НАО «Восточно-Казахстанский технический университет имени Д. Серикбаева»

Дисциплина «Теоретико-числовые методы в криптографии»

САМОСТОЯТЕЛЬНЫЙ ПРАКТИКУМ К ЛЕКЦИИ 15

Простота, умножение и возведение в степень

Методические указания и задания для самостоятельной работы

Статус работы: дополнительный самостоятельный практикум к теме 15. Аудиторные практические часы по этой теме рабочей программой не предусмотрены. Объём выполнения и срок сдачи определяет преподаватель.

1 Цель и результаты работы

Научиться распознавать числа Кармайкла по известной факторизации, сравнивать проверки Ферма и Миллера — Рабина, выполнять умножение Карацубы и двоичную модульную степень. В отчёте требуется объяснить математический смысл результатов и показать промежуточные вычисления.

Сначала изучите методические сведения и разобранные примеры раздела 3. Затем запустите программные примеры раздела 4 и переходите к своему варианту. Для работы достаточно Python 3 и стандартной библиотеки; дополнительные пакеты не нужны. Все аргументы функций — целые числа.

2 Обозначения и правила вычислений

Число n исследуется на простоту и свойство Кармайкла. Числа x и y используются в заданиях на умножение. В задании на степень требуется найти остаток $x^e \bmod n$. Двоичную запись показателя читают слева направо, начиная со старшего бита.

Основания 2 и 3 для сравнения тестов заданы одинаково во всех вариантах. В каждой проверке сначала вычисляют НОД. Полное разложение на простые множители служит независимой точной проверкой итоговой классификации.

Для подсчёта оснований используется множество

$$U_n = \{a : 2 \le a \le n-2,\ \gcd(a,n) = 1\}.$$

Буквой U обозначается его мощность, F — число оснований из него, проходящих проверку Ферма, R — число оснований, проходящих раунд Миллера — Рабина. Слова «проходит проверку» описывают результат данного теста, а не доказательство простоты.

3 Методичка с разобранными примерами

3.1 Как применять критерий Корсельта

Разложите число на простые множители с кратностями. Отдельно отметьте составность и отсутствие повторных множителей. Для каждого различного простого делителя вычислите остаток от деления $n-1$ на $p-1$. Число Кармайкла получается только при одновременном выполнении всех трёх требований: составность, свобода от квадратов и нулевые остатки.

Пример. Для числа 561 имеем $561 = 3 \cdot 11 \cdot 17$. Все множители различны и просты. Деления $560:2$, $560:10$ и $560:16$ дают соответственно 280, 56 и 35 без остатка. Значит, 561 — число Кармайкла.

Для 169 получаем 13^2 и $168 \bmod 12 = 0$. Делимость показателя выполнена, но квадратный множитель исключает свойство Кармайкла. Для простого числа 97 остальные формальные проверки тоже проходят, однако отсутствует необходимая составность.

Не вычисляйте значение функции Кармайкла как наименьшее общее кратное чисел $p-1$ для произвольной факторизации с повторениями: такая замена в общем случае неверна. В этой работе достаточно непосредственных условий критерия Корсельта.

3.2 Проверки Ферма и Миллера — Рабина

Для Ферма вычислите $g = \gcd(a, n)$ и $f = a^{n-1} \bmod n$. При $g > 1$ уже найден делитель. При $g = 1$ остаток, отличный от 1, доказывает составность; остаток 1 означает лишь прохождение проверки.

Для Миллера — Рабина запишите $n - 1 = 2^s d$ с нечётным d. Начните с $a^d \bmod n$ и последовательно возводите остаток в квадрат. Раунд пройден, если начальный остаток равен 1 либо среди остатков с индексами от 0 до $s-1$ встретилось $n-1$. Если после другого остатка возникла 1, число составное; дальнейшие квадраты уже не нужны.

Пример. Для 561 и основания 2 НОД равен 1, а проверка Ферма даёт 1. В сильном тесте $s = 4$, $d = 35$; цепочка — 263, 166, 67, 1. Ни начальная единица, ни остаток 560 не встретились. Следовательно, раунд не пройден.

