Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Кафедра информатики

Отчёт

по лабораторной работе №7

**Криптография с использованием**

**эллиптических кривых**

|  |  |
| --- | --- |
| Выполнил:  студент группы 653502  Куликов А.Д. | Проверил:  Артемьев В.С. |

Минск 2019

# Постановка задачи

Создать программу, читающую данные из файла и шифрующую исходного текста (дешифрующую) их с помощью аналога алгоритма Диффи-Хеллмана.

# Теоретическая справка

## Эллиптические кривые

Эллиптическая кривая над полем K — неособая кубическая кривая на проективной плоскости над K^ (алгебраическим замыканием поля K), задаваемая уравнением 3-й степени с коэффициентами из поля K и «точкой на бесконечности». В подходящих аффинных координатах её уравнение приводится к виду:



В канонической форме если характеристика поля K не равна 2 или 3, то уравнение с помощью замены координат приводится к канонической форме (форме Вейерштрасса):



Если характеристика К = 3, то каноническим видом уравнения является вид:



Если характеристика K=2, то уравнение приводится к одному из видов:

 - суперсингулярные кривые

или

 - несуперсингулярные кривые

Особое удобство суперсингулярных эллиптических кривых в том, что для них легко вычислить порядок, в то время как вычисление порядка несуперсингулярных кривых вызывает трудности. Суперсингулярные кривые особенно удобны для создания самодельной ЕСС-криптосистемы. Для их использования можно обойтись без трудоёмкой процедуры вычисления порядка.

Точки эллиптической кривой над конечным полем представляют собой группу. И для этой группы определена операция сложения. Соответственно, можно представить умножение числа k на точку G как G+G+..+G с k слагаемыми.

Теперь можно представить, что имеется сообщение M представленное в виде целого числа. Мы можем зашифровать его используя выражение  
C=M\*G. Вопрос в том, насколько сложно восстановить M зная параметры кривой E(a,b), шифротекст С и точку G.

Данная задача называется дискретным логарифмом на эллиптической кривой и не имеет быстрого решения. Более того, считается, что задача дискретного логарифма на эллиптической кривой является более трудной для решения, чем задача дискретного логарифмирования в конечных полях.

## Алгоритм Диффи-Хэллмана

В данном описании криптографического протокола, аналогичный известному протоколу Диффи-Хэллмана для установления защищенной связи два пользователя А и В совместно выбирают эллиптическую кривую Е и точку Р на ней.

1 этап. Пользователь А выбирает свое секретное целое число a (b — большое число) и вычисляет произведение. Далее А пересылает вычисленное значение получателю В. Пользователь В генерирует свое секретное большое число b (b — целое число) и вычисляет произведение. В пересылает его получателю А. При этом параметры самой кривой, координаты точки на ней и значения произведений являются открытыми и могут передаваться по незащищенным каналам связи. Предполагается, что злоумышленник может получить оба этих значения, но не модифицировать их.

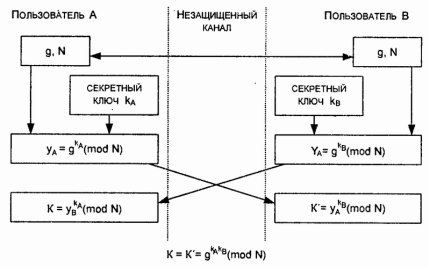
2 этап. А на основе имеющегося у пользователя числа и полученного по сети вычисляет значение В на основе имеющегося у пользователя числа и полученного по сети вычисляет значение В силу свойств операции умножения на число. Таким образом, оба пользователя получат общее секретное значение (координаты точки), которое они могут использовать для получения ключа шифрования.

Злоумышленнику для восстановления ключа потребуется решить сложную с вычислительной точки зрения задачу определения и  по известным  данным. Если эта проблема имеет эффективное решение, то и проблема Диффи-Хеллмана для эллиптических также легко решаема и рассмотренный протокол не имеет смысла. В то же время гипотеза об эквивалентности проблем дискретного логарифма и проблемы Диффи-Хеллмана для эллиптических кривых не доказана. Протокол Диффи-Хеллмана отлично противостоит пассивному нападению, но в случае реализации атаки «человек посередине» он не устоит. В самом деле, Атакующий заменяет сообщения переговоров о ключе на свои собственные и таким образом получает два ключа — свой для каждого из законных участников протокола. Далее он может перешифровывать переписку между участниками, своим ключом для каждого, и таким образом ознакомиться с их сообщениями, оставаясь незамеченным.

