

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Московский Авиационный Институт»
(Национальный Исследовательский Университет)

Институт: №8 «Информационные технологии
и прикладная математика»
Кафедра: 806 «Вычислительная математика
и программирование»

Лабораторная работа № 5
по курсу «Криптография»

Группа: М8О-308Б-21

Студент(ка): А. Ю. Гришин

Преподаватель: А. В. Борисов

Оценка:

Дата: 16.05.2024

Москва, 2024

ОГЛАВЛЕНИЕ

1	Тема	3
2	Задание	3
3	Теория.....	3
	Эллиптические кривые	3
	Операция сложения	3
	Кривые над конечным полем.....	4
	Особенности сложения.....	5
	Порядок.....	5
	Задача дискретного логарифмирования	5
4	Ход лабораторной работы	6
	Подбор параметров	6
	Вычислительная мощность ЭВМ	7
	Поиск порядка кривой	8
	Поиск точки.....	11
	Расчет порядка точки.....	12
5	Выводы.....	12
6	Список используемой литературы	13
7	Листинг.....	13
	curve.py.....	13
	find_dot.py	16
	find_order.p.....	17
	get_divisors.py	17
	calc_speed.py	18

1 Тема

Темой данной лабораторной работы является исследование эллиптических кривых над конечным простым полем Z_p , определение порядка точки с использованием полного перебора, а также анализ времени выполнения и характеристик вычислителя. В работе также рассматриваются алгоритмы и теоремы, которые могут быть использованы для ускорения и облегчения решения задачи полного перебора.

2 Задание

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

Рассмотреть для случая конечного простого поля Z_p .

3 Теория

Эллиптические кривые

Эллиптические кривые представляют собой специальный вид кривых, которые описываются уравнением Вейерштрасса:

$$y^2 = x^3 + a \cdot x + b$$

В криптографии также накладывается дополнительное ограничение на эллиптические кривые: $4 \cdot a^3 + 27 \cdot b^2 \neq 0$. Кривые, удовлетворяющие этому условию, называются **гладкими**.

Операция сложения

Мы можем также определить на эллиптической кривой алгебру.

Сложение трех точек на эллиптической кривой обычно определяется следующим образом: если мы имеем три точки A , B и C на кривой, и они лежат на одной прямой, то их сумма равна нулю (то есть $A + B + C = 0$). Если мы хотим сложить только две точки, скажем A и B , мы просто берем точку C , которая является третьей точкой на прямой через A и B , и определяем $A + B$ как обратную к C . Если же $A = B$, то точка C является точкой пересечения касательной в точке A с кривой.

Обратной точкой точке A будем называть точку $(-A) = (A_x - A_y)$, которая является зеркальной версией точки A относительно оси OX .

Нулевая точка на эллиптической кривой представляет собой "точку в бесконечности". Она играет роль нейтрального элемента в группе, определенной на кривой. Это означает, что для любой точки P на кривой выполняется равенство $P + 0 = P$. Кроме того, если точка P находится на кривой, то $P + (-P) = 0$.

Кривые над конечным полем

В криптографии используется специальный вид кривых, которые определены на конечном поле. Это поле представляет собой кольцо вычетов по модулю простого числа. Такое поле обычно обозначается как Z_p и содержит целые положительные числа в диапазоне $[0 .. (p - 1)]$. Точки на такой кривой должны удовлетворять следующему равенству:

$$y^2 = (x^3 + a \cdot x + b) \bmod p$$

То есть, все операции проводятся в контексте модуля p .

На таком кольце определены операции сложения, вычитания, умножения по модулю. Также, над кольцом определена операция дискретного логарифма, который на самом деле представляет собой аналог операции "деления" над полем Z_p .

Особенности сложения

Операция сложения в таком поле имеет свои особенности. Если мы складываем две точки $P = (x_1, y_1)$ и $Q = (x_2, y_2)$ на эллиптической кривой, результатом будет точка $R = (x_3, y_3)$, где:

$$\begin{aligned}x_3 &= \lambda^2 - x_1 - x_2 \bmod p \\ y_3 &= \lambda \cdot (x_1 - x_3) - y_1 \bmod p\end{aligned}$$

здесь λ представляет наклон прямой, проходящей через точки P и Q .