Если цепочка дала нетривиальный корень z из единицы, вычислите НОД чисел $z-1$ и $z+1$ с модулем. В примере $z = 67$ даёт делители 33 и 17. Для основания 3 вычислять цепочку уже не требуется, поскольку НОД равен 3. Остаток Ферма при этом равен 375; он приведён для сравнения, хотя делитель найден раньше.

3.3 Полный перебор оснований

Для малых учебных чисел можно перебрать всё множество U_n. В примере $\varphi(561) = (3-1)(11-1)(17-1) = 320$, поэтому после исключения двух тривиальных оснований получаем $U = 318$. Итоговые количества: $F = 318$, $R = 8$. Доли равны 1 и $8/318 = 4/159$ соответственно.

При прохождении проверки Ферма всеми основаниями из этого множества необходимо ещё подтвердить составность, прежде чем назвать число числом Кармайкла. Простые числа тоже проходят все проверки. Для составного числа основание, дающее прохождение, называют «ложным свидетелем»; для простого числа это название не применяют.

Для случайного выбора оснований используется другое множество: все целые от 2 до $n-2$, включая необратимые остатки. При независимом равномерном выборе вероятность пропустить фиксированное составное число за k раундов не превосходит 4^{-k}. Чтобы получить границу 2^{-80}, достаточно 40 раундов. Полный перебор в этом разделе является точным экспериментом; он не требует случайных чисел.

3.4 Умножение: столбик и один шаг Карацубы

Для произведения 1234 и 5678 частичные произведения по цифрам второго множителя, начиная с младшей, равны 9872, 86380, 740400 и 6170000. Их сумма — 7006652. В обычной схеме выполнено 16 произведений пар десятичных цифр; сложения и переносы считаются отдельно.

Для Карацубы возьмём $B = 100$. Старшие части равны 12 и 56, младшие — 34 и 78. Вычисляем три произведения:

$$z_0 = 34 \cdot 78 = 2652, \quad z_2 = 12 \cdot 56 = 672, \quad h = 46 \cdot 134 = 6164.$$

Средний коэффициент равен $z_1 = h - z_0 - z_2 = 2840$. Сборка даёт

$$xy = 672 \cdot 100^2 + 2840 \cdot 100 + 2652 = 7006652.$$

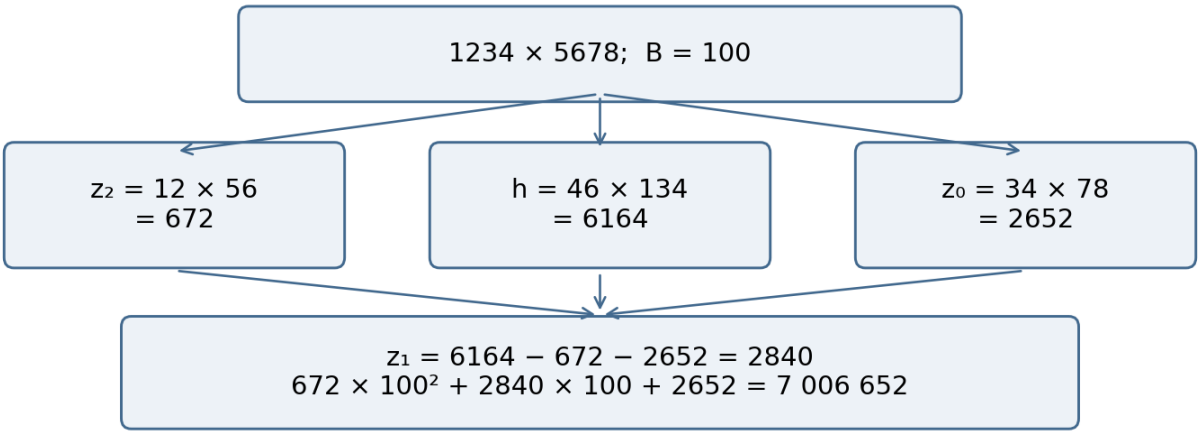


Рисунок 1. Один уровень алгоритма Карацубы в основании 100.

