Министерство науки и высшего образования Российской Федерации

Санкт-Петербургский Политехнический Университет Петра Великого

—

Институт компьютерных наук и кибербезопасности

Высшая школа кибербезопасности

**ЛАБОРАТОРНАЯ РАБОТА № 1**

**«Основы криптосистемы RSA»**

по дисциплине «Криптографические методы защиты информации»

Выполнил

студент гр. 5151004/90101 Кондачков Е.Д.

Преподаватель

Ассистент Ярмак А.В.

Санкт-Петербург

2023

Оглавление

[1. Цель работы 2](#_Toc145171916)

[2. Задачи работы 3](#_Toc145171917)

[3. Ход работы 4](#_Toc145171918)

[1.1 Теоретическая часть 4](#_Toc145171919)

[1.2 Генерация параметров криптосистемы 5](#_Toc145171920)

[1.3 Реализация алгоритмов шифрования 6](#_Toc145171921)

[1.4 Реализация алгоритмов формирования и проверки электронной цифровой подписи 7](#_Toc145171922)

[4. Ответы на контрольные вопросы 9](#_Toc145171923)

[5. Выводы и заключение 11](#_Toc145171924)

[Приложение А 12](#_Toc145171925)

# Цель работы

Изучение криптосистемы RSA, реализация алгоритмов зашифрования и расшифрования сообщений, формирования и проверки электронной цифровой подписи.

# Задачи работы

1. Изучить описание криптосистемы RSA приведенное в методическом пособии к лабораторной работе;
2. Реализовать алгоритмы зашифрования и расшифрования;
3. Проверить работоспособность разработанного приложения и привести в отчете скриншоты зашифрованного сообщения и используемых параметров RSA;
4. Реализовать алгоритмы формирования и проверки цифровой подписи сообщения;
5. Провести тестирование и приложить к отчету скриншоты файла подписи.

# Ход работы

## Теоретическая часть

В соответствии с поставленными в предыдущем пункте задачами первым делом были изучены методические указания к лабораторной работе, в которых содержалась информация о построении криптосистемы RSA.

Параметрами системы является 5 основных значений, а также два ключа: открытый и закрытый. Таким образом, к параметрам относятся:

* простые числа и порядка 2048-4096 бит, которые в будущей реализации выбираются случайно;
* число ;
* число e, называемое открытым показателем, которое ищется как число взаимно простое с ϕ;
* число d, называемое закрытым показателем, которое вычисляется из сравнения ;
* открытый ключ (public key, ключ зашифрования), получаемый отправителем и состоящий из пары ;
* закрытый ключ (private key, ключ расшифрования), не разглашаемый получателем и состоящий из пары .

Эти параметры ложатся в основу криптосистемы RSA в независимости от того, зашифровываем мы или формируем подпись.

Алгоритм зашифрования и расшифрования RSA:

1. Отправитель зашифровывает открытый текст m:
2. Получатель расшифровывает зашифрованный текст c:

Данное шифрование является визуально простым и понятным, но этот подход занимает довольно много времени для больших текстов. По этой причине в лабораторной работе будет использоваться шифр AES, а его ключ, передаваемый вместе с сообщением будет зашифрован при помощи RSA.

Помимо шифрования в данной работе будет также использоваться электронная подпись основанная на RSA. Её алгоритм таков:

1. Отправитель подписи генерирует хэш-значение текста (в этой работе будет использоваться алгоритм SHA256);
2. Генерируется число , соответствующее значению хеша из предыдущего пункта;
3. Генерируется значение : ;
4. Полученное значение отправляется вместе с текстом ;
5. На стороне получателя подписи получается значение от полученного текста;
6. Производится проверка .

