Front matter

title: “Отчет по Лабораторной работе №7 по предмету Математические основы защиты информации и кибер безопасности” author: “Лобов Михаил Сергеевич”

Generic otions

lang: ru-RU toc-title: “Содержание”

Bibliography

bibliography: bib/cite.bib csl: pandoc/csl/gost-r-7-0-5-2008-numeric.csl

Pdf output format

toc: true # Table of contents toc-depth: 2 lof: true # List of figures lot: true # List of tables fontsize: 12pt linestretch: 1.5 papersize: a4 documentclass: scrreprt ## I18n polyglossia polyglossia-lang: name: russian options: - spelling=modern - babelshorthands=true polyglossia-otherlangs: name: english ## I18n babel babel-lang: russian babel-otherlangs: english ## Fonts mainfont: IBM Plex Serif romanfont: IBM Plex Serif sansfont: IBM Plex Sans monofont: IBM Plex Mono mathfont: STIX Two Math mainfontoptions: Ligatures=Common,Ligatures=TeX,Scale=0.94 romanfontoptions: Ligatures=Common,Ligatures=TeX,Scale=0.94 sansfontoptions: Ligatures=Common,Ligatures=TeX,Scale=MatchLowercase,Scale=0.94 monofontoptions: Scale=MatchLowercase,Scale=0.94,FakeStretch=0.9 mathfontoptions: ## Biblatex biblatex: true biblio-style: “gost-numeric” biblatexoptions: - parentracker=true - backend=biber - hyperref=auto - language=auto - autolang=other* - citestyle=gost-numeric ## Pandoc-crossref LaTeX customization figureTitle: “Рис.” tableTitle: “Таблица” listingTitle: “Листинг” lofTitle: “Список иллюстраций” lotTitle: “Список таблиц” lolTitle: “Листинги” ## Misc options indent: true header-includes: -

- keep figures where there are in the text
    - # keep figures where there are in the text

Отчет по лабораторной работе №7

1. Цель работы
Цель данной лабораторной работы – изучить задачу дискретного логарифмирования в конечных полях, понять теоретические аспекты, лежащие в основе сложности этой задачи, ознакомиться с ρ-методом Полларда, который является одним из практических подходов к ее решению, а также реализовать алгоритм на практике. В итоге необходимо получить навык решения подобных задач и осознать значение дискретных логарифмов в криптографии.

2. Задание

- Изучить теоретические основы дискретного логарифмирования, конечных полей, групповой структуры $\mathbb{F}_p^*$ и понятия порядка элемента.
- Рассмотреть свойства и преимущества ρ-метода Полларда для решения задачи дискретного логарифмирования.
- Определить параметры p, a, b и с помощью реализованного кода найти такое x, что $a^x \equiv b \pmod{p}$.

3. Теоретическое введение

3.1. Дискретное логарифмирование и его значение

Дискретный логарифм — одна из фундаментальных задач в теории чисел, лежащая в основе множества криптографических протоколов с открытым ключом. Формально задача формулируется следующим образом:
Пусть p — большое простое число, $\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z}$ — конечное поле, а (_p^* = {1, 2, , p-1}) — мультипликативная группа этого поля. Если $a \in \mathbb{F}_p^*$ является образующим элементом (образующим циклическую подгруппу порядка $p-1$ или порядка некоторого делителя $p-1$), то задача дискретного логарифмирования в группе (_p^*) состоит в нахождении целого числа x, удовлетворяющего сравнению:

$$a^x \equiv b \pmod{p},$$

где даны a, b, p, а требуется найти x.

В отличие от классических логарифмов над вещественными числами, в конечных полях не существует простого метода решения. Наивный перебор занимает экспоненциальное время. Предполагается, что задача дискретного логарифмирования трудна для больших значений p, что обеспечивает безопасность схем Диффи-Хеллмана, алгоритмов Эль-Гамаля, схем подписи (например, DSA) и других криптосистем.

3.2. Конечные поля, кольца вычетов и циклические группы

Конечное поле (_p) определяется для простого p как множество классов вычетов (/p). Группа (_p^*) из $p-1$ ненулевых элементов является циклической. Существование первообразного корня a (т.е. элемента максимального порядка) гарантирует, что каждый элемент $b \in \mathbb{F}_p^*$ может быть представлен как $b = a^x \pmod{p}$ для некоторого x.

