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

**ЛАБОРАТОРНАЯ РАБОТА № 1**

1. **«Основы криптосистемы RSA»**
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 криптосистемы RSA, реализация алгоритмов зашифрования и расшифрования сообщений, формирования и проверки электронной цифровой подписи.

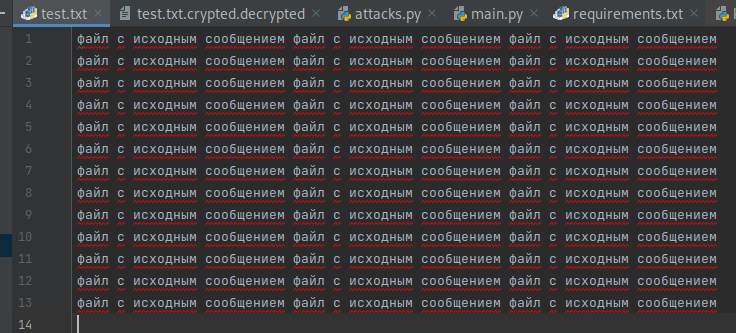
# Задача

Получить у преподавателя вариант задания и разработать программу **П-1**, которая:

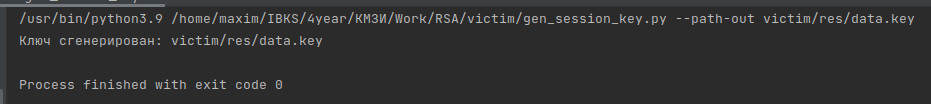
* выполняет зашифрование и расшифрование файла с использованием алгоритма RSA. В режиме зашифрования программа должна принимать на вход файл сообщения (произвольного формата), открытый ключ RSA, ключ симметричного алгоритма; на выходе − возвращать зашифрованный файл, заголовок которого соответствует нотации ASN.1, описанной в *Приложении А*. 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 задан в программе. Случайный ключ шифрования алгоритма AES (32 байта) необходимо представить как число для шифрования алгоритмом RSA, порядок байтов – MSB. Старшие неиспользуемые цифры (байты) числа следует считать нулевыми. В режиме расшифрования программа **П-1** должна принимать на вход зашифрованный файл, закрытый ключ RSA; на выходе – возвращать расшифрованное сообщение;
* выполняет формирование и проверку электронной подписи с использованием алгоритма RSA. При реализации алгоритма электронной подписи рекомендуется использовать хэш-алгоритм SHA-256. Поскольку хэш-образ, вычисленный по алгоритму SHA-256, может быть длиннее модуля RSA, при проверке подписи следует сравнивать значение с подписью сообщения . В режиме формирования подписи программа **П-1** должна принимать на вход файл, который необходимо подписать, ключ подписи RSA; на выходе – возвращать отдельный файл подписи. В режиме проверки подписи программа должна принимать на вход сообщение и файл подписи; на выходе – возвращать результат «Подпись принимается» или «Подпись неверна».

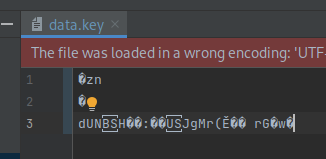
# Ход работы

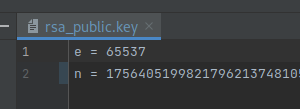
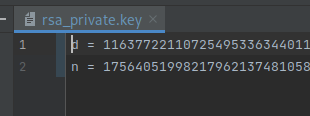
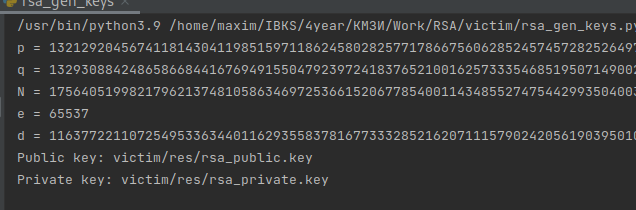
В результате выполнения работы была разработана программа, способная шифровать, расшифровывать, подписывать файлы и осуществлять проверку подписи. Также имеются возможности генерировать сеансовые ключи и ключи для шифрования ключей — ключи RSA. Для разработки программ использовался язык python и библиотека pycryptodome. Продемонстрируем программу. Сформируем тестовый файл:

