**OIB || lab\_1**

## файл с дополнительной информацией для лабораторной работы ##

**[Цель работы:]** цель данной лабораторной работы — изучение математических основ криптографии, в частности модульной арифметики, простых чисел, функций Эйлера, обратных элементов в кольце вычетов и дискретного логарифма. Эти понятия являются базовыми для построения криптографических алгоритмов, таких как RSA, Diffie-Hellman, ECC и другие.

**[Теоретические сведения:]** перед тем как перейти к конкретным математическим примитивам, важно понимать, зачем они нужны.

В криптографии часто используют числа из определённых математических структур, например, кольца вычетов и конечные поля. Эти структуры помогают строить криптографические системы, обеспечивая надёжность за счёт математической сложности операций.

**[Модульная арифметика:]** это арифметика по модулю n (то есть с остатками).

a – делимое, b – остаток | q – частное, r – остаток

a = b \* q + r, 0 <= r < b

*число r называется вычетом числа a по модулю b*, что записывается так:

a = r(mod b) или a = r(b): “a сравнимо с r по модулю b”

**[Шифр Цезаря:]** это простейший подстановочный шифр, в котором каждая буква заменяется на другую, сдвинутую в алфавите на фиксированное число позиций.

Формула шифрования. Для шифрования каждой буквы **x** используется формула:

E(x) = (x + k) mod N, где:

* K – ключ (сдвиг)
* N – кол-во символов в алфавите (например, 26 – в англ. Алф.)
* X – порядковый номер буквы (например, A = 0, B = 1 …)

Формула расшифрования.

D(x) = (x - k) mod N

Т.е. для расшифровки делается сдвиг в обратную сторону.

**Пример (Шифр Цезаря):**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **A** | **B** | **C** | **D** | **E** | **F** | **G** | **H** | **I** | **J** | **K** | **L** | **M** | **N** | **O** | **P** | **Q** | **R** | **S** | **T** | **U** | **V** | **W** | **X** | **Y** | **Z** |
| **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** | **16** | **17** | **18** | **19** | **20** | **21** | **22** | **23** | **24** | **25** |

* Исходный текст: HELLO
* Шифрованный текст (сдвиг k = 3): KHOOR

*Если сдвиг k = -3, то можно расшифровать текст обратно.*

Недостатки:

* Очень легко взломать **перебором** (26 вариантов для английского алфавита).
* Частотный анализ тоже помогает расшифровать текст.

**Вычеты и кольца вычетов**

В криптографии часто работают с числами по модулю. В связи с этим вводятся понятия вычетов и колец вычетов.

1. Вычеты: это остатки от деления чисел на фиксированное число (модуль).

Когда мы делим число **a** на **n**, у нас всегда остается остаток, который называется вычетом числа a по модулю **n**.

A ≡ b mod n - означает, что числа a и b дают одинаковый остаток при делении на n.

Пример:

Пусть n = 5. Тогда **любое число** можно заменить его вычетом по модулю 5:

* 8 / 5 = 1 (остаток 3), значит: 8 ≡ 3 mod 5
* 12 / 5 = 2 (остаток 2), значит: 12 ≡ 2 mod 5
* 25 / 5 (остаток 0), значит: 25 ≡ 0 mod 5

**Таким образом, числа 8, 13, 18, 23, … все эквивалентны числу 3 по модулю 5***(то есть, при делении этих чисел на 5, остаток всегда будет = 3)*

1. Кольцо вычетов.

Если взять **все возможные вычеты** по модулю n и ввести на них операции **сложения и умножения**, то получится **кольцо вычетов.**

Обозначение кольца вычетов:

Zn = {0, 1, 2, …, n - 1}

В этом множестве выполняются операции:

1. Сложение по модулю:  
   (a + b) mod n
2. Умножение по модулю:  
   (a \* b) mod n

Пример кольца вычетов Z5:

{0, 1, 2, 3, 4}

* 2 + 3 = 5 ≡ 0 mod 5
* 2 \* 3 = 6 ≡ 1 mod 5

В криптографии важны кольца вычетов, потому что они **являются основой для работы с шифрами**, включая RSA, Эль-Гамаля и другие алгоритмы.