Три произведения частей не равны по стоимости трём произведениям цифр. В коде ниже первая функция показывает один уровень, а вторая повторяет разбиение рекурсивно. Измерять время на четырёхзначных числах для доказательства асимптотики не требуется.

3.5 Двоичное возведение в степень

Вычислим $7^{181} \bmod 561$. Показатель в двоичной системе — 10110101. Учитываем старшую единицу начальным присваиванием $r = 7$. Каждый следующий бит требует квадрата, а единичный бит — ещё одного умножения на 7.

Таблица 1. Образец полной трассировки.

Префикс	Остаток квадрата	После обработки бита
1	—	7
10	49	49
101	157	538
1011	529	337
10110	247	247
101101	421	142
1011010	529	529
10110101	463	436

В столбце квадрата указан остаток до возможного умножения на основание. Для строки начального присваивания квадрат не вычисляется. Результат — 436; выполнены 7 квадратов и 4 дополнительных умножения.

Если положительный показатель имеет t битов и w единиц, счётчики равны $S = t - 1$ и $M = w - 1$. Последовательная схема с начальным значением a потребовала бы $e - 1$ умножений. Для нулевого показателя наша функция возвращает $1 \bmod m$ без умножений; взаимная простота основания с модулем для самой функции не нужна.

4 Программные примеры

Скопируйте блоки раздела в один файл в указанном порядке. Сообщения об ошибках в коде описывают допустимые входные данные. `pow(a, e, m)` используется как готовая точная операция в

тестах простоты; собственное двоичное возведение реализовано отдельно, чтобы показать ход вычислений.

4.1 Импорт и факторизация

```
from math import gcd
from secrets import randbelow
```

Функция возвращает пары «простой множитель, кратность». Делители выделяются до тех пор, пока их квадрат не превысит оставшуюся часть числа.

```
def factor_pairs(n):
    if n < 2:
        raise ValueError("n must be at least 2")
    out = []
    p = 2
    while p * p <= n:
        k = 0
        while n % p == 0:
            n //= p
            k += 1
        if k:
            out.append((p, k))
        p = 3 if p == 2 else p + 2
    if n > 1:
        out.append((n, 1))
    return out
```

Следующая функция возвращает факторизацию, признак свободы от квадратов, пары «простой делитель, остаток показателя» и итоговый признак Кармайкла. Простое число не принимается за число Кармайкла.

```
def korselt(n):
    factors = factor_pairs(n)
    composite = sum(k for p, k in factors) > 1
    square_free = all(k == 1 for p, k in factors)
    residues = [(p, (n - 1) % (p - 1))
                for p, k in factors]
    carmichael = (composite and square_free
                  and all(r == 0 for p, r in residues))
    return factors, square_free, residues, carmichael
```

4.2 Один раунд и перебор оснований

Результат `mr_round` содержит пять элементов: признак прохождения, s, d, цепочку остатков и НОД основания с модулем. Пустая цепочка означает, что делитель найден сразу. Функция останавливается при первом решающем остатке, поэтому длина цепочки может быть меньше s.

```
def mr_round(n, a):
    if n < 5 or n % 2 == 0 or not 2 <= a <= n - 2:
        raise ValueError("odd n >= 5; 2 <= a <= n-2")
    d, s = n - 1, 0
    while d % 2 == 0:
        d //= 2
        s += 1
    g = gcd(a, n)
    if g > 1:
        return False, s, d, [], g
    x = pow(a, d, n)
    chain = [x]
    if x in (1, n - 1):
        return True, s, d, chain, g
    for _ in range(s - 1):
        x = x * x % n
        chain.append(x)
        if x == n - 1:
            return True, s, d, chain, g
        if x == 1:
            return False, s, d, chain, g
    return False, s, d, chain, g
```