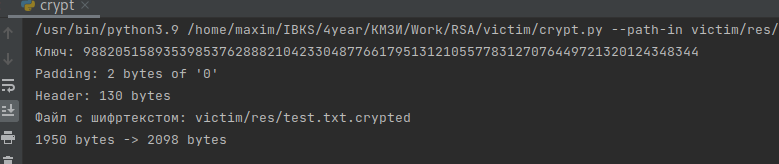
Рисунок 1 — тестовый файл

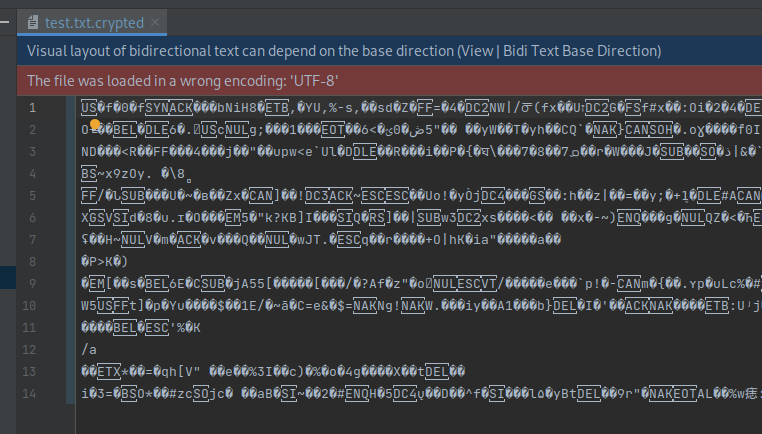
Далее продемонстрируем процесс шифрования-дешифрования. При шифровании использовался алгоритм AES-256 с синхропосылкой «0000000000000000». Ключ данного алгоритма шифруется с помощью RSA и «добавляется» к зашифрованному тексту согласно нотации ASN-1. В качестве дополнения блоков использовался символ «0». Размер RSA-ключей — 1024 бит.



Рисунок 2, 3 — генерируем сеансовый ключ

Рисунок 4, 5, 6 — генерируем ключи RSA



Рисунок 7, 8 — шифруем файл

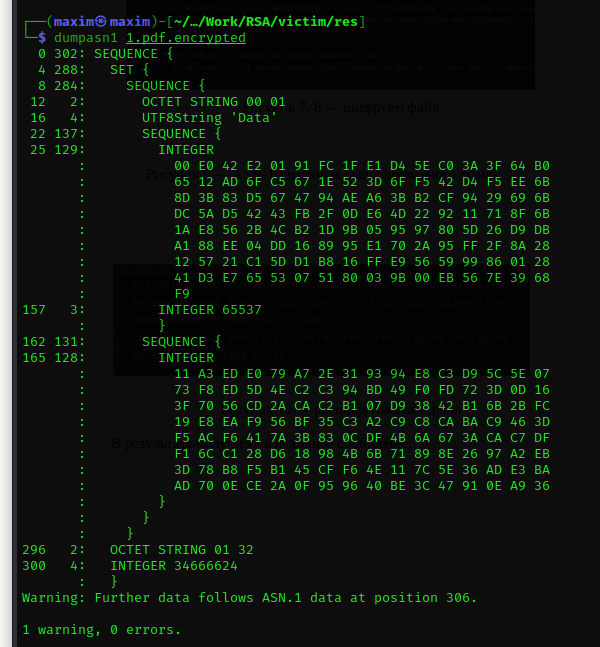
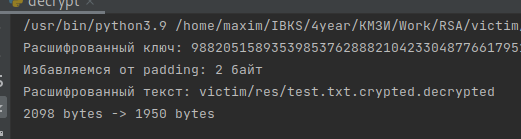
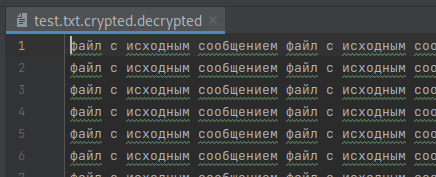


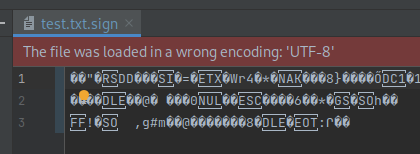
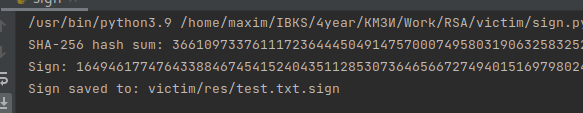
Рисунок 9 — ASN-1 заголовок зашифрованного файла

Рисунок 10 — расшифровываем файл

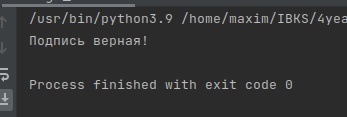
В результате получилось исходное сообщение:

Рисунок 11 -исходное сообщение

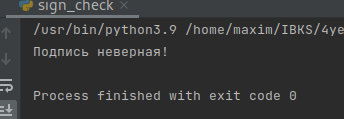
Теперь продемонстрируем подпись файла с помощью RSA:

Рисунки 12, 13 - Подписываем файл цифровой подписью

Проверим цифровую подпись:

Рисунок 14 — проверка цифровой подписи

В случае модификации файла проверка заканчивается неудачей:

Рисунок 15 — неверная цифровая подпись при модификации файла

# Контрольные вопросы

1. Какие задачи положены в основу безопасности системы RSA?