Порядок

Порядок кривой соответствует количеству точек на кривой, включая точку в бесконечности. В контексте эллиптической кривой, порядок точки P означает такое наименьшее положительное число n , при котором $n \cdot P = O$, где O - это точка в бесконечности, а $n \cdot P$ - это результат сложения точки P самой с собой n раз.

Задача дискретного логарифмирования

В контексте эллиптических кривых, задача дискретного логарифмирования формулируется следующим образом. Пусть даны точки P , Q на эллиптической кривой, и требуется найти такое целое число n , которое будет удовлетворять следующему равенству

$$n \cdot P = Q$$

где $n \cdot P$ означает результат сложения точки P с самой собой n раз.

Такая задача является на текущий момент "сложной". То есть, не было придумано алгоритма, который смог бы решить поставленную задачу за приемлемое время. Если бы был найден эффективный алгоритм для решения этой задачи, то большинство современных криптосистем стали бы небезопасными.

Сложность задачи дискретного логарифмирования в контексте эллиптических кривых обусловлена тем, что операции на эллиптических кривых отличаются от операций в обычных группах. В частности, операция сложения на эллиптической кривой не является коммутативной, что затрудняет применение многих известных алгоритмов. Кроме того, эффективные алгоритмы для решения задачи дискретного логарифмирования в других группах, таких как группа целых чисел по модулю p , не могут быть применены к эллиптическим кривым, что делает задачу дискретного логарифмирования на них особенно сложной.

4 **Ход лабораторной работы**

Перед описанием выполнения лабораторной работы стоит отметить, что конечно весь процесс можно было бы сократить до выбора кривой и точки, предоставляемой OpenSSH. Подсчет порядка такой точки при заданных параметрах очевидно будет занимать больше 10 минут. Поэтому, при выполнении лабораторной работы я поставил задачу нахождения такой кривой и точки, время подсчета порядка которой будет как можно ближе к 10 минутам.

Все дальнейшие вычисления производились на ПК со следующими характеристиками:

- Процессор: AMD Ryzen 5 5500U with Radeon Graphics; частота 2.10 GHz; количество ядер 6;
- ОЗУ: 16 Гб; DDR 4

Подбор параметров

При подборе параметров я начал с той идеи, что обычный перебор будет не самым лучшим решением, так как при очень больших значениях параметра p мы будем пробовать замерять время на нахождение порядка точек, не опираясь ни на какие эвристики. Поэтому, при нахождении точки, вычисление порядка которой займет 5 минут, велика вероятность сначала пройти 20 точек, вычисление порядка которых займет по 1 минуте. Итого мы будем тратить в разы больше времени.

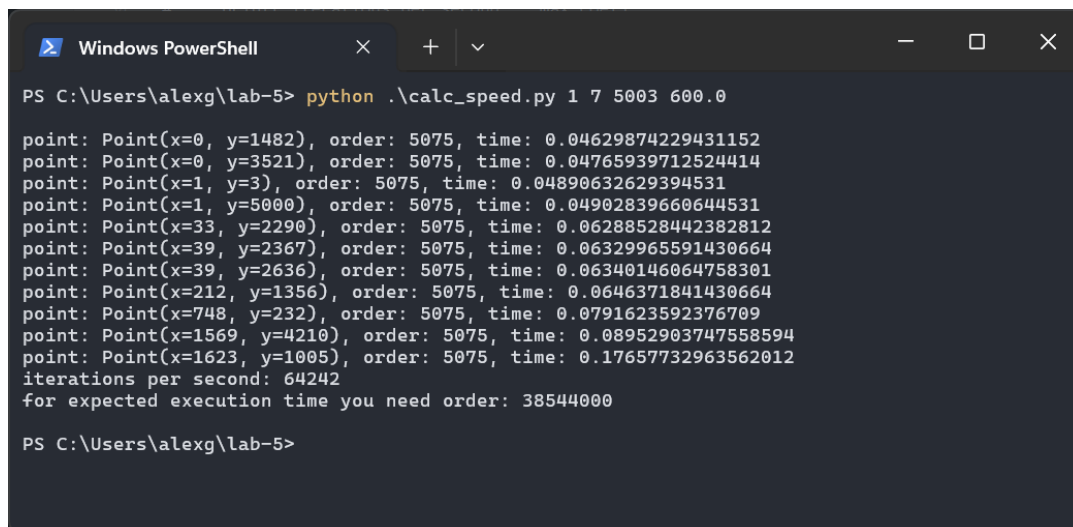
Поэтому первым этапом было сформировать стратегию поиска параметров, используя теоретическую основу эллиптических кривых. Первое, что я заметил - время вычисления порядка точки прямо пропорционально собственно ее порядку. Поэтому, если мы знаем, сколько времени потребуется на 1 итерацию, и требуемое время, с которым должен вычисляться порядок, то само значение порядка мы можем (очень грубо) аппроксимировать следующим выражением