Перебор ниже считает только взаимно простые основания, исключая 1 и $n-1$. Ограничение 100000 отделяет учебный полный перебор от задачи работы с большими параметрами.

```
def count_unit_bases(n):
    if n < 5 or n % 2 == 0 or n > 100000:
        raise ValueError("odd 5 <= n <= 100000")
    units = fermat = strong = 0
    for a in range(2, n - 1):
        if gcd(a, n) != 1:
            continue
        units += 1
        fermat += pow(a, n - 1, n) == 1
        strong += mr_round(n, a)[0]
    return units, fermat, strong
```

4.3 Несколько случайных раундов

В функции используются независимые равномерные выборы с возвращением. Значение `True` означает прохождение всех раундов; значение `False` означает, что число не простое. Отдельно обработаны числа меньше 2, малые простые и чётные числа. Этот пример нужен для понимания вероятностной гарантии; в таблицах вариантов используются фиксированные основания и полный перебор.

```
def probable_prime(n, k=40):
    if k < 1:
        raise ValueError("k must be positive")
    if n < 2:
        return False
    if n in (2, 3):
        return True
    if n % 2 == 0:
        return False
    for _ in range(k):
        a = 2 + randbelow(n - 3)
        if not mr_round(n, a)[0]:
            return False
    return True
```

4.4 Два алгоритма умножения

В `school_mul` цифры записываются от младшей к старшей. Первые два цикла накапливают произведения цифр, следующий цикл выполняет переносы. Возвращаются произведение, число произведений пар цифр и частичные произведения по цифрам второго множителя с учётом разрядов. Последний список служит для ручной проверки и не входит в счётчик основного алгоритма.

```
def school_mul(x, y):
    if x < 0 or y < 0:
        raise ValueError("nonnegative inputs required")
    a = [int(c) for c in str(x)[::-1]]
    b = [int(c) for c in str(y)[::-1]]
    digits = [0] * (len(a) + len(b))
    for i, u in enumerate(a):
        for j, v in enumerate(b):
            digits[i + j] += u * v
    for i in range(len(digits) - 1):
        digits[i + 1] += digits[i] // 10
        digits[i] %= 10
    value = sum(v * 10**i for i, v in enumerate(digits))
    partials = [x * v * 10**j for j, v in enumerate(b)]
    return value, len(a) * len(b), partials
```

Один уровень Карацубы возвращает значения в порядке: z_0, z_2, h, z_1, итоговое произведение. Основание 100 используется во всех индивидуальных вариантах.

```
def karatsuba_step(x, y, B=100):
    if x < 0 or y < 0 or B < 2:
        raise ValueError("x,y >= 0; B >= 2")
    x1, x0 = divmod(x, B)
    y1, y0 = divmod(y, B)
    z0 = x0 * y0
    z2 = x1 * y1
    h = (x0 + x1) * (y0 + y1)
    z1 = h - z0 - z2
    product = z2 * B**2 + z1 * B + z0
    return z0, z2, h, z1, product
```

Рекурсивная версия выбирает длину частей автоматически. Десятичное представление здесь используется для наглядного учебного разбиения. Встроенное целочисленное умножение Python привлекается в базовом случае; код не является измерением скорости внутренней реализации Python.

```
def karatsuba(x, y):
    if x < 0 or y < 0:
        raise ValueError("nonnegative inputs required")
    if min(x, y) < 10:
        return x * y
    half = max(len(str(x)), len(str(y))) // 2
    B = 10**half
    x1, x0 = divmod(x, B)
    y1, y0 = divmod(y, B)
    z0 = karatsuba(x0, y0)
    z2 = karatsuba(x1, y1)
    h = karatsuba(x0 + x1, y0 + y1)
    return z2 * B**2 + (h - z0 - z2) * B + z0
```

