**Министерство науки и высшего образования РФ**

Федеральное государственное бюджетное общеобразовательное учреждение высшего образования

«Чувашский Государственный Университет им. И.Н.Ульянова»

Факультет информатики и вычислительной техники

Кафедра компьютерных технологий

Отчет Лабораторной работы №2

По дисциплине: «Информационная безопасность»

Вариант 4

Выполнил студент группы КТ-43-21

Казаков А.Ю.

Проверил ст. преподаватель:

Мытникова Е. А.

Чебоксары, 2025

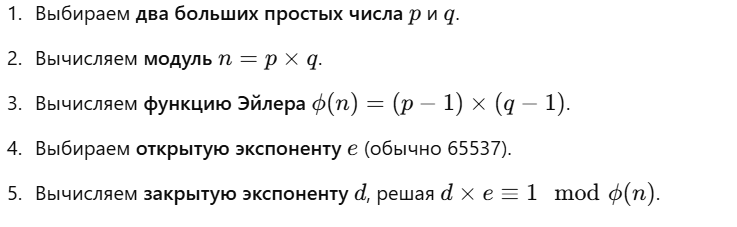
**Лабораторная работа №2**

Задание: **Шифрование файлов с использованием RSA**:  
Разработать приложение для шифрования и дешифрования файлов с использованием RSA.  
Дополнительно: добавить возможность выбора размера ключа (1024, 2048, 4096 бит).

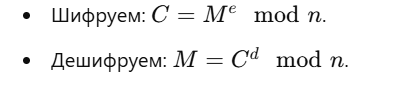
**RSA** (аббревиатура от фамилий Rivest, Shamir и Adleman) — криптографический алгоритм с открытым ключом, основывающийся на вычислительной сложности задачи факторизации больших полупростых чисел. Криптосистема RSA стала первой системой, пригодной и для шифрования и цифровой подписи.

Реализация алгоритма RSA без сторонних библиотек включает:

1. **Генерацию ключей**: Выбор двух простых чисел, вычисление открытого и закрытого ключа.
2. **Шифрование**: Преобразование сообщения с помощью открытого ключа.
3. **Дешифрование**: Восстановление исходного сообщения с помощью закрытого ключа.



* Генерируем случайное число заданного размера.
* Проверяем, является ли оно простым (тест Миллера-Рабина).
* Если не является, ищем следующее простое.



Код:

import random  
import base64  
  
  
def is\_prime(n, k=5):  
 *""" Тест Миллера-Рабина для проверки простоты числа """* if n <= 1:  
 return False  
 if n <= 3:  
 return True  
 if n % 2 == 0:  
 return False  
  
 r, d = 0, n - 1  
 while d % 2 == 0:  
 r += 1  
 d //= 2  
  
 def check(a, d, n, r):  
 x = pow(a, d, n)  
 if x == 1 or x == n - 1:  
 return True  
 for \_ in range(r - 1):  
 x = pow(x, 2, n)  
 if x == n - 1:  
 return True  
 return False  
  
 for \_ in range(k):  
 a = random.randint(2, n - 2)  
 if not check(a, d, n, r):  
 return False  
 return True  
  
  
def generate\_prime(bits):  
 *""" Генерация случайного простого числа заданной битности """* while True:  
 num = random.getrandbits(bits)  
 num |= (1 << bits - 1) | 1 *# Устанавливаем первый и последний биты (делает число нечётным и достаточной длины)* if is\_prime(num):  
 return num  
  
  
def gcd(a, b):  
 *""" Нахождение наибольшего общего делителя (НОД) """* while b:  
 a, b = b, a % b  
 return a  
  
  
def mod\_inverse(e, phi):  
 *""" Нахождение мультипликативного обратного числа d к e по модулю phi (Расширенный алгоритм Евклида) """* a, b, u = 0, phi, 1  
 while e > 0:  
 q = b // e  
 e, a, b, u = b % e, u, e, a - q \* u  
 if b == 1:  
 return a % phi  
  
  
def generate\_keys(key\_size):  
 *""" Генерация открытого и закрытого ключей """* print(f"Генерация {key\_size}-битного ключа...")  
  
 bit\_length = key\_size // 2  
 p = generate\_prime(bit\_length)  
 q = generate\_prime(bit\_length)  
  
 n = p \* q  
 phi = (p - 1) \* (q - 1)  
  
 e = 65537 *# Чаще всего используется это значение* while gcd(e, phi) != 1: *# Проверяем, что e взаимно просто с phi* e = random.randrange(2, phi)  
  
 d = mod\_inverse(e, phi)  
  
 public\_key = (e, n)  
 private\_key = (d, n)  
  
 return public\_key, private\_key  
  
  
def encrypt(message, public\_key):  
 *""" Шифрование сообщения с помощью открытого ключа """* e, n = public\_key  
 cipher\_text = [pow(ord(char), e, n) for char in message]  
 return cipher\_text  
  
  
def decrypt(cipher\_text, private\_key):  
 *""" Дешифрование сообщения с помощью закрытого ключа """* d, n = private\_key  
 message = ''.join(chr(pow(char, d, n)) for char in cipher\_text)  
 return message  
  
  
def encrypt\_file(file\_path, public\_key):  
 with open(file\_path, "rb") as file:  
 data = file.read()  
 data\_str = base64.b64encode(data).decode()  
 encrypted\_data = encrypt(data\_str, public\_key)  
  
 with open(f"e\_{file\_path}", "w") as enc\_file:  
 enc\_file.write(" ".join(map(str, encrypted\_data)))  
  
 print(f"Файл e\_{file\_path} зашифрован!")  
  
  
def decrypt\_file(file\_path, private\_key):  
 with open(f"{file\_path}", "r") as file:  
 encrypted\_data = list(map(int, file.read().split()))  
  
 decrypted\_str = decrypt(encrypted\_data, private\_key) *# Дешифруем строку* original\_data = base64.b64decode(decrypted\_str).decode() *# Декодируем из base64 в байты  
  
 # decrypted\_path = file\_path.replace("e\_", "d\_")  
 # with open(decrypted\_path, "wb") as dec\_file:  
 # dec\_file.write(original\_data)* print(f"{original\_data}")  
  
  
def main():  
 key\_size = 1024 *# Можно выбрать 2048 или 4096* public\_key, private\_key = generate\_keys(key\_size)  
  
 while True:  
 print("\n1. Сгенерировать ключи")  
 print("2. Зашифровать файл")  
 print("3. Расшифровать файл")  
 print("4. Выход")  
 choice = input("Выберите действие: ")  
  
 if choice == "1":  
 key\_size = int(input("Введите размер ключа (1024, 2048, 4096): "))  
 elif choice == "2":  
 file\_path = input("Введите путь к файлу: ")  
 print(f"{file\_path}")  
 encrypt\_file(file\_path, public\_key)  
 elif choice == "3":  
 file\_path = input("Введите путь к зашифрованному файлу: ")  
 decrypt\_file(file\_path, private\_key)  
 elif choice == "4":  
 break  
 else:  
 print("Неверный ввод!")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

