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

**ЛАБОРАТОРНАЯ РАБОТА № 3**

1. **«Атака на криптосистему RSA при шифровании коротких сообщений»**
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 атак на криптосистему RSA, связанных с использованием малого открытого показателя при шифровании коротких сообщений, реализация схемы дополнения RSA-OAEP

# Задача

* Разработать программу **П-1**, которая реализует атаку на криптосистему RSA для малого открытого показателя и короткого сообщения. Программа **П-1** должна принимать на вход составное число , шифртексты , , открытый показатель ; на выходе – возвращать восстановленное сообщение.
* Разработать программу **П-2**, которая реализует зашифрование и расшифрование файла согласно схеме преобразования RSA-OAEP. Зашифрование файла необходимо производить с использованием вспомогательного симметричного алгоритма AES-256 в режиме CBC, реализация симметричного алгоритма не требуется. Ключ шифрования симметричного алгоритма – случайный или изначально задан в программе. В режиме зашифрования программа должна принимать на вход файл сообщения (произвольного формата), открытый ключ RSA, ключ симметричного алгоритма; на выходе − возвращать зашифрованный файл, заголовок которого соответствует нотации ASN.1, описанной в *Приложении А*. В режиме расшифрования **П-2** должна принимать на вход зашифрованный файл, закрытый ключ RSA; на выходе – возвращать расшифрованное сообщение.

# Ход работы

## Атака на криптосистему

Была написана программа, реализующая атаку при следующих условиях: нарушитель помимо открытого ключа перехватил два зашифрованных сообщения, таких, что, в открытом виде сообщения связаны Афинным преобразованием: (считается что параметры a, b также известны нарушителю). Программа работает следюущим образом: с помощью библиотеки rsa генерируется пара ключей (открытый, закрытый), случайным образом генерируется сообщение m1 и считается сообщение m2 и высчитываются шифртексты c1, c2. Далее запускается атака, которая возврашает m1, m2. В случае, если эти параметры совпадают с исходными, выводится сообщение об успешности атаки, в противном случае, о безуспешности. Продемонстрируем работу программы для следующих параметров:

p = 199

q = 167

n = 33233

e = 5

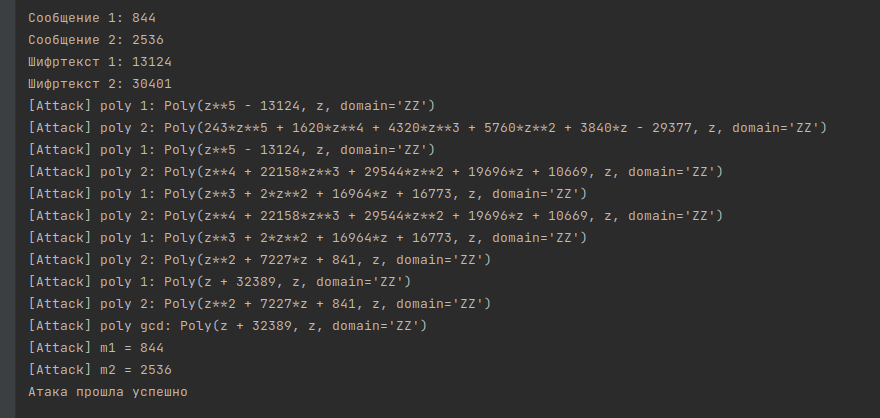
d = 19721

a = 3

b = 4

m1 = 844

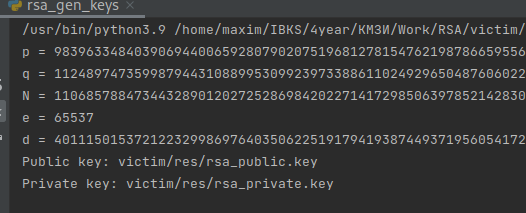
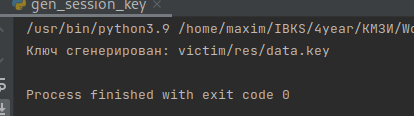
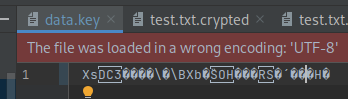
m2 = a\*m1+b = 2536