**[Алгоритм Евклида]**

\**Взаимно простые числа* – числа, которые не имеют нетривиального общего делителя (отличный от 1)

Алгоритм Евклида – это быстрый метод нахождения наибольшего общего делителя (НОД) двух чисел. НОД – это наибольшее число, на которое делятся оба числа без остатка.

1. Основная идея

Если мы хотим найти НОД двух чисел a и b, то:

gcd(a, b) = gcd(b, a mod b)

Мы заменяем большее число на его остаток от деления на меньшее, и повторяем, пока не получим 0.

Последнее ненулевое число и есть НОД.

1. Пример работы алгоритма

Задача: Найти НОД (56, 15).

Шаги алгоритма:

1. 56 / 15 = 3, остаток 11 -> gcd(56, 15) = gcd(15, 11)
2. 15 / 11 = 1, остаток 4 -> gcd(15, 11) = gcd(11, 4)
3. 11 / 4 = 2, остаток 3 -> gcd(11, 4) -> gcd(4, 3)
4. 4 / 3 = 1, остаток 1 -> gcd(4, 3) -> gcd(3, 1)
5. 3 / 1 = 3, остаток 0 -> gcd(3, 1) –> gcd(1, 0) = 1

Ответ: gcd(56, 15) = 1, значит числа взаимно просты.

1. Применение алгоритма Евклида в криптографии

* **Взаимно простые числа**: если gcd(a, n) = 1, то числа a и n взаимно просты, что **важно для RSA**
* **Обратные элементы**: расширенный алгоритм Евклида используется в RSA и других криптосистемах для нахождения обратного по модулю числа.
* **Шифры и подписи**: например, в RSA модульные операции зависят от НОД.

1. Расширенный алгоритм Евклида

Помимо нахождения НОД, расширенный алгоритм Евклида находит коэффициенты x и y, такие что:

ax + by = gcd(a, b)

*\*Эти коэффициенты важны при нахождении обратного элемента по модулю.*

Пример (находим x и y для 30x + 21y = gcd(30, 21))

1. gcd(30, 21) = 3 (вычислили по обычному алгоритму)
2. Записываем через деления:

30 = 1 \* 21 + 9

21 = 2 \* 9 + 3

9 = 3 \* 3 + 0

Теперь выражаем 3 через 30 и 21:  
 3 = 21 – 2 \* 9

9 = 30 – 1 \* 21

Подставляем 9 во второе уравнение:

3 = 21 – 2 \* (30 – 1 \* 21)

3 = 21 – 2 \* 30 + 2 \* 21

3 = - 2 \* 30 + 3 \* 21

Ответ: x = -2, y = 3.

*Это значит, что обратный элемент 30 по модулю 21 можно выразить через линейную комбинацию*

1. Обычный алгоритм Евклида быстро находит НОД
2. Расширенный алгоритм Евклида помогает находить обратные элементы.

**[Простые и составные числа]**

1. Простые и составные числа.

Число называется **простым**, если у него есть только два делителя: 1 и оно само. Например, 2, 3, 5, 7, 11 — это простые числа.

Число называется **составным**, если у него есть другие делители, кроме 1 и самого себя. Например, 4, 6, 8, 9, 12 — составные числа, потому что они делятся на другие числа (например, 4 делится на 2).

1. Проверка числа на простоту.

Разложение на множители — самый очевидный способ узнать, является ли число простым, но он слишком затратный. Поэтому существуют более быстрые методы:

* Малая теорема Ферма: если число N простое, то для любого числа a, не делящегося на N выполняется:

a^(N - 1) ≡ 1 (mod N)

*Если хотя бы для одного a это не выполняется, N точно составное.*

* Числа Кармайкла: есть особые составные числа, которые ведут себя как простые в проверке по теореме Ферма. Их сложнее выявлять. (Они с условием, что НОД(a, N) = 1)

1. Алгоритм Миллера-Рабина.

