Лабораторная работа № 4 по курсу "Криптография"

Выполнил студент группы М8О-308 МАИ *Скворцов Александр.*

Условие

Подобрать такую эллиптическую кривую над конечным простым полем порядка p, порядок точки которой полным перебором находится за 10 минут на ПК. Упомянуть в отчёте какие алгоритмы и теоремы существуют для облегчения и ускорения решения задачи полного перебора.

Оборудование студента

Процессор Intel Core i5-6200U 2 @ 2.3GHz, память: 4096Gb. ОС Windows 10, разрядность системы 64.

Теоретические сведения

Эллиптическая кривая над полем K — множество точек, описываемых уравнением:

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

Если характеристика поля не равна 2 или 3, то то уравнение с помощью замены координат приводится к форме Вейерштрасса:

$$y^2 = x^3 + ax + b$$

Определение эллиптической кривой также требует, чтобы кривая не имела особых точек. Алгебраически, достаточно проверить, что дискриминант

$$\Delta = -16(4a^3 + 27b^2)$$

не равен нулю. Есди дискриминант положительный, то график имеет две связные компоненты, а если отрицательный, то одну компоненту.

Для эллиптических кривых можно определить группу:

1. элементы группы являются точками эллиптической кривой
2. единичный элемент — это бесконечно удалённая точка 0
3. обратная величина точки P — это точка, симметричная относительно оси x
4. сложение задаётся следующим правилом: сумма трёх ненулевых точек P, Q и R, лежащих на одной прямой, будет равна $P + Q + R = 0$.

В криптографии используются эллиптические кривые над конечными полемя, например множество целых по модулю p.

$$y^2 = x^3 + ax + b \ (mod\ p)$$

Две точки $P(x_1, y_1)$ и $Q(x_2, y_2)$ эллиптической кривой, определенной над конечным полем $\mathbb{Z}_p$ складываются по правилу:

$$\begin{aligned}
&P + Q = R \equiv (x_3, y_3),\\
&x_3 = \lambda^2 - x_1 - x_2 \pmod p,\\
&y_3 = -y_1 + \lambda(x_1 - x_3) \pmod p
\end{aligned}$$

где

$$\lambda = \begin{cases} \dfrac{y_2 - y_1}{x_2 - x_1} \pmod p, & P \neq Q,\\ \dfrac{3x_1^2 + a}{2y_1} \pmod p. & P = Q. \end{cases}$$

Можно определить умножение точки на число:

$$nP = \underbrace{P + P + \cdots + P}_{n \text{ раз}}$$

Порядком точки называется такое число n, что $nP = 0$.