$$n_p = \frac{T}{s}$$

где n_p - порядок точки, T - ожидаемое время вычисления порядка, s - сколько времени требуется на 1 итерацию.

Вычислительная мощность ЭВМ

Возникает проблема в поиске значения s . Для этого я написал небольшой скрипт на Python, который при не слишком большом параметре p начинает вычислять порядки всех точек кривой. Для каждой точки также вычисляется потраченное время. Далее, для каждой точки я посчитал отношение порядка к затраченному времени и нашел среднее арифметическое среди таких отношений.



```
Windows PowerShell
PS C:\Users\alexg\lab-5> python .\calc_speed.py 1 7 5003 600.0

point: Point(x=0, y=1482), order: 5075, time: 0.04629874229431152
point: Point(x=0, y=3521), order: 5075, time: 0.04765939712524414
point: Point(x=1, y=3), order: 5075, time: 0.04890632629394531
point: Point(x=1, y=5000), order: 5075, time: 0.04902839660644531
point: Point(x=33, y=2290), order: 5075, time: 0.06288528442382812
point: Point(x=39, y=2367), order: 5075, time: 0.06329965591430664
point: Point(x=39, y=2636), order: 5075, time: 0.06340146064758301
point: Point(x=212, y=1356), order: 5075, time: 0.06463718414306664
point: Point(x=748, y=232), order: 5075, time: 0.0791623592376709
point: Point(x=1569, y=4210), order: 5075, time: 0.08952903747558594
point: Point(x=1623, y=1005), order: 5075, time: 0.17657732963562012
iterations per second: 64242
for expected execution time you need order: 38544000

PS C:\Users\alexg\lab-5>
```

Итого, мы получаем, что интересующая нас точка должна иметь порядок, близкий к найденному значению. Теперь задача сводится к нахождению:

- Параметра p , в котором будет интересующая нас точка
- Интересующей нас точки $P = (x, y)$

Так как полный перебор здесь будет крайне не эффективен, то я снова воспользовался теоретическими основами эллиптических кривых. Возьмем произвольную точку P , принадлежащую кривой с порядком N . Тогда, из теоремы Лагранжа следует, что порядком точки P является один из делителей N . Также, по определению, порядком точки P является такое число n , что $n \cdot P = 0$. Итого, мы приходим к следующей стратегии поиска параметров:

1. Возьмем произвольный p .
2. Для кривой с таким параметров ищем ее порядок N .
3. Находим все делители N и берем из них максимальный – n .
4. Возьмем точку P , принадлежащую кривой.
5. Если $n \cdot P = 0$ и при всех делителях k меньших n , это равенство не соблюдается, то мы можем сказать, что порядок текущей точки P равен n .

Теперь задача сводится к поиску порядка самой эллиптической кривой и более эффективному вычислению умножения, так как сейчас он ни чем не отличается от сложения за линейное время.

Поиск порядка кривой

Поиск порядка кривой является "сложной" задачей. Это означает, что она не может быть выполнена достаточно быстро на современных машинах.