Этот алгоритм — один из самых быстрых и популярных способов проверки, является ли число N простым. Он основан на модификации малой теоремы Ферма и позволяет с высокой вероятностью выявлять составные числа.

1. **Основная идея алгоритма**

Малая теорема Ферма утверждает, что если N — простое, то для любого числа a, не делящегося на N, выполняется:

a^(N-1) ≡ 1 (mod N)

Однако бывают составные числа, для которых это тоже выполняется (например, числа Кармайкла). Поэтому Миллер и Рабин предложили более строгую проверку.

**Разложение N – 1**

Любое нечетное число N можно представить в виде:

N – 1 = 2^s \* t, где:

* t – нечетное число,
* s – степень двойки.

Пример: для N = 37 имеем:

37 – 1= 36 = 2\*2 \* 9 (s = 2, t = 9)

1. **Как работает алгоритм?**

Мы выбираем случайное число a (называемое “свидетелем”), и проверяем:

1. Вычисляем a^t (mod N).

Если получилось 1, то число N может быть простым.

1. Проверяем последовательность квадратов:

* Вычисляем a^(2t) (mod N), а затем a^(4t) (mod N) и так далее, пока не дойдем до a^(2^(s-1) \* t) (mod N)
* Если на каком-то шаге результат становится -1 (или то же самое, что N - 1), то число может быть простым

1. Если оба условия не выполняются, N точно составное.

Пример: проверим N = 37 с a = 5

* N – 1 = 36 = 2^9 \* 9, значит t = 9, s = 2
* Считаем 5^9 mod (37): 5^9 (mod 37) = 5
* Затем 5^18 ≡ -1 (mod 37)
* Так как появилось -1, алгоритм не говорит, что число составное.

1. **Вероятностный характер алгоритма**

Если хотя бы для одного a проверка не проходит, число точно составное.

Но если для нескольких a проверка прошла, N скорее всего простое.

Чем больше разных a мы проверяем, тем выше вероятность, что тест дал правильный ответ.

* Если мы проверили k случайных a, то вероятность ошибки не превышает 1/4k.
* Например, если k=10, то вероятность ошибки — 1/1048576 (очень мала).

1. **Алгоритм Миллера-Рабина шаг за шагом**

Вход: число N, количество проверок: k.

1. Разложить N – 1 как 2^s \* t (найти s и t)
2. Выполнить k раз:
   1. Выбрать случайное число a, 1 <a <N
   2. Вычислить a^t (mod N).
   3. Если a^t ≡ 1 (mod N), перейти к следующей проверке.
   4. Последовательно вычислить a^(2t), a^(4t), …, a^(2\*(s-1) \*t) (mod N)
   5. Если на каком-то шаге результат стал -1 (или N - 1), перейти к следующей проверке.
   6. Если ничего из этого не выполнено, N составное.
3. Если ни одна проверка на выявила составное число, N считается простым с высокой вероятностью.

\*\* **a^t (mod N) простыми словами:** *мы возводим число a в степень t и берем остаток от деления на N.*

Пример: 3^4 (mod 5)

1. Сначала считаем обычное 3^4: 81
2. Теперь делим 81 на 5 и берем остаток: 81 / 5 = 16 (целая часть), 81 – 80 = 1 (остаток)

То есть:

**3^4 ≡ 1 (mod 5)** – что и есть ответ.

**[Поиск числа, обратного по модулю]**

Обратное число по модулю m для числа e — это такое число d, что выполняется условие:

e\*d ≡ 1 (mod m)

То есть при умножении e на d результат дает 1 по модулю m. Это важно в криптографии, например, при вычислении закрытого ключа в алгоритме RSA.

Как найти обратное число по модулю?

Для нахождения обратного числа используется расширенный алгоритм Евклида.