3.3. Сложность и алгоритмы решения дискретного логарифма

На данный момент не известны полиномиальные по количеству бит в p алгоритмы для решения дискретного логарифма над большими простыми модулями. Известны методы:
- Наивный перебор: ((p)) операций, практически не применим для больших p.
- Метод «Гигантских шагов и крошечных шагов» Шенкса: (()) операций.
- Алгоритм Полларда «ро» (ρ-метод) для дискретного логарифма: также работает примерно за (()) операций, но обычно проще в реализации и требует меньше памяти.

- Более продвинутые алгоритмы (индексный калькулятор, алгоритм Ленстры, NFS-DLog для очень больших значений), но они сложны в реализации и зависят от структуры модуля.

3.4. ρ-метод Полларда

Р-метод (ро-метод) Полларда по форме напоминает ρ-метод для факторизации чисел. Он использует идею поиска цикла в последовательности значений, генерируемых «случайным» отображением $f: \mathbb{F}_p^* \to \mathbb{F}_p^*$.

Идея состоит в следующем:
1. Определим функцию $f(c)$, которая в зависимости от значения c преобразует его либо умножением на a, либо на b или с добавлением некоторых операций. Например, ветвящееся отображение:

$$ f(c) = \begin{cases} a \cdot c \pmod{p}, & c < \frac{p}{2} \\ b c , & c \end{cases} $$

При этом мы отслеживаем логарифмы выражений относительно a. Это необходимо, чтобы впоследствии, найдя коллизию $c = d$, можно было записать два выражения для их логарифмов и решить уравнение для x.

2. Стартуем с некоторого значения $c = a^u b^v \pmod{p}$, где u, v выбираются случайно. Аналогично определяем $d = c$.

3. Применяем отображение f к c один раз за итерацию (медленный «черепаший» ход) и к d два раза за итерацию (быстрый «заячий» ход). Движение c и d по циклической последовательности, генерируемой f, продолжится до тех пор, пока не обнаружится коллизия $c = d$.

4. Когда найдена коллизия, мы имеем два разных представления полученного элемента через u, v и U, V, отвечающие за накопленные при движении показатели. Образуется уравнение:

$$a^u b^v \equiv a^U b^V \pmod{p}.$$

   Это уравнение позволяет выразить x через уже известные величины, решив сравнения по модулю порядка r (где r — порядок элемента a по модулю p).

Главное преимущество ρ-метода Полларда — это скорость ((())) и низкие затраты памяти по сравнению с методом Шенкса. Несмотря на то, что задача остаётся сложной, данный метод позволяет решать задачи дискретного логарифма для достаточно больших значений p, что без него было бы практически невозможно.

4. Выполнение лабораторной работы

4.1. Исходные данные

Для конкретных значений p, a, b, заданных преподавателем, необходимо найти x, такое что:

$$a^x \equiv b \pmod{p}.$$

Предположим, что p — простое число, a и b выбраны так, что задача не слишком велика для демонстрации.

4.2. Реализация ρ-метода Полларда

Ниже приведен пример кода на Julia, реализующий описанный выше подход. В коде используются функции:

