*Лабораторная работа №6*

*студент группы ИТз-41*

*Котик Дмитрия Степановича*

*Выполнение:\_\_\_\_\_\_\_\_\_ Защита:\_\_\_\_\_\_\_\_\_\_\_*

**Реализация комбинированных алгоритмов шифрования данных**

Цель работы: изучить принцип работы ассиметричных алгоритмов. Разработать приложение, совмещающее в себе достоинства симметричных и ассиметричных алгоритмов шифрования.

## Содержание работы

1. Реализовать криптоалгоритм RSA:

Требования к реализации:

* + простые числа p и q генерируются программой или задаются из файла, где они представлены в битовой форме.
  + Длина генерируемых значение p и q должна задаваться пользователем;
  + сгенерированные ключи записываются и хранятся в разных файлах.

1. Реализовать систему шифрования/дешифрования, в которой основным алгоритмом шифрования/дешифрования является AES. Ассиметричный алгоритм RSA используется для шифрования/дешифрования ключа.

Требования к реализации:

* + ключ для симметричного алгоритма должен генерироваться случайным образом;
  + зашифрованный текст должен сохраняться в первом файле, дешифрованный текст - во втором, а зашифрованный асимметричным алгоритмом ключ симметричного алгоритма – в третьем файле;
  + Приложение расшифровывает зашифрованный ключ симметричного шифрования. После этого расшифровывает шифртекст.
  + программа должна уметь работать с текстом произвольной длины.

## Основные понятия

При всех своих достоинствах симметричные алгоритмы обладают существенными недостатками. Во-первых, при симметричном шифровании необходимо, чтобы приемник и передатчик сообщения имели общий **ключ**, который каким-то образом должен быть им заранее передан. Один из основоположников шифрования с открытым ключом У. Диффи, заметил, что данное требование отрицает всю суть криптографии, а именно возможность поддерживать всеобщую секретность при коммуникациях.

Во-вторых, все участники обмена данными должны быть убеждены, что электронное сообщение было послано конкретным участником. Это более сильное требование, чем аутентификация.

Диффи и Хеллман достигли значительных результатов, предложив способ решения обеих задач, который радикально отличается от всех предыдущих подходов к шифрованию.

Ключ, используемый в симметричном шифровании, будем называть **секретным ключом**. Два ключа, используемые при шифровании с **открытым ключом**, будем называть открытым ключом и **закрытым ключом**. Закрытый ключ держится в секрете, создается локально каждым пользователем, открытый ключ – общеизвестен. Закрытый ключ обозначается KR, открытый ключ -KU.

Диффи и Хеллман описывают требования, которым должен удовлетворять алгоритм шифрования с открытым ключом.

**Односторонней функцией** называется такая функция, у которой каждый аргумент имеет единственное обратное значение, при этом вычислить саму функцию легко, а вычислить обратную функцию трудно.

## Ход работы

Разработаем программы:

* + Для реализации алгоритма RSA
  + Комбинированный алгоритм RSA и AES

**RSA.py**:

# -- Программа реализации алгоритма шифрования RSA --

from gen\_simple\_numbers import gen\_sp, test\_rabin\_miller, gcd

# Функция получение p и q

def get\_p\_and\_q():

choice = int(input("p и q генерируем программно или задаём из файла? (1/2): "))

if choice == 1:

# Программная генерация

# получаем p

n = int(input("Скольки битное число {0} должно быть сгенерировано?: ".format('p')))

p = gen\_sp.gen\_sp(n)

