Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Кафедра информатики

Лабораторная работа №3. Компьютерная реализация алгоритмов криптографии с открытым ключом на примере RSA

Выполнил: cтудент гр. 853501

Яковлев А.Б.

Проверил:

Протько М.И.

Минск 2021

# Постановка задачи и описание алгоритма

**Цель:** создать программу, читающую данные из файла и шифрующую

(дешифрующую) их помощью алгоритма RSA.

Алгоритм, разработанный Ривестом, Шамиром и Адлеманом, использует выражения с экспонентами. Данные шифруются блоками, каждый блок рассматривается как число, меньшее некоторого числа n. Шифрование и дешифрование имеют следующий вид для некоторого незашифрованного блока М и зашифрованного блока С.

С = Ме (mod n)

M = Cd (mod n) = (Me)d (mod n) = Med (mod n)

Как отправитель, так и получатель должны знать значение n. Отправитель знает значение е, получатель знает значение d. Таким образом, *открытый ключ* есть KU = {e, n} и *закрытый ключ* есть KR = {d, n}. При этом должны выполняться следующие условия:

1. Возможность найти *е*, *d* и *n* такие, что *Med = M mod n* для всех М < n .

2. Относительная легкость вычисления Ме и Сd для всех значений М < n.

3. Невозможность определить d, зная е и n.

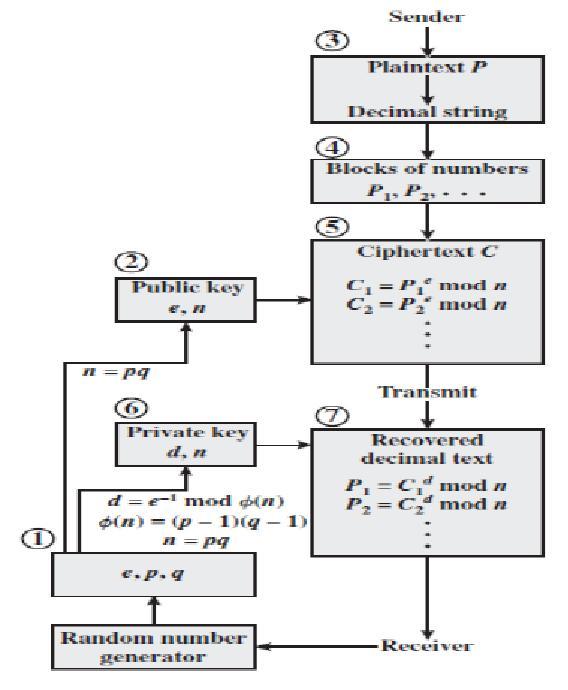
**Шифрование**

|  |
| --- |
| 1. Незашифрованный текст: М < n |
| 1. Зашифрованный текст: С = М е (mod n) |

**Расшифрование**

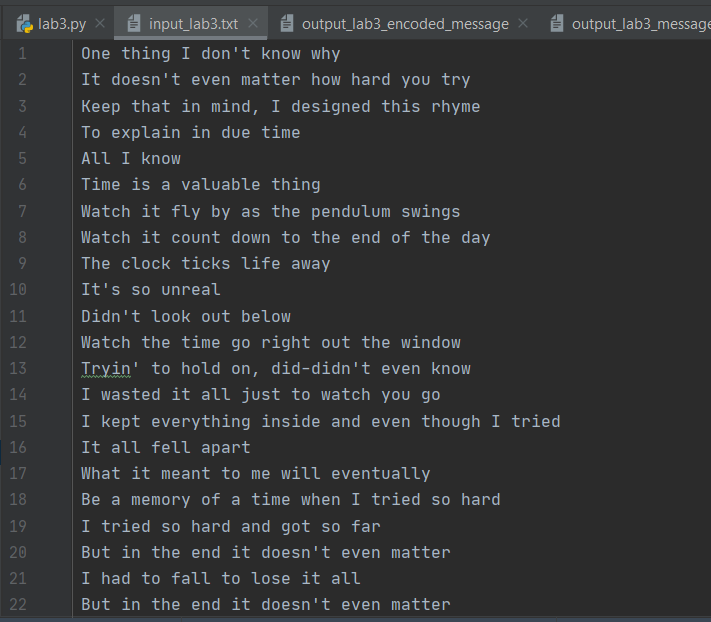
|  |
| --- |
| 1. Зашифрованный текст: С |
| 1. Незашифрованный текст: М = Сd (mod n) |

