Федеральное государственное автономное образовательное учреждение высшего профессионального образования Национальный исследовательский университет «Высшая школа экономики»

Московский институт электроники и математики Факультет прикладной математики и кибернетики

Кафедра «Компьютерная безопасность»

ОТЧЕТ

по дисциплине «Теоретико-числовые методы в криптографии»

Программная реализация алгоритма Полига - Хеллмана (с использованием Sage)

Выполнили студенты
группы СКБ-171:
Зайцева А. А.
Дундуков С. В.

Проверил: Доцент
Нестеренко А. Ю.

МОСКВА 2021

Теоретическое описание

Китайская теорема об остатках [3]:

Если у нас есть система уравнений вида:

$$\begin{cases} x = r_1(mod\ a_1) \\ x = r_2(mod\ a_2) \\ \dots \\ x = r_n(mod\ a_n) \end{cases}$$

Где все a_iпопарно взаимно просты, а r_i – целое и $0 \le r_i < a_i\ \forall i \in \{1,2,\dots,n\}$. Тогда найдется такое X, которое при делении на a_i дает остаток $r_i\ \ \forall i \in \{1,2,\dots,n\}$

Из доказательства теоремы следует, что решением этой системы будет: $X = \sum_{i=1}^{n} r_i\, M_i {M_i}^{-1}$, где

$$M = \prod_{i=1}^{n} a_i\,, \qquad M_i = \frac{M}{a_i}, \quad {M_i}^{-1} = \frac{1}{M_i}(mod\ a_i)$$

Теорема Лагранжа:

Пусть группа G конечна, и H — её подгруппа. Тогда порядок G равен порядку H, умноженному на количество её левых или правых классов смежности.

Алгоритм Гельфонда – Шенкса (алгоритм больших и малых шагов) [2]:

Задана циклическая группа G порядка p, генератор группы a, и некоторый элемент b ∈ G. Нужно найти целое число x, для которого выполняется: $a^x = b$.

Для этого используется представление x = i * m – j, где m = [√p] + 1, i ∈ [1, m], j ∈ [0, m).

Если совместить нужный нам результат и представление x, то можно получить такое равенство: $a^{im} = ba^j$.

Соответственно, чтобы найти искомый x, нужно найти i и j. Для этого переберем значения a^{im} для всех i и сравним их со значениями ba^j для всех j. Когда получится равенство – мы нашли i и j. Остается только подставить их в формулу для x.

Этот алгоритм работает для любого p (даже не простого). Сложность этого алгоритма O(√p).

Алгоритм Полига – Хеллмана [1][5]:

Задана циклическая группа G порядка p^k (где p – простое), генератор группы a, и некоторый элемент b ∈ G. Нужно найти целое число x, для которого выполняется: $a^x = b$.

Основная идея алгоритма состоит в итерационном вычислении k логарифмов, которые являются элементами разложения $x = \sum_{i=0}^{k-1} p^i d_i$. Таким образом, вычисляя поочередно x_0, x_1, …, x_k, получаем итоговый результат $x = x_k$.

Алгоритм:

> Задаются начальные значения $x_0 = 0$ и $A = a^{p^{k-1}}$ (по теореме Лагранжа A будет иметь порядок p)
>
> Затем идут k итераций по i ∈ [0,k):
>
> 1. $b_i = (a^{-x_i} b)^{p^{k-1-i}}$
> 2. $d_i = \mathrm{Log}_A b_i$ с помощью алгоритма больших и малых шагов
> 3. $x_{i+1} = x_i + p^i d_i$
>
> На выходе получается $x = x_k = d_0 + d_1 p + d_2 p^2 + \cdots + d_{k-1} p^{k-1}$.

Сложность данного алгоритма O(k√p)

Общий алгоритм вычисления логарифма:

Задана циклическая группа G порядка p (где p – простое),

$p - 1 = \prod_{i=1}^{k} p_i^{q_i}$, генератор группы a, и некоторый элемент b ∈ G. Нужно найти целое число x, для которого выполняется: $a^x = b$.