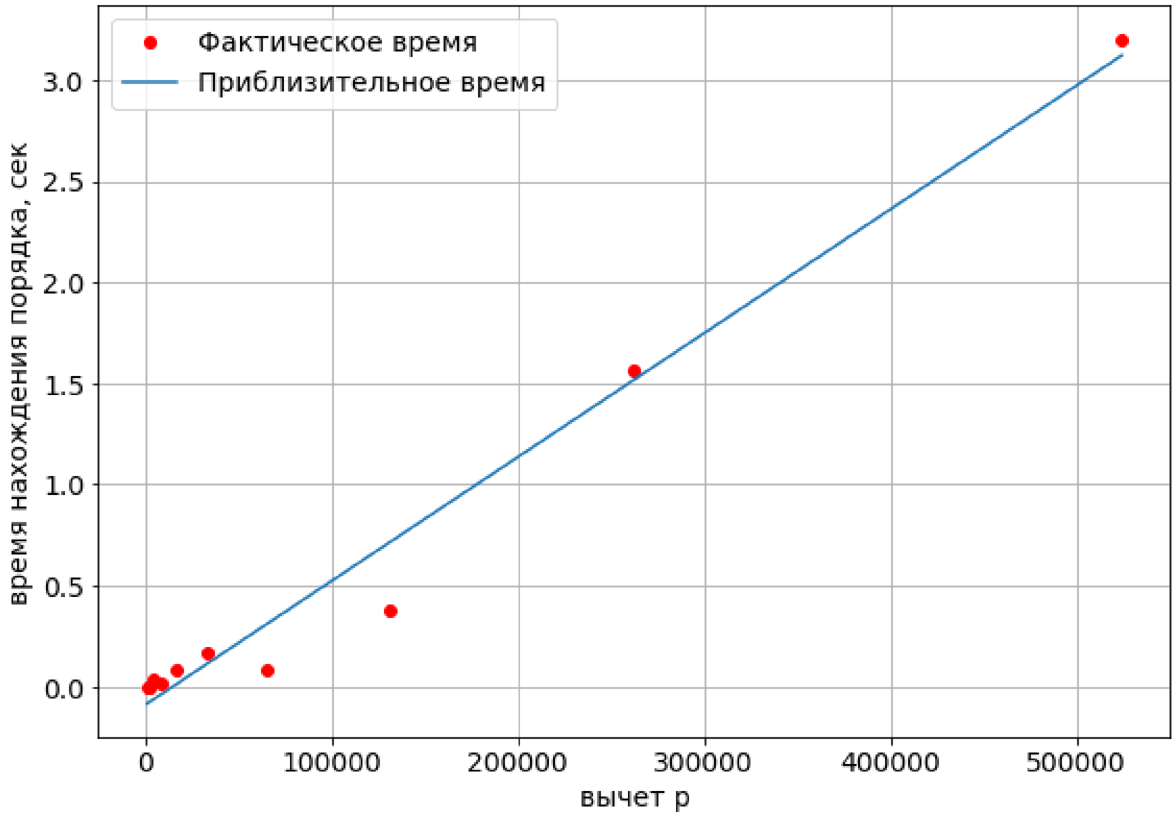
Метод решения

В качестве эллиптической кривой я выбрал кривую в форме Вейерштрасса:

$$y^2 = x^2 + ax + b \ (mod\ p)$$

Если зафиксировать параметры a и b, то порядок точки P будет зависить только от вычета p. Пусть $a = 2$, а $b = 4$. В этом случае точка $P = (0, 2)$ будет всегда принадлежать кривой для любого p. Тогда будем считать порядок этой точки.

Выясним, как меняется время нахождения порядка в зависимости от вычета p:



Из графика видно, что время растет линейно. На основе фактических данных с помощью метода наименьших квадратов можно подобрать аппроксимирующий полином первой степени для приблизительного вычисления вычета p, которому соответствует время 600 секунд.

Таким методом я подобрал $p = 84203143$, а для вычисления мне потребовалась 651 секунда. Итоговый ответ:

$$y^2 = x^3 + 2x + 4 \ (mod\ 84203143)$$

Выводы

С помощью эллиптических кривых можно построить асимметричную криптосистему, где закрытым ключом является число d, а открытым ключом является точка $H = dG$. При этом, даже если мы знаем H и G, то поиск закрытого ключа d является «сложной» задачей. По моим подсчетам, если использовать в качестве вычета значение близкое к max_uint32, то перебор всех значений займет около 18 дней, а это только 4 байта, современные ключи гораздо длиннее.

Стоит отметить, что существуют куда более быстрые алгоритмы нахождения числа d, например алгоритм Шенкса и ро-метод Полларда, которые работают за $O(\sqrt{n})$, в то время как полный перебор осуществляется за $O(n)$.

Код программы

```
import matplotlib.pyplot as plt
import numpy as np
import time

def extended_euclidean_algorithm(a, b):
    s, old_s = 0, 1
    t, old_t = 1, 0
    r, old_r = b, a

    while r != 0:
        quotient = old_r // r
        old_r, r = r, old_r - quotient * r
        old_s, s = s, old_s - quotient * s
        old_t, t = t, old_t - quotient * t

    return old_r, old_s, old_t


def inverse_of(n, p):
    gcd, x, y = extended_euclidean_algorithm(n, p)
    assert (n * x + p * y) % p == gcd

    if gcd != 1:
        raise ValueError(
            '{} has no multiplicative inverse '
            'modulo {}'.format(n, p))
    else:
        return x % p

class Elliptic_curve:
    def __init__(self, a, b):
        self.a = a
        self.b = b

    def __call__(self, x):
        return x ** 3 + self.a * x + self.b

    def add(self, P1, P2, p):
        if P1 == 0:
            return P2
        if P2 == 0:
```

```
            return P1

        x1, y1 = P1
        x2, y2 = P2

        if x1 == x2 and y1 == -y2 % p:
            return 0

        if x1 == x2 and y1 == y2:
            k = (3 * x1 ** 2 + self.a) * inverse_of(2 * y1, p) % p
        else:
            k = (y2 - y1) * inverse_of(x2 - x1, p) % p

        x3 = (k ** 2 - x1 - x2) % p
        y3 = (-y1 + k * (x1 - x3)) % p

        return (x3, y3)

    def order_of_point(self, P, p):
        order = 1
        Q = P
        while Q:
            Q = self.add(P, Q, p)
            order += 1
        return order

if __name__ == '__main__':
    a = 2
    b = 4
    e = Elliptic_curve(a, b)
    P = (0, 2)

    primes = [1031, 2053, 4099, 8209, 16411, 32771, 65537, \
              131101, 262147, 524309]
    time_intervals = []

    for p in primes:
        start = time.process_time_ns()
        e.order_of_point(P, p)
        end = time.process_time_ns()
        time_intervals.append((end - start) / 10 ** 9)
```

```
poly = np.polyfit(primes, time_intervals, 1)
approx_line = np.poly1d(poly)

pp = (600 - poly[1]) / poly[0]
print(pp)

plt.figure(figsize=(10, 7))
plt.rcParams.update({'font.size': 14})

plt.plot(primes, time_intervals, 'ro')
plt.plot(primes, approx_line(primes))
plt.grid()

plt.show()
```