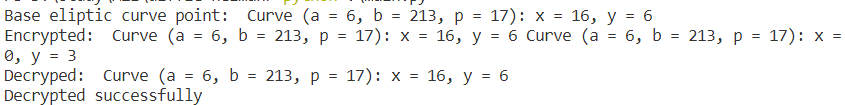
## Безопасность алгоритма на основе эллиптических кривых

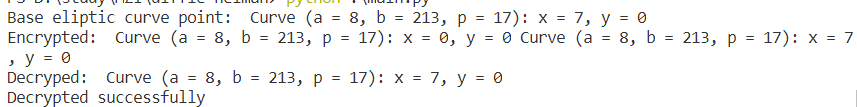
Безопасность, обеспечиваемая криптографическим подходом на основе эллиптических кривых, зависит от того, насколько трудной для решения оказывается задача определения {\displaystyle k}k по данным {\displaystyle kP}kP и {\displaystyle P}P. Эту задачу обычно называют проблемой логарифмирования на эллиптической кривой. Наиболее быстрым из известных на сегодня методов логарифмирования на эллиптической кривой является так называемый {\displaystyle p}p-метод Полларда. В таблице (см. ниже) сравнивается эффективность этого метода и метод разложения на простые множители с помощью решета в поле чисел общего вида. Из нее видно, что по сравнению с RSA в случае применения методов криптографии на эллиптических кривых примерно тот же уровень защиты достигается со значительно меньшими значениями длины ключей.

# Блок-схема алгоритма



# Результат исполнения программы





# Вывод

В ходе выполнения лабораторной работы были изучены алгоритмы шифрования и дешифрования для аналога алгоритма Диффи-Хеллмана на основе эллиптических кривых, а также написаны их программные реализации. Были получены навыки усложнения и увеличения криптостойкости алгоритма Диффи-Хеллмана на основе эллиптических кривых, а также изучены модификации и режимы работы алгоритма Диффи-Хеллмана на основе эллиптических кривых.

# Приложение

**hellman.py**

class DiffieHellman:

    def \_\_init\_\_(self, point):

        self.point = point

        self.n = point.order

    def get\_public\_key(self, private\_key):

        return self.point \* private\_key

    def encrypt(self, data\_point, public\_key, q):

        return self.point \* q, data\_point + public\_key \* q

    def decrypt(self, encrypted\_points, private\_key):

        return encrypted\_points[1] + -(encrypted\_points[0] \* private\_key)

**main.py**

import random

from ec import ECPoint, EllipticCurve

from hellman import DiffieHellman

EC\_A = 8

EC\_B = 213

EC\_P = 17

EC\_POINT\_COORD = 18

def main():

    curve = EllipticCurve(EC\_A, EC\_B, EC\_P)

    base\_point = curve.create\_point(EC\_POINT\_COORD)

    diffie = DiffieHellman(base\_point)

    private\_key = random.randint(1, 1000)

    public\_key = diffie.get\_public\_key(private\_key)

    q = random.randint(1, 100)

    to\_encrypt = base\_point \* 2

    encrypted\_points = diffie.encrypt(to\_encrypt, public\_key, q)

    decrypted = diffie.decrypt(encrypted\_points, private\_key)

    print('Base eliptic curve point: ', to\_encrypt)

    print('Encrypted: ', encrypted\_points[0], encrypted\_points[1])

    print('Decryped: ', decrypted)

    if decrypted == to\_encrypt:

        print('Decrypted successfully')

    else:

        print('Something went wrong')

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**ec.py**

from utils import mod\_inverse, mod\_sqrt

class EllipticCurve:

    def \_\_init\_\_(self, a, b, p):

        assert (4 \* a \*\* 3 + 27 \* b \*\* 2) % p != 0

        self.a = a

        self.b = b

        self.p = p

    def create\_point(self, x):

        ysq = (x \*\* 3 + self.a \* x + self.b) % self.p

        \_, my = mod\_sqrt(ysq, self.p)

        return ECPoint(self, x, my)

    def \_\_str\_\_(self):

        return 'a = {}, b = {}, p = {}'.format(self.a, self.b, self.p)

class ECPoint:

    def \_\_init\_\_(self, curve, x, y):

        self.curve = curve

        self.x = x

        self.y = y

    @property

    def check\_zero(self):

        return self.x == 0 and self.y == 0

    @property

    def order(self):

        for i in range(1, self.curve.p + 1):

            if (self \* i).check\_zero:

                return i

    def \_\_neg\_\_(self):

        return ECPoint(self.curve, self.x, -self.y % self.curve.p)

    def \_\_add\_\_(self, other):

        if (self.x, self.y) == (0, 0):

            return other

        if (other.x, other.y) == (0, 0):

            return ECPoint(self.curve, self.x, self.y)

        if self.x == other.x and (self.y != other.y or self.y == 0):

            return ECPoint(self.curve, 0, 0)

        if self.x == other.x:

            l = (3 \* self.x \*\* 2 + self.curve.a) \* \

                mod\_inverse(2 \* self.y, self.curve.p) % self.curve.p

        else:

            l = (other.y - self.y) \* mod\_inverse(other.x -

                                                 self.x, self.curve.p) % self.curve.p

        x = (l \* l - self.x - other.x) % self.curve.p

        y = (l \* (self.x - x) - self.y) % self.curve.p

        return ECPoint(self.curve, x, y)

    def \_\_mul\_\_(self, number):

        result = ECPoint(self.curve, 0, 0)

        tmp = ECPoint(self.curve, self.x, self.y)

        while 0 < number:

            if number & 1 == 1:

                result += tmp

            number, tmp = number >> 1, tmp + tmp

        return result

    def \_\_eq\_\_(self, other):

        return (self.curve, self.x, self.y) == (other.curve, other.x, other.y)

    def \_\_str\_\_(self):

        return 'Curve ({}): x = {}, y = {}'.format(self.curve, self.x, self.y)