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 №10**

**по курсу**

**«КРИПТОГРАФИЧЕСКИЕ ПРОТОКОЛЫ»**

Работу выполнил

Студенты 46 группы

Прозоров М.С.

Преподаватель:

Крамаренко А.А.

Краснодар 2024

**Постановка задачи**

Реализовать криптосистему RSA для шифрования и расшифрования вводимых сообщений. Открытый ключ показывать пользователю, закрытый ключ записывать в файл.

Алгоритм реализации криптосистемы RSA для расшифрования вводимых сообщений состоит из нескольких ключевых шагов, включая генерацию ключей, шифрование и расшифрование сообщений. Вот подробное объяснение каждого шага:

1. Генерация ключей

a. Выбор двух больших простых чисел p и q:

Для начала выбираются два больших простых числа p и q. В реализации на Python мы использовали функцию `randprime` для генерации этих чисел, но в практических приложениях они должны быть значительно больше для обеспечения безопасности.

b. Вычисление n = p \* q:

Затем вычисляется n, которое является произведением p и q. Значение n используется как часть открытого и закрытого ключей и служит модулем для шифрования и расшифрования.

c. Вычисление функции Эйлера от n, phi(n) = (p-1) \* (q-1):

Функция Эйлера от n, обозначаемая phi(n), необходима для определения количества целых чисел, взаимно простых с n. В контексте RSA это значение используется для генерации ключей.

d. Выбор открытого экспонента e:

Выбирается число e, которое является взаимно простым с phi(n) и меньше phi(n). Обычно используют стандартные значения, например, 65537, поскольку они обеспечивают хороший баланс между безопасностью и скоростью.

e. Вычисление закрытого экспонента d:

d является мультипликативным обратным к e по модулю phi(n), что означает, что e \* d = 1 mod phi(n). Это можно вычислить с помощью расширенного алгоритма Евклида.

f. Формирование ключей:

Открытый ключ состоит из пары (e, n), а закрытый ключ — из пары (d, n).

2. Шифрование (на стороне отправителя):

Для шифрования сообщения каждый символ сообщения преобразуется в числовой код (например, с использованием ASCII), после чего каждое число m шифруется с помощью открытого ключа по формуле c = m^e mod n, где `c` — зашифрованный текст.

3. Расшифрование (на стороне получателя):

Для расшифрования полученного сообщения используется закрытый ключ. Каждый зашифрованный символ `c` расшифровывается по формуле m = c^d \mod n, где m — исходный числовой код символа. Затем числовые коды преобразуются обратно в текст.

4. Запись закрытого ключа в файл:

Закрытый ключ сохраняется в файл для последующего использования при расшифровке. Это обеспечивает безопасность закрытого ключа, позволяя его использовать только тем, кто имеет доступ к файлу.

**Вывод**

Реализовал криптосистему RSA для шифрования и расшифрования вводимых сообщений.

**Листинг:**

from sympy import randprime, gcd

from math import prod

# Функция для генерации ключей RSA

def generate\_rsa\_keys():

# Генерируем два простых числа (в реальности должны быть гораздо больше)

p, q = randprime(100, 1000), randprime(100, 1000)

while q == p:

q = randprime(100, 1000)

# Вычисляем n и функцию Эйлера от n

n = p \* q

phi = (p - 1) \* (q - 1)

# Выбираем e, взаимно простое с phi

e = 2

while gcd(e, phi) != 1:

e += 1

# Вычисляем d, обратное к e по модулю phi

d = pow(e, -1, phi)

# Возвращаем открытый и закрытый ключи

return ((e, n), (d, n))

# Функция для шифрования сообщения

def encrypt\_message(message, public\_key):

e, n = public\_key

# Шифруем сообщение, применяя преобразование RSA

encrypted\_msg = [pow(ord(char), e, n) for char in message]

return encrypted\_msg

# Функция для расшифровки сообщения

def decrypt\_message(cipher\_text, private\_key):

d, n = private\_key

# Расшифровываем сообщение, применяя преобразование RSA

decrypted\_msg = [chr(pow(char, d, n)) for char in cipher\_text]

return ''.join(decrypted\_msg)

# Генерируем ключи

public\_key, private\_key = generate\_rsa\_keys()

# Показываем открытый ключ пользователю

print(f'Public key: {public\_key}')

# Демонстрация шифрования и расшифрования

message = "RSA Test!"

encrypted\_message = encrypt\_message(message, public\_key)

print(f'Encrypted message: {encrypted\_message}')

decrypted\_message = decrypt\_message(encrypted\_message, private\_key)

print(f'Decrypted message: {decrypted\_message}')

# Запись закрытого ключа в файл

private\_key\_path = "private\_key\_lr10.txt"

with open(private\_key\_path, "w+") as file:

file.write(f"{private\_key[0]}, {private\_key[1]}")

print(f"private key: {private\_key}")