4.5 Модульная степень с трассировкой

Возвращаются результат, список строк трассировки, число квадратов и число дополнительных умножений. Каждая строка содержит двоичный префикс, остаток после квадрата и остаток после обработки бита. В первой строке `None` обозначает отсутствие операции квадрата.

```
def pow_mod_trace(a, e, m):
    if e < 0 or m < 1:
        raise ValueError("e >= 0; m >= 1")
    if e == 0:
        return 1 % m, [], 0, 0
    a %= m
    bits = bin(e)[2:]
    r = a
    prefix = "1"
    trace = [(prefix, None, r)]
    squares = products = 0
    for bit in bits[1:]:
        squared = r * r % m
        r = squared
        squares += 1
        if bit == "1":
            r = r * a % m
            products += 1
        prefix += bit
        trace.append((prefix, squared, r))
    return r, trace, squares, products
```

4.6 Проверка разобранных примеров

Запустите следующий блок после определений функций. Его вывод должен совпасть с приведённым ниже. Две функции умножения и встроенная операция дают одинаковый результат.

```
f, sf, rem, carm = korselt(561)
print("Factors:", f)
print("Square free:", sf, "Carmichael:", carm)
print("Remainders:", rem)
print("Fermat:", pow(2, 560, 561), pow(3, 560, 561))
print("MR:", mr_round(561, 2))
print("Bases:", count_unit_bases(561))
print("School:", school_mul(1234, 5678))
print("Karatsuba:", karatsuba_step(1234, 5678))
print("Recursive:", karatsuba(1234, 5678))
r, trace, squares, products = pow_mod_trace(7, 181, 561)
print("Power:", r, squares, products)
assert r == pow(7, 181, 561)
print("Edge:", pow_mod_trace(9, 0, 1)[0])
```

Ожидаемый вывод

```
Factors: [(3, 1), (11, 1), (17, 1)]
Square free: True Carmichael: True
Remainders: [(3, 0), (11, 0), (17, 0)]
Fermat: 1 375
MR: (False, 4, 35, [263, 166, 67, 1], 1)
Bases: (318, 318, 8)
School: (7006652, 16, [9872, 86380, 740400, 6170000])
Karatsuba: (2652, 672, 6164, 2840, 7006652)
Recursive: 7006652
Power: 436 7 4
Edge: 0
```

Для своего варианта замените только исходные данные в первой строке следующего блока. Вывод трассировки позволяет проверить каждую строку ручного решения.

```
# Replace these values with your variant.
n, x, y, e = 561, 1234, 5678, 37
print("Korselt:", korselt(n))
for a in (2, 3):
    print("Base:", a, gcd(a, n), pow(a, n - 1, n))
    print("MR:", mr_round(n, a))
print("Bases:", count_unit_bases(n))
print("School:", school_mul(x, y))
print("Karatsuba:", karatsuba_step(x, y))
assert karatsuba(x, y) == x * y
r, trace, S, M = pow_mod_trace(x, e, n)
print("Power:", r, "S:", S, "M:", M)
for row in trace:
    print(row)
assert r == pow(x, e, n)
```

5 Индивидуальные задания

Задание 1. Факторизация и критерий Корсельта. Разложите n на простые множители, подтвердите простоту найденных множителей. Для каждого различного простого делителя выпишите остаток $(n-1) \bmod (p-1)$. Укажите составность, свободу от квадратов и вывод о свойстве Кармайкла. При отрицательном ответе назовите невыполненное условие.

Задание 2. Сравнение проверок. Для оснований 2 и 3 вычислите НОД и остаток Ферма. Затем выполните раунд Миллера — Рабина: выпишите s, d и фактически вычисленную цепочку до остановки. Если НОД больше 1, укажите найденный делитель и оставьте цепочку пустой. При появлении нетривиального корня из единицы найдите делители через НОД. Сопоставьте результаты с точной факторизацией.

Задание 3. Перебор оснований. Программно найдите количества U, F, R и запишите точные доли F/U и R/U. Проверьте знаменатель с помощью функции Эйлера. Объясните, почему результаты согласуются с заданием 1. Отдельно укажите число независимых случайных раундов, достаточное для границы ошибки 2^{-80}, и условия применимости этой оценки.

