 64 бит:

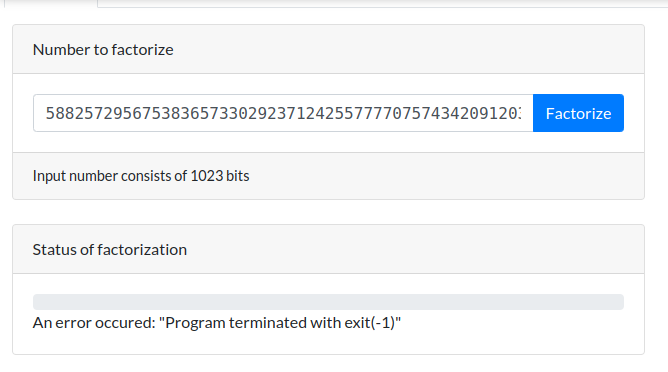
Рисунок 5 — генерация ключа размерность 64 бит

Проанализируем данный ключ с помощью CryptTools:

Рисунок 6 — данный ключ легко раскладывается на множители

Тогда покажем, что ключ размерностью 2048 бит уже будет стойким:

Рисунок 7 - генерация ключа размерность 2048 бит

Рисунок 8 — CryptTools заканчивает работу с ошибкой

# Контрольные вопросы

1. При каком условии возможна атака Винера?

, (достаточно малый показатель d)

1. Перечислите методы противодействия атаке Винера.

Исползовать

Использовать ,

1. Почему числа и необходимо выбирать так, чтобы   
    имели большие простые делители?

В противном случае у параметра e будет маленький порядок в

и возможна атака «Безключевое дешифрование»

1. Предложите способы защиты от атаки бесключевого дешифрования широковещательных сообщений без отказа от использования общих малых закрытых показателей.

Перед шифрованием применять некое преобразование над сообщением, например , тогда после преобразования сообщения m будут уникальны для каждого пользователя и применения К.Т.О. будет невозможным.

# Вывод

В результате выполнения лабораторной работы были исследованы некоторые атаки на криптосистему RSA. Можно сделать следующие выводы: для каждого абонента необходимо генерировать не только уникальные параметры e, d но и уникальные параметры n, причем данный параметр должен быть произведением уникальных p и q, таких, что p-1 и q-1 не является сильно составным числом. В идеале: p-1=2p1, q-1=2q1. Также необходимо, чтобы параметр d был достаточно большим (больше, чем ) и чтобы у параметра e был большой порядок в кольце Z/nZ. Рекомендуется брать e такой, что его двоичное представление имеет вид 10...01. Также, размер ключа должен составлять как минимум 1024 бит.

# Приложение

import random

import math

import rsa

from rsa.common import inverse

class AttackCommonModule:

def \_\_init\_\_(self, n, e, d, logs=True):

self.\_\_e\_b = e

self.\_\_d\_b = d

self.\_\_n = n

self.\_\_logs = logs

def \_\_call\_\_(self, e):

"""

:return: private\_key

"""

m = self.\_\_e\_b \* self.\_\_d\_b - 1

if self.\_\_logs:

print('[Attack] m = {}'.format(m))

s = m

while (s & 1) == 0:

s >>= 1

print('[Attack] s = {}'.format(s))

last\_b = self.\_\_n - 1

while last\_b == self.\_\_n - 1:

a = random.randint(0, self.\_\_n)

if self.\_\_logs:

print('[Attack] a = {}'.format(a))

b = pow(a, s, self.\_\_n)

if self.\_\_logs:

print('[Attack] wait...')

while b != 1:

# print('[Attack] b = {}'.format(b))

last\_b = b

b = pow(b, 2, self.\_\_n)

t = last\_b

p, q = math.gcd(t + 1, self.\_\_n), math.gcd(t - 1, self.\_\_n)

f\_n = (p - 1) \* (q - 1)

d = inverse(e, f\_n)

if self.\_\_logs:

print('[Attack] p = {}'.format(p))

print('[Attack] q = {}'.format(q))

print('[Attack] f(n) = {}'.format(f\_n))

print('[Attack] d = {}'.format(d))

return p, q, d

class AttackViner:

def \_\_init\_\_(self, logs=True):

self.\_\_logs = logs

def \_\_call\_\_(self, public\_key):

"""

:return: private\_key.d

"""

m = 100

if self.\_\_logs:

print('[Attack] m = {}'.format(m))

c = pow(m, public\_key.e, public\_key.n)

if self.\_\_logs:

print('[Attack] m^e (mod n) = {}'.format(c))

frac = self.\_\_to\_continued\_fraction(public\_key.e, public\_key.n)

if self.\_\_logs:

print('[Attack] e/n = {}'.format(frac))

Q\_, Q\_\_, P\_, P\_\_ = 0, 1, 1, 0

for a in frac:

if self.\_\_logs:

print('[Attack] a = {}'.format(a))

