Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Кафедра информатики

Лабораторная работа №4. Компьютерная реализация алгоритма шифрования Эль-Гамаля

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 Эль-Гамаля.

Схема Эль-Гамаля (Elgamal) — криптосистема с открытым ключом, основанная на трудности вычисления дискретных логарифмов в конечном поле. Криптосистема включает в себя алгоритм шифрования и алгоритм цифровой подписи. Схема Эль-Гамаля лежит в основе бывших стандартов электронной цифровой подписи в США (DSA) и России (ГОСТ Р 34.10-94).

Генерация ключей:

* Генерируется случайное простое число p.
* Выбирается целое число g — первообразный корень p.
* Выбирается случайное целое число x такое, что 1 &lt; x &lt; p - 1.
* Вычисляется y = g^x mod p.
* Открытым ключом является y, закрытым ключом — число x.

**Шифрование**

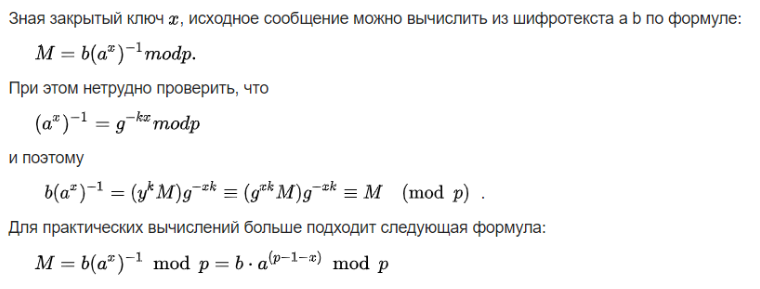
Сообщение M должно быть меньше числа p. Сообщение шифруется следующим образом:

1. Выбирается сессионный ключ — случайное целое число К такое, что

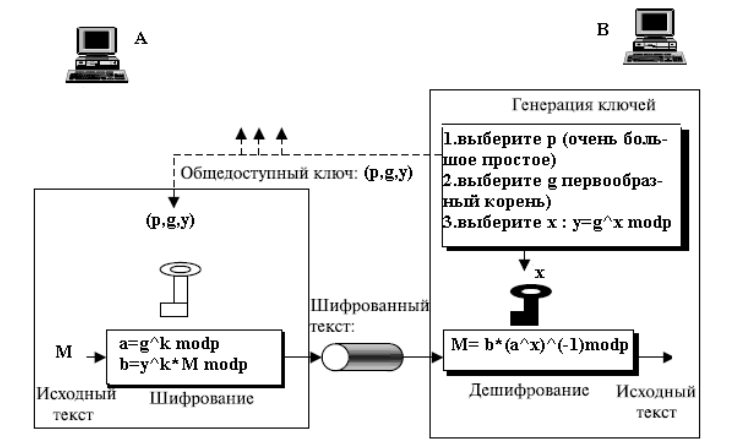
2. Вычисляются числа

3. Пара чисел *a, b* является шифротекстом.

