Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Методы защиты информации

ОТЧЕТ

к лабораторной работе №7

на тему

**Криптография с использованием**

**эллиптических кривых.**

Выполнил Д.С. Шевцова

Проверил Герчик А. В.

Минск 2024

СОДЕРЖАНИЕ

[Введение 3](#_Toc158758843)

[1 Краткие теоретические сведения 4](#_Toc158758844)

[2 Результаты выполнения лабораторной работы](#_Toc158758845) 6

[Заключение](#_Toc158758846)

[Приложение А](#_Toc158758848) [(обязательное)](#_Toc158758849) [Листинг кода](#_Toc158758850) 8

## ВВЕДЕНИЕ

В данной лабораторной работе нужно реализовать схему шифрования (дешифрования) для аналога алгоритма Эль-Гамаля на основе эллиптических кривых

## 1 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Криптография с использованием эллиптических кривых представляет собой мощный метод обеспечения безопасности данных, который основывается на математических свойствах эллиптических кривых и теории чисел. Этот подход широко используется в различных криптографических протоколах, включая шифрование, цифровую подпись и обмен ключами. Основное преимущество криптографии на основе эллиптических кривых заключается в том, что она позволяет добиться высокого уровня безопасности при использовании коротких ключей, что делает её более эффективной по сравнению с традиционными методами, такими как RSA.

Эллиптические кривые определяются уравнением вида y2=x3+ax+b, где параметры a и b выбираются таким образом, чтобы кривая не имела вырожденных точек. Важной особенностью этих кривых является наличие групповой структуры, которая позволяет проводить операции сложения и умножения, что является основой для построения криптографических схем. Асимметричная криптография, использующая эллиптические кривые, опирается на создание пары ключей: приватного, который должен оставаться в секрете, и публичного, доступного для всех. Приватный ключ используется для создания подписи или шифрования данных, в то время как публичный ключ – для проверки подписи или расшифровки данных.

Процесс генерации ключей в криптографии с использованием эллиптических кривых включает в себя выбор случайного приватного ключа и вычисление соответствующего публичного ключа через умножение базовой точки на приватный ключ. Базовая точка выбирается заранее и известна всем участникам системы. Далее следует этап шифрования или создания подписи. При шифровании сообщения используется случайное число для обеспечения уникальности шифрования каждой отдельной сессии, что делает атаки на основе анализа шифрованного текста менее эффективными. При создании цифровой подписи оригинальное сообщение сначала обрабатывается хеш-функцией, которая преобразует его в фиксированный размер хэш. Этот хэш затем шифруется с использованием приватного ключа, формируя подпись, которая передается вместе с сообщением.

Процесс проверки подписи начинается с получения сообщения и его подписи. Получатель сначала вычисляет хэш полученного сообщения и затем расшифровывает подпись с использованием публичного ключа отправителя, получая оригинальный хэш. Сравнив два хэша, полученный из сообщения и расшифрованный из подписи, получатель может установить целостность и подлинность сообщения. Если хэши совпадают, это подтверждает, что сообщение не было изменено и действительно отправлено тем, кто подписал его.

Криптография с использованием эллиптических кривых находит применение в различных областях, включая электронные платежи, безопасный обмен данными и создание цифровых подписей. Она стала ключевым компонентом современных систем безопасности, обеспечивая надежную защиту и аутентификацию данных. Благодаря своей эффективности и высокому уровню безопасности, криптография на основе эллиптических кривых играет важную роль в поддержании доверия и защищенности в цифровом мире.

Ниже на рисунке 1.1 представлена блок схема алгоритма.

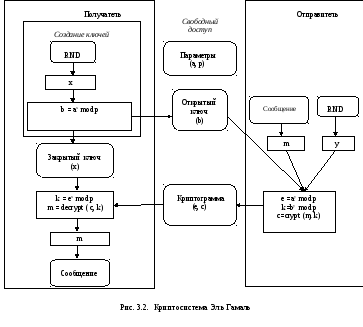


Рисунок 1.1 – Блок схема алгоритма

# 2 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

В ходе выполнения лабораторной работы мы шифруем текст с помощью алгоритма Эль-Гамаля. На рисунке 2.1 представлен текст, что мы будем шифровать.

Получатель использует свой приватный ключ для расшифровки зашифрованных точек, восстанавливая оригинальные точки сообщения.

Далее на рисунке 2.2 мы видим результат пар точек, а на рисунке 2.3 уже расшифрованное сообщение, что совпадает с исходным.

