Министерство науки и высшего образования Российской Федерации

Санкт-Петербургский Политехнический Университет Петра Великого

—

Институт компьютерных наук и кибербезопасности

Высшая школа кибербезопасности

**ЛАБОРАТОРНАЯ РАБОТА № 2**

**«Атаки на криптосистему RSA, использующие данные о показателях»**

по дисциплине «Криптографические методы защиты безопасности»

Выполнил

студент гр. 5151004/90101 Кондачков Е.Д.

Преподаватель

Ассистент Ярмак А.В.

Санкт-Петербург

2023

Содержание

[1. Цель работы 2](#_Toc146885998)

[2. Задачи работы 3](#_Toc146885999)

[3. Ход работы 4](#_Toc146886000)

[1.1 Реализация разложения составного числа на множители по известным показателям RSA 4](#_Toc146886001)

[1.2 Реализация атаки Винера на криптосистему RSA 5](#_Toc146886002)

[1.3 Реализация бесключевого дешифрования сообщения 6](#_Toc146886003)

[1.4 Генерация безопасных параметров RSA 7](#_Toc146886004)

[4. Ответы на контрольные вопросы 10](#_Toc146886005)

[5. Выводы 11](#_Toc146886006)

[Приложение А 12](#_Toc146886007)

# Цель работы

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

# Задачи работы

1. Изучить описание трех атак на криптосистему RSA, приведенное в методическом пособии к лабораторной работе;
2. Реализовать алгоритм разложения составного числа на множители по известным показателям RSA;
3. Реализовать атаку Винера на криптосистему RSA;
4. Реализовать бесключевое дешифрование сообщения в случае малого порядка элемента 𝑒 в группе .

# Ход работы

## Реализация разложения составного числа на множители по известным показателям RSA

В качестве первой атаки рассматривалось разложения составного числа на множители по известным показателям RSA. Основной задачей данной атаки являлось нахождение разложения числа n на множители и , что обычно является весьма трудоемкой задачей. Алгоритм основывается на том, что отправитель и получатель имеют один и тот же модуль . Это означает, что, например, отправитель, зная собственные открытый показатель и закрытый показатель , а также открытый показатель получателя и общий модуль может найти числа и – делители общего модуля. Уже по известным и можно было найти получателя.

Алгоритм данной атаки выглядит следующим образом:

*Вход*. Число , показатели , такие, что , открытый показатель другого пользователя.

*Выход*. Делители и числа , закрытый показатель другого пользователя.

1. Представить разность в виде , где – нечетное число.
2. Выбрать случайное и вычислить .
3. Вычислять , ,  
   , до получения 𝑙 такого, что . Если , то вернуться на шаг 2, иначе положить   
   .
4. Положить 𝑝, .
5. Вычислить и найти закрытый ключ другого пользователя: .
6. Результат: , .

Реализация данного алгоритма в коде представлена на рисунке Рисунок 1.

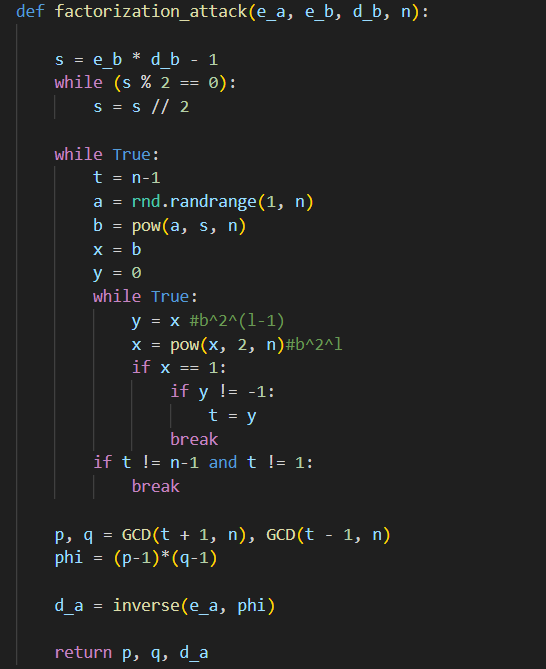


Рисунок – Реализация атака разложения на множители

## Реализация атаки Винера на криптосистему RSA

Второй атакой была атака Винера, позволяющая получить закрытый показатель , зная открытый показатель и модуль , то есть в случае получения открытого ключа пользователя. Данная атака реализуется при помощи разложения отношения в виде непрерывной дроби и ноходения подходящей к ней дроби . При нахождении корректной дроби, число в ее знаменателе будет совпадать с , обратным уже известному . Важно отметить, что из-за сложности и некоторой непредсказуемости описанного механизма параметры RSA для данной атаки были зафиксированы.

