

Министерство науки и высшего образования Российской Федерации
Санкт-Петербургский Политехнический Университет Петра Великого

Институт Кибербезопасности и Защиты Информации

ЛАБОРАТОРНАЯ РАБОТА № 1

«Основы криптосистемы RSA»

по дисциплине «Криптографические методы защиты информации»

Выполнил
студент гр. 4851003/80802

<подпись>

Сошнев М.Д.

Проверил
преподаватель

<подпись>

Ярмак А.В.

Санкт-Петербург

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 $h(m)(mod n)$ с подписью s сообщения m . 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 ruscryptodome. Продemonстрируем программу. Сформируем тестовый файл:

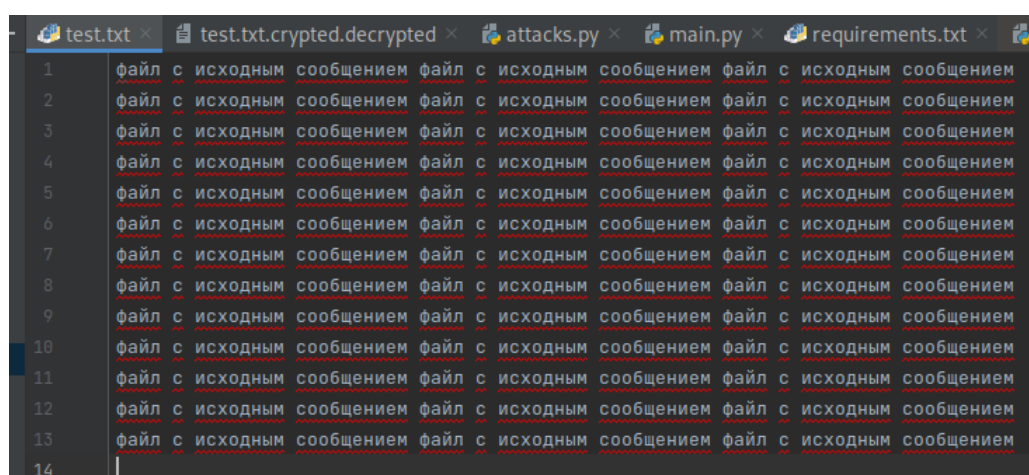


Рисунок 1 — тестовый файл

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.

```
/usr/bin/python3.9 /home/maxim/IBKS/4year/КМЗИ/Work/RSA/victim/gen_session_key.py --path-out victim/res/data.key
Ключ сгенерирован: victim/res/data.key

Process finished with exit code 0
```

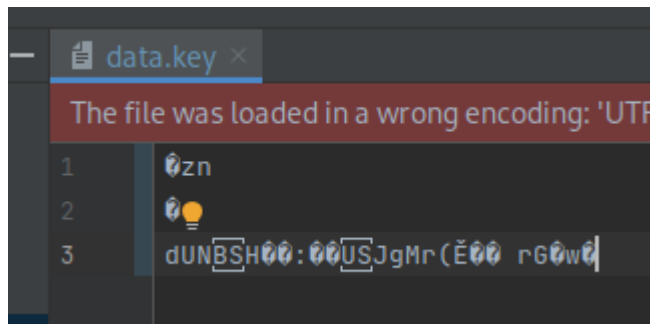


Рисунок 2, 3 — генерируем сеансовый ключ

```
rsa_gen_keys <
/usr/bin/python3.9 /home/maxim/IBKS/4year/КМ3И/Work/RSA/victim/rsa_gen_keys.py
p = 13212920456741181430411985159711862458028257717866756062852457457282526497
q = 13293088424865866844167694915504792397241837652100162573335468519507149003
N = 17564051998217962137481058634697253661520677854001143485527475442993504003
e = 65537
d = 11637722110725495336344011629355837816773332852162071115790242056190395010
Public key: victim/res/rsa_public.key
Private key: victim/res/rsa_private.key
```

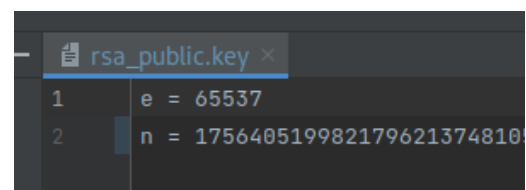
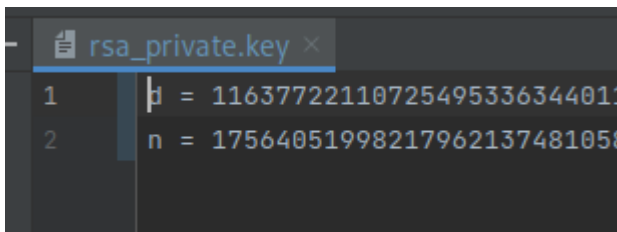


