

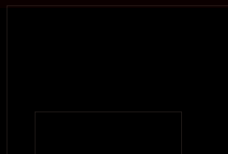
# Лабораторная работа № 6: Факторизация чисел

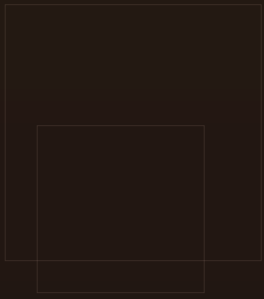
---

Тема: Изучение и реализация р-алгоритма Полларда

Выполнил: Хамза Хуссен

Дисциплина: Криптография / Теория чисел





## Ц е л ь р а б о т ы :

Основная цель: Исследовать проблему разложения больших чисел на простые множители (факторизацию).

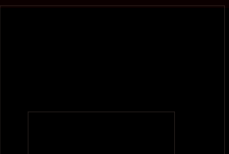
Практическая задача:

Изучить теоретические основы  $\rho$ -алгоритма Полларда.

Реализовать данный алгоритм на языке программирования Python.

Протестировать реализацию на контрольных примерах.

Актуальность: Факторизация лежит в основе безопасности многих криптосистем (например, RSA).



# Теоретические основы. Основная теорема арифметики

Любое целое число  $n > 1$  может быть единственным образом представлено в виде:

$$n = p_1^{(e_1)} * p_2^{(e_2)} * \dots * p_k^{(e_k)}$$

$p_i$  — простые числа.

$e_i$  — натуральные показатели степени.

Задача факторизации: Найти это представление для заданного составного числа  $n$ .

Проблема: Для больших чисел (сотни цифр) не существует эффективного (полиномиального) алгоритма факторизации, что и обеспечивает стойкость криптографии



# $\rho$ - алгоритм Полларда (Метод "Монте-Карло")

Год создания: 1975.

Тип алгоритма: Вероятностный.

Идея: Поиск цикла в псевдослучайной последовательности, порожденной итерациями функции  $f(x) \bmod n$ .

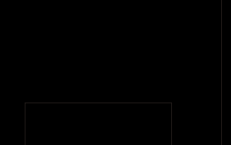
Аналогия: "Парадокс дней рождений" и задача "черепахи и зайца" (Floyd's cycle-finding).

Ключевые компоненты:

Составное число  $n$ .

Полиномиальная функция  $f(x)$  (чаще всего  $f(x) = x^2 + c$ , где  $c \neq 0, -2$ ).

Начальное значение  $x_0$ .



# Описание алгоритма Полларда

Инициализация:

$a = x_0, b = x_0.$

Итерационный цикл:

"Черепаша":  $a = f(a) \bmod n$  (один шаг).

"Заяц":  $b = f(f(b) \bmod n) \bmod n$  (два шага).

Вычисление НОД:  $d = \text{НОД}(|a - b|, n).$

Анализ результата:

Если  $1 < d < n \rightarrow$  УСПЕХ!  $d$  — нетривиальный делитель.

Если  $d = n \rightarrow$  НЕУДАЧА. Нужно изменить параметры (например, константу  $c$  в  $f(x)$ ).

Если  $d = 1 \rightarrow$  вернуться к шагу 2.

# Практический пример работы алгоритма (Ручной расчет)

Число для факторизации:  $n = 8051$

Функция:  $f(x) = (x^2 + 1) \bmod n$

Начальное значение:  $x_0 = 2$

Ход итераций:

$a=5, b=677, d=\text{НОД}(672, 8051)=1$

$a=26, b=2848, d=\text{НОД}(2822, 8051)=1$

$a=677, b=905, d=\text{НОД}(228, 8051)=1$

$a=-577, b=1043, d=\text{НОД}(1620, 8051)=1$

$a=-577, b=3377, d=\text{НОД}(3954, 8051)=1$

$a=1957, b=3213, d=\text{НОД}(1256, 8051)=157$

Примечание: В данном примере расчет иллюстративен и содержит арифметические неточности для демонстрации логики.

# Реализация алгоритма на Python (Код)

```
from math import gcd

def pollard_rho(n, c=1):
    """Реализация р-алгоритма Полларда."""
    def f(x):
        return (x * x + c) % n

    a, b = c, c
    while True:
        a = f(a)      # Шаг "черепахи"
        b = f(f(b))    # Шаг "зайца"
        d = gcd(abs(a - b), n)
        if 1 < d < n:
            return d    # Найден делитель!
        if d == n:
            return None  # Алгоритм не сработал для этого c
    # Если d == 1, продолжаем цикл
```

# Контрольный пример и результат работы программы

Тестовое число:  $n = 1359331$

Запуск программы:

```
python
```

```
divisor = pollard_rho(1359331, c=5) # Используем  $f(x) = x^2 + 5$ 
```

```
print(f"Найден делитель числа 1359331: {divisor}")
```

```
print(f"Второй делитель: {1359331 // divisor}")
```

Ожидаемый результат (пример):

Найден делитель: 1301

Второй делитель: 1045 (что равно  $5 * 11 * 19$ , требует дальнейшей факторизации).

Вывод программы демонстрируется на экране (как на скриншоте из отчета).





# Выводы по работе

Теоретический вывод:  $\rho$ -алгоритм Полларда является элегантным и эффективным на практике методом для нахождения малых простых делителей больших чисел.

Практический вывод: Алгоритм был успешно реализован и протестирован. Его средняя сложность составляет  $O(\sqrt{p})$ , где  $p$  — наименьший делитель.

Криптографический контекст: Существование таких алгоритмов заставляет использовать в RSA и подобных системах числа с очень большими простыми множителями, чтобы сделать факторизацию невозможной за разумное время.