## Генерация параметров криптосистемы

Как уже было сказано ранее, первичные параметры p и q генерируются случайным образом при помощи функции . Позднее из этих значений получается и ϕ. В дальнейшем при помощи функций и получаются и Полный код генерации базовый параметров представлен на рисунке Рисунок 1. Пример же сгенерированных параметров представлен на рисунке Рисунок 2

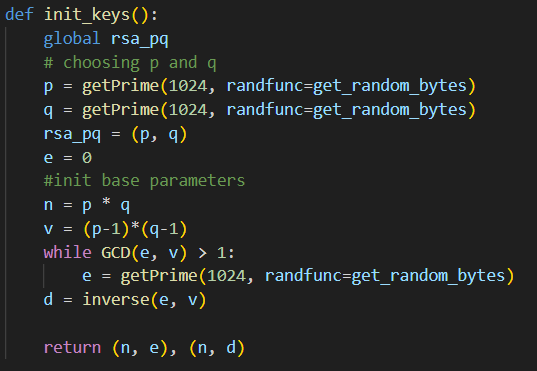


Рисунок 1 – Функция генерации параметров криптосистемы

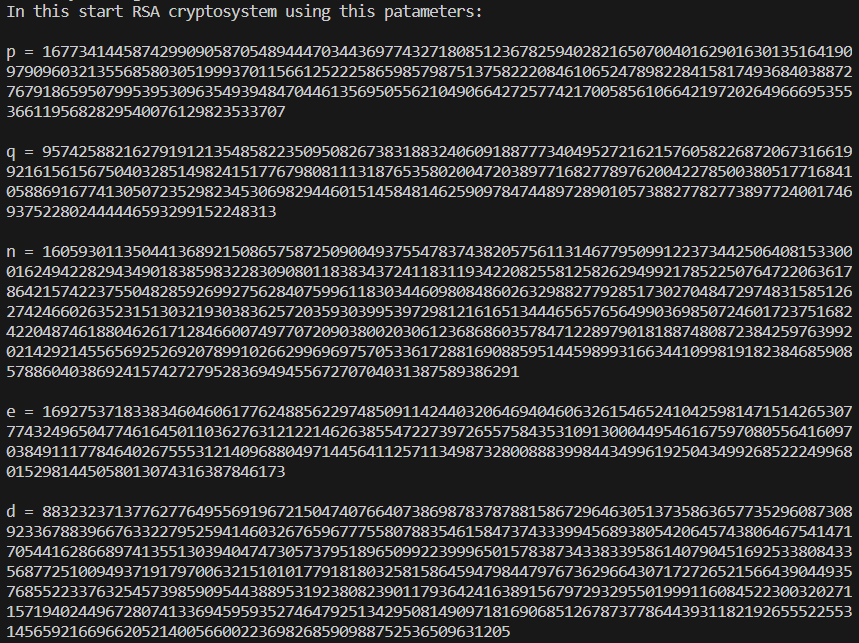


Рисунок 2 – Пример сгенерированных параметров системы

## Реализация алгоритмов шифрования

Алгоритм шифрования и расшифрования описаны в теоретической части. В этом пункте стоит показать реализацию данных функций и результат, даваемый функцией зашифрования (структуру ASN-файла, который должен передаваться получателю).

Как и было указано ранее, текст шифруется при помощи алгоритма AES, что продемонстрировано на рисунке Рисунок 3.

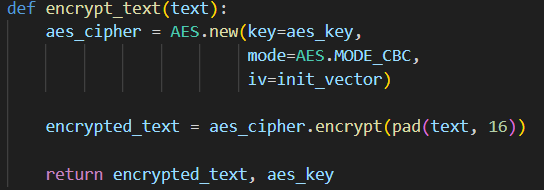


Рисунок 3 – Шифрование текста алгоритмом AES

Ключ AES шифруется при помощи RSA, как на рисунке Рисунок 4.

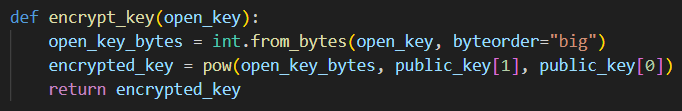


Рисунок 4 – Шифрование ключа при помощи RSA

В итоге после зашифрования получается файл, указанный на рисунке Рисунок 5.