# получаем q

n = int(input("Скольки битное число {0} должно быть сгенерировано?: ".format('q')))

while True:

q = gen\_sp.gen\_sp(n)

if q == p:

continue

break

# Ручной ввод

# p = int(input("p = "))

# if test\_rabin\_miller.test\_Rabin\_Miller(p, 10) != True:

# print("Число p должно быть простым. ")

# continue

# else:

# break

# while True:

# q = int(input("q = "))

# if test\_rabin\_miller.test\_Rabin\_Miller(q, 10) != True:

# print("Число q должно быть простым. ")

# continue

# else:

# break

# Задание из файла из файла

else:

p, q = '', ''

path\_to\_file = input("Введите путь к 'p' (D:\Учёба\8 семестр (-)\(-) Информационная безопасность\Лабораторные работы\Лабораторная работа №6 \_\Лабораторная работа №6\\RSA\\texts\p.txt): ")

with open(path\_to\_file) as file:

for line in file:

p += line.strip()

while True:

if test\_rabin\_miller.test\_Rabin\_Miller(int('0b' + p, 2) ,10) != True:

p = bin(int(input("Число p должно быть простым. Поробуйте заново. p = ")))[2:]

continue

else:

p = int('0b' + p, 2)

break

path\_to\_file = input("Введите путь к 'q' (D:\Учёба\8 семестр (-)\(-) Информационная безопасность\Лабораторные работы\Лабораторная работа №6 \_\Лабораторная работа №6\\RSA\\texts\q.txt): ")

with open(path\_to\_file) as file:

for line in file:

q += line.strip()

while True:

if test\_rabin\_miller.test\_Rabin\_Miller(int('0b' + q, 2), 10) != True:

q = bin(int(input("Число q должно быть простым. Поробуйте заново. q = ")))[2:]

continue

else:

q = int('0b' + q, 2)

break

return p, q

# Функция получения n

def get\_n(p, q):

return p \* q

# Фукнкция получения функции Эйлера

def get\_function\_Eiler(p, q):

return (p-1) \* (q-1)

# Функция получения открытой экспоненты

def get\_e(f\_n, start=5):

for e in range(start, f\_n):

NOD = gcd.gcd(f\_n, e)

if NOD == 1:

return e

# Функция получения d. d - это такое число, мультипликативно обратное к числу e по модулю f\_n

def get\_d(f\_n, e):

e = e

while True:

for k in range(f\_n):

d = (k \* f\_n + 1) / e

if d % 1 == 0:

return int(d), e

for i in range(e, f\_n):

e += 1

NOD = gcd.gcd(f\_n, e)

if NOD == 1:

break

# Функция записи в файл

def write\_in\_file(path, key):

with open(path, 'w', encoding='utf-8') as file:

for path\_key in key:

file.write(str(path\_key) + '\n')

# Функция быстрого возведения в степень по модулю

def fast\_pow(base, degree, module):

degree = bin(degree)[2:]

r = 1

for i in range(len(degree) - 1, -1, -1):

r = (r \* base \*\* int(degree[i])) % module

base = (base \*\* 2) % module

return r

# Шифрование (c = m^e mod n)

def rsa\_encrypt(m, e, n):

return fast\_pow(m, e, n)

# Дешифрование (m = c^d mod n)

def rse\_decrypt(c, d, n):

return fast\_pow(c, d, n)

# Точка входа в программу

def rsa(from\_main=True, \_key=bytearray()):

p, q = get\_p\_and\_q()

print("p = {0}, q = {1}".format(p, q))

n = get\_n(p, q)

print("n = {0}".format(n))

f\_n = get\_function\_Eiler(p, q)

print("f(n) = {0}".format(f\_n))

e = get\_e(f\_n)

d, e = get\_d(f\_n, e)

print("d = {0}, e = {1}".format(d, e))

open\_key = (n, e)

secret\_key = (n, d)

if from\_main == True:

write\_in\_file('D:\Учёба\8 семестр (-)\(-) Информационная безопасность\Лабораторные работы\Лабораторная работа №6 \_\Лабораторная работа №6\\RSA\\texts\open\_key.txt', open\_key)

write\_in\_file('D:\Учёба\8 семестр (-)\(-) Информационная безопасность\Лабораторные работы\Лабораторная работа №6 \_\Лабораторная работа №6\\RSA\\texts\secret\_key.txt', secret\_key)

else:

write\_in\_file('D:\Учёба\8 семестр (-)\(-) Информационная безопасность\Лабораторные работы\Лабораторная работа №6 \_\Лабораторная работа №6\\AES\_and\_RSA\\texts\open\_key.txt', open\_key)

write\_in\_file('D:\Учёба\8 семестр (-)\(-) Информационная безопасность\Лабораторные работы\Лабораторная работа №6 \_\Лабораторная работа №6\\AES\_and\_RSA\\texts\secret\_key.txt', secret\_key)

