Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Кафедра информатики

Отчет по лабораторной работе №4

Выполнил:

студент гр. 953504

Кондрашов И.Д.

Проверил:

Протько М.И.

Минск 2022

**Оглавление**

[**1. Постановка задачи 2**](#_Toc116311529)

[**2. Теоретическая сведения 2**](#_Toc116311530)

[**Генерация ключей: 2**](#_Toc116311531)

[**Шифрование 2**](#_Toc116311532)

[**Расшифрование 3**](#_Toc116311533)

[**3. Блок-схема алгоритма 4**](#_Toc116311534)

[**4.Пример работы программы 5**](#_Toc116311535)

[**5. Код программы 6**](#_Toc116311536)

[**6. Вывод 9**](#_Toc116311537)

# **1. Постановка задачи**

Необходимо реализовать программные средства шифрования и дешифрования текстовых файлов при помощи алгоритма Эль-Гамаля*.*

**ЗАДАНИЕ:**

1. Изучить теоретические сведения.
2. Создать программы, читающие данные из файла и шифрующие (дешифрующие) их с помощью алгоритма Эль-Гамаля.

# **2. Теоретическая сведения**

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 < x < 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:

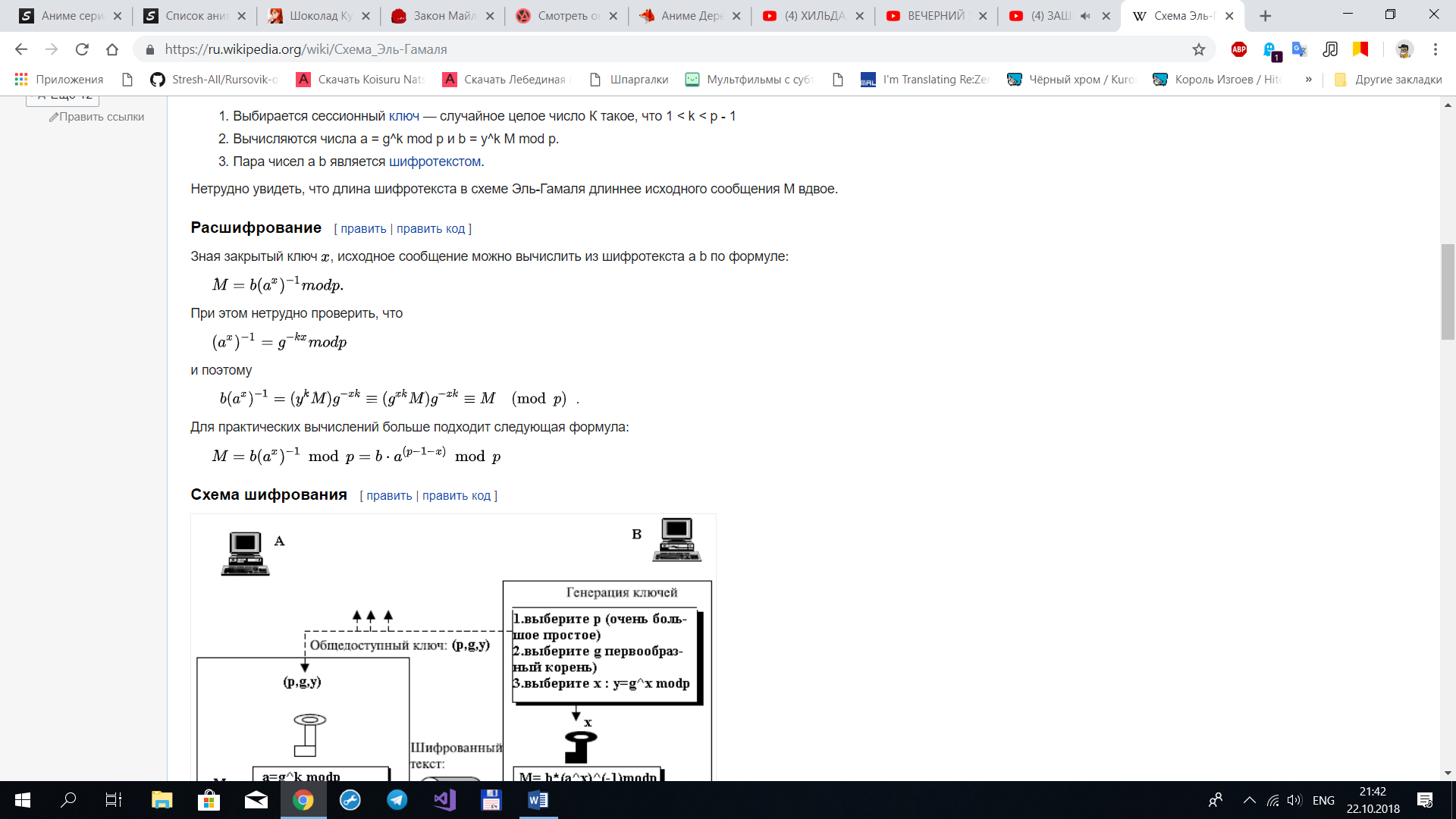
1. Выбирается сессионный ключ — случайное целое число К такое, что

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.

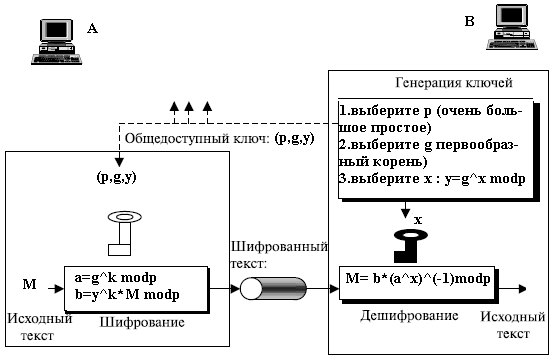
Нетрудно увидеть, что длина шифротекста в схеме Эль-Гамаля длиннее исходного сообщения M вдвое.