Алгоритм выглядит следующим образом:

*Вход*. Число , показатель .

*Выход.* Закрытый показатель .

1. Представить в виде обыкновенной непрерывной дроби , где .
2. Для выполнить следующие действия.
   1. Вычислить -ю подходящую дробь к : , где .
   2. Для произвольного сообщения проверить выполнение сравнения . Если сравнение выполняется, то положить и перейти на шаг 3.
3. Результат: .

Реализация данной атаки представлена на рисунке Рисунок 2. Параметры, выбранные для демонстрации атаки представлены на рисунке Рисунок 3.

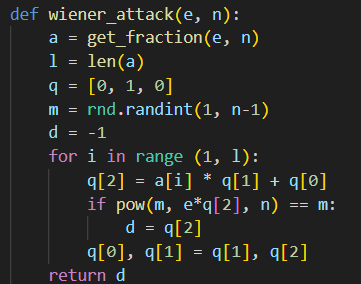


Рисунок – Реализация атаки Винера

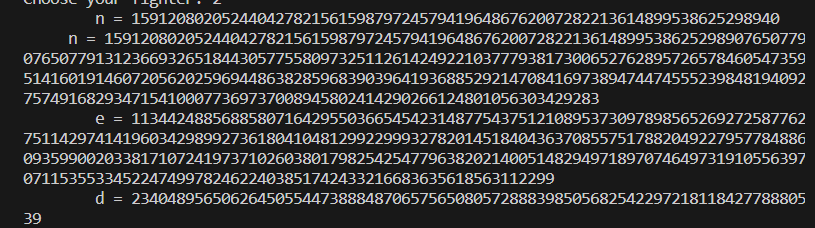


Рисунок – Параметры RSA для демонстрации атаки Винера

## Реализация бесключевого дешифрования сообщения

В качестве последней атаки был рассмотрен метод дешифрования сообщений без использования закрытого показателя d, что реализуемо, если порядок открытого показателя e мал. Для демонстрации этой атаки размерность всех параметров RSA была сильно снижена в угоду скорости работы программы.

Алгоритм данной атаки таков:

*Вход.* Составное число , шифртекст , открытый показатель .

*Выход.* Открытый текст .

1. Для выполнить следующие действия.
   1. Вычислить , где .
   2. Проверить выполнение сравнения . Если сравнение выполняется, то положить и перейти на шаг 2.
2. Результат: .

Генерация параметров представлена на рисунке Рисунок 4. Реализация алгоритма представлена на рисунке Рисунок 5.

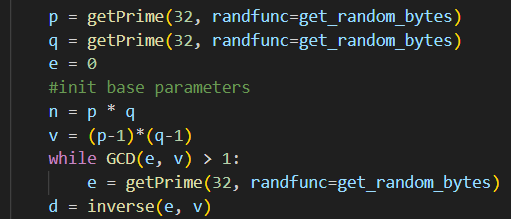


Рисунок – Генерация параметров для демонстрации атаки

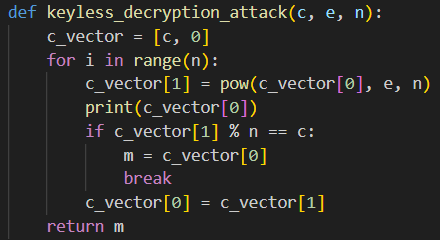


Рисунок – Реализация атаки

## Генерация безопасных параметров RSA

Для программы были выбраны большие простые числа *p* и *q* с разницей в 12 байт. Также они являются безопасными, то есть , где – простое число. Показатель *е* был выбран 16 бит, не слишком большой или маленький.

Программа принимает на выход битовую длину модуля RSA, на выходе возвращает параметры криптосистемы: *n*, *e*, *d*. Результат отражен на рисунке Рисунок 6.