* Шаг 1: используем алгоритм Евклида для нахождения НОД

Алгоритм Евклида используется для нахождения наибольшего общего делителя (НОД) двух чисел. Он работает так:

1. Берем два числа: s и t, где s> t.
2. Делим s на t, записываем частное Q и остаток t1:

S = Q1t + t1

1. Повторяем для t и t1:

t = Q2t1 + t2

1. Повторяем, пока остаток не станет нулем. Последний ненулевой остаток – это НОД.

* Шаг 2: Расширенный алгоритм Евклида (нахождение коэффициентов)

Если НОД(e, m) = 1, можно выразить 1 как:

ex + my = 1

Число x будет искомым обратным числом d.

Мы используем обратное распространение из алгоритма Евклида, чтобы выразить 1 через e и m.

Пример: Найдем 3^(-1) (mod 7)

1. Применяем алгоритм Евклида:

7 = 2 \* 3 + 1

3 = 3 \* 1 + 0

НОД – это 1, значит обратное число существует

1. Расписываем уравнение:

1 = 7 – 2 \* 3

Переписываем:

1 = -2 \* 3 + 1 \* 7

Значит: x = -2, но так как остатки должны быть положительными, приводим к положительному остатку:

-2 ≡ 5 (mod 7)

Ответ: 3^(-1) ≡ 5 (mod 7)

**[Алгоритм шифрования RSA]**

RSA — это один из самых популярных методов шифрования. Он основан на математических свойствах простых чисел и операций с большими числами.

1. **Как работает RSA?**
2. Создание ключей:
   * Выбираем два больших простых числа p и q.
   * Перемножаем их: n = p \* q. Это число становится частью открытого ключа.
   * Считаем функцию Эйлера: φ(n) = (p - 1) \* (q - 1).
   * Выбираем число e, которое взаимно просто с φ(n). Это показатель шифрования.
   * Выбираем число d, обратное к e по модулю φ(n). Это показатель дешифрования.
3. Открытый и закрытый ключ:
   * Открытый ключ: (n, e). Им шифруют данные.
   * Закрытый ключ: (n, d). Им расшифровывают данные.
4. Шифрование:
   * Чтобы зашифровать число x, вычисляем:

y = x^e (mod n)

* + Полученный результат y отправляется получателю.

1. Дешифрование:
   * Чтобы расшифровать y, вычисляем:

x = y^d (mod n)

* + В результате получаем исходное число x.

1. **Как используется RSA?**
   * Передача зашифрованных сообщений: Один человек использует открытый ключ получателя для шифрования сообщения. Только владелец закрытого ключа может его расшифровать.
   * Цифровая подпись: Отправитель подписывает сообщение своим закрытым ключом, а получатель проверяет подпись с помощью открытого ключа.
2. **Безопасность RSA**

RSA держится на сложности разложения больших чисел на множители. Если n очень большое (например, 2048 бит), разложить его на p и q почти невозможно за разумное время.

**[Ранцевая криптосистема Меркля-Хеллмана]**

1. Введение в задачу рюкзака

Представьте, что у нас есть рюкзак с ограниченной вместимостью, и набор предметов, у каждого из которых есть вес. Нужно выбрать такие предметы, чтобы их суммарный вес не превышал вместимость рюкзака, а сумма стоимостей (ценностей) была максимальной.

В криптографии мы заменяем «стоимости» на двоичные числа и рассматриваем ситуацию, когда задача рюкзака используется для шифрования.

1. Основная идея шифрования

Криптосистема Меркля-Хеллмана использует сверхвозрастающую последовательность чисел. Это последовательность, где каждый следующий элемент больше суммы всех предыдущих. Например:

2, 3, 6, 13, 27, 52, 105, 210

Главная идея в том, что такую последовательность можно легко разложить обратно на сумму элементов (решить задачу рюкзака), но, если ее замаскировать, задача станет сложной.

