Министерство образования и науки Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г. ЧЕРНЫШЕВСКОГО»

Кафедра теоретических основ компьютерной безопасности и криптографии

Лабораторная работа №6

студента 5 курса 531 группы
специальности 10.05.01 «Компьютерная безопасность»
факультета компьютерных наук и информационных технологий
Енца Михаила Владимировича

Преподаватель
профессор _____ В. А. Молчанов
подпись, дата

Заведующий кафедрой
д.ф.-м.н., доцент _____ М. Б. Абросимов
подпись, дата

Саратов 2018

1. Постановка задачи.

Изучение основных методов дискретного логарифмирования в конечном поле и их программная реализация.

2. Теоретические сведения по рассмотренным темам с их обоснованием.

Определение. Дискретным логарифмом (показателем) элемента h группы G по основанию g называется число $x \in \{0,1,\dots,m-1\}$, являющееся решением уравнения

$g^x = h$. (1)

Алгоритм Гельфонда-Шенкса

Вход. Конечная циклическая группа $G = \langle g \rangle$, верхняя оценка для порядка группы $|G| \leq B$, элемент $h \in G$.

Выход. Число $x = \log_g h$

Шаг 1. Вычислить $r = \left[\sqrt{B}\right] + 1$.

Вычислить $g^a, 0 \leq a \leq r-1$, упорядочить массив пар (a, g^a) по второй координате.

Шаг 2. Вычислить $g_1 = g^{-r}$. Для каждого $b, 0 \leq b \leq r-1$ проверить, является ли элемент $g_1^b h$ второй координатой какой-либо пары из упорядоченного на шаге 1 массива пар. Если $g_1^b h = g^a$, то запомнить число $x = a + rb$.

Шаг 3. Среди всех чисел, найденный на втором этапе, выбрать наименьшее. Оно и будет искомым значением $x = \log_g h$.

Достоинством приведенного алгоритма является его детерминированный характер, а также отсутствие необходимости знать точное значение порядка группы G.

ρ-метод Полларда.

Дана конечная циклическая группа $G = \langle g \rangle$, известен ее порядок $|G| = m$ и $h \in G$.

Метод Полларда применим к любой циклической группе G, чьи элементы представлены таким образом, что их можно разбить на три примерно равные, попарно не пересекающиеся части $G = U_1 \cup U_2 \cup U_3$. При этом должен существовать эффективный способ проверки, к какому из этих подмножеств принадлежит данный элемент группы.

Будем формировать группы по следующему правилу:

$$U_1 = \left\{a \in G \middle| 0 < a < \frac{p}{3}\right\},$$

$$U_2 = \left\{a \in G \middle| \frac{p}{3} \leq a < \frac{2p}{3}\right\},$$

$$U_3 = \left\{a \in G \middle| \frac{2p}{3} \leq a < p\right\}.$$

Интуитивно ясно, что эти множества примерно равны по величине.

Определим функцию f на G таким образом, что

$$f(a) = \begin{cases} ha, a \in U_1 \\ a^2, a \in U_2 \\ ga, a \in U_3 \end{cases}$$

Идея р-метода логарифмирования в некотором смысле повторяет идею р-метода Полларда факторизации. Будет построена рекуррентная последовательность $y_i = f(y_{i-1}), i \geq 1, y_0 = g^s$. Из определения функции f нетрудно заметить, что при любом $i \geq 0, y_i = h^{b_i} g^{a_i}$ для некоторых $a_i, b_i \in Z_m$. Также последовательности $\{a_i\}, \{b_i\}$ задаются следующим рекуррентными соотношениями:

$$a_0 = 0, a_{i+1} = \begin{cases} a_i + 1 \ (mod\ m-1), & y_i \in U_1; \\ 2a_i \ (mod\ m-1), & y_i \in U_2; \\ a_i \ (mod\ m-1), & y_i \in U_3. \end{cases}$$

$$b_0 = s, b_{i+1} = \begin{cases} b_i \ (mod\ m-1), & y_i \in U_1; \\ 2b_i \ (mod\ m-1), & y_i \in U_2; \\ b_i + 1 \ (mod\ m-1), & y_i \in U_3. \end{cases}$$

При вычислении очередного члена последовательности $y_i = f(y_{i-1})$ числа a_i, b_i вычисляются по известным a_{i-1}, b_{i-1} очень легко. При этом для любого $i \geq 0$ выполняется равенство

$$\log_g y_i = b_i x + a_i \ (mod\ m).$$

Вход: конечная циклическая группа $G = \langle g \rangle$ порядка m, элемент $h \in G$, функция f, заданная соотношением выше, $\varepsilon > 0$ – точность алгоритма.

Выход: $x = \log_g h$.

Шаг 1. Вычислить $T = \left[\sqrt{2m \ln(1/\varepsilon)}\right] + 1$.

Шаг 2. Положить $i = 1$, выбрать случайное $s \in \mathbb{Z}_m$, вычислить $y_1 = f(g^s (mod\ m))$, $y_2 = f(y_1)$. Запомнить две тройки (y_1, a_1, b_1), (y_2, a_2, b_2) и перейти к шагу 4.

Шаг 3. Положить $i = i + 1$, найти $y_i = f(y_{i-1})$, $y_{2i} = f(f(y_{2i-2}))$. Запомнить две тройки (y_i, a_i, b_i), (y_{2i}, a_{2i}, b_{2i}) и перейти к шагу 4.