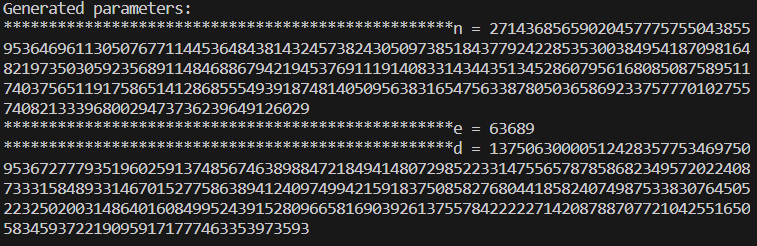


Рисунок – Результат работы программы генерации параметров

Сгенерированные параметры были проверены утилитой Cryptools.

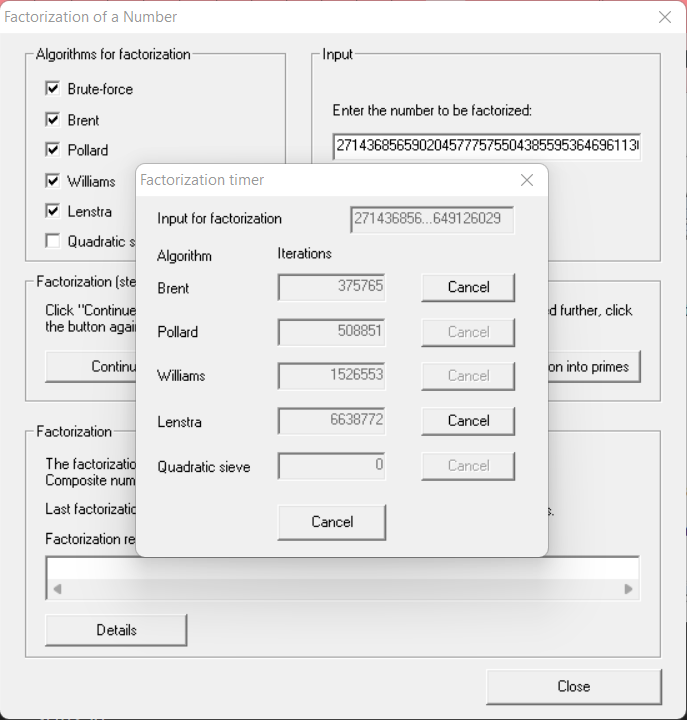


Рисунок – Разложение модуля на множители

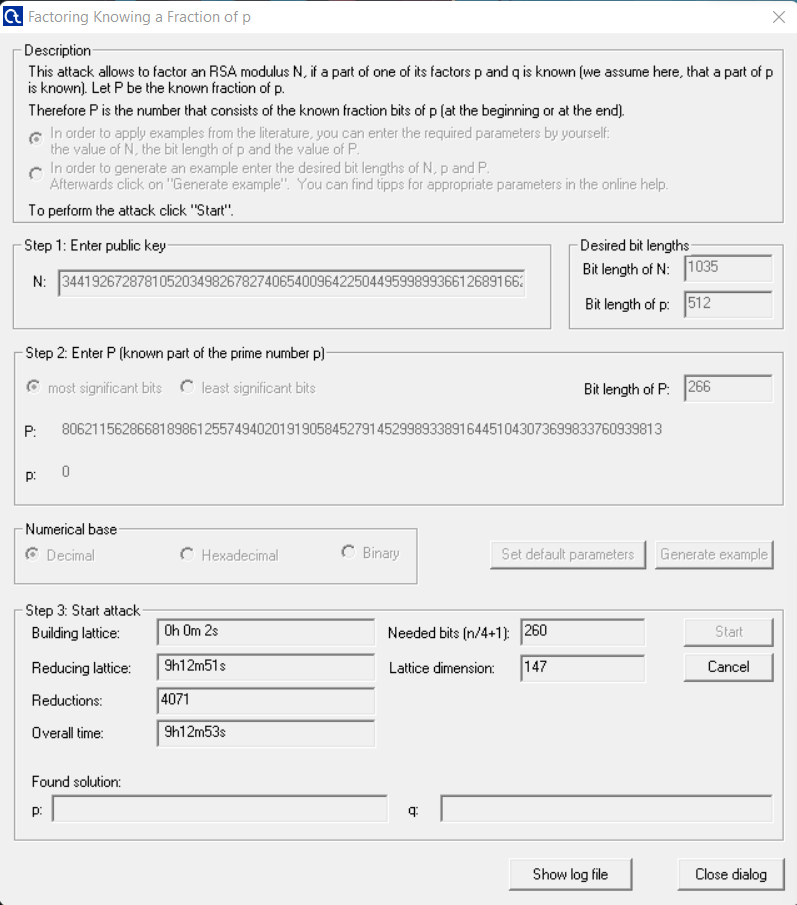


Рисунок – Нахождение p и q по части p

# Ответы на контрольные вопросы

1. При каком условии возможна атака Винера?

Атака Винера возможна, если не выполняется условие

1. Перечислите методы противодействия атаке Винера.

Соблюдение условия .

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

Для усложнения поиска числа n при помощи алгоритмов факторизации.

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

После превышения количества блоков шифртекстов обновлять параметры криптосистемы, чтобы сменить параметр e. Не допускать превышения количества блоков шифртекстов над параметром e путем увеличения размера блока шифртекста

# Выводы

В рамках выполнения настоящей лабораторной работы были изучены методы реализации атак на криптосистему RSA. Для реализации атак на криптосистему RSA при генерации ее параметров нужно допустить ошибки, которые были описаны и изучены в методических материалах к данной лабораторной работе. По итогу проведения различных атак на криптосистему RSA генератор параметров криптосистемы был оснащен дополнительными проверками для повышения уровня криптостойкости криптосистемы RSA.

# Приложение А

Листинг разработанной программы на языке Python3:

import random as rnd

import math

import time