1. Генерация ключей

Чтобы создать криптографическую систему, необходимо сделать следующее:

1. Выбрать сверхвозрастающую последовательность S = {s1, s2, …, sn}
2. Выбрать модуль m, который больше суммы всех элементов последовательности.
3. Выбрать число n, которое взаимно просто с m.
4. Преобразовать исходную последовательность по формуле:

Pi = (si \* n) (mod m)

Полученные числа образуют открытый ключ.

Пример:

* Берем сверхвозрастающую последовательность: {1, 3, 5, 10, 21, 42, 84, 167}
* Выбираем m = 335, n = 19
* Строим открытый ключ:

(1 \* 19) mod 335 = 19, (3 \* 19) mod 335 = 57, …, (167 \* 19) mod 335 = 158

Итог:

* Закрытый ключ: {1, 3, 5, 10, 21, 42, 84, 167}
* Открытый ключ: {19, 57, 95, 190, 64, 128, 256, 158}

1. Шифрование сообщения

Чтобы зашифровать сообщение:

1. Переводим его в двоичный код. Например, буква O (11001110).
2. Берем соответствующие элементы открытого ключа, где в двоичном коде стоит 1 и складываем их:

19 + 57 + 64 + 128 + 256 = 524

1. Это число (524) – зашифрованное сообщение (шифрограмма).

Пример для слова “ОИБ”: (таблица в пособии)

* О -> 524
* И -> 140
* Б -> 234

1. Расшифровка сообщения
2. Нужно найти число n^(-1), обратное к n по модулю m, используя расширенный алгоритм Евклида. В нашем примере n^(-1) = 194 (потому что 19 \* 194 mod 335 = 1).
3. Домножаем шифртекст на n^(-1) по модулю m:

524 \* 194 mod 335 = 151

1. Теперь разлагаем 151 по закрытому ключу:

151 = 1 + 3 + 21 + 42 + 84

Значит двоичный код: **11001110,** что соответствует букве O.

*\*Повторяем для остальных символов и получаем расшифрованное сообщение.*

**Итог:**

* Ранцевая криптосистема строится на сложной задаче рюкзака, но с использованием сверхвозрастающей последовательности, которую легко решить, если знать секрет.
* Открытый ключ — это измененная версия исходной последовательности, а закрытый ключ позволяет легко расшифровать сообщение.
* Однако со временем были найдены эффективные методы взлома, поэтому сегодня Меркль-Хеллман не используется в реальной криптографии.

**[Расширенный алгоритм Евклида]**

Расширенный алгоритм Евклида — это расширение обычного алгоритма Евклида, который не только находит наибольший общий делитель (НОД) двух чисел aa и bb, но и выражает его в виде линейной комбинации:

**gcd(a, b) = a\*x + b\*y**

где x и y - целочисленные коэффициенты (так называемые коэффициенты Безу).

**Основная идея**

* **Обычный алгоритм** Евклида использует деление с остатком: gcd(a,b)=gcd(b,a mod  b), пока остаток не станет 0.
* **Расширенный алгоритм** идет дальше: на каждом шаге он отслеживает, как текущий НОД может быть выражен через исходные a и b, обновляя коэффициенты x и y рекурсивно.

Принцип работы:

1. **Базовый случай**:

Если b = 0, то gcd(a, 0) = a, и линейная комбинация тривиальна:

a = a \* 1 + 0 \* 0, то есть:

**x = 1, y = 0**

1. Рекурсивный шаг:

Для gcd(a,b) мы переходим к gcd(b,a mod  b). Пусть:

* a = qb + r (где q = [a/b], r = a mod b)
* gcd(b, r) = b\*x1 + r\*y1 (предположим, что мы уже нашли x1 и y1 для следующего шага)

Тогда подставим r = a – qb в выражение:

gcd(a, b) = gcd(b, r) = b\*x1 + (a - qb)\*y1

Перегруппируем:

Gcd(a, b) = b\*x1 + a\*y1 - q\*b\*y1 = a\*y1 + b(x1 – q\*y1)

Таким образом:

* **x = y1**
* **y = x1 – q\*y1**

Процесс повторяется рекурсивно, пока не дойдет до базового случая, а затем коэффициенты "поднимаются" обратно через все шаги.