# Блок-схемы алгоритмов

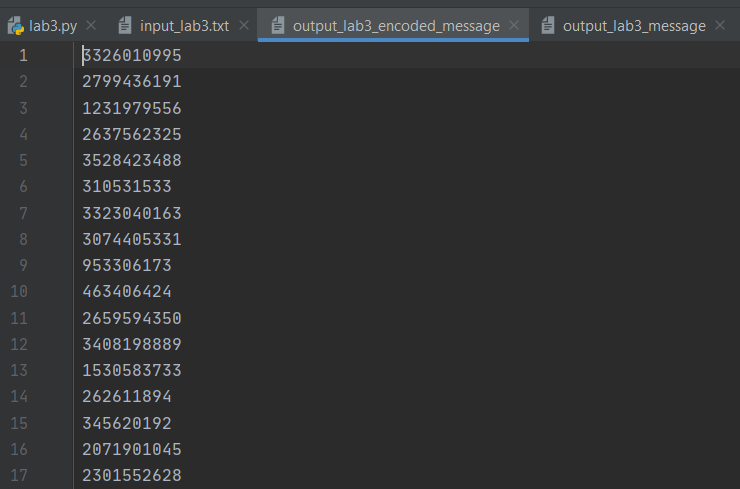


# Результаты выполнения программы

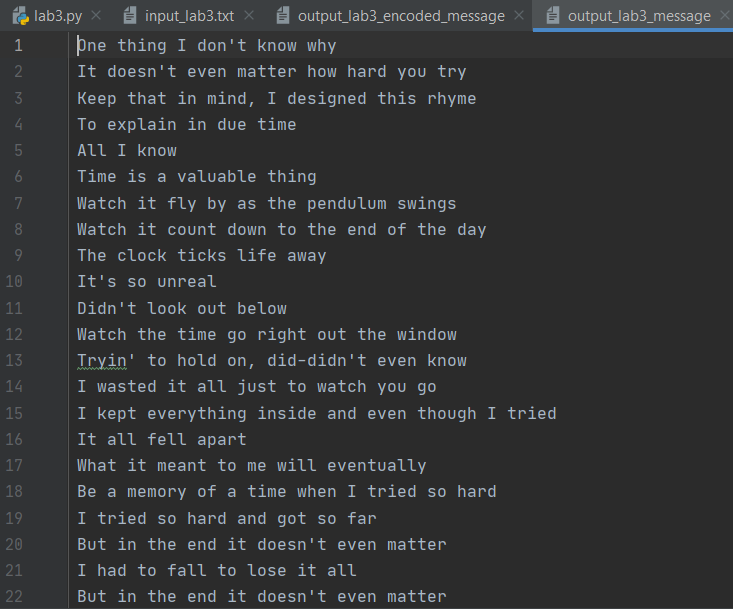
Файл с входными данными:



Зашифрованное сообщение:



Расшифрованное сообщение:



# Исходный код

import typing as tp  
  
from numpy.random import randint  
from functions import is\_prime, are\_relatively\_prime  
  
  
class RSA:  
 def \_\_init\_\_(self, private\_key: tp.Tuple[int, int], public\_key: tp.Tuple[int, int]):  
 self.\_public\_key = public\_key  
 self.\_private\_key = private\_key  
  
 @staticmethod  
 def generate\_key\_pair(p: int, q: int) -> tp.Tuple[tp.Tuple[int, int], tp.Tuple[int, int]]:  
 assert is\_prime(p)  
 assert is\_prime(q)  
 n = p \* q  
 phi = (p - 1) \* (q - 1)  
 d = randint(1, n)  
 while not are\_relatively\_prime(d, phi):  
 d = randint(1, n)  
 e = pow(d, -1, phi)  
 return (d, n), (e, n)  
  
 @staticmethod  
 def slice\_into\_chunks(array: bytes, chunk\_size: int) -> tp.Generator[tp.SupportsBytes, None, None]:  
 *"""  
 Allows to slice bytes array into evenly sized chunks* ***:param*** *array: array to slice* ***:param*** *chunk\_size: size of each chunk* ***:return****: yields chunks one by one  
 """* for i in range(0, len(array) - chunk\_size, chunk\_size):  
 yield list(array[i: i + chunk\_size])  
 output = list(reversed(array[len(array) - len(array) % chunk\_size: len(array)]))  
 while len(output) % chunk\_size > 0:  
 output.append(0x00)  
 yield list(reversed(output))  
  
 @staticmethod  
 def \_int\_to\_str(num: int) -> str:  
 result = ''  
 while num > 0:  
 result += chr(num & 255)  
 num >>= 8  
 return ''.join(reversed(result))  
  
 @staticmethod  
 def encode\_message(message: str) -> tp.List[int]:  
 return [int.from\_bytes(chunk, byteorder='big', signed=False)  
 for chunk in RSA.slice\_into\_chunks(message.encode('UTF-8'), 4)]  
  
 @staticmethod  
 def decode\_message(encoded\_message: tp.List[int]) -> str:  
 return ''.join(map(RSA.\_int\_to\_str, encoded\_message))  
  
 def encrypt(self, message: str) -> tp.List[int]:  
 return [pow(m, \*self.\_public\_key) for m in RSA.encode\_message(message)]  
  
 def decrypt(self, message: tp.List[int]) -> str:  
 return RSA.decode\_message([pow(m, \*self.\_private\_key) for m in message])  
  
  
def main() -> None:  
 p, q = 68963, 66047  
 private\_key, public\_key = RSA.generate\_key\_pair(p, q)  
 encoder = RSA(private\_key, public\_key)  
  
 with open('input\_files/input\_lab3.txt', 'r') as f:  
 message = f.read()  
 encoded\_message = encoder.encrypt(message)  
 with open('output\_files/output\_lab3\_encoded\_message', 'w') as f:  
 f.writelines([str(value) + '\n' for value in encoded\_message])  
  
 with open('output\_files/output\_lab3\_message', 'w') as f:  
 f.write(encoder.decrypt(encoded\_message))  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

# Вывод

В результате выполнения работы была реализована программа, позволяющая при заданных исходных параметрах шифровать и расшифровывать текстовые значения формата UTF-8 с помощью алгоритма RSA.