

Дисциплина	Лабораторная	ФИО
Математические основы защиты информации и информационной безопасности	№7	Александра Миличевич

##Цель лабораторной работы

Цель лабораторной работы №7 заключается в ознакомлении студентов с алгоритмом Полларда для дискретного логарифмирования. Студенты должны изучить и реализовать алгоритм, который позволяет находить дискретные логарифмы в конечных полях, а также понять его применение в криптографии и теории чисел.

Как работает:

1. **Базовый случай:** Если b равно 0, возвращается кортеж $(a, 1, 0)$.
2. **Рекурсия:** Рекурсивно вызывается `extended_euclidean(b, a % b)`, результат разворачивается, и вычисляются новые коэффициенты x и y .

1. Функция `modular_inverse(a, n)`

Эта функция вычисляет обратное к a по модулю n .

Описание:

- **Вход:**
 - a (int): Число, для которого ищется обратное.
 - n (int): Модуль.
- **Выход:**
 - Обратное к a по модулю n .

Как работает:

1. Используется функция `extended_euclidean(a, n)` для получения коэффициентов Безу.
2. Возвращается коэффициент x (второй элемент в кортеже), который является

```
def modular_inverse(a, n):
    """
    Вычисляет обратное к 'a' по модулю 'n'.

    Args:
        a (int): Число, для которого ищется обратное.
        n (int): Модуль.

    Returns:
        int: Обратное к 'a' по модулю 'n'.
    """
    return extended_euclidean(a, n)[1]
```

обратным к a по модулю n .

2. Функция pollard_step(x, a, b, params)

Эта функция реализует один шаг алгоритма Полларда для дискретного логарифмирования.

Описание:

- **Вход:**
 - x (int): Текущее значение x .
 - a (int): Текущее значение a .
 - b (int): Текущее значение b .
 - $params$ (tuple): Параметры (G, H, P, Q) .
- **Выход:**
 - Кортеж обновленных значений (x, a, b) .

Как работает:

1. **Разделение на подмножества:** Использует $x \% 3$ для определения подмножества.
2. **Обновление значений в зависимости от подмножества:**
 - Если $x \% 3 == 0$: x умножается на G по модулю P , a увеличивается на 1 по модулю Q .
 - Если $x \% 3 == 1$: x умножается на H по модулю P , b увеличивается на 1 по модулю Q .
 - Если $x \% 3 == 2$: x возводится в квадрат по модулю P , a и b умножаются на 2 по модулю Q .

3. Функция pollard_rho_discrete_log(generator, value, prime)

Эта функция реализует алгоритм Полларда для дискретного логарифмирования.

Описание:

- **Вход:**
 - $generator$ (int): Генератор группы.
 - $value$ (int): Значение, для которого ищется дискретный логарифм.

```

def pollard_step(x, a, b, params):
    """
    Шаг алгоритма Полларда для дискретного логарифмирования.

    Args:
        x (int): Текущее значение x.
        a (int): Текущее значение a.
        b (int): Текущее значение b.
        params (tuple): Параметры (G, H, P, Q).

    Returns:
        tuple: Обновленные значения x, a, b.
    """
    G, H, P, Q = params
    subset = x % 3 # Выбираем подмножество

    if subset == 0:
        x = (x * G) % P
        a = (a + 1) % Q

    if subset == 1:
        x = (x * H) % P
        b = (b + 1) % Q

    if subset == 2:
        x = (x * x) % P
        a = (a * 2) % Q
        b = (b * 2) % Q

    return x, a, b

```

Figure 1: pollard step

- `prime (int)`: Простое число (порядок группы).
- **Выход:**
 - Дискретный логарифм (если найден) или сообщение об ошибке.

Как работает:

1. **Инициализация:** Устанавливаются начальные значения `Q, x, a, b, X, A, B`.
2. **Основной цикл:**
 - Используются “заяц” и “черепаха” для поиска коллизии, где заяц делает два шага за итерацию, а черепаха - один.
 - Функция `pollard_step` применяется для каждого шага.
 - Цикл выполняется до тех пор, пока не будет найдена коллизия (`x == X`).
3. **Вычисление дискретного логарифма:**
 - Вычисляется числитель `a - A` и знаменатель `B - b`.
 - Вычисляется обратный элемент знаменателя по модулю `Q` с помощью функции `modular_inverse`.
 - Вычисляется дискретный логарифм: $(\text{inverse_denominator} * \text{numerator}) \% Q$.
4. **Верификация:** Вызывается функция `verify` для проверки правильности найденного логарифма.
 - Если логарифм верный, то он возвращается.
 - Если логарифм не верный (при `pow(generator, result, prime) != value`), то `result` увеличивается на `Q` и возвращается.

4. Функция `verify(generator, value, prime, x)`

Эта функция проверяет правильность вычисленного дискретного логарифма.

Описание:

- **Вход:**
 - `generator (int)`: Генератор группы.
 - `value (int)`: Значение, для которого ищется дискретный логарифм.
 - `prime (int)`: Простое число (порядок группы).
 - `x (int)`: Вычисленный дискретный логарифм.
- **Выход:**
 - `True`, если логарифм верный, `False` в противном случае.

Как работает:

1. Вычисляет `generatorx mod prime`.
2. Сравнивает с `value`. Возвращает `True`, если равны, иначе `False`.

```

def pollard_rho_discrete_log(generator, value, prime):
    """
    Реализация алгоритма Полларда для дискретного логарифмирования.

    Args:
        generator (int): Генератор группы.
        value (int): Значение, для которого ищется дискретный логарифм.
        prime (int): Простое число (порядок группы).

    Returns:
        int: Дискретный логарифм (если найден) или сообщение об ошибке.
    """
    Q = (prime - 1) // 2 # Порядок подгруппы
    x = (generator * value) % prime # Начальное значение x
    a = 1 # Начальное значение a
    b = 1 # Начальное значение b

    X = x # Текущее значение X
    A = a # Текущее значение A
    B = b # Текущее значение B

    # Основной цикл поиска коллизии
    for i in range(1, prime):
        # Заяц
        x, a, b = pollard_step(x, a, b, (generator, value, prime, Q))

        # Черепаха
        X, A, B = pollard_step(X, A, B, (generator, value, prime, Q))
        X, A, B = pollard_step(X, A, B, (generator, value, prime, Q))

        # Если найдена коллизия, выходим из цикла
        if x == X:
            break

    numerator = a - A # Вычисляем числитель
    denominator = B - b # Вычисляем знаменатель

    try:
        # Вычисляем обратный элемент к знаменателю по модулю Q
        inverse_denominator = modular_inverse(denominator, Q)
    except:
        return "Не удалось найти обратный элемент"

    # Вычисляем дискретный логарифм
    result = (inverse_denominator * numerator) % Q

    if verify(generator, value, prime, result):
        return result
    else:
        return result + Q

```

Figure 2: pollard discrete log

```

def verify(generator, value, prime, x):
    """
    Проверяет правильность вычисленного дискретного логарифма.

    Args:
        generator (int): Генератор группы.
        value (int): Значение, для которого ищется дискретный логарифм.
        prime (int): Простое число (порядок группы).
        x (int): Вычисленный дискретный логарифм.

    Returns:
        bool: True, если логарифм верный, False в противном случае.
    """
    return pow(generator, x, prime) == value

```

Figure 3: verify