P = a \* P\_ + P\_\_

Q = a \* Q\_ + Q\_\_

P\_\_, Q\_\_ = P\_, Q\_

P\_, Q\_ = P, Q

if pow(c, Q, public\_key.n) == m:

return Q

return None

@staticmethod

def \_\_to\_continued\_fraction(a, b):

a, q = divmod(a, b)

t, res = b, [a]

while q != 0:

next\_t = q

a, q = divmod(t, q)

t = next\_t

res.append(a)

return res

class AttackKeyLessDecrypt:

def \_\_init\_\_(self, iter=1\_000\_000, logs=True):

self.\_\_iter = iter

self.\_\_logs = logs

def \_\_call\_\_(self, public\_key, cipher\_text):

"""

:return: open\_text

"""

c = cipher\_text

for i in range(self.\_\_iter):

if self.\_\_logs:

print('[Attack] i={} c={}'.format(i, c))

c\_pred = c

c = pow(c, public\_key.e, public\_key.n)

if (c - cipher\_text) % public\_key.n == 0:

return c\_pred

return None

import random

import rsa.common

import rsa.prime

import argparse

from attacks import \*

def generate\_keys\_with\_common\_n(bits):

# Первый ключ

\_, private\_key\_1 = rsa.newkeys(bits)

# Второй ключ на тех же p, q, n но новые e, d

e = 0b1000000000000000000000001

d = rsa.common.inverse(e, (private\_key\_1.p - 1) \* (private\_key\_1.q - 1))

private\_key\_2 = rsa.PrivateKey(private\_key\_1.n, e, d, private\_key\_1.p, private\_key\_1.q)

print('p = {}'.format(private\_key\_1.p))

print('q = {}'.format(private\_key\_1.q))

print('n = {}'.format(private\_key\_1.n))

print('\nПервый ключ:')

print('e = {}'.format(private\_key\_1.e))

print('d = {}'.format(private\_key\_1.d))

print('\nВторой ключ:')

print('e = {}'.format(private\_key\_2.e))

print('d = {}\n'.format(private\_key\_2.d))

return private\_key\_1, private\_key\_2

def try\_attack\_1():

private\_key\_1, private\_key\_2 = generate\_keys\_with\_common\_n(1024)

attack = AttackCommonModule(private\_key\_1.n, private\_key\_1.e, private\_key\_1.d)

p\_attacked, q\_attacked, d\_attacked = attack(private\_key\_2.e)

if private\_key\_2.p == p\_attacked and private\_key\_2.q == q\_attacked and \

private\_key\_2.d == d\_attacked:

print('Атака прошла успешно')

else:

print('Атака прошла безуспешно')

def try\_attack\_2():

# public\_key, private\_key = rsa.newkeys(128)

public\_key = rsa.PublicKey(793097, 678271)

private\_key = rsa.PrivateKey(793097, 678271, 7, 863, 919)

print('Ключ:')

print('p = {}'.format(private\_key.p))

print('q = {}'.format(private\_key.q))

print('n = {}'.format(private\_key.n))

print('e = {}'.format(private\_key.e))

print('d = {}'.format(private\_key.d))

attack = AttackViner()

private\_key\_attacked = attack(public\_key)

if private\_key.d == private\_key\_attacked:

print('Атака прошла успешно')

else:

print('Атака прошла безуспешно')

def try\_attack\_3():

# public\_key, private\_key = rsa.newkeys(64, exponent=50)

public\_key = rsa.PublicKey(793097, 678271)

private\_key = rsa.PrivateKey(793097, 678271, 7, 863, 919)

print('Ключ:')

print('p = {}'.format(private\_key.p))

print('q = {}'.format(private\_key.q))

print('n = {}'.format(private\_key.n))

print('e = {}'.format(private\_key.e))

print('d = {}\n'.format(private\_key.d))

m = random.randint(0, public\_key.n)

print('Исходное сообщение: {}\n'.format(m))

c = pow(m, public\_key.e, public\_key.n)

print('Шифртекст: {}'.format(c))

attack = AttackKeyLessDecrypt(logs=True)

m\_attacked = attack(public\_key, c)

if m\_attacked == m:

print('Атака прошла успешно')

else:

print('Атака прошла безуспешно {} {}'.format(m\_attacked, m))

def generate(bits):

left = 1 << (bits - 1)

right = (1 << bits) - 1

# generate p

while True:

p = random.randint(left, right)

if rsa.prime.is\_prime(p):

break

# generate q

while True:

q = random.randint(left, right)

if rsa.prime.is\_prime(q) and p != q:

break

n = p \* q

f\_n = (p - 1) \* (q - 1)

# generate e

while True:

e = random.randint(1, f\_n - 1)

if math.gcd(e, f\_n) == 1:

break

d = rsa.common.inverse(e, f\_n)

print('p = {}'.format(p))

print('q = {}'.format(q))

print('n = {}'.format(n))

print('e = {}'.format(e))

print('d = {}'.format(d))

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

parser = argparse.ArgumentParser()

parser.add\_argument('--attack', required=False)

parser.add\_argument('--generate', required=False)

args = parser.parse\_args()

if args.attack == '1':

try\_attack\_1()

elif args.attack == '2':

try\_attack\_2()

elif args.attack == '3':

try\_attack\_3()

if args.generate:

generate(int(args.generate))