Самое простое решение - посчитать количество точек, принадлежащих кривой. В этом случае мы проходимся по всем возможным значениям (x, y) , где каждая координата может иметь целое значение в диапазоне $[0; p)$, проверяем каждую точку на принадлежность. Однако, как видно из описания алгоритма, его временная сложность составляет $O(p^2)$, из-за чего алгоритм становится пригодных при не очень больших p .

Однако, есть улучшенный алгоритм, который имеет такой же смысл - подсчет всех точек. Этот алгоритм основывается на той идее, что в уравнении эллиптической кривой, левая и правая часть содержат ровно по 1 координате. Итого, мы можем пройтись по всем возможным значениям $x \in [0; p)$, $y \in [0; p)$, для каждой части посчитать значение соответствующей части и сохранить результат.

Результат сохраняется в виде отображения "значение" \rightarrow "кол-во значений координаты". Далее, нам нужно пройтись по всем возможным значениям, взять кол-во значения каждой координаты и перемножить их. Суммой всех найденных значений и будет порядок кривой.

Такой способ работает уже за линейную сложность $O(p)$, однако, он его пространственная сложность также увеличивается - $O(p)$. Итого, при значениях $p \sim 10^6$ нам придется хранить уже 2 словаря по миллиону элементов, что крайне затратно по памяти, и алгоритм становится непригодным при огромных значениях p .

Однако, существует алгоритм Шуфа, который эффективно делает подсчет числа точек на эллиптической кривой над конечным полем. Для этого я использовал уже готовую реализацию алгоритма Шуфа. представленную в виде исполняемого файла.