- `funf(h, j, k)`: реализует ветвящееся отображение f, обновляя значение h в зависимости от условия $h < r$ или $h \geq r$. При этом изменяются счетчики j и k, которые отражают накопление логарифмических коэффициентов.
- `find_collision(c, d, u, v, U, V)`: организует процесс поиска цикла, применяя `funf` к c (один раз) и к d (два раза) на каждой итерации.
- `compute_x(u, v, U, V, r)`: решает полученное после нахождения коллизии сравнение по модулю r.

```
# Предположим, что p, a, b, r заданы заранее
# p - простое число
# a - элемент с порядком r по модулю p
# b - элемент для которого ищем x

function funf(h, j, k)
    if h < r
        j += 1
        return mod(a * h, p), j, k
    else
        k += 1
        return mod(b * h, p), j, k
    end
end

function invmod(a, m)
    g, x, _ = gcdx(a, m)
    if g != 1
        throw(ArgumentError("No inverse"))
    else
        return mod(x, m)
    end
end

function compute_x(u, v, U, V, r)
    delta_v = mod(v - V, r)
    delta_u = mod(U - u, r)
    if delta_v == 0
        return "No solutions"
    end
    delta_v_inv = invmod(delta_v, r)
    return mod(delta_u * delta_v_inv, r)
end

function find_collision(c, d, u, v, U, V)
```

```
    while c != d
        c, u, v = funf(c, u, v)
        d, U, V = funf(d, U, V)
        d, U, V = funf(d, U, V)
    end
    return c, d, u, v, U, V
end

# Инициализация
u, v = 2, 2
U, V = 2, 2
c = mod(a^u * b^v, p)
d = c
c, u, v = funf(c, u, v)
d, U, V = funf(d, U, V)
d, U, V = funf(d, U, V)

# Поиск коллизии
c, d, u, v, U, V = find_collision(c, d, u, v, U, V)

# Вычисление x
x = compute_x(u, v, U, V, r)
println("x = ", x)

# Проверка решения
if x != "No solutions" && mod(a^x, p) == b
    println("Проверка пройдена: a^x ≡ b (mod p)")
else
    println("Решений нет или проверка не пройдена.")
end
```

4.3. Пояснение шагов

1. **Инициализация:**
Задаются начальные значения u, v и U, V, а также начальное значение $c = a^u b^v \pmod p$. Параллельно переменная d начинает с того же значения, чтобы применить впоследствии «быстрый» ход.

2. **Функция funf:**
При каждом вызове `funf` определяет, к какому «сегменту» принадлежит текущее значение и обновляет его умножением либо на a, либо на b по модулю p, а также корректирует u, v, U, V (в данном случае через j, k).

3. **Поиск цикла (find_collision):**
Используется идея Флойда: одна переменная обновляется один раз за итерацию (медленно), другая — два раза (быстро). Когда значения совпадают ($c = d$), обнаружен цикл в последовательности. На этом этапе мы имеем два различных представления найденного элемента через параметры u, v и U, V.

4. **Решение сравнения (compute_x):**
   Используя полученные соотношения для логарифмов, формируем линейное уравнение по модулю r. Решение этого уравнения даёт нам искомый x.

5. **Проверка результата:**
   Подставляем найденный x в $a^x \pmod p$ и сравниваем с b. Если равно, значит логарифм найден верно.

5. Выводы

В ходе данной лабораторной работы было подробно рассмотрено теоретическое обоснование трудности задачи дискретного логарифмирования, изучены основные понятия конечных полей и циклических групп, а также подробно разобран ρ-метод Полларда.

Реализация алгоритма и успешная демонстрация нахождения дискретного логарифма показали на практике, каким образом можно применить теоретические знания для решения сложных задач. Это полезно в кибербезопасности, поскольку криптосистемы с открытым ключом, такие как Диффи-Хеллман, RSA, Эль-Гамаль и схемы электронной подписи, полагаются на сложность дискретного логарифма или факторизации. Понимание и умение реализовывать алгоритмы для решения таких задач (как ρ-метод Полларда) позволяют оценивать надёжность и прочность криптографических схем, а также тестировать их устойчивость к потенциальным атакам.

6. Список литературы

Pollard, 1974. Karaarslan E. Primality Testing Techniques and The Importance of Prime Numbers in Security Protocols (англ.) // ICMCA'2000: Proceedings of the Third International Symposium Mathematical & Computational Applications — Konya: 2000. — P. 280—287. Василенко, 2003, с. 60. Ишмухаметов, 2011, с. 53—55. Cohen, 2000, pp. 439. Montgomery, Silverman, 1990. Циммерман, Поль. Record Factors Found By Pollard's p-1 Method (англ.). Les pages des personnels du LORIA et du Centre Inria NGE. Дата обращения: 10 октября 2016. Архивировано 11 октября 2016 года. InriaForge: GMP-ECM (Elliptic Curve Method): Project Home. Дата обращения: 15 ноября 2012. Архивировано 21 июля 2012 года.