Рисунок 4, 5, 6 — генерируем ключи RSA

```
crypt <
/usr/bin/python3.9 /home/maxim/IBKS/4year/КМ3И/Work/RSA/victim/crypt.py --path-in victim/res/
Ключ: 98820515893539853762888210423304877661795131210557783127076449721320124348344
Padding: 2 bytes of '0'
Header: 130 bytes
Файл с шифртекстом: victim/res/test.txt.crypted
1950 bytes -> 2098 bytes
```

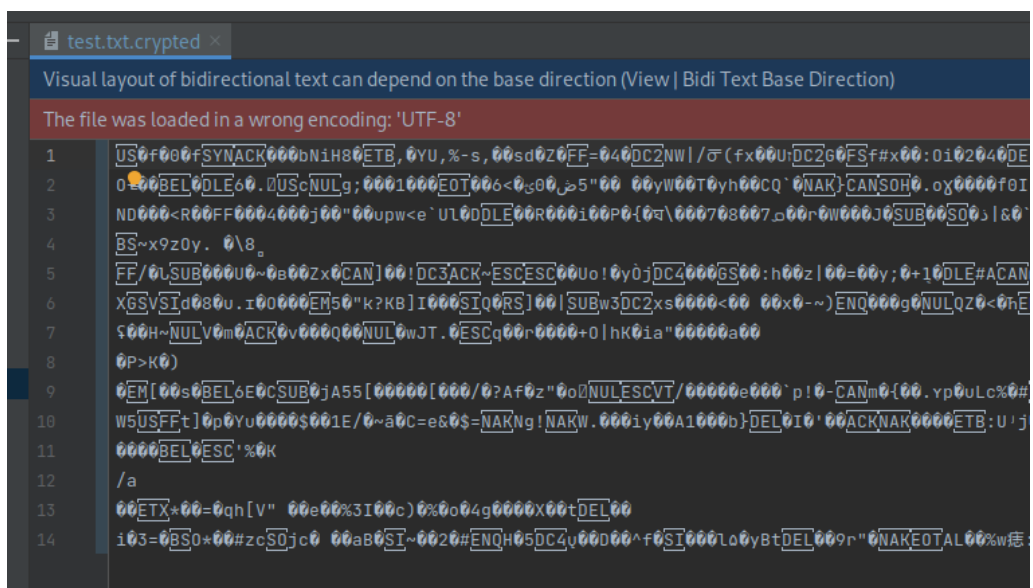


Рисунок 7, 8 — шифруем файл

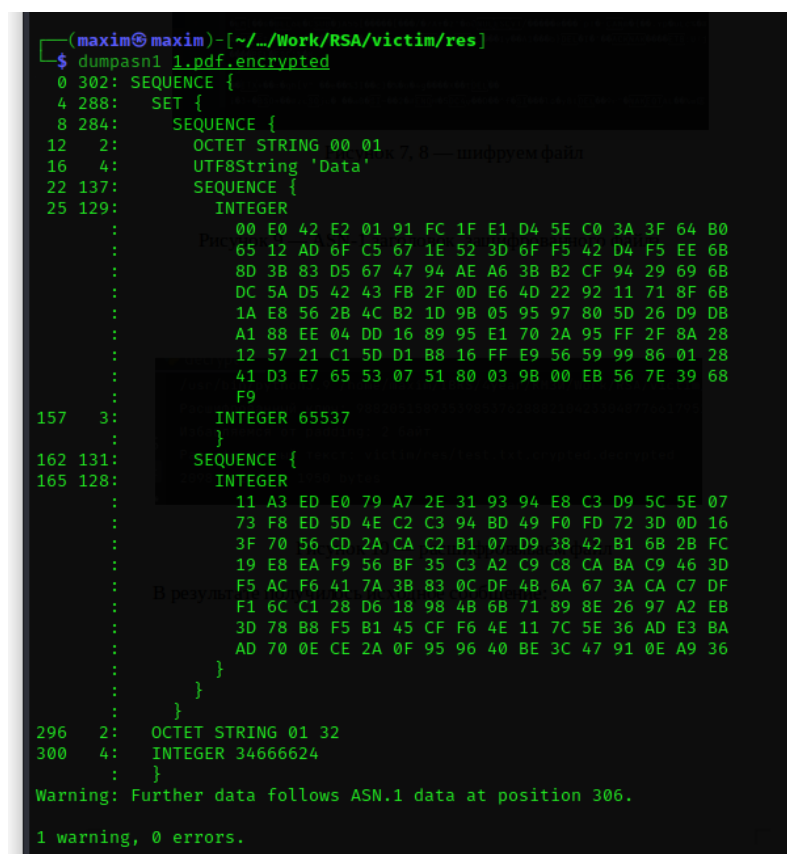


Рисунок 9 — ASN-1 заголовок зашифрованного файла

```
decrypt x
/usr/bin/python3.9 /home/maxim/IBKS/4year/KM3I/Work/RSA/victim.py
Расшифрованный ключ: 98820515893539853762888210423304877661795
Избавляемся от padding: 2 байт
Расшифрованный текст: victim/res/test.txt.crypted.decrypted
2098 bytes -> 1950 bytes
```