from sympy import isprime, sqrt

from Crypto.Util.number import getPrime, inverse, GCD

from Crypto.Random import get\_random\_bytes

#--------FACTORIZATION BLOCK--------#

def factorization\_attack(e\_a, e\_b, d\_b, n):

s = e\_b \* d\_b - 1

while (s % 2 == 0):

s = s // 2

while True:

t = n-1

a = rnd.randrange(1, n)

b = pow(a, s, n)

x = b

y = 0

while True:

y = x #b^2^(l-1)

x = pow(x, 2, n)#b^2^l

if x == 1:

if y != -1:

t = y

break

if t != n-1 and t != 1:

break

p, q = GCD(t + 1, n), GCD(t - 1, n)

phi = (p-1)\*(q-1)

d\_a = inverse(e\_a, phi)

return p, q, d\_a

def get\_e\_and\_d(phi):

e = 0

while GCD(e, phi) > 1:

e = getPrime(1024, randfunc=get\_random\_bytes)

d = inverse(e, phi)

return e, d

def get\_first\_program\_param():

p = getPrime(1024, randfunc=get\_random\_bytes)

q = getPrime(1024, randfunc=get\_random\_bytes)

e\_a = 0

e\_b = 0

n = p \* q

phi = (p-1)\*(q-1)

e\_a, d\_a = get\_e\_and\_d(phi)

while True:

e\_b, d\_b = get\_e\_and\_d(phi)

if e\_a != e\_b and d\_a != d\_b:

break

return e\_a, e\_b, d\_a, d\_b, n

def first\_program():

e\_a, e\_b, d\_a, d\_b, n = get\_first\_program\_param()

print(f' n = {n}\n\

e\_a = {e\_a}\n\

d\_a = {d\_a}\n\

e\_b = {e\_b}\n\

d\_b = {d\_b}')

p\_predicted, q\_predicted, d\_a\_predicted = factorization\_attack(e\_a, e\_b, d\_b, n)

print(f'\

Predicted:\n\

p by factorisation {p\_predicted}\n\

q by factorisation {q\_predicted}\n\

d\_a by factorisation {d\_a\_predicted}')

if d\_a\_predicted == d\_a:

print(f'Success')

else:

print(f'Failure')

