МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КУБГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**Отчет**

**по практическому заданию №2**

**по курсу**

**«КРИПТОГРАФИЧЕСКИЕ ПРОТОКОЛЫ»**

Работу выполнил

Студенты 49 группы

Прозоров М.С.

Преподаватель:

Крамаренко А.А.

Краснодар 2024

**Постановка задачи**

Реализовать класс вычисления целых степеней по заданному модулю (для вычисления положительной степени воспользоваться малой теоремой Ферма и реализованными отдельном методами умножения и сложения по заданному модулю, для вычисления отрицательной степени воспользоваться теоремой Эйлера).

Метод euclid\_extended(a, b) - реализует Расширенный алгоритм Евклида. Он вычисляет наибольший общий делитель (НОД) двух чисел a и b, а также коэффициенты x и y такие, что ax + by = gcd(a, b).

Метод euler\_phi(m) - вычисляет значение функции Эйлера φ(m), которое равно количеству натуральных чисел, меньших m и взаимно простых с m.

Класс ModularExponentiation - класс предоставляет методы для выполнения различных операций по модулю заданного числа modulus.

Метод add\_mod(a, b) - Сложение по модулю: возвращает сумму a и b по модулю modulus.

Метод mul\_mod(a, b) - Умножение по модулю: возвращает произведение a и b по модулю modulus.

Метод inv\_mod(a) - Нахождение обратного элемента по модулю: если существует число x такое, что ax ≡ 1 (mod modulus), то x является обратным для a по модулю modulus.

Метод pow\_mod(base, exponent) - Вычисление степени по модулю: возвращает результат возведения числа base в степень exponent по модулю modulus. Этот метод поддерживает как положительные, так и отрицательные степени. Для отрицательных степеней используется теорема Эйлера, позволяя вычислять обратные степени при наличии взаимной простоты между основанием степени и модулем.

**Вывод**

Был реализован класс вычисления целых степеней по заданному модулю. Для вычисления положительной степени использовалась малая теоремой Ферма, для вычисления отрицательной степени использовалась теорема Эйлера.

**Листинг:**

**def** euclid\_extended(a, b):

"""Расширенный алгоритм Евклида."""

**if** a **==** 0:

**return** b, 0, 1

**else**:

gcd, x1, y1 **=** euclid\_extended(b **%** a, a)

x **=** y1 **-** (b **//** a) **\*** x1

y **=** x1

**return** gcd, x, y

**def** euler\_phi(m):

"""Вычисляет значение функции Эйлера."""

result **=** 1

**for** i **in** range(2, m):

**if** euclid\_extended(i, m)[0] **==** 1:

result **+=** 1

**return** result

**class** ModularExponentiation:

**def** \_\_init\_\_(self, modulus):

self**.**modulus **=** modulus

**def** add\_mod(self, a, b):

"""Сложение по модулю."""

**return** (a **+** b) **%** self**.**modulus

**def** mul\_mod(self, a, b):

"""Умножение по модулю."""

**return** (a **\*** b) **%** self**.**modulus

**def** inv\_mod(self, a):

"""Нахождение обратного элемента по модулю."""

gcd, x, \_ **=** euclid\_extended(a, self**.**modulus)

**if** gcd **!=** 1:

**raise** ValueError("Обратный элемент не существует.")

**return** x **%** self**.**modulus

**def** pow\_mod(self, base, exponent):

"""Вычисление степени по модулю."""

**if** exponent **==** 0:

**return** 1

**elif** exponent **>** 0:

# Малая теорема Ферма для положительных степеней

result **=** 1

**for** \_ **in** range(exponent):

result **=** self**.**mul\_mod(result, base)

**return** result

**else**:

# Теорема Эйлера для отрицательных степеней

phi **=** euler\_phi(self**.**modulus)

inv\_base **=** self**.**inv\_mod(base)

positive\_exponent **=** (**-**exponent) **%** phi

result **=** 1

**for** \_ **in** range(positive\_exponent):

result **=** self**.**mul\_mod(result, inv\_base)

**return** result