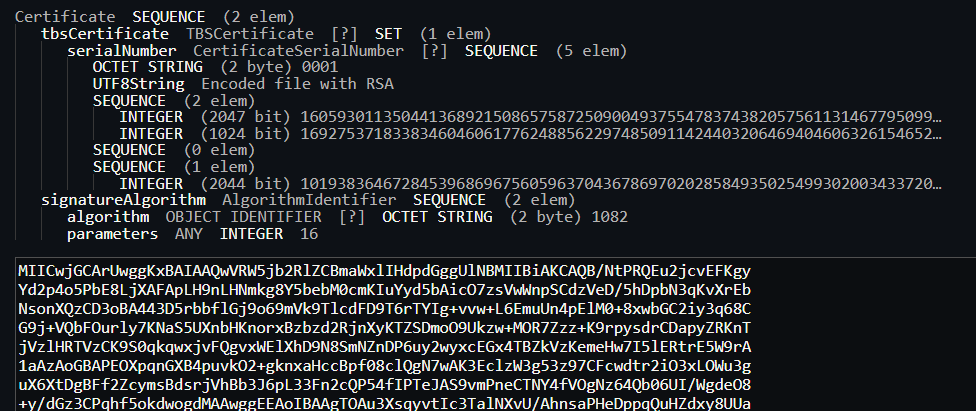


Рисунок 5 – ASN файл, полученный после зашифрования текста

В указанном выше ASN файле стоит уточнить значение полей типа . Данные поля содержат (сверху вниз соотвественно):

1. Число ;
2. Число ;
3. Зашифрованный ключ AES.

Шифрование происходит в обратном порядке идентичными методами, поэтому их реализация здесь показана не будет

## Реализация алгоритмов формирования и проверки электронной цифровой подписи

Как и в случаен с алгоритмами шифрования, описание формирования и проверки подписи было дано ранее. По этой причине в этой части так же, как и в предыдущей, буду представлены фактические реализации методов формирования и проверки, а также вид файла подписи.

Функция формирования выглядит как на рисунке Рисунок 6.

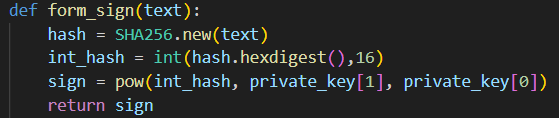


Рисунок 6 – Функция формирования подписи

Файл, получаемый в ходе формирования подписи, выглядит подобно примеру на рисунке Рисунок 7.

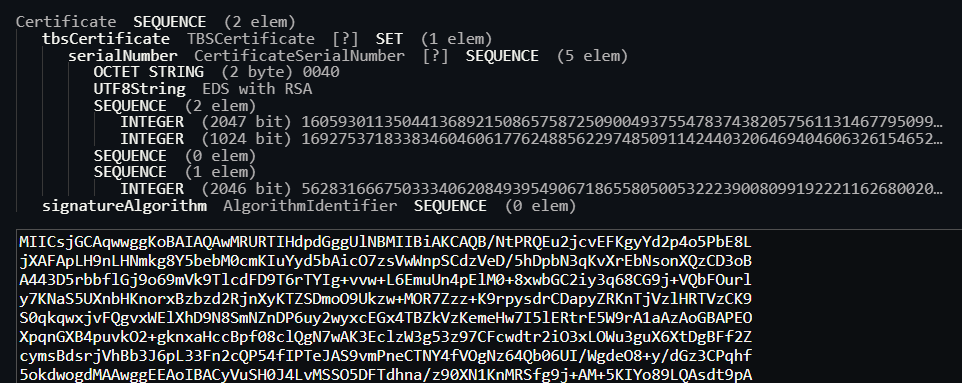


Рисунок 7 – Файл с подписью

Процедура проверки подписи показана на рисунке Рисунок 8.

