Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

КАФЕДРА ИНФОРМАТИКИ

Отчёт по лабораторной работе №4

По дисциплине «Методы защиты информации»

По теме «Асимметричная криптография. Алгоритм Эль-Гамаля»

Выполнила:

Студентка гр. 653501

Д. А. Гуринович

Проверил:

В. С. Артемьев

Минск 2019

Содержание

[1. Постановка задачи](#_2et92p0)

2. Краткие теоретические сведения

3. Схема алгоритма

4[. Результаты выполнения](#_tyjcwt)

5[. Вывод](#_3dy6vkm)

[ПРИЛОЖЕНИЕ 1. Исходный код программы.](#_1t3h5sf)

# 1. Постановка задачи

В современном мире остро стоит вопрос о безопасности. Обеспечение безопасности является важным аспектом деятельности любой компании. Для обеспечения безопасности используется множество различных средств, как аппаратных, так и программных. Одним из таких средств является шифрование данных.

Шифрование — обратимое преобразование информации в целях сокрытия от неавторизованных лиц, с предоставлением, в это же время, авторизованным пользователям доступа к ней. Главным образом, шифрование служит задачей соблюдения конфиденциальности передаваемой информации.

В ходе данной лабораторной работы необходимо изучить теоретическую часть об алгоритме шифрования Эль-Гамаля, который необходимо использовать для шифрования данных, и на ее основании создать приложение, реализующее алгоритм Эль-Гамаля.

**2. Краткие теоретические 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).

Схема была предложена Тахером Эль-Гамалем в 1985 году. Эль-Гамаль разработал один из вариантов алгоритма Диффи-Хеллмана. Он усовершенствовал систему Диффи-Хеллмана и получил два алгоритма, которые использовались для шифрования и для обеспечения аутентификации. В отличие от RSA алгоритм Эль-Гамаля не был запатентован и, поэтому, стал более дешевой альтернативой, так как не требовалась оплата взносов за лицензию. Считается, что алгоритм попадает под действие патента Диффи-Хеллмана.

**Алгоритм создания 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.

**Алгоритм шифрования:**

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

1. Выбирается сессионный ключ —случайное целое число k такое, что 1<k<p-1.

2. Вычисляются числа a=g^k (mod p) и b=y^(k)\*M (mod p).

3. Пара чисел (a,b) является шифротекстом.

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

**Алгоритм расшифрования:**

Зная закрытый ключ x, исходное сообщение можно вычислить из шифротекста (a,b) по формуле:

M=b(a^x)^(-1) (mod p)

При этом нетрудно проверить, что (a^x)^(-1) (mod p) =g^(-kx) (mod p)

и поэтому b(a^x)^(-1)=(y^k)M)g^(-xk) = (g^(xk)M)g^(-xk) = M (mod p).

Для практических вычислений больше подходит следующая формула: M=b(a^x)^(-1)=ba^(p-1-x) (mod p)

**3. Схема алгоритма**

# 

# 4. Результаты выполнения

# 

Рисунок 4.1. Исходный файл

# 

Рисунок 4.2. Результат работы программы

**5. Вывод**

В рамках данной лабораторной работы был рассмотрен алгоритм для асимметричного шифрования «Алгоритм Эль-Гамаля».

Для демонстрации работы описанных выше алгоритмов было разработано и реализовано приложение для шифрования и дешифрования файлов с использованием указанного выше алгоритма.

# ПРИЛОЖЕНИЕ 1. ИСХОДНЫЙ ТЕКСТ ПРОГРАММЫ

import random

from fractions import gcd

# GENERATE KEYS

def generate\_keys(bits\_number=256, test\_accuracy=32):

p = find\_prime(bits\_number, test\_accuracy)

g = pow(find\_primitive\_root(p), 2, p)

x = random.randint(1, (p - 1) // 2 )

y = pow(g, x, p)

return (p, g, y), (p, g, x)

#https://ru.wikipedia.org/wiki/Тест\_Соловея\_—\_Штрассена

def is\_prime(number, test\_accuracy):

for \_ in range(test\_accuracy):

a = random.randint(2, number - 1)

if gcd(a, number) > 1:

return False

if not pow(a, (number - 1) // 2, number) == jacobi(a, number) % number:

return False

return True

def jacobi(a, n):

if a == 0:

if n == 1:

return 1

return 0

if a == -1:

if n % 2 == 0:

return 1

return -1

if a == 1:

return 1

if a == 2:

if n % 8 == 1 or n % 8 == 7:

return 1

if n % 8 == 3 or n % 8 == 5:

return -1

if a >= n:

return jacobi(a % n, n)

if a % 2 == 0:

return jacobi(2, n) \* jacobi(a // 2, n)

if a % 4 == 3 and n % 4 == 3:

return -1 \* jacobi(n, a)

return jacobi(n, a)

def find\_primitive\_root(p):

if p == 2:

return 1

p1 = 2

p2 = (p - 1) // p1

while True:

g = random.randint(2, p - 1)

if not pow(g, (p - 1) // p1, p) == 1 and not pow(g, (p - 1) // p2, p) == 1:

return g

def find\_prime(bits\_number, test\_accuracy):

while True:

while True:

p = 0

while p % 2 == 0:

p = random.randint(2 \*\* (bits\_number - 2), 2 \*\* (bits\_number - 1))

if is\_prime(p, test\_accuracy):

break

p = p \* 2 + 1

if is\_prime(p, test\_accuracy):

return p

# ENCRYPT

def encrypt(text, key, bits\_number=256):

z = encode(text, bits\_number)

p, g, y = key

cipher\_pairs = []

#i is an integer in z

for i in z:

k = random.randint(0, p)

a = pow(g, k, p)

b = (i \* pow(y, k, p)) % p

cipher\_pairs.append([a, b])

encrypted\_string = ""

for pair in cipher\_pairs:

encrypted\_string += str(pair[0]) + ' ' + str(pair[1]) + ' '

return encrypted\_string

def encode(text, bits\_number):

byte\_array = bytearray(text, 'utf-16')

encoded\_bytes = []

bytes\_number = bits\_number // 8

j = -1 \* bytes\_number

for i in range(len(byte\_array)):

if i % bytes\_number == 0:

j += bytes\_number

encoded\_bytes.append(0)

encoded\_bytes[j//bytes\_number] += byte\_array[i] \* (2 \*\* (8 \* (i % bytes\_number)))

return encoded\_bytes

# DECRYPT

def decrypt(text, key, bits\_number=256):

plain\_text = []

p, g, x = key

cipher\_array = text.split()

if not len(cipher\_array) % 2 == 0:

return "Malformed Cipher Text"

for i in range(0, len(cipher\_array), 2):

a = int(cipher\_array[i])

b = int(cipher\_array[i+1])

s = pow(a, x, p)

plain = (b \* pow(s, p - 2, p)) % p

plain\_text.append(plain)

decryptedText = decode(plain\_text, bits\_number)

decryptedText = "".join([ch for ch in decryptedText if ch != '\x00'])

return decryptedText

def decode(text, bits\_number):

bytes\_array = []

bytes\_number = bits\_number // 8

for byte in text:

for i in range(bytes\_number):

temp = byte

for j in range(i + 1, bytes\_number):

temp = temp % (2 \*\* (8 \* j))

letter = temp // (2 \*\* (8 \* i))

bytes\_array.append(letter)

byte = byte - (letter \* (2 \*\* (8 \* i)))

decodedText = bytearray(b for b in bytes\_array).decode('utf-16')

return decodedText