Отчет по лабораторной работе №3 по курсу «Криптография»

Выполнил Моисеенков Илья Павлович, М8О-308Б-19.

Задание

Подобрать такую эллиптическую кривую, порядок точки которой полным перебором находится за 10 минут на ПК. Упомянуть в отчёте результаты замеров работы программы, характеристики вычислителя. Также указать какие алгоритмы и/или теоремы существуют для облегчения и ускорения решения задачи полного перебора.

Рассмотреть для случая конечного простого поля Z_p.

Ход работы

Рассмотрим эллиптическую кривую $y^2 = x^3 + ax + b$, где $4a^3 + 27b^2 \neq 0$ в конечном простом поле $\mathbb{Z}_p$. Коэффициенты a и b я задал рандомно.

Для нахождения порядка эллиптической кривой я нахожу количество целочисленных точек из множества Z_p, принадлежащих заданной кривой. Эта операция выполняется за $O(p^2)$.

Затем я выбираю рандомную точку и начинаю искать ее порядок. Для этого я складываю точку саму с собой до тех пор, пока не получится точка (0, 0). Количество итераций, потребовавшихся на это, и есть порядок точки. Все вычисления проводятся по модулю p. Алгебраическое сложение двух точек в Z_p проводится по следующим правилам:

если $P = (x_P, y_P)$, $Q = (x_Q, y_Q)$ и $R = (x_R, y_R)$, то $P + Q = -R$ можно вычислить следующим способом:

$$\begin{aligned} x_R &= (m^2 - x_P - x_Q) \bmod p \\ y_R &= [y_P + m(x_R - x_P)] \bmod p \\ &= [y_Q + m(x_R - x_Q)] \bmod p \end{aligned}$$

Если $P \neq Q$, то наклон m принимает форму:

$$m = (y_P - y_Q)(x_P - x_Q)^{-1} \bmod p$$

Иначе, если $P = Q$, мы получаем:

$$m = (3x_P^2 + a)(2y_P)^{-1} \bmod p$$

(взято с https://habr.com/ru/post/335906/)

Будем рассматривать кривую $y^2 = x^3 + 2683x + 2399$ в поле Z_{32003}. Порядок поля был подобран экспериментально. Я брал разные простые числа и смотрел на время, которое требовалось для нахождения порядка точки в соответствующем поле.

Код

```
import time
import random
a = 2683
b = 2399


def elliptic_curve(x, y, p):
    """
    Check if point (x, y) is in elliptic curve y^2 = x^3 + ax + b in Z_p
    Returns true or false
    """
    return (y ** 2) % p == (x ** 3 + (a % p) * x + (b % p)) % p


def extended_euclidean_algorithm(a, b):
    """
    Returns (gcd, x, y): ax + by == gcd(a, b)
    Complexity: O(log b)
    stolen from https://habr.com/ru/post/335906/
    """
    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):
    """
    Returns m: (n * m) % p == 1
    stolen from https://habr.com/ru/post/335906/
    """
    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


def points_sum(A, B, p):
    """
    Get algebraic sum of two points A, B in Z_p
    Algorithm: https://habr.com/ru/post/335906/
    Returns R = (x_r, y_r) = A + B
    """
    if A == (0, 0):
        return B
    if B == (0, 0):
```

```
        return A
    if A[0] == B[0] and A[1] != B[1]:
        return 0, 0

    if A != B:
        m = ((A[1] - B[1]) * inverse_of(A[0] - B[0], p)) % p
    else:
        m = ((3 * A[0] ** 2 + a) * inverse_of(2 * A[1], p)) % p

    x_r = (m ** 2 - A[0] - A[1]) % p
    y_r = (A[1] + m * (x_r - A[0])) % p
    return x_r, -y_r % p


def get_point_order(point, p):
    """
    Get order of the point in Z_p
    """
    ans = 0
    found_point_order = False
    prev_point = point
    while not found_point_order:
        ans += 1
        point_sum = points_sum(point, prev_point, p)
        if point_sum == (0, 0):
            found_point_order = True
        else:
            prev_point = point
            point = point_sum
    return ans


if __name__ == '__main__':
    p = 32003

    start = time.time()
    points = []
    for x in range(p):
        for y in range(p):
            if elliptic_curve(x, y, p):
                points.append((x, y))

    curve_order = len(points)
    print('Curve order:', curve_order)

    point = random.choice(points)
    point_order = get_point_order(point, p)
    print('Point', point, 'order:', point_order)

    end = time.time()
    print('Time:', end - start, 's')
```

Результат

```
Curve order: 31986
Point (30080, 14559) order: 59998
Time: 856.1665139198303 s
```

Вычисления производились на процессоре Intel Core i5-8250U.

Выводы

Выполняя эту работу, я познакомился с криптографией на эллиптических кривых. Эллиптическая кривая — это кривая вида $y^2 = x^3 + ax + b$, заданная на поле Z_p. С виду кажется, что это уравнение совсем не похоже на что-то, что может использоваться в шифровании. Но это не так.

С помощью эллиптических кривых может производиться шифрование и передача сообщений с использованием открытых/закрытых ключей. Через эллиптические кривые можно реализовать электронные подписи. Используя эллиптические кривые, можно факторизовать большие числа (я уже сталкивался с таким алгоритмом в прошлой лабораторной).

В этой лабораторке я смог сам подобрать такую эллиптическую кривую, поиск порядка точек которой занимает довольно большое время. Я разобрался в деталях, как вычисляется порядок кривой и порядок ее точек. Но я вычислял порядок кривой за квадрат - простым перебором. Этот этап можно упростить, используя алгоритм Шуфа, который работает за полиномиальное время.