Федеральное государственное бюджетное образовательное учреждение
высшего образования «Петрозаводский государственный университет»
Институт математики и информационных технологий
Кафедра информатики и математического обеспечения

Отчёт по лабораторной работе №2
по дисциплине: «Криптографические средства»
Вариант 2.2

Выполнил студент группы 22307:
Гордеев Никита Владиславович

Проверил преподаватель:
Кафедры прикладной математики и кибернетики
Института математики и информационных технологий
Ларионов Дмитрий Дмитриевич

Петрозаводск
2024

Оглавление

1 Формулировка задания

Напишите программу шифрования и расшифрования алгоритмом RSA. Рекомендуется использовать библиотеку для работы с длинными числами. В случае применения этой библиотеки разрешается использовать функции сложения, вычитания, умножения, целочисленного деления, вычисления остатка от деления. Функции возведения числа в степень, нахождения наибольшего общего делителя, обратного элемента в мультипликативной группе вычетов, генерации простого числа реализовать самостоятельно. Выполняемые функции программы:

1) генерация пары открытый/закрытый ключ, при этом число e задается пользователем;

2) шифрование данных (целого числа);

3) расшифрование шифртекста (целого числа).

2 Описание метода решения

Позволяет двум сторонам, находящимся в открытом канале связи, безопасно согласовать общий секретный ключ, который можно использовать для дальнейшего шифрования сообщений.

1) Стороны договариваются о двух числах - простом модуле p (17) и генераторе g (3).
2) Первый собеседник выбирает приватное число x (54)
3) Вычисляет открытый ключ a = g^x mod p (3^54 mod 17 = 15)
4) Отправляет открытый ключ второму собеседнику (15)
5) Второй собеседник выбирает секретное число y (24)
6) Вычисляет открытый ключ b = g^y mod p (3^24 mod 17 = 16)
7) Отправляет открытый ключ первому собеседнику (16)
8) Первый собеседник вычисляет общий секретный ключ b^x mod p (16^54 mod 17 = 1)
9) Второй собеседник вычисляет общий секретный ключ a^y mod p // 15^24 mod 17 = 1)
10) Теперь у собеседников одинаковый секретный ключ // 1

3 Примеры кода программ:

3.1 Реализован класс Random для генерации случайных чисел:

Реализованы методы getrandbits, randrange, randint, generate_large_number и generate_prime.

3.1.1 Метод getrandbits генерирует k случайных битов.

```
def getrandbits(self, k):
    """
    Генерация k случайных битов.
    :param k: Количество битов.
    :return: Случайное число с k битами.
    """
    if self.seed is None:
        raise ValueError("Seed is not set")
    self.index += 1
    return (self.seed + self.index) % (2 ** k)
```

3.1.2 Метод randrange генерирует случайное число из заданного диапазона.

```
def randrange(self, start, stop=None, step=1):
    """
    Генерация случайного числа из диапазона.
    :param start: Начало диапазона.
    :param stop: Конец диапазона.
    :param step: Шаг.
    :return: Случайное число из диапазона.
    """
    if stop is None:
        start, stop = 0, start
    if step == 1:
        return start + self.getrandbits(self._bit_length(stop - start))
    else:
        return start + step * self.getrandbits(self._bit_length((stop - start) // step))
```

3.1.3 Метод randint генерирует случайное целое число из заданного диапазона.

```
def randint(self, a, b):
    """
    Генерация случайного целого числа из диапазона [a, b].
    :param a: Начало диапазона.
    :param b: Конец диапазона.
    :return: Случайное целое число.
    """
    return self.randrange(a, b + 1)
```

3.1.4 Метод generate_large_number генерирует большое случайное число заданной длины.

```
def generate_large_number(self, length):
    """
    Генерация большого случайного числа длины length.
    :param length: Длина числа.
    :return: Случайное число.
    """
    return self.randint(2**(length-1), 2**length)
```

3.1.5 Метод generate_prime генерирует случайное простое число заданной длины.

```
def generate_prime(self, length):
    """
    Генерация случайного простого числа длины length.
    :param length: Длина числа.
    :return: Простое число.
    """
    while True:
        p = self.generate_large_number(length)
        if Math().is_prime(p):
            return p
```

3.2 Реализован класс Math:

Реализованы методы **pow**, **gcd**, **is_prime** и **mod_inverse**.