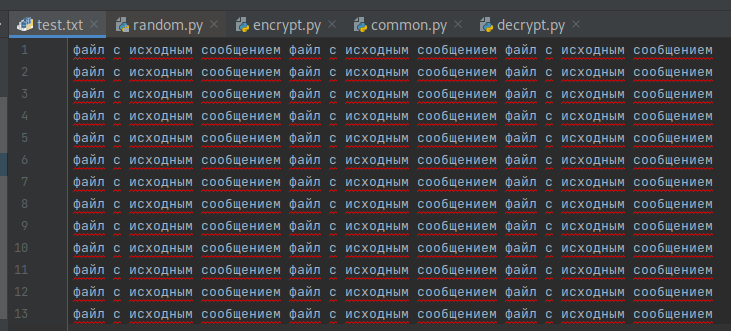
Рисунок 1 — реализация атаки

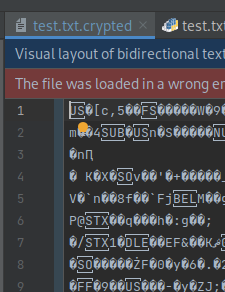
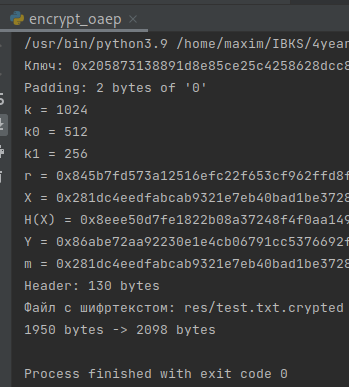
На рисунке 1 показаны отладочные сообщения злоумышленника — он составил полиномы и нашел их НОД с помощью алгоритма Евклида. Стоит отметить что при высоких значениях параметра e данная задача будет трудоемкой. В нашем примере e=5 и нарушитель очень быстро посчитал значения m1, m2 — атака прошла успешно.

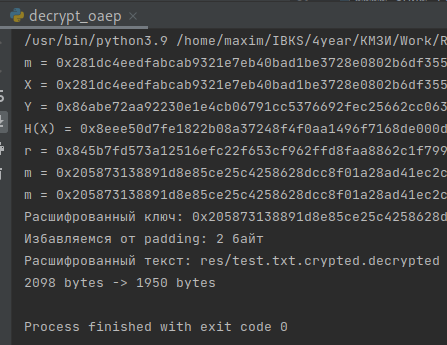
## Реализация криптосистемы RSA-OAEP

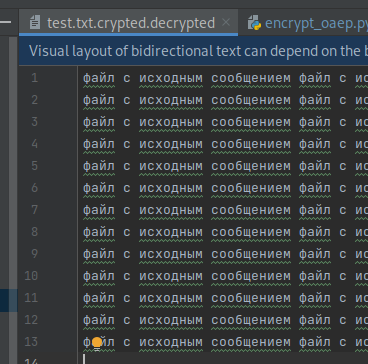
Во второй части лабораторной работы была реализована криптосистема RSA-OAEP — были написаны программы для шифрования/дешифрования сообщений. Сеансовый ключ и ключи RSA генерируются и записываются в файл, сообщение шифруется с помощью алгоритма AES в режиме CBC с длиной ключа 64 бита. В качесвте синхропосылки используется 0. Ключ алгоритма AES дополнительно шифруется алгоритмом ESA-OAEP и записывается в зашифрованном виде в заголовок файла, согласно нотации ASN-1. Продемонстрируем работу программы:

Рисунки 2, 3, 4 — генерация ключа AES и ключа RSA

Рисунок 5 — исходное сообщение

Рисунки 6, 7 — шифрование сообщения



Рисунки 8, 9 — расшифрование сообщения

# Контрольные вопросы

1. Обеспечивают ли часто используемые открытые показатели вида стойкость к методу криптоанализа на основе частично известных открытых текстов?
2. При какой длине дополнения по двум шифртекстам можно восстановить исходное сообщение при и длине модуля 1024 бит?
3. Перечислите преимущества RSA-OAEP по сравнению с классической схемой RSA.

RSA-OAEP защищает от атак по выбранному шифртексту

1. Какие существуют модификации RSA на основе схемы OAEP?

