Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

КАФЕДРА ИНФОРМАТИКИ

Лабораторная работа № 4

Компьютерная реализация алгоритма шифрования Эль-Гамаля

Выполнил: Нетецкая Ю.В.

Проверил: Олисейчик В.В.

Минск 2021

**Постановка задачи**

Реализовать программные средства шифрования и дешифрования текстовых файлов при помощи алгоритма Эль-Гамаля. Проверить корректность выполнения программных средств на примерах.

**Описание использованных алгоритмов**

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.

Генерация ключей:

1. Генерируется случайное простое число p.
2. Выбирается целое число g — первообразный корень p.
3. Выбирается случайное целое число x такое, что 1 < x < p - 1.
4. Вычисляется y = g^x mod p.
5. 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

1 < k < p - 1

1. Вычисляются числа a = g^k mod p и b = y^k M mod p.
2. Пара чисел a b является шифротекстом.

Нетрудно увидеть, что длина шифротекста в схеме Эль-Гамаля длиннее исходного сообщения M вдвое.

Расшифрование

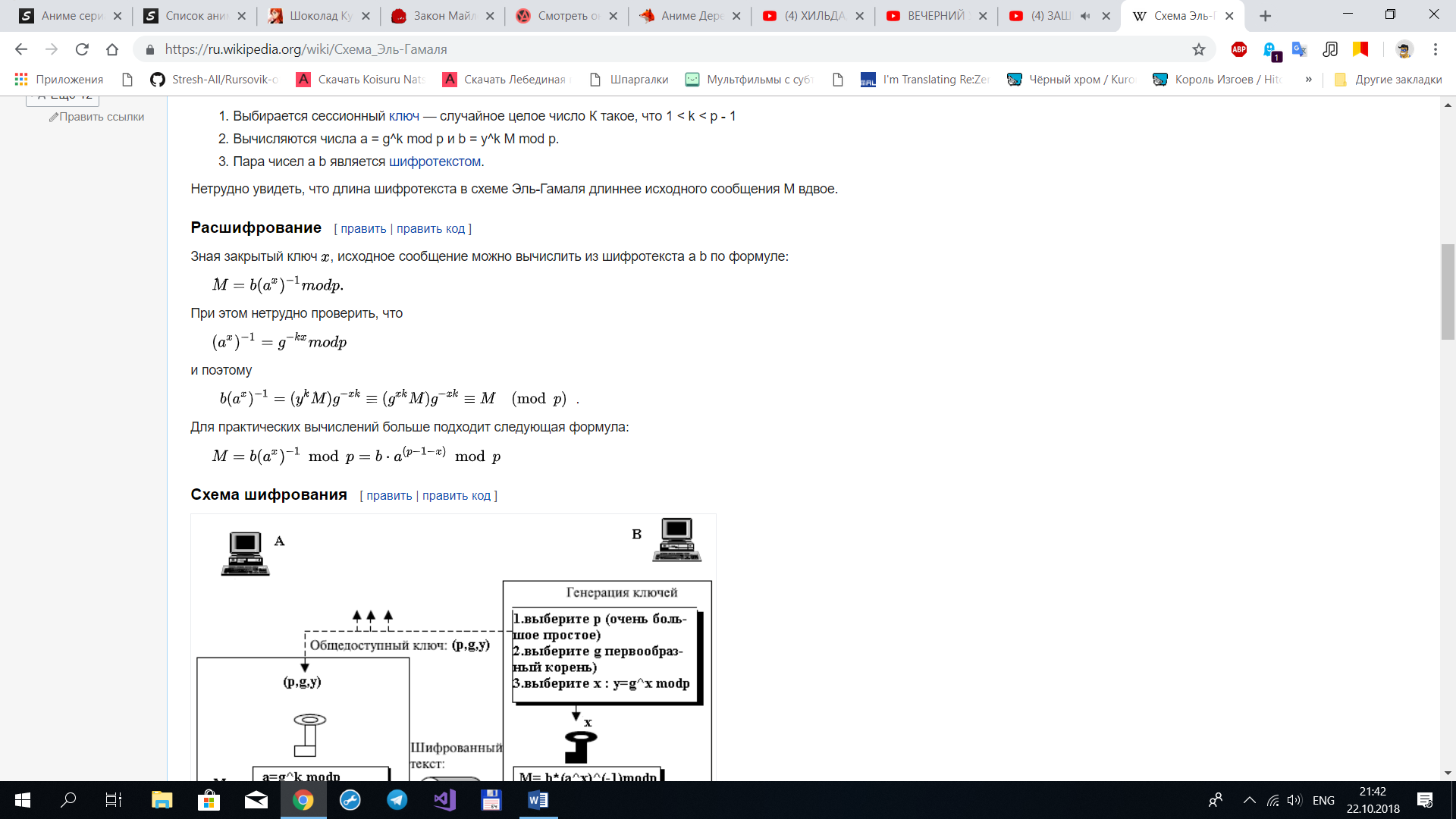
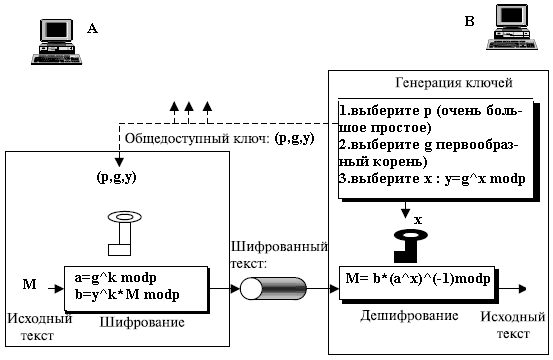
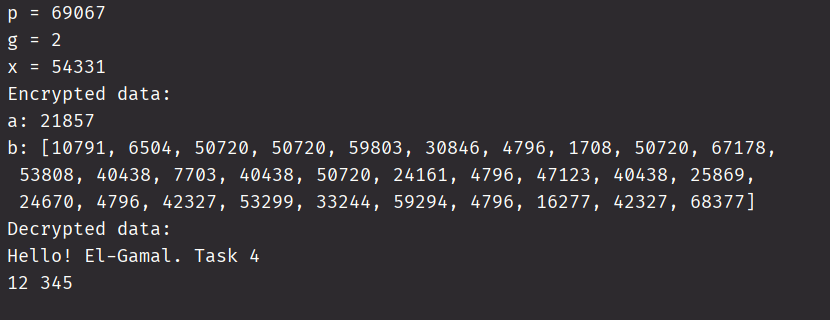