**Расшифрование**

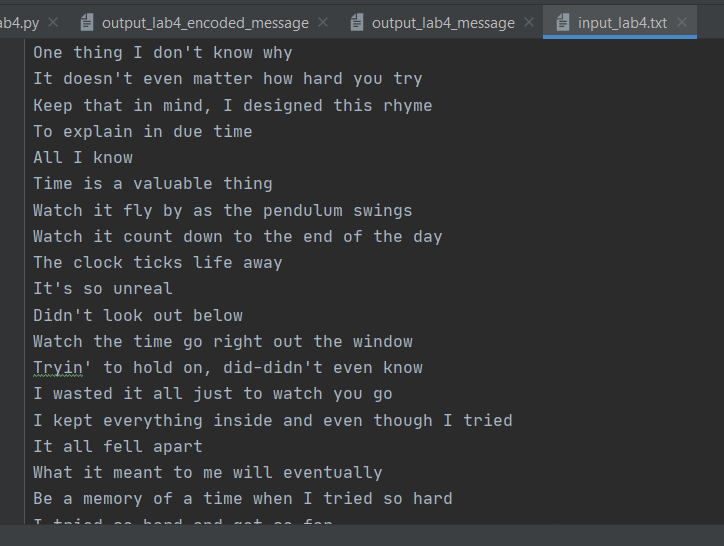


# Блок-схемы алгоритмов

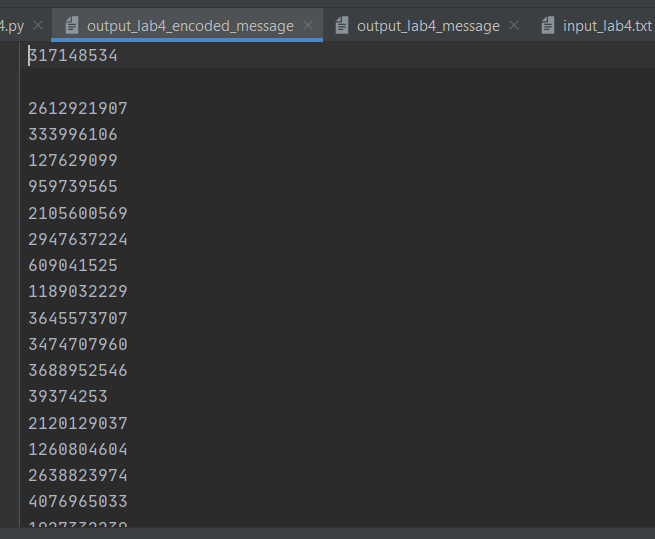


# Результаты выполнения программы

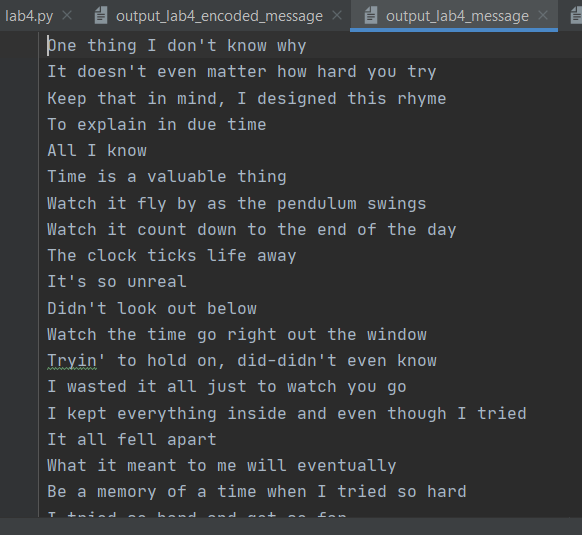
Файл с входными данными:



Зашифрованное сообщение:



Расшифрованное сообщение:



# Исходный код

import typing as tp  
from numpy.random import randint  
from functions import factorize, is\_prime  
  
  
class ElGamal:  
 def \_\_init\_\_(self, p: int) -> None:  
 *"""* ***:param*** *p: random prime number, larger than 2  
 """* assert is\_prime(p)  
 self.p = p  
 self.session\_key = randint(1, self.p)  
 self.x = randint(1, self.p)  
 self.g = self.find\_primitive\_root()  
 assert self.g is not None  
 self.y = pow(self.g, self.x, self.p)  
  
 def find\_primitive\_root(self) -> tp.Optional[int]:  
 *"""  
 For prime numbers primitive root can generate any element of multiplicative  
 group of integers modulo p (except 0) by raising it into some power* ***:return****: smallest primitive root if there's any. For primes above 2 there's always at least one  
 (more precisely phi(p - 1))  
 """* for i in range(2, self.p - 1):  
 flag = False  
 for key in factorize(self.p - 1).keys():  
 if pow(i, (self.p - 1) // key, self.p) == 1:  
 flag = True  
 break  
 if not flag:  
 return i  
  
 @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 ElGamal.slice\_into\_chunks(message.encode('UTF-8'), 4)]  
  
 @staticmethod  
 def decode\_message(encoded\_message: tp.List[int]) -> str:  
 return ''.join(map(ElGamal.\_int\_to\_str, encoded\_message))  
  
 def encrypt(self, message: str) -> tp.Tuple[int, tp.List[int]]:  
 first = pow(self.g, self.session\_key, self.p)  
 second = [(item \* pow(self.y, self.session\_key, self.p)) % self.p  
 for item in ElGamal.encode\_message(message)]  
 return first, second  
  
 def decrypt(self, first: int, second: tp.List[int]) -> str:  
 return ElGamal.decode\_message([item \* pow(first, -self.x, self.p) % self.p for item in second])  
  
  
def main() -> None:  
 prime = 4294969633  
 encoder = ElGamal(prime)  
 with open('input\_files/input\_lab4.txt', 'r') as f:  
 message = f.read()  
 a, b = encoder.encrypt(message)  
 with open('output\_files/output\_lab4\_encoded\_message', 'w') as f:  
 f.write(str(a) + '\n\n')  
 f.writelines([str(value) + '\n' for value in b])  
  
 with open('output\_files/output\_lab4\_message', 'w') as f:  
 f.write(encoder.decrypt(a, b))  
  
  
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 Эль-Гамаля.