Т.к. разложение p-1 на множители изначально считается неизвестным, то сначала находим его.

Затем, для каждого i-го элемента разложения делаем следующее:

Находим A и B (полученные из a и b), порядком которых будет i-ый элемент: $A = a^{p_i^{\frac{p-1}{q_i}}}$, $B = b^{p_i^{\frac{p-1}{q_i}}}$

Для полученных A и B вычисляем $\mathrm{Log}_A B$ с помощью одного из алгоритмов:

- Если $q_i = 1$, то используем алгоритм больших и малых шагов.
- Если $q_i > 1$ – используем алгоритм Полига – Хеллмана.

Полученный результат записываем в систему уравнений вида [полученный результат, i-ый элемент]

К полученной системе применяем китайскую теорему об остатках, результатом которой будет искомый x.

Сложность данного алгоритма $O(\sum_{i=1}^{k} q_i(\sqrt{p_i} + \log p))$

Использованные Sage функции [4]

sqrt(a) – вычисления корня из a

a.powermod(m,p) – вычисления $a^m \pmod p$

xgcd(a,b) – реализация расширенного алгоритма Евклида для нахождения обратного элемента

factor(a) – разложение a на множители

sum(f(i) for i in (k..n)) – вычисление суммы функции

cputime() – для замера времени работы функции

Реализация

Программа состоит из 4 функций

1. Ord(a, b, p1, p) – реализация алгоритма больших и маленьких шагов.
   На вход подается 4 параметра:
   - a – основание логарифма,
   - b – число,
   - p1 – маленький модуль (один из делителей p-1),
   - p – общий модуль.

   На выходе получается одно значение – результат вычисление логарифма или информации о отсутствии результата.

2. Polig_Helman(a, b, i, p) – реализация алгоритма Полига-Хеллмана.
   На вход подается 4 параметра:
   - a – основание логарифма,
   - b – число,
   - i – список из двух значений: основания и степени одного из делителей p-1,
   - p – общий модуль.

   На выходе получается одно значение – результат вычисление логарифма или информации о отсутствии результата.

3. System(m, a, b, p) – создания системы уравнений для китайской теоремы об остатках.
   Данная функция реализует Общий алгоритм до момента применения китайской теоремы.
   На вход подается 4 параметра:
   - m – список, в котором содержится разложение p-1 на множители
   - a – основание логарифма,
   - b – число,
   - p – общий модуль

   На выходе – система уравнений в виде списка.

4. Log(a, b, p) – вычисление логарифма.
   В этой функции p-1 раскладывается на множители. Затем, через System(m,a,b,p), вырабатывается система, которая затем решается с

помощью китайской теоремы об остатках.
На вход подается 3 параметра:

- a – основание логарифма,
- b – число,
- p – общий модуль

На выходе – кортеж из двух значений: результат вычисления логарифма и проверка правильности результата.

Примеры работы программы

Пример 1:

Входные значения:

```
a = 2^34 - 23453
b = 2^13 - 216
p = 2^40 - 195
print('40: ',Log(a,b,p),'\n\n')

a = 2^34 - 23
b = 2^45 - 12645678
p = 2^45 - 69
print('45: ',Log(a,b,p),'\n\n')

a = 2^34 - 23453
b = 2^45 - 345765436
p = 2^50 - 51
print('50: ',Log(a,b,p),'\n\n')

a = 10
b = 2^68 - 8765456
p = 2^70 - 923
print('70: ',Log(a,b,p),'\n\n')

a = 2^45 - 654567
b = 2^90
p = 2^100 - 15
print('100: ',Log(a,b,p),'\n\n')

a = 2^100 - 876543567
b = 2^110
p = 2^120 - 6083
print('120: ',Log(a,b,p),'\n\n')

a = 2^75 - 3456754
b = 2^90 - 123
p = 2^150 - 3
print('150: ',Log(a,b,p),'\n\n')

a = 2^75 - 3456754
b = 2^90 - 123
p = 2^170 - 255
print('170: ',Log(a,b,p),'\n\n')
```