Задание 4. Умножение. Выполните умножение x на y в столбик: покажите четыре частичных произведения с учётом разрядов и сумму. Затем выполните один шаг Карацубы при $B = 100$: выпишите четыре части, три произведения, средний коэффициент и результат сборки. Проверьте результат рекурсивным кодом. Поясните, почему 16 произведений цифр и 3 произведения частей нельзя считать одинаковыми единицами трудоёмкости.

Задание 5. Модульная степень. Вычислите $x^e \bmod n$ алгоритмом слева направо. В отчёте приведите двоичную запись, её длину и число единиц; полную таблицу «префикс — остаток квадрата — итоговый остаток»; счётчики S, M и их сумму. Сравните сумму с $e-1$ для последовательного алгоритма. Проверьте результат встроенной функцией и объясните инвариант.

Задание 6. Граничные случаи и вывод. Проверьте модульную степень на входах (7,0,561), (3,560,561), (9,7,1) и при отрицательном показателе. Проверьте умножение на ноль и произведение 34 на 123456. Объясните каждый результат и сформулируйте, что в вашей работе доказано точно, а что означает лишь прохождение проверки.

6 Варианты исходных данных

Номер варианта назначает преподаватель. Все числа в таблице десятичные. Для задания 2 во всех строках используются основания 2 и 3; для Карацубы — основание 100.

№	n	x	y	e
1	561	1234	5678	37
2	341	1371	5871	53
3	1105	1508	6064	73
4	645	1645	6257	91
5	1729	1782	6450	127
6	169	1919	6643	129
7	2465	2056	6836	173
8	97	2193	7029	255
9	2821	2330	7222	257
10	6601	2467	7415	341
11	8911	2604	7608	511
12	10585	2741	7801	513
13	15841	2878	7994	769
14	29341	3015	8187	1023
15	41041	3152	8380	1025
16	46657	3289	8573	1365
17	52633	3426	8766	2047
18	62745	3563	8959	2049
19	63973	3700	9152	2731
20	75361	3837	9345	4095

7 Требования к отчёту и оценивание

Отчёт содержит тему, Ф. И. О. студента, группу и вариант; исходные данные; решения заданий 1–6 с промежуточными таблицами; использованный код и его результаты; краткий вывод. Для задания 3 достаточно программного перебора с проверкой знаменателя; в заданиях 1, 2, 4 и 5 необходимы пояснения и ручные вычисления. Полную распечатку всех оснований прикладывать не нужно.

Оценивание из 100 баллов: критерий Корсельта — 20; сравнение проверок — 20; перебор и вероятность — 10; умножение — 20; модульная степень — 20; граничные случаи и обоснованный вывод — 10. Правильный конечный остаток без хода вычислений не заменяет решение соответствующего задания.

Контрольные вопросы

1. Почему нулевые остатки в критерии Корсельта без двух других условий недостаточны?
2. Почему основание 3 не обязано проходить проверку Ферма для числа 561?
3. Как интерпретировать пустую цепочку и позднее появление единицы в `mr_round`?
4. Почему из числа взаимно простых остатков вычитается 2?
5. Почему два заранее выбранных основания не дают общей гарантии ошибки в одну шестнадцатую?
6. Как восстанавливается средний коэффициент в Карацубе и что считается в школьном алгоритме?
7. Почему для положительного показателя счётчики равны $t - 1$ и $w - 1$?
8. Чем проверка инварианта полезнее простого совпадения конечных ответов?

Литература

Лекция 15 данного комплекта. Для теории: Katz, Lindell, *Introduction to Modern Cryptography*, 2-е изд., гл. 8 и приложение B; Trappe, Washington, *Introduction to Cryptography with Coding Theory*, 2-е изд., § 3.13, упражнение 16, и § 6.3. Для умножения: Erickson, *Algorithms*, § 1.9. Точные сведения об изданиях и ссылки приведены в лекции.