Далее, путем подбора значения p я нашел такую кривую, у которой порядок был больше интересующего нас количества итераций, так как значения ниже нас не интересуют ведь все делители такого числа будут меньше, и максимальный делитель которого также больше интересующего нас количества.

```
Windows PowerShell
PS C:\Users\alexg\lab-5> .\Shoof.exe 50021 1 7
NP= 50157
PS C:\Users\alexg\lab-5> python .\get_divisors.py 50157
divisors: 3, 9, 5573, 16719
PS C:\Users\alexg\lab-5> .\Shoof.exe 178371091 1 7
NP= 178353200
PS C:\Users\alexg\lab-5> python .\get_divisors.py 178353200
divisors: 2, 4, 5, 8, 10, 16, 20, 25, 40, 50, 80, 100, 200, 400, 445883, 891766, 1783532,
2229415, 3567064, 4458830, 7134128, 8917660, 11147075, 17835320, 22294150, 35670640, 44588
300, 89176600
PS C:\Users\alexg\lab-5> .\Shoof.exe 38544001 1 7
NP= 38548224
PS C:\Users\alexg\lab-5> python .\get_divisors.py 38548224
divisors: 2, 3, 4, 6, 8, 9, 11, 12, 13, 16, 18, 22, 24, 26, 27, 32, 33, 36, 39, 44, 48, 52
, 54, 64, 66, 72, 78, 81, 88, 96, 99, 104, 108, 117, 128, 132, 143, 144, 156, 162, 169, 17
6, 192, 198, 208, 216, 234, 256, 264, 286, 288, 297, 312, 324, 338, 351, 352, 384, 396, 41
6, 429, 432, 468, 507, 528, 572, 576, 594, 624, 648, 676, 702, 704, 768, 792, 832, 858, 86
4, 891, 936, 1014, 1053, 1056, 1144, 1152, 1188, 1248, 1287, 1296, 1352, 1404, 1408, 1521,
1584, 1664, 1716, 1728, 1782, 1859, 1872, 2028, 2106, 2112, 2288, 2304, 2376, 2496, 2574,
2592, 2704, 2808, 2816, 3042, 3168, 3328, 3432, 3456, 3564, 3718, 3744, 3861, 4056, 4212,
4224, 4563, 4576, 4752, 4992, 5148, 5184, 5408, 5577, 5616, 6084, 6336, 6864, 6912, 7128,
7436, 7488, 7722, 8112, 8424, 8448, 9126, 9152, 9504, 9984, 10296, 10368, 10816, 11154, 1
1232, 11583, 12168, 12672, 13689, 13728, 14256, 14872, 14976, 15444, 16224, 16731, 16848,
18252, 18304, 19008, 20592, 20736, 21632, 22308, 22464, 23166, 24336, 25344, 27378, 27456,
28512, 29744, 29952, 30888, 32448, 33462, 33696, 36504, 36608, 38016, 41184, 43264, 44616
, 44928, 46332, 48672, 50193, 54756, 54912, 57024, 59488, 61776, 64896, 66924, 67392, 7300
8, 76032, 82368, 89232, 89856, 92664, 97344, 100386, 109512, 109824, 114048, 118976, 12355
2, 129792, 133848, 134784, 146016, 150579, 164736, 178464, 185328, 194688, 200772, 219024,
228096, 237952, 247104, 267696, 269568, 292032, 301158, 329472, 356928, 370656, 389376, 4
01544, 438048, 475904, 494208, 535392, 584064, 602316, 713856, 741312, 803088, 876096, 988
416, 1070784, 1168128, 1204632, 1427712, 1482624, 1606176, 1752192, 2141568, 2409264, 2965
248, 3212352, 3504384, 4283136, 4818528, 6424704, 9637056, 12849408, 19274112
```

```
Windows PowerShell
PS C:\Users\alexg\lab-5> .\Shoof.exe 713484217 1 7
NP= 713455008
PS C:\Users\alexg\lab-5> python .\get_divisors.py 713455008
divisors: 2, 3, 4, 6, 7, 8, 12, 14, 16, 21, 24, 28, 32, 42, 48, 56, 84, 96, 112, 168, 224,
336, 672, 1061689, 2123378, 3185067, 4246756, 6370134, 7431823, 8493512, 12740268, 148636
46, 16987024, 22295469, 25480536, 29727292, 33974048, 44590938, 50961072, 59454584, 891818
76, 101922144, 118909168, 178363752, 237818336, 356727504
PS C:\Users\alexg\lab-5> .\Shoof.exe 356742149 1 7
NP= 356762784
PS C:\Users\alexg\lab-5> python .\get_divisors.py 356762784
divisors: 2, 3, 4, 6, 7, 8, 12, 14, 16, 21, 24, 28, 32, 42, 48, 56, 84, 96, 112, 168, 224,
336, 672, 530897, 1061794, 1592691, 2123588, 3185382, 3716279, 4247176, 6370764, 7432558,
8494352, 11148837, 12741528, 14865116, 16988704, 22297674, 25483056, 29730232, 44595348,
50966112, 59460464, 89190696, 118920928, 178381392
PS C:\Users\alexg\lab-5> |
```

Ближе всего к интересующему меня количеству итераций оказалось значение 356727504, которое соответствует значению p , равному 713484217.

Поиск точки

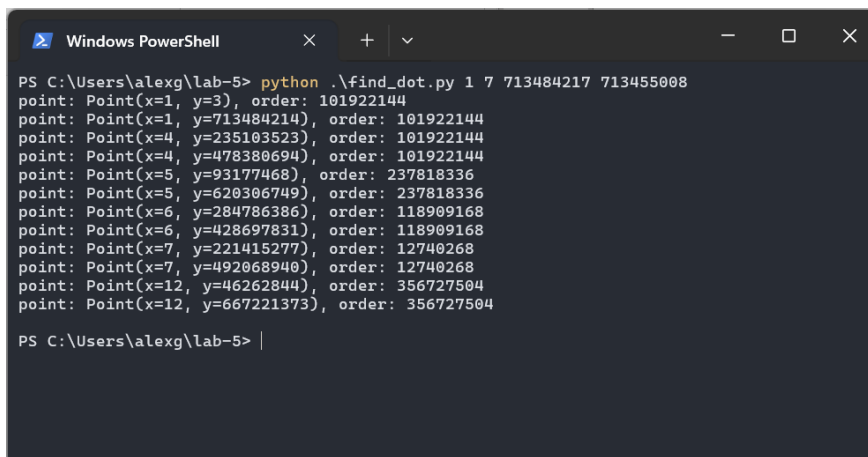
И так, мы выяснили, что кривая с параметрами $a = 1, b = 7, p = 713484217$ содержит точку, порядок которой удовлетворяет нашим требованиям. В итоге, остается найти конкретную точку с необходимым порядком.