## **Расшифрование**



# **3. Блок-схема алгоритма**

**Схема шифрования:**



# **4.Пример работы программы**

A picture containing graphical user interface

Description automatically generated

# **5. Код программы**

import random  
  
class PrivateKey(object):  
 def \_\_init\_\_(self, p=None, g=None, x=None, numBits=0):  
 self.p = p  
 self.g = g  
 self.x = x  
 self.numBits = numBits  
  
class PublicKey(object):  
 def \_\_init\_\_(self, p=None, g=None, y=None, numBits=0):  
 self.p = p  
 self.g = g  
 self.y = y  
 self.numBits = numBits  
  
def gcd( a, b ):  
 while b != 0:  
 c = a % b  
 a = b  
 b = c  
 return a  
  
def modexp( base, exp, modulus ):  
 return pow(base, exp, modulus)  
  
def solovayStrassen( num, confidence ):  
 for i in range(confidence):  
 a = random.randint( 1, num-1 )  
  
 if gcd( a, num ) > 1:  
 return False  
  
 if not jacobi( a, num ) % num == modexp ( a, (num-1)//2, num ):  
 return False  
  
 return True  
  
def jacobi( a, n ):  
 if a == 0:  
 if n == 1:  
 return 1  
 else:  
 return 0  
 elif a == -1:  
 if n % 2 == 0:  
 return 1  
 else:  
 return -1  
 elif a == 1:  
 return 1  
 elif a == 2:  
 if n % 8 == 1 or n % 8 == 7:  
 return 1  
 elif n % 8 == 3 or n % 8 == 5:  
 return -1  
 elif a >= n:  
 return jacobi( a%n, n)  
 elif a%2 == 0:  
 return jacobi(2, n)\*jacobi(a//2, n)  
 else:  
 if a % 4 == 3 and n%4 == 3:  
 return -1 \* jacobi( n, a)  
 else:  
 return jacobi(n, a )  
  
def find\_primitive\_root( p ):  
 if p == 2:  
 return 1  
 p1 = 2  
 p2 = (p-1) // p1  
 while( 1 ):  
 g = random.randint( 2, p-1 )  
 if not (modexp( g, (p-1)//p1, p ) == 1):  
 if not modexp( g, (p-1)//p2, p ) == 1:  
 return g  
  
def find\_prime(numBits, confidence):  
 while(1):  
 p = random.randint( 2\*\*(numBits-2), 2\*\*(numBits-1) )  
 while( not solovayStrassen(p, confidence) ):  
 p = random.randint( 2\*\*(numBits-2), 2\*\*(numBits-1) )  
 while( p % 2 == 0 ):  
 p = random.randint(2\*\*(numBits-2), 2\*\*(numBits-1))  
 p = p \* 2 + 1  
 if solovayStrassen(p, confidence):  
 return p  
  
def generate\_keys(numBits=256, confidence=32):  
 p = find\_prime(numBits, confidence)  
 g = find\_primitive\_root(p)  
 g = modexp( g, 2, p )  
 x = random.randint( 2, p - 1)  
 y = modexp( g, x, p )  
  
 publicKey = PublicKey(p, g, y, numBits)  
 privateKey = PrivateKey(p, g, x, numBits)  
  
 return {'privateKey': privateKey, 'publicKey': publicKey}  
  
def encrypt(key, plainText):  
 z = bytearray(plainText, 'utf-8')  
 cipher\_pairs = []  
 for i in z:  
 k = random.randint(2, key.p - 1)  
 a = modexp( key.g, k, key.p )  
 b = (i\*modexp( key.y, k, key.p)) % key.p  
 cipher\_pairs.append( [a, b] )  
  
 encryptedStr = ""  
 for pair in cipher\_pairs:  
 encryptedStr += str(pair[0]) + ' ' + str(pair[1]) + ' '  
  
 return encryptedStr  
  
def decrypt(key, cipher):  
 plaintext = []  
  
 cipherArray = cipher.split()  
 if (not len(cipherArray) % 2 == 0):  
 return "Malformed Cipher Text"  
 for i in range(0, len(cipherArray), 2):  
 a = int(cipherArray[i])  
 b = int(cipherArray[i+1])  
 s = modexp( a, key.x, key.p )  
 plain = (b\*modexp( s, key.p-2, key.p)) % key.p  
 plaintext.append( plain )  
  
 decryptedText = bytearray(plaintext).decode('utf-8')  
  
 return decryptedText  
  
def main():  
 keys = generate\_keys()  
 priv = keys['privateKey']  
 pub = keys['publicKey']  
 message = "Hello World!"  
 print("Message: %r\n" % message)  
  
 cipher = encrypt(pub, message)  
 print("Encrypted: %r\n" % cipher)  
  
 plain = decrypt(priv, cipher)  
 print("Decrypted: %r" % plain)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

# **6. Вывод**

В настоящее время криптосистемы с открытым ключом считаются наиболее перспективными. К ним относится и схема Эль-Гамаля, криптостойкость которой основана на вычислительной сложности проблемы дискретного логарифмирования, где по известным p, g и y требуется вычислить x, удовлетворяющий сравнению:

*y* ≡ *gx*(*mod p*)

В ходе написания лабораторной работы были изучены алгоритмы шифрования и дешифрования Эль-Гамаля, а также написаны их программные реализации.