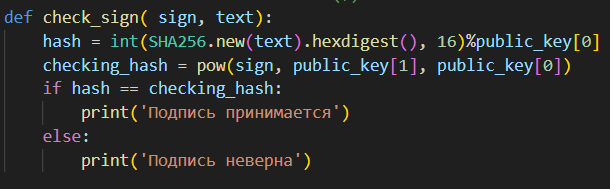


Рисунок 8 – Процедура проверки подписи

# Ответы на контрольные вопросы

1. Какие задачи положены в основу безопасности системы RSA?

В основу безопасности системы RSA положена задача факторизации большого простого числа. Зная открытый ключ и разложение показателя , злоумышленник сможет рассчитать функцию Эйлера , затем вычислить секретный показатель и расшифровать сообщение .

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

При подборе параметров криптосистемы RSA требуется выполнение равенства:

То есть это означает, что числа e и d являются взаимно простыми по модулю функции Эйлера .

Тогда существует коэффициент k, при котором справедливо равенство:

.

В схеме RSA расшифрование будет иметь следующий вид:

Аналогично:

По КТО:

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

Пускай существует разложение , где и– простые числа. Тогда функция Эйлера вычисляется следующим образом:

Пускай известны значения и . Тогда будет существовать система уравнения следующего вида:

Тогда и являются корнями некоторого квадратичного уравнения (по т. Виета) следующего вида:

Отсюда:

Для вычисления и по известным и в большинстве используются «быстрые» операции – сложение и деление на 2, и только по одному разу происходит вычисление квадратного корня и возведение в квадрат. Однако из-за однократности этих операций ими можно пренебречь. В свою очередь, при вычислении функции Эйлера также используются операции произведения и сложения.

Таким образом, в силу использования одинаковых по сложности операций при вычислении функции Эйлера и при вычислении множителей числа можно говорить о том, что данные операции являются полиномиально эквивалентными.

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

Гомоморфизм определяется следующим образом:

В случае RSA , где – исходное сообщение,  *–* открытый показатель. Тогда:

# Выводы и заключение

В ходе работы была изучена криптосистема RSA. Как было выявлено, система не подходит для шифрования больших сообщений в связи с сложностью операции возведение в степень по модулю. В связи с этим связывание RSA с другими алгоритмами шифрования выглядит мудро.

Сама криптосистема проста в реализации, что определенно является её преимуществом. Но этот признак вызывает и дополнительные опасности при использовании системы. Например, при нахождении и может возникнуть ситуация, когда один из параметров является недостаточно большим и может быть подобран.

Другая проблема или, вернее, особенность системы – необходимость наличия у всех абонентов различных n, чтобы обеспечить невозможность нахождения параметров одного абонента, используя параметры другого.

Данная криптосистема, как и многие другие не лишена преимуществ и недостатков. В следствие своего раннего появления, её можно и нужно считать хорошей опорой многих алгоритмов.

# Приложение А

Листинг разработанной программы на языке Python3:

from rsa import RSA

from asn import ASN

from Crypto.Cipher import AES

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

from Crypto.Random import get\_random\_bytes

from Crypto.Hash import SHA256

from Crypto.Util.Padding import pad, unpad

# init keys

public\_key = (0, 0)

private\_key = (0, 0)

aes\_key = get\_random\_bytes(32)

init\_vector = get\_random\_bytes(16)

rsa\_pq = (0, 0)

def get\_text(filename = None):

if filename == None:

filename = input("Enter the path to the file: ")

text = b''

with open(filename, "rb") as file:

for line in file:

text += line

return text, filename

#------------------ENCRYPTION BLOCK(#)------------------#

def encrypt\_text(text):

aes\_cipher = AES.new(key=aes\_key,

mode=AES.MODE\_CBC,

iv=init\_vector)

encrypted\_text = aes\_cipher.encrypt(pad(text, 16))

return encrypted\_text, aes\_key

def encrypt\_key(open\_key):

open\_key\_bytes = int.from\_bytes(open\_key, byteorder="big")

encrypted\_key = pow(open\_key\_bytes, public\_key[1], public\_key[0])

return encrypted\_key

def encrypt\_file(): # #

text, filename = get\_text()

encrypted\_text, open\_key = encrypt\_text(text)

encrypted\_key = encrypt\_key(open\_key)

asn\_text = ASN.encrypt\_rsa\_cipher(b'\x00\x01',

b'Encoded file with RSA',

public\_key[0],

public\_key[1],

encrypted\_key,

b'\x10\x82',

len(encrypted\_text),

encrypted\_text)

with open("#" + filename, "wb") as enc:

enc.write(asn\_text)