Так как вычислять порядок каждой точки "прямо" будет занимать слишком много времени, я решил использовать более оптимизированный способ, основанный на определении порядка точки. Как упоминалось ранее, порядком точки называется такое n , что $n \cdot P = 0$. Поэтому, достаточно пройти по всем делителям порядка кривой и посмотреть, при каком делителе результатом будет равен нулевой (бесконечно удаленной) точке.

Для более оптимального вычисления произведения $n \cdot P$ я воспользовался стандартным алгоритмом бинарного умножения, который часто применяется на практике. Его суть заключается в представлении числа n в двоичном виде

$$n = a_0 \cdot 2^0 + a_1 \cdot 2^1 + a_2 \cdot 2^2 + \dots$$

В таком виде значение умножения может быть вычислено за $O(\log n)$ вместо $O(n)$.



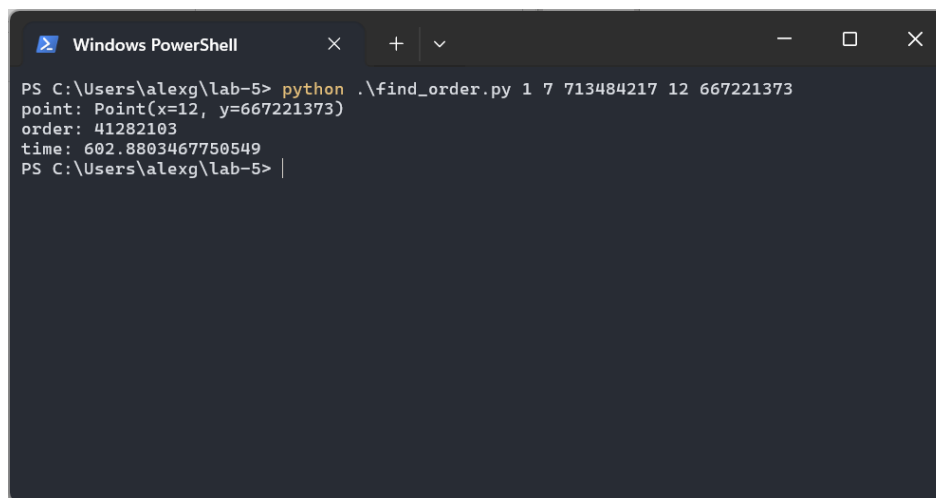
```
PS C:\Users\alexg\lab-5> python .\find_dot.py 1 7 713484217 713455008
point: Point(x=1, y=3), order: 101922144
point: Point(x=1, y=713484214), order: 101922144
point: Point(x=4, y=235103523), order: 101922144
point: Point(x=4, y=478380694), order: 101922144
point: Point(x=5, y=93177468), order: 237818336
point: Point(x=5, y=620306749), order: 237818336
point: Point(x=6, y=284786386), order: 118909168
point: Point(x=6, y=428697831), order: 118909168
point: Point(x=7, y=221415277), order: 12740268
point: Point(x=7, y=492068940), order: 12740268
point: Point(x=12, y=46262844), order: 356727504
point: Point(x=12, y=667221373), order: 356727504

PS C:\Users\alexg\lab-5> |
```

Итого, мы видим, что точка $(12, 667221373)$ имеет порядок 356727504. Это значение близко к интересующему нас, что дает нам основания предполагать, что подсчет такой точки будет занимать около 10 минут.