# c = m^e mod n

encrypt\_key = []

for \_byte in \_key:

# print(\_byte)

encrypt\_key.append(rsa\_encrypt(\_byte, e, n))

# print(encrypt\_key)

print("ENCRYPT KEY: {0}".format(encrypt\_key))

return encrypt\_key

# Дешифрование ключа

def rsa\_dec(\_key):

key, numbers = bytearray(), []

with open('D:\Учёба\8 семестр (-)\(-) Информационная безопасность\Лабораторные работы'

'\Лабораторная работа №6 \_\Лабораторная работа №6\\AES\_and\_RSA\\texts'

'\secret\_key.txt', 'r', encoding='utf-8-sig') as file:

for line in file:

numbers.append(int(line.strip()))

n, d = numbers[0], numbers[1]

for num in \_key:

key.append(rse\_decrypt(num, d, n))

return key

# При вызове модуля, как основной программы, выполнить следующее:

if \_\_name\_\_ == "\_\_main\_\_":

rsa()

**generate\_key.py**:

# МОДУЛЬ генерации случайного ключа для AES-128

import random

def get\_key():

key = bytearray()

for i in range(16):

key.append(random.getrandbits(8))

print("KEY from MODULE: {0}".format(key))

return key

# def write\_key()

**AES-128\_and\_RSA.py**:

# -\*- coding: utf-8 -\*-

# -- Программа шафрования/дешифрования по алгоритму AES-128 --

from AES\_and\_RSA import encrypt

from AES\_and\_RSA import decrypt

from AES\_and\_RSA import generate\_key

from RSA import RSA

# Работа ведется в кодировке UTF-8

# Инициализация некоторых исходных данных

from AES\_and\_RSA import table\_data

# Функция ввода исходных данных. Принимает от пользователя ключ в виде символов кириллицы (для удобства).

def input\_data():

choices\_function = int(input("Шифровать/Дешифровать? (1/2): "))

if choices\_function == 1:

while True:

# key = input("Введите ключ (кириллицей): ")

key = generate\_key.get\_key()

encript\_key = RSA.rsa(from\_main=False, \_key=key)

with open('D:\Учёба\8 семестр (-)\(-) Информационная безопасность\Лабораторные работы'

'\Лабораторная работа №6 \_\Лабораторная работа №6\\AES\_and\_RSA\\texts'

'\encrypt\_key.txt', 'w', encoding='utf-8-sig') as file:

for \_byte in encript\_key:

file.write(str(\_byte) + '\n')

break

# if len(key.encode()) == 16:

# key = bytearray(key.encode())

# break

# else:

# print("Ключ должен быть длиной 8 символов (16 байт в utf-8)")

type\_input\_data = int(input("Данные вводим с клавиатуры или из файла? (1/2): "))

if type\_input\_data == 1:

string = input("Введите текст: ")

else:

path\_to\_file = input(

"Введите путь (D:\Учёба\8 семестр (-)\(-) Информационная безопасность\Лабораторные работы"

"\Лабораторная работа №6 \_\Лабораторная работа №6\\AES\_and\_RSA\\texts\encrypt\_text.txt): ")

with open(path\_to\_file, "r", encoding='utf-8-sig') as file: # encoding='UTF-8'

string = file.read()

if choices\_function == 2:

# Считываение ключа

new\_key = list()

with open('D:\Учёба\8 семестр (-)\(-) Информационная безопасность\Лабораторные работы'

'\Лабораторная работа №6 \_\Лабораторная работа №6\\AES\_and\_RSA\\texts'