Шаг 4. Если $y_i \neq y_{2i}$ и $i < T$ – перейти к шагу 3. Если $i = T$, то остановить алгоритм и сообщить, что $x = \log_g h$ вычислить не удалось. Если $y_i = y_{2i}$, то перейти к шагу 5.

Шаг 5. Вычислить $d = \text{НОД}(a_{2i} - a_i, m - 1)$. Если $1 < d \leq \sqrt{m}$, то сравнение

$$b_i - b_{2i} \equiv (a_{2i} - a_i) x \ (mod\ m - 1)$$

имеет d различных решений по модулю m. Для каждого из этих решений проверить выполнимость равенства $g^x = h \ (mod\ m)$ и найти истинное решение $x = \log_g h$. Если $d > \sqrt{m}$ – перейти на шаг 2 и выбрать новое значение s.

Коды программ, реализующей рассмотренные алгоритмы

Программа реализована на языке Python (версия интерпретатора 3.6).

```
def shanks_gelfond(m, g, h):
    def square_root(n):
        x1 = n
        x2 = int((x1 + (n / x1)) / 2)
        while x2 < x1:
            x1, x2 = x2, int((x2 + (n / x2)) / 2)
        return x1

    # Step 1
    r = square_root(m) + 1
    pairs = {pow(g, a, m): a for a in range(r)}
```

```
            # Step 2
            g1 = pow(utils.inverse(g, m), r, m)
            for b in range(r):  # r-1 ?
                value = (pow(g1, b, m) * h) % m
                if value in pairs:
                    return pairs[value] + r * b

        def ppollard(m, g, h, e=0.05):
            def equation(a, b, m):
                a, b = a % m, b % m
                d = math.gcd(a, m)
                if b % d != 0:
                    return

                a_new, b_new, m_new = a // d, b // d, m // d
                d_new, q, r = euclid.euclid_extended(a_new, m_new)
                q, r = q % m, r % m

                x0 = (b_new * q) % m_new
                for j in range(d):
                    yield x0 + m_new * j

            def calculate_yab(y, a, b, params):
                g, h, m = params
                if y <= m // 3:
                    y = (y * h) % m
                    a = (a + 1) % (m - 1)
                elif m // 3 < y <= 2 * m // 3:
                    y = (y * y) % m
                    a = (a * 2) % (m - 1)
                    b = (b * 2) % (m - 1)
                elif 2 * m // 3 < m:
                    y = (y * g) % m
                    b = (b + 1) % (m - 1)
                return y, a, b

            # Step 1
            t = square_root(2 * m * math.log(1 / e)) + 1

            while True:
                # Step 2
                i = 1
                s = random.randint(0, m - 2)  # m-1 ?

                yi, ai, bi = calculate_yab(pow(g, s, m), 0, s, (g, h, m))
                y2i, a2i, b2i = calculate_yab(yi, ai, bi, (g, h, m))

                # Step 4
                while i < t and yi != y2i:
                    # Step 3
                    i += 1
                    yi, ai, bi = calculate_yab(yi, ai, bi, (g, h, m))
                    y2i, a2i, b2i = calculate_yab(*calculate_yab(y2i, a2i,
b2i, (g, h, m)), (g, h, m))

                if yi == y2i:
                    # Step 5
```

```
aa, bb = (a2i - ai) % (m - 1), (bi - b2i) % (m - 1)
d = sympy.gcd(aa, m - 1)
if d < square_root(m - 1):
    for x in equation(aa, bb, m - 1):
        if pow(g, x, m) == h:
            return x
```

Оценки сложности рассмотренных алгоритмов

Сложность алгоритма Гельфонда-Шенкса составляет $O(\sqrt{B}\log B)$ операций в группе G. Объем использованной памяти составляет $O(\sqrt{B})$ ячеек.

Сложность алгоритма p-метод Полларда составляет $O\left(\sqrt{\ln(1/\varepsilon)}\sqrt{m}\right)$ операций в группе G.

Результаты тестирования программ

Оценим корректность работы алгоритмов. Для этого возьмем достаточно большую циклическую группу и на нескольких образующих и нескольких значениях степеней проверим результаты работы алгоритмов.

m	g	h	Алгоритм Гельфонда-Шенкса	p-метод Полларда
2163547	11	10012	682681	682681
2163547	11	100500	2069622	2069622
2163547	29	6789	1137605	1137605
2163547	28	184	1798464	1798464
89765387	5	79823	80999464	80999464
89765387	14	89765000	8104158	8104158
89765387	18	56473	28285821	28285821
89765387	18	2	19451523	19451523

4. Тестирование скорости работы алгоритмов

Протестируем скорость работы алгоритмов на 1000 запусков с различными параметрами, в ячейки таблицы запишем суммарное время работы (в секундах) на все запуски для каждого алгоритма.

m	g	h	Алгоритм Гельфонда-Шенкса	ρ-метод Полларда
5471	7	101	0.293 с	1.254 с
5471	7	1012	0.437 с	1.261 с
5471	7	5470	0.289 с	1.606 с
5471	7	1	0.189 с	0.466 с

Скорость работы алгоритма Гельфонда-Шенкса варьируется для различных чисел до 2 раз, случай $h = 1$ не многим быстрее других значений h. ρ-метод Полларда работает дольше в среднем в 3 раза, для любых h.