#------------------DECRYPTION BLOCK(~)------------------#

def decrypt\_key(encrypted\_key):

open\_key\_bytes = pow(encrypted\_key, private\_key[1], private\_key[0])

open\_key = open\_key\_bytes.to\_bytes(32, byteorder="big")

return open\_key

def decrypt\_text(text, key):

aes\_cipher = AES.new(key=key,

mode=AES.MODE\_CBC,

iv=init\_vector)

decrypted\_text = aes\_cipher.decrypt(text)

decrypted\_text = unpad(decrypted\_text, 16)

return decrypted\_text

def decrypt\_file():

text, filename = get\_text()

\_, \_, encrypted\_key, encrypted\_text = ASN.decrypt\_rsa\_cipher(text)

open\_key = decrypt\_key(encrypted\_key)

open\_text = decrypt\_text(encrypted\_text, open\_key)

with open("~" + filename, "wb") as dec:

dec.write(open\_text)

#------------------CREATING BLOCK(^)------------------#

def form\_sign(text):

hash = SHA256.new(text)

int\_hash = int(hash.hexdigest(),16)

sign = pow(int\_hash, private\_key[1], private\_key[0])

return sign

def create\_signature():# ^

text, filename = get\_text()

sign = form\_sign(text)

asn\_text = ASN.encrypt\_rsa\_eds(b'\x00\x40',

b'EDS with RSA',

public\_key[0],

public\_key[1],

sign)

with open("^" + filename, "wb") as enc:

enc.write(asn\_text)

#------------------VERIFYING BLOCK($)------------------#

def check\_sign( sign, text):

hash = int(SHA256.new(text).hexdigest(), 16)%public\_key[0]

checking\_hash = pow(sign, public\_key[1], public\_key[0])

if hash == checking\_hash:

print('Подпись принимается')

else:

print('Подпись неверна')

def verify\_signature():#

text, filename = get\_text()

sign\_asn, \_ = get\_text('^' + filename)

\_,\_,sign = ASN.decrypt\_rsa\_eds(sign\_asn)

check\_sign(sign, text)

#------------------VIEWING BLOCK------------------#

def view\_parameters():

print('''In this start RSA cryptosystem using this patameters:''')

print(f'''

p = {rsa\_pq[0]}\n

q = {rsa\_pq[1]}\n

n = {public\_key[0]}\n

e = {public\_key[1]}\n

d = {private\_key[1]}\n

''')

def init\_keys():

global rsa\_pq

# choosing p and q

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

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

rsa\_pq = (p, q)

e = 0

#init base parameters

n = p \* q

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

while GCD(e, v) > 1:

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

d = inverse(e, v)

return (n, e), (n, d)

def main():

global public\_key, private\_key

public\_key, private\_key = init\_keys()

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. Robin Hood (encrypt file)

2. James Bond (decrypt file)

3. Spider-man (create signature)

4. Captain Nemo (verify signature)

5. Albert Einstein (view parameters)

''' )

command = input("Choose your fighter: ")

if command == '1' or command == 'Robin Hood':

encrypt\_file()

elif command == '2' or command == 'James Bond':

decrypt\_file()

elif command == '3' or command == 'Spider-man':

create\_signature()

elif command == '4' or command == 'Captain Nemo':

verify\_signature()

elif command == '5' or command == 'Albert Einstein':

view\_parameters()

else:

print("incorrect input, try again")

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

main()