# Вывод

В результате выполнения лабораторной работы была изучена и реализована новая атака на криптосистему RSA. Также была разработана криптосистема RSA-OAEP.

# Приложение

import random

import tkinter

from common import \*

def crypt\_key(key, e, n):

k0, k1 = 512, 256

k = RSA\_KEY\_LEN

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

print('k0 = {}'.format(k0))

print('k1 = {}'.format(k1))

key <<= k1

r = random.randbytes(k0 // 8)

G = SHA512.new()

H = SHA512.new()

G.update(r)

r = G.digest()

r = int.from\_bytes(r, byteorder='big')

print('r = {}'.format(hex(r)))

X = r ^ key

print('X = {}'.format(hex(X)))

H.update(X.to\_bytes(length=(k-k0)//8, byteorder='big'))

H\_X = H.digest()

H\_X = int.from\_bytes(H\_X, byteorder='big')

print('H(X) = {}'.format(hex(H\_X)))

Y = r ^ H\_X

print('Y = {}'.format(hex(Y)))

X\_size = (k - k0)

m1 = (X << X\_size) | Y

print('m = {}'.format(hex(m1)))

c1 = pow(m1, e, n)

return c1

def main(context):

key = from\_file(context.data\_key, 32)

print('Ключ: {}'.format(hex(int.from\_bytes(key, byteorder='big'))))

open\_text = from\_file(context.path\_in, os.path.getsize(context.path\_in))

open\_text\_len = len(open\_text)

open\_text = padding\_make(open\_text)

cipher\_instance = AES.new(key, AES.MODE\_CBC, iv=b'0000000000000000')

cipher\_text = cipher\_instance.encrypt(open\_text)

key = int.from\_bytes(key, byteorder='big')

e, n = parse\_rsa\_key(context.rsa\_key, is\_public=True)

cipher\_key = crypt\_key(key, e, n)

cipher\_key = cipher\_key.to\_bytes(RSA\_KEY\_LEN//8, byteorder='big')

l, type = RSA\_KEY\_LEN // 8, 31

header = type.to\_bytes(1, 'big') + l.to\_bytes(1, 'big') + cipher\_key

print('Header: {} bytes'.format(len(header)))

path\_out = context.path\_in + '.crypted'

f = open(path\_out, 'wb')

f.write(header)

f.write(cipher\_text)

f.close()

print('Файл с шифртекстом: {}'.format(path\_out))

print('{} bytes -> {} bytes'.format(open\_text\_len, len(cipher\_text) + len(header)))

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

parser = argparse.ArgumentParser()

parser.add\_argument('--path-in', required=True)

parser.add\_argument('--rsa-key', required=True)

parser.add\_argument('--data-key', required=True)

args = parser.parse\_args()

main(args)

from common import \*

def decrypt\_key(cipher\_key, d, n):

k0, k1 = 512, 256

k = RSA\_KEY\_LEN

m1 = pow(cipher\_key, d, n)

print('m = {}'.format(hex(m1)))

X\_size = (k - k0)

H = SHA512.new()

G = SHA512.new()

X = m1 >> X\_size

Y = m1 & ((1<<X\_size)-1)

H.update(X.to\_bytes(byteorder='big', length=X\_size//8))

H\_X = int.from\_bytes(H.digest(), byteorder='big')

print('X = {}'.format(hex(X)))

print('Y = {}'.format(hex(Y)))

print('H(X) = {}'.format(hex(H\_X)))

r = Y ^ H\_X

print('r = {}'.format(hex(r)))

G.update(r.to\_bytes(byteorder='big', length=k0//8))

m = X ^ r

print('m = {}'.format(hex(m)))

mask = (1 << k1) - 1

if m & mask:

raise Exception('Incorrect ciphertext')

m >>= k1

print('m = {}'.format(hex(m)))

return m

def main(context):

data = from\_file(context.path\_in, os.path.getsize(context.path\_in))

cipher\_key, cipher\_text = parse\_header(data)

cipher\_key = int.from\_bytes(cipher\_key, byteorder='big')

d, n = parse\_rsa\_key(context.rsa\_key, is\_public=False)

key = decrypt\_key(cipher\_key, d, n)

print('Расшифрованный ключ: {}'.format(hex(key)))

key = key.to\_bytes(32, byteorder='big')

cipher\_instance = AES.new(key, AES.MODE\_CBC, iv=b'0000000000000000')

decrypted\_text = cipher\_instance.decrypt(cipher\_text)

decrypted\_text = padding\_remove(decrypted\_text)

path\_out = context.path\_in + '.decrypted'

f = open(path\_out, 'wb')

f.write(decrypted\_text)

f.close()

print('Расшифрованный текст: {}'.format(path\_out))

print('{} bytes -> {} bytes'.format(len(data), len(decrypted\_text)))