3.2.1 Метод pow возводит число в степень по модулю.

```
def pow(self, x, y, z=None):
    """
    Возведение числа x в степень y по модулю z.
    :param x: Основание.
    :param y: Показатель степени.
    :param z: Модуль.
    :return: Результат возведения в степень по модулю.
    """
    if z is None:
        return x ** y
```

```
    result = 1
    while y:
        if y & 1:
            result = result * x % z
        x = x * x % z
        y >>= 1
    return result
```

3.2.2 Метод gcd находит наибольший общий делитель двух чисел.

```
def gcd(self, a, b):
    """
    Нахождение наибольшего общего делителя чисел a и b.
    :param a: Первое число.
    :param b: Второе число.
    :return: НОД(a, b).
    """
    while b != 0:
        a, b = b, a % b
    return a
```

3.2.3 Метод is_prime проверяет, является ли число простым.

```
def is_prime(self, n, k=5):
    """
    Проверка, является ли число простым.
    :param n: Число для проверки.
    :param k: Количество итераций теста Миллера-Рабина.
    :return: True, если число простое, иначе False.
    """
    if n <= 1:
        return False
    if n <= 3:
        return True
    def miller_rabin(n, d):
        a = self.random.randint(2, n - 2)
        while self.gcd(a, n) != 1:
            a = self.random.randint(2, n - 2)
        x = self.pow(a, d, n)
        if x == 1 or x == n - 1:
            return True
        while d != n - 1:
            x = self.pow(x, 2, n)
            d *= 2
            if x == 1:
                return False
            if x == n - 1:
                return True
        return False
```

```
    d = n - 1
    while d % 2 == 0:
        d //= 2

    for _ in range(k):
        if not miller_rabin(n, d):
            return False
    return True
```

3.3 Реализован класс DiffieHellman для работы с алгоритмом DiffieHellman:

Реализованы методы create_module_and_generator, generate_keys и generate_shared_secret.

3.3.1 Метод create_module_and_generator генерирует открытый и закрытый ключи.

```
    def create_module_and_generator(self):
        """
        Генерация простого числа p и генератора g.
        :return: Кортеж (p, g).
        """
        p = self.random.generate_prime(512)
        g = self.random.randint(2, p - 2)

        return p, g
```

3.3.2 Метод generate_keys выполняет генерацию закрытого ключа и соответствующего ему открытого ключа.

```
    def generate_keys(self, g, p):
        """
        Генерация закрытого ключа и соответствующего ему открытого ключа.
        :param g: Генератор.
        :param p: Простое число.
        :return: Кортеж (private_key, public_key).
        """
        private_key = self.random.generate_large_number(self.key_length)
        public_key = self.math.pow(g, private_key, p)
        return private_key, public_key
```

3.3.3 Метод generate_shared_secret выполняет генерацию общего секретного ключа.

```
def generate_shared_secret(self, other_public_key, private_key, p):
    """
    Генерация общего секретного ключа на основе открытого ключа другой стороны и собственного закрытого ключа.
    :param other_public_key: Открытый ключ другой стороны.
    :param private_key: Закрытый ключ.
    :param p: Простое число.
    :return: Общий секрет.
    """
    return self.math.pow(other_public_key, private_key, p)
```