'\encrypt\_key.txt', 'r', encoding='utf-8-sig') as file:

for line in file:

new\_key.append(int(line.strip()))

print("READ\_KEY: {0}".format(new\_key))

key = RSA.rsa\_dec(new\_key)

print("DECRYPT\_KEY: {0}".format(key))

# print("Код.стр.: " + str(string))

# print("Ключ : " + str(key))

# choices\_function - кодировать или декодировать

# ley - ключ

# string - кодируемые байты

return choices\_function, key, string

# Функция парсит последовательность строковых символов в байты

def string\_to\_hex(string):

text\_bytes = bytearray()

i = 0

while i < len(string):

print("2 символа: {0}".format(string[i+3 : i+5]))

print("тип символов: {0}".format(type(string[i+3 : i+5])))

\_byte = bytes.fromhex(string[i+3 : i+5])[0]

print("byte: {0}".format(\_byte))

text\_bytes.append(\_byte)

print("text byte: {0}".format(text\_bytes))

i += 5

print('закодированная: {0}'.format(text\_bytes))

return text\_bytes

# \0x14\0xcc\0x26\0xd4\0xfd\0xdc\0xa5\0xca\0x06\0x02\0x09\0xf1\0xa9\0x1d\0x02\0xa6

# Функция нормализует текст, т.е. если не хватает байтов до 16, заполняет символом '('; ord(40) == '('; hex = 0x28

def normalize\_text(string):

if str == type(string):

array\_bytes = bytearray(string.encode(encoding='utf-8'))

else:

array\_bytes = string

ost = len(array\_bytes) % 16

if ost != 0:

for i in range(16-ost):

array\_bytes.append(0x28)

# print('normalize array\_bytes: {0}'.format(array\_bytes))

return array\_bytes

# Основная функция

def main():

choices\_function, key, string = input\_data()

# string = bytearray(b'\x00\x04\x12\x14\x12\x04\x12\x00\x0C\x00\x13\x11\x08\x23\x19\x19\x00\x04\x12\x14\x12\x04\x12\x00\x0C\x00\x13\x11\x08\x23\x19\x19')

print("STRING: {0}, len(STRING) = {1}".format(string, len(string)))

if choices\_function == 1:

text\_bytes = normalize\_text(string)

else:

text\_bytes = string\_to\_hex(string)

\_round = 0

output\_encrypt\_bytes = list()

output\_decrypt\_bytes = list()

# testing bytes from book (страница 241)

# text\_bytes = bytearray(b'\x00\x04\x12\x14\x12\x04\x12\x00\x0C\x00\x13\x11\x08\x23\x19\x19') # <--- text for encrypt

# testing encrypt\_bytes

# text\_bytes = bytearray(b'\xbc\x02\x8b\xd3\xe0\xe3\xb1\x95\x55\x0d\x6d\xf8\xe6\xf1\x82\x41') # <--- text for decrypt

if choices\_function == 1:

# testing keys

# key = bytearray(b'\x0f\x15\x71\xc9\x47\xd9\xe8\x59\x0c\xb7\xad\xdf\xaf\x7f\x67\x98')

# key = bytearray(b'\x24\x75\xA2\xB3\x34\x75\x56\x88\x31\xE2\x12\x00\x13\xAA\x54\x87')

# key for book

# key = bytearray(b'\x24\x75\xA2\xB3\x34\x75\x56\x88\x31\xE2\x12\x00\x13\xAA\x54\x87') # <--- key

print("KEY = {0}".format(key))

rounds\_key = encrypt.expand\_key(key) # !!!!!!!!!!!!!!!!!!!!!!!!!! расширение ключа

for key in rounds\_key:

print("key: {0}".format(key))

# Разбиваем текст на блоки и оперируем ими, но при этом каждая первая итерация над блоком будет проводиться отдельно

for i in range(int(len(text\_bytes)/16)):

print('---' \* 20 + '\n' + 'round: {0}'.format(\_round) + '\n' + '---' \* 20)

block = [] # !!!!!!!!!!!!!!!!!!!!!!!!!! блок

text = text\_bytes[i\*16 : (i+1)\*16]

for j in range(4):

block.append(text[j\*4 : (j+1)\*4])

state = encrypt.add\_round\_key(\_round, rounds\_key, block) # !!!!!!!!!!!!!!!!!!!!!!!!!! матрица состояний