Схема шифрования



**Результат работы программы**



**Код программы**

import random  
from sympy import \*  
  
MIN\_N = 10000  
MAX\_N = 100000  
  
  
def prime\_number():  
 while True:  
 number = random.randrange(MIN\_N, MAX\_N)  
 if isprime(number):  
 break  
 return number  
  
  
def primitive\_root(*number*):  
 o = 1  
 r = 0  
 while r < *number*:  
 k = *pow*(r, o, *number*)  
 while k > 1:  
 o = o + 1  
 k = (k \* r) % *number* if o == (*number* - 1):  
 return r  
 o = 1  
 r += 1  
 return None  
  
  
def are\_relatively\_prime(*a*, *b*):  
 for n in *range*(2, *min*(*a*, *b*) + 1):  
 if *a* % n == *b* % n == 0:  
 return False  
 return True  
  
  
def get\_relatively\_prime(*n*):  
 while True:  
 res = random.randint(2, *n* - 1)  
 if are\_relatively\_prime(res, *n*):  
 return res  
  
  
def get\_keys():  
 p = prime\_number()  
 *print*('p = ' + *str*(p))  
 *# g - первообразный корень p* g = primitive\_root(p)  
 *print*('g = ' + *str*(g))  
 assert g is not None  
 x = get\_relatively\_prime(p - 1)  
 *print*('x = ' + *str*(x))  
 y = (g \*\* x) % p  
 return p, g, x, y  
  
  
def encrypt(*p*, *g*, *y*, *text*):  
 key = random.randint(2, *p* - 2)  
 for M in *text*:  
 if *ord*(M) > *p*:  
 *print*("BAD")  
 a = (*g* \*\* key) % *p* b = []  
 for i in *range*(*len*(*text*)):  
 b.append(*y* \*\* key \* *ord*(*text*[i]) % *p*)  
 return a, b  
  
  
def decrypt(*p*, *x*, *a*, *b*):  
 res = []  
 for bi in *b*:  
 res.append(*chr*(*int*(bi \* *a* \*\* (*p* - 1 - *x*) % *p*)))  
 return ''.join(res)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 file = *open*("data.txt", "r")  
 text = file.read()  
 file.close()  
 p, g, x, y = get\_keys()  
 a, b = encrypt(p, g, y, text)  
 res\_text = decrypt(p, x, a, b)  
 *print*('Encrypted data:\n' + 'a: ' + *str*(a) + '\nb: ' + *str*(b))  
 *print*('Decrypted data:\n' + *str*(res\_text))

**Вывод**

В ходе данной лабораторной работе были изучены теоретические сведения.

Создана программа, читающая данные из файла и шифрующая (дешифрующая) их помощью алгоритма Эль-Гамаля.