Рисунок 10 — расшифровываем файл

В результате получилось исходное сообщение:

```
test.txt.crypted.decrypted x
1 файл с исходным сообщением файл с исходным соо
2 файл с исходным сообщением файл с исходным соо
3 файл с исходным сообщением файл с исходным соо
4 файл с исходным сообщением файл с исходным соо
5 файл с исходным сообщением файл с исходным соо
6 файл с исходным сообщением файл с исходным соо
7 файл с исходным сообщением файл с исходным соо
```

Рисунок 11 -исходное сообщение

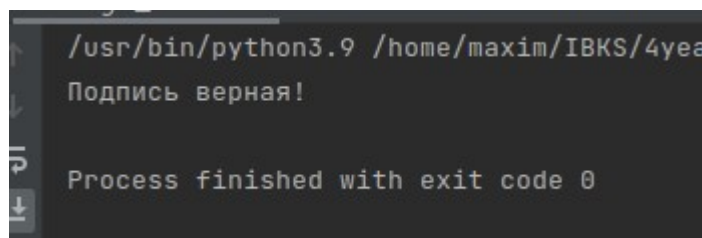
Теперь продемонстрируем подпись файла с помощью RSA:

```
sign
/usr/bin/python3.9 /home/maxim/IBKS/4year/KM3I/Work/RSA/victim/sign.py
SHA-256 hash sum: 366109733761117236444504914757000749580319063258325
Sign: 1649461774764338846745415240435112853073646566727494015169798024
Sign saved to: victim/res/test.txt.sign
```

```
test.txt.sign x
The file was loaded in a wrong encoding: 'UTF-8'
1 00"0RSDD000SI0=0ETX0Wr40*0NAK0008}00000DC101
2 0000DLE00@0 0000NUL00ESC0000600*0GS0S0h00
3 FF!0S0 ,g#m00@000000080DLE0EOT:r00
```

Рисунки 12, 13 - Подписываем файл цифровой подписью

Проверим цифровую подпись:

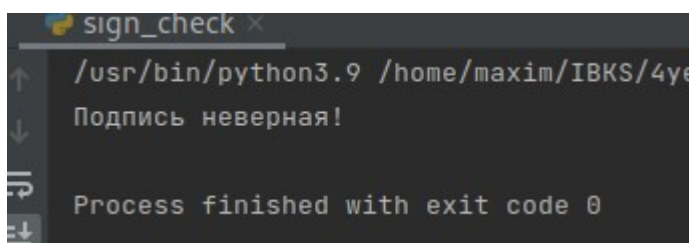


```
/usr/bin/python3.9 /home/maxim/IBKS/4yea
Подпись верная!

Process finished with exit code 0
```

Рисунок 14 — проверка цифровой подписи

В случае модификации файла проверка заканчивается неудачей:



```
sign_check x
/usr/bin/python3.9 /home/maxim/IBKS/4yea
Подпись неверная!

Process finished with exit code 0
```

Рисунок 15 — неверная цифровая подпись при модификации файла

Контрольные вопросы

1. Какие задачи положены в основу безопасности системы RSA?

Задача факторизации чисел

2. Показать, что схема RSA работает корректно для любого сообщения $m \in \mathbb{Z}/n\mathbb{Z}$.

Докажем $m^{ed} = m \pmod{p}$:

$$ed = 1 + k * (p-1) * (q-1) \text{ для некоторого } k \in \mathbb{Z}$$

$$m^{ed} = m * m^{k * (p-1) * (q-1)} = m * (m^{p-1})^{k * (q-1)} = m \pmod{p} \text{ по малой Теореме Ферма}$$

$$\text{Аналогично } m^{ed} = m \pmod{q}$$

По Китайской теореме об остатках: $m^{ed} = m \pmod{p * q}$ ч.т.д.

3. Доказать, что задача разложения числа n на множители и задача вычисления функции Эйлера $\varphi(n)$ полиномиально эквивалентны.

Обе задачи имеют вычислительную сложность $O(\sqrt{n})$. Вычисление $\varphi(n)$ - поиск чисел из промежутка $1 \dots \sqrt{n}$, таких что их НОД с n будет 1. Факторизация n — поиск чисел из промежутка $1 \dots \sqrt{n}$, таких, что их НОД с n будет не равен 1.

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

$$\text{Пусть } E(m_1) = c_1, E(m_2) = c_2$$

$$\text{Тогда } E(m_1 * m_2) = (m_1 * m_2)^e \pmod{n} = (m_1)^e \pmod{n} * (m_2)^e \pmod{n} = c_1 * c_2 \text{ ч.т.д.}$$

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 + '.encrypted'
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)

```