Результат:

1 – разложение числа p-1

2 – система уравнений для китайской теоремы об остатках

3 – результат вычисления логарифма

```
2^2 * 3^2 * 5 * 61 * 191 * 269 * 1949
x:  [[0, 4], [0, 9], [4, 5], [51, 61], [136, 191], [112, 269], [1944, 1949]]
40:  (805601299104, True)


2 * 13 * 29761 * 45470417
x:  [[1, 2], [6, 13], [8193, 29761], [42910994, 45470417]]
45:  (25207068749531, True)


2^2 * 3 * 103 * 1063 * 6491 * 132019
x:  [[0, 4], [2, 3], [33, 103], [20, 1063], [3272, 6491], [11549, 132019]]
50:  (288108765128252, True)


2^2 * 5^4 * 71 * 26387 * 53089 * 4747957
x:  [[2, 4], [4670, 625], [60, 71], [482, 26387], [34851, 53089], [1271099, 474
7957]]
70:  (47201136612185764670, True)


2^4 * 3^2 * 5 * 7 * 13 * 17 * 97 * 193 * 241 * 257 * 673 * 65537 * 22253377
x:  [[36, 16], [0, 9], [0, 5], [5, 7], [7, 13], [13, 17], [63, 97], [72, 193],
[44, 241], [203, 257], [15, 673], [2816, 65537], [9189275, 22253377]]
100:  (992527403962474944640041053940, True)


2^2 * 7 * 11 * 23 * 1151 * 4783 * 15583 * 6719593 * 8303821 * 39198839
x:  [[2, 4], [4, 7], [0, 11], [18, 23], [766, 1151], [867, 4783], [11222, 1558
3], [6396636, 6719593], [2619813, 8303821], [995074, 39198839]]
120:  (1200743032895028145253559117051472282, True)


2^2 * 3 * 5 * 149 * 223 * 593 * 1777 * 25781083 * 184481113 * 231769777 * 61631
8177
x:  [[0, 4], [2, 3], [1, 5], [67, 149], [78, 223], [120, 593], [1493, 1777], [8
782238, 25781083], [65787687, 184481113], [20207190, 231769777], [557922790, 61
6318177]]
150:  (1102181832797178792022066693957054609653084116, True)


2^8 * 3^5 * 7 * 19 * 73 * 163 * 2593 * 71119 * 87211 * 135433 * 262657 * 976858
39 * 272010961
x:  [[2992, 256], [2637, 243], [2, 7], [17, 19], [41, 73], [55, 163], [1187, 25
93], [60075, 71119], [5336, 87211], [113228, 135433], [134272, 262657], [554928
36, 97685839], [191281707, 272010961]]
170:  (1222145254418441009386962150411774318951296759012016, True)
```