Зачада факторизации чисел

1. Показать, что схема RSA работает корректно для любого сообщения .

Докажем :

для некоторого

по малой Теореме Ферма

Аналогично

По Китайской теореме об остатках: ч.т.д.

1. Доказать, что задача разложения числа *n* на множители и задача вычисления функции Эйлера полиномиально эквивалентны.

Обе задачи имеют вычислительную сложность . Вычисление - поиск чисел из промежутка 1..., таких что их НОД с n будет 1. Факторизация n — поиск чисел из промежутка 1..., таких, что их НОД c n будет не равен 1.

1. Показать, что схема RSA обладает свойством гомоморфности относительно операции умножения.

Пусть ,

Тогда ч.т.д.

# Вывод

В результате выполнения работы была изучена криптосистема RSA и использована на практике: разработана программа реализующая шифрование/дешифрование файлов а также цифровую подпись и ее проверку.

# Приложение

from Crypto.Cipher import AES

from Crypto.Hash import SHA256

from Crypto.PublicKey import RSA

from sys import argv

import random

import argparse

import os

import math

RSA\_KEY\_LEN = 1024

def from\_file(path, n):

f = open(path, 'rb')

data = f.read(n)

f.close()

return data

def parse\_rsa\_key(path, is\_public):

f = open(path, 'r')

e\_d = f.readline()

n = f.readline()

n = int(n.replace('n = ', ''))

if is\_public:

return int(e\_d.replace('e = ', '')), n

else:

return int(e\_d.replace('d = ', '')), n

import os.path

from common import \*

def padding\_forward(data):

padding = AES.block\_size - (len(data) % AES.block\_size)

data = data + (b'0' \* padding) + padding.to\_bytes(length=AES.block\_size, byteorder='big')

print('Padding: {} bytes of \'0\''.format(padding))

return data

def main(context):

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

print('Ключ: {}'.format(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\_forward(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 = pow(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 parse\_header(data):

type, l = data[0], data[1]

assert type == 31

return data[2: 2 + l], data[2 + l:]

def padding\_backward(data):

padding = int.from\_bytes(data[-AES.block\_size:], byteorder='big')

data = data[:-AES.block\_size - padding]

print('Избавляемся от padding: {} байт'.format(padding))

return data

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 = pow(cipher\_key, d, n)

print('Расшифрованный ключ: {}'.format(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\_backward(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)

import random

import argparse

def main(context):

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

f.write(random.randbytes(32))

f.close()

print('Ключ сгенерирован: {}'.format(context.path\_out))

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

parser = argparse.ArgumentParser()

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

args = parser.parse\_args()

main(args)

from common import \*

def main(context):

private\_key = RSA.generate(RSA\_KEY\_LEN)

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))

f = open(context.path\_out\_public, 'w')

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

f.write('\n')

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

f.close()

f = open(context.path\_out\_private, 'w')

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

f.write('\n')

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

f.close()

print('Public key: {}'.format(context.path\_out\_public))

print('Private key: {}'.format(context.path\_out\_private))

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

parser = argparse.ArgumentParser()

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

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

args = parser.parse\_args()

main(args)

from common import \*

def main(context):

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

hash\_instance = SHA256.new()

hash\_instance.update(text)

sha256\_hash = hash\_instance.digest()

sha256\_hash = int.from\_bytes(sha256\_hash, byteorder='big')

print('SHA-256 hash sum: {}'.format(sha256\_hash))

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

s = pow(sha256\_hash, d, n)

print('Sign: {}'.format(s))

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

path\_sign = context.path\_in + '.sign'

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

f.write(s)

f.close()

print('Sign saved to: {}'.format(path\_sign))

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)

import os.path

from common import \*

def main(context):

s = from\_file(context.path\_sign, RSA\_KEY\_LEN//8)

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

# print('Sign: {}'.format(s))

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

hash\_instance = SHA256.new()

hash\_instance.update(text)

sha256\_hash = hash\_instance.digest()

sha256\_hash = int.from\_bytes(sha256\_hash, byteorder='big')

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

if pow(s, e, n) == sha256\_hash:

print('Подпись верная!')

else:

print('Подпись неверная!')

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

parser = argparse.ArgumentParser()

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

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

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

args = parser.parse\_args()

main(args)