#--------WIENER BLOCK--------#

def get\_fraction(e, n):

a, q = divmod(e, n)

t = n

x = [a]

while q != 0:

next\_t = q

a, q = divmod(t, q)

t = next\_t

x.append(a)

return x

def wiener\_attack(e, n):

a = get\_fraction(e, n)

l = len(a)

q = [0, 1, 0]

m = rnd.randint(1, n-1)

d = -1

for i in range (1, l):

q[2] = a[i] \* q[1] + q[0]

if pow(m, e\*q[2], n) == m:

d = q[2]

q[0], q[1] = q[1], q[2]

return d

def second\_program():

n = 159120802052440427821561598797245794196486762007282213614899538625298940765077913123669326518443057755809732511261424922103777938173006527628957265784605473595141601914607205620259694486382859683903964193688529214708416973894744745552398481940927574916829347154100077369737008945802414290266124801056303429283

e = 11344248856885807164295503665454231487754375121089537309789856526927258776275114297414196034298992736180410481299229993278201451840436370855751788204922795778488609359900203381710724197371026038017982542547796382021400514829497189707464973191055639707115355334522474997824622403851742433216683635618563112299

d = 23404895650626450554473888487065756508057288839850568254229721811842778880539

print(f'\

n = {n}\n\

e = {e}\n\

d = {d}')

d\_predicted = wiener\_attack(e, n)

print(f'Predicted d by wiener {d\_predicted}')

if d == d\_predicted:

print(f'Success')

else:

print(f'Failure')

#--------KEYLESS DECRYPTION BLOCK--------#

def keyless\_decryption\_attack(c, e, n):

c\_vector = [c, 0]

for i in range(n):

c\_vector[1] = pow(c\_vector[0], e, n)

print(c\_vector[0])

if c\_vector[1] % n == c:

m = c\_vector[0]

break

c\_vector[0] = c\_vector[1]

return m

def third\_program():

m = 156

p = getPrime(32, randfunc=get\_random\_bytes)

q = getPrime(32, randfunc=get\_random\_bytes)

e = 0

#init base parameters

n = p \* q

v = (p-1)\*(q-1)

while GCD(e, v) > 1:

e = getPrime(32, randfunc=get\_random\_bytes)

d = inverse(e, v)

c = pow(m, e, n)

print(f'\

n = {n}\n\

e = {e}\n\

d = {d}\n\

m = {m}\n\

c = {c}')

start = time.time()

m\_predicted = keyless\_decryption\_attack(c, e, n)

end = time.time() - start

print(f'Predicted d by wiener {m\_predicted}')

print(f'spended time = {end} seconds')

if m == m\_predicted:

print(f'Success')

else:

print(f'Failure')

#--------PARAMETERS GENERATOR-------#

def gen\_security\_parameters():

# p и q - безопасные простые числа, z = 2\*z1 + 1

while True:

p = getPrime(512, randfunc=get\_random\_bytes)

p\_1 = (p - 1) // 2

if(isprime(p\_1) == True):

break

while True:

q = getPrime(512 + 12, randfunc=get\_random\_bytes) # разница в 12 байт между p и q

q\_1 = (q - 1) // 2

if(isprime(q\_1) == True):

break

n = p \* q

v = (p - 1) \* (q - 1)

while True:

e = 2

while (GCD(e, v) > 1):

e = getPrime(16, randfunc=get\_random\_bytes) # 16 байт, не слишком большое и не слишком маленькое

d = inverse(e, v)

if(d >= sqrt(sqrt(n))):

break

return n, e, d, p, q

def main():

while True:

print('''

Warning: You will not be able to decrypt the file or verify

the signature if the action was not performed at the current start of the program.

Available fighters:

1. Harry Potter (factorization)

2. Tom Sawyer (wiener)

3. Optimus Prime (keyless decryption)

4. John Snow (generate parameters)

''' )

command = input("Choose your fighter: ")

if command == '1' or command == 'Harry Potter':

first\_program()

elif command == '2' or command == 'Tom Sawyer':

second\_program()

elif command == '3' or command == 'Optimus Prime':

third\_program()

elif command == '4' or command == 'John Snow':

n, e, d, p, q = gen\_security\_parameters()

print(f'Generated parameters:\n\

{"\*"\*50}\nn = {n}\n\

{"\*"\*50}\ne = {e}\n\

{"\*"\*50}\nd = {d}\n\

{"\*"\*50}\np = {p}\n\

{"\*"\*50}\nq = {q}')

else:

print("incorrect input, try again")

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

main()