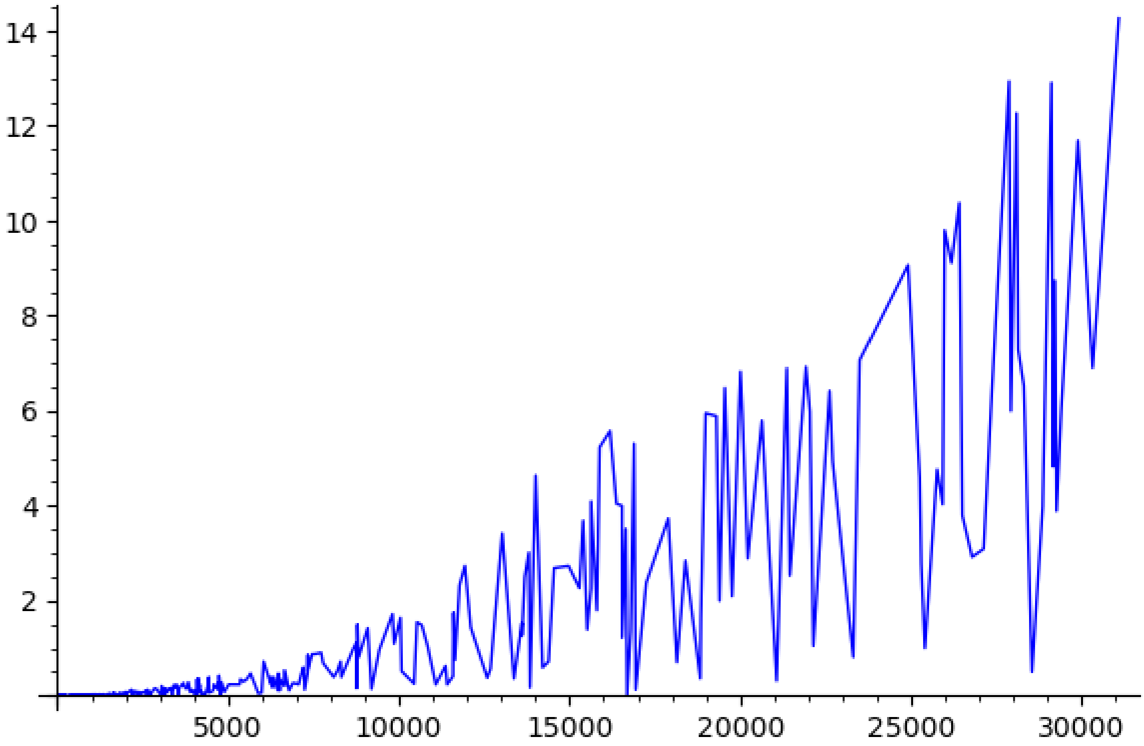
Пример 2:

Нет решений для данных входных значений

```
a = 2^75 - 3456754
b = 2^90 - 12
p = 2^150 - 3
print('150: ',Log(a,b,p),'\n\n')
```

```
2^2 * 3 * 5 * 149 * 223 * 593 * 1777 * 25781083 * 184481113 * 231769777 * 61631
8177
nema
150:  нет решения
```

График зависимости времени выполнения от корня максимального числа в разложении p-1.



Из графика можно сказать, что зависимость получилась квадратичной.

Пики возникают, когда в разложении есть хотя бы одно маленькое число в степени. В таком случае программа обращается к алгоритму Полги-Хеллмана и количество итераций увеличивается в несколько раз.

Выводы

Быстрее всего данный алгоритм работает, когда p-1 раскладывается на маленькие p_i. Т.к. при больших p_i величина $\sqrt{p_i}$ тоже велика, а значит и количество итераций в алгоритме большое.

Из-за этого не удалось посчитать логарифм для $p>2^{170}$, т.к. не было возможности найти разложение с достаточно маленькими элементами.

Список литературы

1. S. Pohlig and M. Hellman (1978). «An Improved Algorithm for Computing Logarithms over GF(p) and its Cryptographic Significance» – IEEE Transactions on Information Theory (24): 106–110.

2. А.О. Гельфонд, Ю.В. Линник. «Элементарные методы в аналитической теории чисел». — М.: Физматгиз, 1962. — 272 с.

3. Василенко О. Н. «Теоретико-числовые алгоритмы в криптографии» — М.: МЦНМО, 2003. — 328 с

4. Уильям Стайн «Краткое Руководство по Sage» (основано на работе П. Джипсен) Лицензия свободной документации GNU с Уильям Стайн 2014

5. Menezes, Alfred J.; van Oorschot, Paul C.; Vanstone, Scott A. (1997) «Handbook of Applied Cryptography»