Расчет порядка точки

Проверим нашу теорию на практике, посчитав перебором порядок точки $(12, 667221373)$ на кривой $(1, 7, 713484217)$. Для этого я написал скрипт на Python, который путем использования операции сложения до тех пор, пока не получит результат $(0, 0)$, вычисляет порядок точки.



```
Windows PowerShell
PS C:\Users\alexg\lab-5> python .\find_order.py 1 7 713484217 12 667221373
point: Point(x=12, y=667221373)
order: 41282103
time: 602.8803467750549
PS C:\Users\alexg\lab-5> |
```

5 Выводы

В ходе выполнения данной лабораторной работы я выполнил поиск эллиптической кривой и точки, вычисление порядка которой наивным способом занимает около 10 минут на ПК.

Я на практике убедился, почему в современных конфигурациях эллиптических кривых, предоставляемых, например, OpenSSH, параметр p имеет такие большие значения. Я также понял, что точка P не выбирается произвольным образом, а выбирается такая, которая будет удовлетворять необходимым требованиям безопасности.

В ходе данной лабораторной работы я убедился, насколько неэффективно наивное решение для задачи нахождения порядка точки, которая является частным случаем задачи дискретного логарифмирования. В настоящее время существуют более оптимизированные алгоритмы нахождения порядка точки, такие как алгоритм **ρ Полларда** и **Baby-step giant-step**.

Эллиптические кривые могут служить заменой традиционным методам криптографии, таким как RSA.

Эллиптические кривые имеют ряд преимуществ по сравнению с RSA, что позволяет им быть хорошей заменой:

1. Более высокая эффективность: Эллиптические кривые требуют меньше вычислительных ресурсов для генерации ключей, шифрования и дешифрования.
2. Меньший размер ключа: Для достижения такого же уровня криптостойкости, как и в RSA, требуются ключи меньшего размера. Это уменьшает затраты на хранение и передачу информации.
3. Безопасность: На сегодняшний день неизвестны субэкспоненциальные алгоритмы для решения задачи дискретного логарифмирования в группах точек эллиптических кривых.

6 Список используемой литературы

- “Доступно о криптографии на эллиптических кривых” Хабр: <https://habr.com/ru/articles/335906/>
- Статья на сайте secuteck: http://secuteck.ru/articles2/Inf_security/elept_krivaya_nov_etap_razv_kripto
- “Что такое шифрование на основе эллиптических кривых” keepersecurity: <https://www.keepersecurity.com/blog/ru/2023/06/07/what-is-elliptic-curve-cryptography/>

7 Листинг

`curve.py`

```
from dataclasses import dataclass
```

```

from typing import Iterator
from itertools import product
from multiprocessing import Pool
import random

@dataclass(slots=True, frozen=True)
class Point:
    x: int
    y: int

def extended_gcd(a: int, b: int) -> tuple[int, int, int]:
    stack = []

    while a:
        stack.append((a, b))
        a, b = b % a, a

    gcd, x, y = b, 0, 1
    while stack:
        a, b = stack.pop()
        x, y = y - (b // a) * x, x

    return gcd, x, y

def mod_inverse(a, p):
    gcd, x, _ = extended_gcd(a, p)

    if gcd != 1:
        raise Exception("Can't find inverse element")

    else:
        return x % p

class Curve:
    def __init__(self, a: int, b: int, p: int) -> None:
        self._a = a
        self._b = b
        self._p = p

    def _check_curve_coefs(self) -> None:
        a, b, p = self._a, self._b, self._p

        if (4 * a ** 3 + 27 * b ** 2) % p == 0:
            raise ValueError("Invalid curve coefficients")

    def contains(self, point: Point) -> bool:
        x, y = point.x, point.y
        a, b, p = self._a, self._b, self._p

        return (y * y) % p == (x * x * x + a * x + b) % p