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

parser = argparse.ArgumentParser()

parser.add\_argument('--path-in', required=True)

parser.add\_argument('--rsa-key', required=True)

args = parser.parse\_args()

main(args)

class AttackTwoAffineMessages:

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

self.\_\_a = a

self.\_\_b = b

self.\_\_logs = logs

@staticmethod

def \_\_poly\_normalize(poly, n, z):

new\_coeffs = []

hight\_coeff = poly.coeffs()[0]

for c in poly.coeffs():

x = 0

while c % n != (x \* hight\_coeff) % n:

x += 1

# while c % hight\_coeff != 0:

# c += n

new\_coeffs.append(x)

return Poly(new\_coeffs, z)

def \_\_poly\_gcd(self, poly\_1, poly\_2, public\_key, z):

while poly\_1.degree() != 1 and poly\_2.degree() != 1:

if poly\_2.degree() >= poly\_1.degree():

poly\_2 %= poly\_1

poly\_2 = self.\_\_poly\_normalize(poly\_2, public\_key.n, z)

else:

poly\_1 %= poly\_2

poly\_1 = self.\_\_poly\_normalize(poly\_1, public\_key.n, z)

if self.\_\_logs:

print('[Attack] poly 1: {}'.format(poly\_1))

print('[Attack] poly 2: {}'.format(poly\_2))

if poly\_1.degree() == 1:

return poly\_1

else:

return poly\_2

def \_\_call\_\_(self, public\_key, cipher\_text\_1, cipher\_text\_2):

"""

:return: open\_text\_1, open\_text\_2

"""

z = symbols('z', real=True)

poly\_1 = Poly(z \*\* public\_key.e - cipher\_text\_1)

poly\_2 = Poly((self.\_\_a \* z + self.\_\_b) \*\* public\_key.e - cipher\_text\_2)

if self.\_\_logs:

print('[Attack] poly 1: {}'.format(poly\_1))

print('[Attack] poly 2: {}'.format(poly\_2))

poly\_gcd = self.\_\_poly\_gcd(poly\_1, poly\_2, public\_key, z)

if self.\_\_logs:

print('[Attack] poly gcd: {}'.format(poly\_gcd))

m1 = public\_key.n - poly\_gcd.coeffs()[1]

m2 = (m1 \* self.\_\_a + self.\_\_b) % public\_key.n

if self.\_\_logs:

print('[Attack] m1 = {}'.format(m1))

print('[Attack] m2 = {}'.format(m2))

return m1, m2

def try\_attack\_4():

a, b = 3, 4

public\_key, private\_key = rsa.newkeys(16, exponent=5)

# public\_key = rsa.PublicKey(81089, 17)

# private\_key = rsa.PrivateKey(81089, 17, 42533, 131, 619)

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))

m1 = random.randint(0, public\_key.n)

m2 = (a \* m1 + b) % public\_key.n

# m1, m2 = 12345, 12346

print('Сообщение 1: {}'.format(m1))

print('Сообщение 2: {}'.format(m2))

c1 = pow(m1, public\_key.e, public\_key.n)

c2 = pow(m2, public\_key.e, public\_key.n)

print('Шифртекст 1: {}'.format(c1))

print('Шифртекст 2: {}'.format(c2))

attack = AttackTwoAffineMessages(a, b)

m1\_attacked, m2\_attacked = attack(public\_key, c1, c2)

if m1\_attacked == m1 and m2\_attacked == m2:

print('Атака прошла успешно')

else:

print('Атака прошла безуспешно {}, {} -> {}, {}'.format(m1, m2, m1\_attacked, m2\_attacked))