Московский авиационный институт
(национальный исследовательский университет)

Институт №8 «Информационные технологии и прикладная математика»

Кафедра 806 «Вычислительная математика и программирование»

Лабораторная работа №3 по курсу «Криптография»

Тема: Эллиптические кривые

Студент: А.С. Федоров

Преподаватель: А.В. Борисов

Группа: М8О-307Б-19

Дата:

Оценка:

Подпись:

Москва, 2022

Эллиптические кривые

Задача:

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

Описание

Криптография на эллиптических кривых строится на “сложности” задачи дискретного логарифмирования. Доказательства сложности данной задачи нет, в прочем, как и не найден алгоритм, позволяющий решать данную задачу за полиномиальное время. Сама кривая, а точнее ее параметры, порождают модулярную алгебру, в поле, образованном кривой и параметром p (по какому остатку брать), позволяют использовать точки кривой и их порядки для крипто протокола ECDH, а также для алгоритма цифровой подписи ECDSA. Решение задач определения порядка точки и порядка кривой являются ключевыми во взломе крипто протокола, основанного на данной технике.

Ход работы

Буду пытаться решить задачу определения всех точек кривой, определения их порядков и порядка кривой самым простым способом. Реализую класс эллиптической кривой и все необходимые вспомогательные функции. Программа написана на языке Python. Для ускорения разработки, готовые функции, описанные на сторонних интернет-ресурсах, были использованы намерено.

Вспомогательные функции:

```
def extended_euclidean_algorithm(a, b):
    """
    Возвращает кортеж из трёх элементов (gcd, x, y), такой, что
    a * x + b * y == gcd, где gcd - наибольший
    общий делитель a и b.

    В этой функции реализуется расширенный алгоритм
    Евклида и в худшем случае она выполняется O(log b).
    """
    s, old_s = 0, 1
    t, old_t = 1, 0
    r, old_r = b, a
```

```python
    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):
    """
    Возвращает обратную величину
    n по модулю p.

    Эта функция возвращает такое целое число m, при котором
    (n * m) % p == 1.
    """
    gcd, x, y = extended_euclidean_algorithm(n, p)
    assert (n * x + p * y) % p == gcd

    if gcd != 1:
        # Или n равно 0, или p не является простым.
        raise ValueError(
            '{} has no multiplicative inverse '
            'modulo {}'.format(n, p))
    else:
        return x % p

def sqrt(n, q):
    """sqrt on PN modulo: returns two numbers or exception if not exist
    >>> assert (sqrt(n, q)[0] ** 2) % q == n
    >>> assert (sqrt(n, q)[1] ** 2) % q == n
    """
    assert n < q
    for i in range(1, q):
        if i * i % q == n:
            return (i, q - i)
    return -1
```

Опишу собственный класс эллиптической кривой и наивные алгоритмы решения вышеописанных задач.

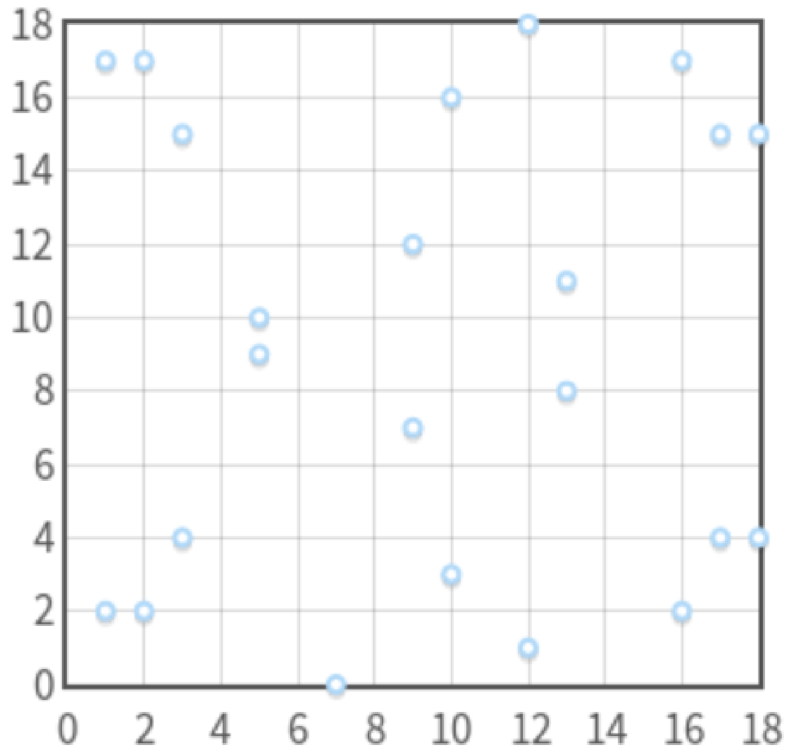
Класс EllipticCurve:

```python
class EllipticCurve:
    def __init__(self, p, a, b):
        assert 4 * a ** 3 + 27 * b ** 2 != 0
        self.p = p
        self.a = a
        self.b = b
        self.n = -1
        self.Points = []
        self.Orders = []
```

```
    def findGroupOrderAndPoints(self):
        self.n = 0
        for x in range(self.p):
            sqrtY = sqrt((x ** 3 + self.a * x + self.b) % self.p, self.p)
            if sqrtY != -1:
                self.n += 2
                y1, y2 = sqrtY
                self.Points.append([x, y1])
                self.Points.append([x, y2])
            elif (x ** 3 + self.a * x + self.b) % self.p == 0:
                self.Points.append([x, 0])

    def findPointOrder(self, P):
        curP = P.copy()
        cnt = 0
        for i in range(len(self.Points) + 1):
            cnt += 1
            curP = self.addition(curP, P)
            if curP[0] == P[0] and curP[1] == P[1]:
                return cnt

    def findAllPointOrgers(self):
        for p in self.Points:
            self.Orders.append(self.findPointOrder(p))

    def addition(self, P, Q):
        if P[0] == 0 and P[1] == 0:
            return Q
        if Q[0] == 0 and Q[1] == 0:
            return P
        R = [0, 0]
        if P[0] != Q[0]:
            s = ((P[1] - Q[1]) * inverse_of(P[0] - Q[0], self.p)) % self.p
            R[0] = (s ** 2 - P[0] - Q[0]) % self.p
            R[1] = (-P[1] + s * (P[0] - R[0])) % self.p
        else:
            if P[1] == -Q[1]:
                return np.array([0, 0])
            elif P[1] == Q[1] and Q[1] != 0:
                s = ((3 * P[0] ** 2 + self.a) * inverse_of(2 * P[1], self.p))
% self.p
                R[0] = (s ** 2 - 2 * P[0]) % self.p
                R[1] = (-P[1] + s * (P[0] - R[0])) % self.p
        return R
```