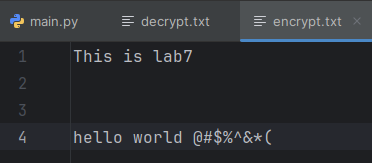


Рисунок 2.1 – Исходный текст

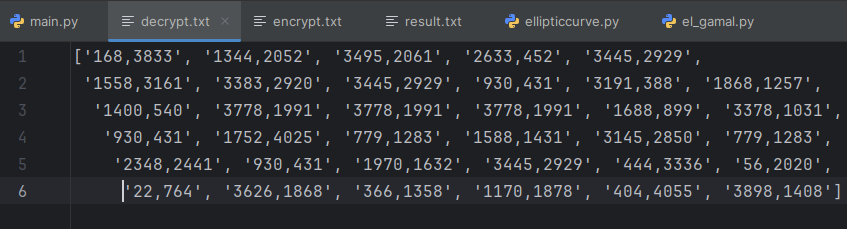


Рисунок 2.2 – Пары точек, полученные в ходе шифрования

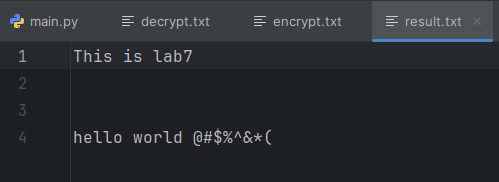


Рисунок 2.3 – Результат расшифровывания

## ЗАКЛЮЧЕНИЕ

В ходе выполнения данной лабораторной работы была написана реализация схемы шифрования (дешифрования) для аналога алгоритма Эль-Гамаля на основе эллиптических кривых.

# **ПРИЛОЖЕНИЕ А**

## (обязательное)

## Листинг кода

el\_gamal.py

from ellipticcurve import EllipticCurve, Point

import random

import math

class Gamal:

# Параметры кривых

def \_\_init\_\_(self, curve\_a, curve\_b, prime, random\_k, sender\_random, private\_key, encoding='ascii'):

self.private\_key = private\_key

self.prime\_number = prime

self.sender\_random\_value = sender\_random

self.elliptic\_curve = EllipticCurve(curve\_a, curve\_b, prime)

self.random\_k = random\_k

self.encoding = encoding

# Генерируем случайную точку для базы

self.\_\_generateBase()

# Генерируем публичный ключ

self.public\_key = self.elliptic\_curve.multiplication(self.base\_point, private\_key)

def encrypt(self, plaintext):

"""Входное сообщение должно быть массивом точек"""

ciphertext\_points = []

encoded\_points = self.\_\_encode\_plaintext(plaintext)

for point in encoded\_points:

ciphertext\_points.append(self.\_\_encrypt\_point(point))

return ciphertext\_points

def decrypt(self, cipherpoints):

plaintext\_points = []

for point\_pair in cipherpoints:

plaintext\_points.append(self.\_\_decrypt\_point\_pair(point\_pair))

return self.\_\_decode\_points(plaintext\_points)

def \_\_generateBase(self):

"""Генерируем базовую начальную точку"""

found = False

while not found:

x = random.choice(range(1, self.prime\_number))

y\_list = self.elliptic\_curve.generatePoints(x)

if len(y\_list) > 0:

found = True

self.base\_point = Point(x, random.choice(y\_list))

def \_\_encode\_plaintext(self, plaintext):

"""переводим входное сообщение в массив точек"""

# добавляем русский язык

byte\_array = bytearray(plaintext, "1251")

chunks\_list = []

for byte in byte\_array:

for j in range(self.random\_k):

# решаем для y

x = byte \* self.random\_k + j

list\_of\_y = self.elliptic\_curve.generatePoints(x)

if len(list\_of\_y) > 0:

y = random.choice(list\_of\_y)

chunks\_list.append(Point(x, y))

break

return chunks\_list

def \_\_decode\_points(self, plaintext\_points):

"""Декодируем наши точки"""

proto = ""

for point in plaintext\_points:

dec\_representation = int(math.floor(point.x / self.random\_k))

hex\_representation = hex(dec\_representation)[2:]

if len(hex\_representation) == 1:

hex\_representation = "0" + hex\_representation

proto += bytes.fromhex(hex\_representation).decode("cp1251")

return proto

def \_\_encrypt\_point(self, point):

"""Здесь мы берём каждую точку, после чего шифруем её"""

first\_cipher = self.elliptic\_curve.multiplication(self.base\_point, self.sender\_random\_value)

second\_cipher = self.elliptic\_curve.addition(point, self.elliptic\_curve.multiplication(self.public\_key,

self.sender\_random\_value))

return [first\_cipher, second\_cipher]

def \_\_decrypt\_point\_pair(self, point\_pair):

"""Декодируем пару точек из метода выше"""

first\_block = self.elliptic\_curve.multiplication(point\_pair[0], self.private\_key)

first\_block\_inv = Point(first\_block.x, (-first\_block.y) % self.prime\_number)

return self.elliptic\_curve.addition(point\_pair[1], first\_block\_inv)

main.py

from el\_gamal import Gamal

import random

PRIVATE\_KEY = 100

prime\_number = 4093

SENDER\_RANDOM = random.choice(range(1, prime\_number))

CURVE\_A = 1

CURVE\_B = 6

ENCODING\_RANDOM\_K = 5

gamal = Gamal(

CURVE\_A,

CURVE\_B,

prime\_number,

ENCODING\_RANDOM\_K,

SENDER\_RANDOM,

PRIVATE\_KEY

)

def get\_points():

with open('encrypt.txt', 'r', encoding="utf-8") as file:

plain\_message = file.read()

cipherpoints = gamal.encrypt(plain\_message)

return cipherpoints

def main():

cipherpoints = get\_points()

with open('decrypt.txt', 'w', encoding="utf-8") as file:

file.write(str([str(pair[1]) for pair in cipherpoints]))

file.close()

plaintext = gamal.decrypt(cipherpoints)

with open('result.txt', 'w', encoding="utf-8") as file:

file.write(plaintext)

if \_\_name\_\_ == '\_\_main\_\_':

main()