Пример:

Найдем gcd(51581124,911) и коэффициенты x и y:

**Шаг 1: Прямой ход – обычный алгоритм Евклида**

1. 51581124 / 911 = 56620 (остаток: 304)

51581124 = 56620 \* 911 + 304

1. 911 / 304 = 2 (остаток: 303)

911 = 2 \* 304 + 303

1. 304 / 303 = 1 (остаток: 1)

304 = 1 \* 303 + 1

1. 303 / 1 = 303 (остаток 0)

303 = 1 \* 303 + 0

=> GCD = 1

**Шаг 2: Обратный ход – расширенный алгоритм Евклида**

1. Из Шага 1: пункт 3: 304 = 1 \* 303 + 1

Это первое выражение, где 1 выражено через 304 и 303

1. Из пункта 2: 911 = 2 \* 304 + 303, выразим 303:

303 = 911 – 2 \* 304

Подставим 303 в 1 = 304 – 1 \* 303:

**1 = 304 – 911 + 2 \* 304 = 3 \* 304 – 911**

Теперь 1 выражено через 911 и 304

1. Из пункта 1: 51581124 = 56620 \* 911 + 304, выразим 304:

304 = 51581124 – 56620 \* 911

Подставим 304 в 1 = 3 \* 304 – 911:

1 = 3 \* 51581124 – 3 \* 56620 \* 911 – 911

Итого:

1 = **3** \* 51581124 **– 169861** \* 911

Трассировка выполнения для a = 51581124, b = 911

1. Первый вызов: extended\_gcd(51581124, 911, &x, &y)

* a = 51581124, b = 911
* 51581124 % 911 = 304
* Рекурсия: extended\_gcd(911, 304, &x1, &y1)

1. Второй вызов: extended\_gcd(911, 304, &x1, &y1)

* a = 911, b = 304
* 911 % 304 = 303
* Рекурсия: extended\_gcd(304, 303, &x1, &y1)

1. Третий вызов: extended\_gcd(304, 303, &x1, &y1)

* a = 304, b = 303
* 304 % 303 = 1
* Рекурсия: extended\_gcd(303, 1, &x1, &y1)

1. Четвертый вызов: extended\_gcd(303, 1, &x1, &y1)

* a = 303, b = 1
* 303 % 1 = 0
* Рекурсия: extended\_gcd(1, 0, &x1, &y1)

1. Пятый вызов: extended\_gcd(1, 0, &x1, &y1) – **Базовый случай**

* b = 0
* \*x1 = 1, \*y1 = 0
* Возвращаем gcd = 1

1. Обратный ход:

* Из extendedgcd(303,1):
  + x1 = 1, y1 = 0
  + q = 303 / 1 = 303
  + ∗x = y1 = 0
  + ∗y = x1 − (303 / 1) \* y1 = 1 – 303 \* 0 = 1
  + Возвращаем gcd = 1
* Из extendedgcd(304,303):
  + x1 = 0, y1 = 1
  + q = 304 / 303 = 1
  + ∗x = y1 = 1
  + ∗y = x1 − (304 / 303) \* y1 = 0 – 1 \* 1 = -1
  + Возвращаем gcd = 1
* Из extendedgcd(911,304):
  + x1 = 1, y1 = -1
  + q = 911 / 304 = 2
  + ∗x = y1 = -1
  + ∗y = x1 − (911 / 304) \* y1 = 1 – 2 \* (-1) = 1 + 2 = 3
  + Возвращаем gcd = 1
* Из extendedgcd(51581124,911):
  + x1 = -1, y1 = 3
  + q = 51581124/ 911 = 56620
  + ∗x = y1 = 3
  + ∗y = x1 − (51581124/ 911) \* y1 = -1 – 56620 \* 3 = -1 – 169860 = -169861
  + Возвращаем gcd = 1