Работа программы была проверена на относительно простых примерах со статьи на Хабр, показанной на одной из лекций.

Сравнение результата с графической иллюстрацией:



Кривая $y^2 \equiv x^3 - 7x + 10 \pmod p$ с $p = 19$

Вычисленные точки и порядки:

```
curve.findGroupOrderAndPoints()
curve.Points
```

```
[[1, 2],
 [1, 17],
 [2, 2],
 [2, 17],
 [3, 4],
 [3, 15],
 [5, 9],
 [5, 10],
 [7, 0],
 [9, 7],
 [9, 12],
 [10, 3],
 [10, 16],
 [12, 1],
 [12, 18],
 [13, 8],
 [13, 11],
 [16, 2],
 [16, 17],
 [17, 4],
 [17, 15],
 [18, 4],
 [18, 15]]
```

```
curve.findAllPointOrgers()
curve.Orders
```

```
[7,
 7,
 23,
 23,
 23,
 23,
 11,
 11,
 1,
 7,
 7,
 23,
 23,
 2,
 2,
 11,
 11,
 5,
 5,
 23,
 23,
 3,
 3]
```

Полагая, что сложность решения задачи для кривой зависит только от размера поля вычетов. Так что параметры оставлю прежними за исключением p.

Возьму какое-нибудь достаточно большое простое число и взгляну на время вычисления. Для получения такого числа использую сайт-экспертную систему WolframAlpha.

Попробую число 19 387. Решение задачи заняло 17 минут, возьму простое число поменьше. Для p = 13259 задача была решена почти за 10 минут. Итоговые параметры подобранной кривой: p = 13259, a = -7, b = 10.

Характеристики вычислителя: Ноутбук ASUS FX570UD, процессор Intel® Core™ i5-8250U 1.6 ГГц CPU 1.60GHz 2.29GHz, память 12ГБ, 64-разрядная система.

Ускорение решения

Существует несколько алгоритмов, способных ускорить решение данной задачи. Так, есть алгоритм Шуфа, способный решить задачу вычисления порядка кривой за сложность $O(log^8q)$, что намного лучше наивного решения.

Также есть возможность ускорить решение задачи поиска порядка точки. Для этого можно применить алгоритмы «baby-step, giant-step» и ρ-алгоритм Полларда оба алгоритма решают задачу дискретного логарифмирования: найти для двух заданных точек P и Q целое число x, удовлетворяющее уравнению Q = xP. Временная сложность обоих $O(\sqrt{n})$, , однако, ρ – алгоритм Полларда обладает привлекательной пространственной сложностью $O(1)$. Можно попытаться применить их для решения задачи определения порядка точки приравняв Q и P и обозначив, что $x \neq 1$.

Также, полезным фактом будет то, что порядок дочки всегда является делителем порядка кривой. Данное свойство происходит из теоремы Лагранжа и на его основе можно придумать оптимизацию. Для определения порядка точки нам достаточно определились порядок кривой, факторизовать его (каким-нибудь быстрым алгоритмом) и затем перебирать делители в качестве потенциальных ответов, существенно сократив область поиска.

Вывод

В ходе выполнения лабораторной работы я познакомился эллиптическими кривыми в криптографии. Реализовал решение задачи определения порядка кривой и порядков точек наивным способом. Подобрал кривую, решение задачи для которой заняло 10 минут на мощностях моего ПК. Провел сравнительный анализ более эффективных алгоритмов решения данных задач и изучил свойства, полезные в решении. Хоть математического доказательства нет, но сложность решения задачи дискретного логарифмирования в рамках эллиптических кривых, кажется, “сложнее”, чем факторизация числа. Этот вывод можно сделать, сравнивая размеры ключей, дающих одинаковый уровень защиты у RSA и ECC. ECC в этом плане показывает себя лучше, что делает его более предпочтительным.