3.4 Просмотр работы программы пользователю

В программе предусмотрена возможность просмотра пользователем генерации чисел, имитации обмена данными между пользователей и получения общего ключа.

```
    # Генерация простого числа p и примитивного корня g
    p, g = diffie_hellman.create_module_and_generator()

    print("Открытый канал связи:")
    print("простой модуль p =", p)
    print("генератор g =", g)

    print("\nПервый пользователь: генерирует свой приватный и публичный ключи:")
    first_private_key, first_public_key = diffie_hellman.generate_keys(g, p)
    print("Приватный ключ:", first_private_key)
    print("Публичный ключ:", first_public_key)

    print("\nПервый пользователь: отправляет публичный ключ")

    print("\nВторой пользователь: генерирует свой приватный и публичный ключи:")
    second_private_key, second_public_key = diffie_hellman.generate_keys(g, p)
    print("Приватный ключ:", second_private_key)
    print("Публичный ключ:", second_public_key)

    print("\nВторой пользователь: отправляет публичный ключ")

    print("\nРасчет общего секретного ключа:")
    first_shared_secret = diffie_hellman.generate_shared_secret(second_public_key, first_private_key, p)
    second_shared_secret = diffie_hellman.generate_shared_secret(first_public_key,
second_private_key, p)

    print("Общий секретный ключ у Первого пользователя:", first_shared_secret)
    print("Общий секретный ключ у Второго пользователя:", second_shared_secret)
```

4 Тестовые данные

```
import unittest
from diffie_hellman import Random, Math, DiffieHellman

class TestRandom(unittest.TestCase):

    def setUp(self):
        self.random = Random(seed=42)

    def test_generate_prime(self):
        # Убеждаемся, что числа, сгенерированные методом generate_prime, действительно простые.
        self.assertTrue(Math().is_prime(self.random.generate_prime(512)))
        self.assertTrue(Math().is_prime(self.random.generate_prime(1024)))

class TestMath(unittest.TestCase):
```

```
    """Тесты для класса Math."""

    def setUp(self):
        self.math = Math()

    def test_pow(self):
        # Проверяем возведение в степень по модулю.
        self.assertEqual(self.math.pow(2, 3, 5), 3)
        self.assertEqual(self.math.pow(2, 10, 11), 1)
        self.assertEqual(self.math.pow(3, 4), 81)
        self.assertEqual(self.math.pow(5, 3, 7), 6)

    def test_gcd(self):
        # Проверяем нахождение наибольшего общего делителя.
        self.assertEqual(self.math.gcd(10, 25), 5)
        self.assertEqual(self.math.gcd(14, 28), 14)
        self.assertEqual(self.math.gcd(15, 17), 1)
        self.assertEqual(self.math.gcd(25, 100), 25)

    def test_is_prime(self):
        # Проверяем, что числа правильно определяются как простые или составные.
        self.assertTrue(self.math.is_prime(7))
        self.assertTrue(self.math.is_prime(13))
        self.assertTrue(self.math.is_prime(23))
        self.assertFalse(self.math.is_prime(9))
        self.assertFalse(self.math.is_prime(15))
        self.assertFalse(self.math.is_prime(21))

class TestDiffieHellman(unittest.TestCase):
    def setUp(self):
        self.diffie_hellman = DiffieHellman()

    def test_create_module_and_generator(self):
        # Проверяем корректность генерации простого модуля и генератора.
        p, g = self.diffie_hellman.create_module_and_generator()
        self.assertTrue(Math().is_prime(p))
        self.assertTrue(2 < g < p)

    def test_generate_keys(self):
        # Проверяем генерацию закрытого и открытого ключей.
        p, g = self.diffie_hellman.create_module_and_generator()
        private_key, public_key = self.diffie_hellman.generate_keys(g, p)
        self.assertTrue(1 < private_key < p - 1)

    def test_generate_shared_secret(self):
        # Тест метода generate_shared_secret
        p, g = 23, 5
        private_key_A, public_key_A = self.diffie_hellman.generate_keys(g, p)
        private_key_B, public_key_B = self.diffie_hellman.generate_keys(g, p)
```

```
        shared_secret_A = self.diffie_hellman.generate_shared_secret(public_key_B, private_key_A, p)
        shared_secret_B = self.diffie_hellman.generate_shared_secret(public_key_A, private_key_B, p)

        self.assertEqual(shared_secret_A, shared_secret_B)

if __name__ == '__main__':
    unittest.main()
```