```

```

def add(self, p: Point, q: Point) -> Point:
    px, py, qx, qy = p.x, p.y, q.x, q.y

    if p == Point(0, 0):
        return q

    if q == Point(0, 0):
        return p

    if py == (-qy) % self._p:
        return Point(0, 0)

    if p == q:
        coef = mod_inverse(2 * py % self._p, self._p)
        m = ((3 * px ** 2 + self._a) * coef) % self._p
    else:
        coef = mod_inverse((px - qx) % self._p, self._p)
        m = ((py - qy) * coef) % self._p

    rx = (m ** 2 - px - qx) % self._p
    ry = (qy + m * (rx - qx)) % self._p

    return Point(rx, self._p - ry)

def mult(self, n: int, p: Point) -> Point:
    result = Point(0, 0)

    while n:
        if n & 1:
            result = self.add(result, p)
        p = self.add(p, p)
        n >>= 1

    return result

def get_order(self) -> int:
    N = 1

    for x in range(0, self._p):
        for y in range(0, self._p):
            left = (y * y) % self._p
            right = (x * x * x + self._a * x + self._b) % self._p
            N += (left == right)

    return N

def _check(args):
    a, b, p, x, y = args
    result = (y * y) % p == (x * x * x + a * x + b) % p
    return x, y, result

def iter_points(curve: Curve) -> Iterator[Point]:
    p = curve._p

```

```

    for x in range(0, p):
        for y in range(0, p):
            point = Point(x, y)

            if curve.contains(point):
                yield point

def get_random_point(curve: Curve) -> Point:
    index = random.randint(128, 256)
    for number, point in enumerate(iter_points(curve), 1):
        if number == index:
            return point
    return point

def point_order(curve: Curve, point: Point) -> int:
    order = 1
    zero_point = Point(0, 0)
    curr = point

    while curr != zero_point:
        order += 1
        curr = curve.add(curr, point)

    return order

```

find_dot.py

```

import sys
from typing import Iterable, Iterator
from curve import Curve, Point, iter_points
from get_divisors import find_divisors

def find_dot_orders(
    curve: Curve,
    points: Iterable[Point],
    divisors: list[int],
) -> Iterator[tuple[int, Point]]:
    for point in points:
        for d in divisors:
            if curve.mult(d, point) == Point(0, 0):
                yield d, point
                break

if __name__ == "__main__":
    a, b, p = map(int, sys.argv[1:4])
    n = int(sys.argv[4])

    curve = Curve(a, b, p)

```



```

d = find_divisors(n)
points = iter_points(curve)

for order, point in find_dot_orders(curve, points, d):
    print(f"point: {point}, order: {order}")

```

find_order.p

```

from curve import Curve, Point, point_order
import time
import sys

```

```

if __name__ == "__main__":
    a, b, p = map(int, sys.argv[1:4])
    x, y = map(int, sys.argv[4:6])

    curve = Curve(a, b, p)
    point = Point(x, y)

    start = time.time()
    order = point_order(curve, point)
    end = time.time()

    print(f'point: {point}')
    print(f'order: {order}')
    print(f'time: {end - start}')

```

get_divisors.py

```

import sys

def find_divisors(n: int) -> list[int]:
    divisors_left = []
    divisors_right = []

    for k in range(2, int(n ** 0.5) + 1):
        if n % k == 0:
            divisors_left.append(k)
            divisors_right.insert(0, n // k)

    return divisors_left + divisors_right

if __name__ == "__main__":

```

```

n = int(sys.argv[1])
d = find_divisors(n)
print("divisors:", ", ".join(map(str, d)))

```

calc_speed.py

```

from curve import Curve, Point, iter_points, point_order
import time
import sys

def calc_order_time(curve: Curve, point: Point) -> tuple[int, float]:
    start = time.time()
    order = point_order(curve, point)
    end = time.time()

    return order, end - start

if __name__ == "__main__":
    a, b, p = map(int, sys.argv[1:4])
    expected = float(sys.argv[4])
    curve = Curve(a, b, p)

    results = []
    max_time = 0.0

    for point in iter_points(curve):
        order, elapsed = calc_order_time(curve, point)

        if elapsed > max_time:
            max_time = elapsed
            print(f"point: {point}, order: {order}, time: {elapsed}")

        results.append((order, elapsed))

    coefs = [order / elapsed for order, elapsed in results if elapsed]
    max_coef = int(max(coefs))

    print("iterations per second:", max_coef)
    print(
        "for expected execution time you need order:",
        int(expected * max_coef),
    )

```