\_round = 1

for i in range(1, 10):

print('---' \* 20 + '\n' + 'round: {0}'.format(\_round) + '\n' + '---' \* 20)

state = encrypt.sub\_bytes(state) # !!!!!!!!!!!!!!!!!!!!!!!!!! операция замены байтов

# print("state после sub\_bytes: {0}".format(state))

print("sub\_bytes Выполнилось")

state = encrypt.shift\_rows(state) # !!!!!!!!!!!!!!!!!!!!!!!!!! операция циклического сдвига байтов

print("shift\_rows Выполнилось")

state = encrypt.mix\_columns(state)

print("mix\_columns Выполнилось")

state = encrypt.add\_round\_key(\_round, rounds\_key, state) # !!!!!!!!!!!!!!!!!!!!!!!!!! операция добавления ключа

print("add\_round\_key Выполнилось")

\_round += 1

print('---' \* 20 + '\n' + 'round: {0}'.format(\_round) + '\n' + '---' \* 20)

state = encrypt.sub\_bytes(state)

state = encrypt.shift\_rows(state)

state = encrypt.add\_round\_key(\_round, rounds\_key, state)

output\_encrypt\_bytes += state

\_round = 0

result = bytearray()

\_str\_result = ''

for i in output\_encrypt\_bytes:

result += i

for j in i:

if len(hex(j)) == 3:

\_str\_result += '\\' + hex(j)[0:2] + '0' + hex(j)[-1]

else:

\_str\_result += '\\' + hex(j)

with open('D:\Учёба\8 семестр (-)\(-) Информационная безопасность\Лабораторные работы'

'\Лабораторная работа №6 \_\Лабораторная работа №6\\AES\_and\_RSA\\texts'

'\encrypt\_text.txt', 'w', encoding='utf-8-sig') as file:

file.write(\_str\_result)

print('Зашифрованные байты: {0}'.format(result))

print('Зашифрованные байты \_str\_: {0}'.format(\_str\_result))

else:

# key = bytearray(b'\x24\x75\xA2\xB3\x34\x75\x56\x88\x31\xE2\x12\x00\x13\xAA\x54\x87')

rounds\_key = decrypt.expand\_key(key)

for key in rounds\_key:

print("key: {0}".format(key))

# Разбиваем текст на блоки и оперируем ими, но при этом каждая первая итерация над блоком будет проводиться отдельно

for i in range(int(len(text\_bytes)/16)):

print('---' \* 20 + '\n' + 'round: {0}'.format(\_round) + '\n' + '---' \* 20)

block = [] # !!!!!!!!!!!!!!!!!!!!!!!!!! блок

text = text\_bytes[i\*16 : (i+1)\*16]

for j in range(4):

block.append(text[j \* 4: (j + 1) \* 4])

state = decrypt.add\_round\_key(\_round, rounds\_key, block) # !!!!!!!!!!!!!!!!!!!!!!!!!! матрица состояний

\_round = 1

for i in range(1, 10):

print('---' \* 20 + '\n' + 'round: {0}'.format(\_round) + '\n' + '---' \* 20)

state = decrypt.inv\_shift\_rows(state) # !!!!!!!!!!!!!!!!!!!!!!!!!! операция циклического сдвига байтов

print("shift\_rows Выполнилось")

state = decrypt.inv\_sub\_bytes(state) # !!!!!!!!!!!!!!!!!!!!!!!!!! операция замены байтов

# print("state после sub\_bytes: {0}".format(state))

print("sub\_bytes Выполнилось")

state = decrypt.add\_round\_key(\_round, rounds\_key, state) # !!!!!!!!!!!!!!!!!!!!!!!!!! операция добавления ключа

print("add\_round\_key Выполнилось")

state = decrypt.inv\_mix\_columns(state)

print("inv\_mix\_columns Выполнилось")

