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 2](#_Toc165405813)

[2 Описание метода решения 3](#_Toc165405814)

[3 Примеры кода программ: 3](#_Toc165405815)

[3.1 Реализован класс Random для генерации случайных чисел: 3](#_Toc165405816)

[3.1.1 Метод getrandbits генерирует k случайных битов. 3](#_Toc165405817)

[3.1.2 Метод randrange генерирует случайное число из заданного диапазона. 3](#_Toc165405818)

[3.1.3 Метод randint генерирует случайное целое число из заданного диапазона. 4](#_Toc165405819)

[3.1.4 Метод generate\_large\_number генерирует большое случайное число заданной длины. 4](#_Toc165405820)

[3.1.5 Метод generate\_prime генерирует случайное простое число заданной длины. 4](#_Toc165405821)

[3.2 Реализован класс Math: 4](#_Toc165405822)

[3.2.1 Метод pow возводит число в степень по модулю. 4](#_Toc165405823)

[3.2.2 Метод gcd находит наибольший общий делитель двух чисел. 5](#_Toc165405824)

[3.2.3 Метод is\_prime проверяет, является ли число простым. 5](#_Toc165405825)

[3.2.4 Метод mod\_inverse находит мультипликативно обратное число по модулю. 6](#_Toc165405826)

[3.3 Реализован класс RSA для работы с алгоритмом шифрования RSA: 6](#_Toc165405827)

[3.3.1 Метод generate\_keys генерирует открытый и закрытый ключи. 6](#_Toc165405828)

[3.3.2 Метод encrypt выполняет шифрование данных. 7](#_Toc165405829)

[3.3.3 Метод decrypt выполняет расшифрование данных. 7](#_Toc165405830)

[3.4 Возможность работы пользователю 7](#_Toc165405831)

[4 Тестовые данные 8](#_Toc165405832)

# 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).

# 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

# Примеры кода программ:

## Реализован класс Random для генерации случайных чисел:

Реализованы методы getrandbits, randrange, randint, generate\_large\_number и generate\_prime.

### 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)

### 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))

### 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)

### 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)

### 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

## Реализован класс Math:

Реализованы методы **pow**, **gcd**, **is\_prime** и **mod\_inverse**.

### 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

### 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

### 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

## Реализован класс DiffieHellman для работы с алгоритмом DiffieHellman:

Реализованы методы create\_module\_and\_generator, generate\_keys и generate\_shared\_secret.

### 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

### 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

### 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)

## 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)

# 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()