\_round += 1

print('---' \* 20 + '\n' + 'round: {0}'.format(\_round) + '\n' + '---' \* 20)

state = decrypt.inv\_shift\_rows(state)

state = decrypt.inv\_sub\_bytes(state)

state = decrypt.add\_round\_key(\_round, rounds\_key, state)

output\_decrypt\_bytes += state

\_round = 0

print('---' \* 20 + '\n' + '---' \* 20)

result = bytearray()

for i in output\_decrypt\_bytes:

result += i

while True:

if result[-1] == 40:

result.pop()

else:

break

print('Дешифрованные байты: {0}'.format(result.decode()))

with open('D:\Учёба\8 семестр (-)\(-) Информационная безопасность\Лабораторные работы'

'\Лабораторная работа №6 \_\Лабораторная работа №6\\AES\_and\_RSA\\texts'

'\decrypt\_text.txt', 'w', encoding='utf-8-sig') as file:

file.write(str(result.decode()))

# Точка входа

main()

# key: [bytearray(b'$u\xa2\xb3'), bytearray(b'4uV\x88'), bytearray(b'1\xe2\x12\x00'), bytearray(b'\x13\xaaT\x87')]

# key: [bytearray(b'\x89U\xb5\xce'), bytearray(b'\xbd \xe3F'), bytearray(b'\x8c\xc2\xf1F'), bytearray(b'\x9fh\xa5\xc1')]

# key: [bytearray(b'\xceS\xcd\x15'), bytearray(b'ss.S'), bytearray(b'\xff\xb1\xdf\x15'), bytearray(b'`\xd9z\xd4')]

# key: [bytearray(b'\xff\x89\x85\xc5'), bytearray(b'\x8c\xfa\xab\x96'), bytearray(b'sKt\x83'), bytearray(b'\x13\x92\x0eW')]

# key: [bytearray(b'\xb8"\xde\xb8'), bytearray(b'4\xd8u.'), bytearray(b'G\x93\x01\xad'), bytearray(b'T\x01\x0f\xfa')]

# key: [bytearray(b'\xd4T\xf3\x98'), bytearray(b'\xe0\x8c\x86\xb6'), bytearray(b'\xa7\x1f\x87\x1b'), bytearray(b'\xf3\x1e\x88\xe1')]

# key: [bytearray(b'\x86\x90\x0b\x95'), bytearray(b'f\x1c\x8d#'), bytearray(b'\xc1\x03\n8'), bytearray(b'2\x1d\x82\xd9')]

# key: [bytearray(b'b\x83>\xb6'), bytearray(b'\x04\x9f\xb3\x95'), bytearray(b'\xc5\x9c\xb9\xad'), bytearray(b'\xf7\x81;t')]

# key: [bytearray(b'\xeea\xac\xde'), bytearray(b'\xea\xfe\x1fK'), bytearray(b'/b\xa6\xe6'), bytearray(b'\xd8\xe3\x9d\x92')]

# key: [bytearray(b'\xe4?\xe3\xbf'), bytearray(b'\x0e\xc1\xfc\xf4'), bytearray(b'!\xa3Z\x12'), bytearray(b'\xf9@\xc7\x80')]

# key: [bytearray(b'\xdb\xf9.&'), bytearray(b'\xd58\xd2\xd2'), bytearray(b'\xf4\x9b\x88\xc0'), bytearray(b'\r\xdbO@')]

# Зашифрованные байты bytearray(b'\x14\xcc&\xd4\xfd\xdc\xa5\xca\x06\x02\t\xf1\xa9\x1d\x02\xa6'):

# Зашифрованные байты \_str\_: \0x14\0xcc\0x26\0xd4\0xfd\0xdc\0xa5\0xca\0x06\0x02\0x09\0xf1\0xa9\0x1d\0x02\0xa6

## Вывод:

В этой лабораторной работе были реализованы два алгоритма – симметричный и ассиметричный. Достоинства и недостатки есть у каждого из них.