Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Пермский национальный исследовательский политехнический университет»

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

Направление 09.03.01 – «Информатика и вычислительная техника»

Дисциплина: «Безопасность и защита информации»

Профиль: «Компьютерные системы»

Семестр 6

ОТЧЕТ

по лабораторной работе №2

Тема: «Шифры перестановки и замены»

Выполнил: студент группы КС-21-1б

Чирков А.В. \_\_\_\_\_\_\_\_\_\_\_

Проверил: доцент кафедры ИТАС

Шереметьев В. Г. \_\_\_\_\_\_\_\_\_\_\_

Дата \_\_\_\_\_\_

**ЦЕЛЬ РАБОТЫ**

Получить практические навыки по использованию ассиметричных алгоритмов шифрования, на примере использования алгоритма RSA.

**ЗАДАНИЕ**

Реализовать шифрование текстового сообщения, используя алгоритм RSA.

**ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Алгоритм работы RSA таков. Сначала надо получить открытый и секретный ключи:

1. Выбираются два простых числа p и q

2. Вычисляется их произведение n(=p\*q)

3. Выбирается произвольное число e (e<n), такое, что НОД(e,(p-1)(q-1))=1, то есть e должно быть взаимно простым с числом (p-1)(q-1).

4. Методом Евклида решается в целых числах уравнение e\*d+(p-1)(q-1)\*y=1. Здесь неизвестными являются переменные d и y – метод Евклида как раз и находит множество пар (d,y), каждая из которых является решением уравнения в целых числах.

5. Два числа (e,n) – публикуются как открытый ключ.

6. Число d хранится в строжайшем секрете – это и есть закрытый ключ, который позволит читать все послания, зашифрованные с помощью пары чисел (e,n).

Шифрование:

1. Отправитель разбивает свое сообщение на блоки, равные k=[log2(n)] бит, где квадратные скобки обозначают взятие целой части от дробного числа.

2. Подобный блок может быть интерпретирован как число из диапазона (0;2k-1). Для каждого такого числа (назовем его mi) вычисляется выражение ci=((mi)e)mod n. Блоки ci и есть зашифрованное сообщение.

3. Если число n представимо в виде двух простых чисел p и q, то для любого x имеет место равенство (x(p-1)(q-1))mod n = 1. Для дешифрования RSA-сообщений воспользуемся этой формулой. Возведем обе ее части в степень (-y) : (x(-y)(p-1)(q-1))mod n = 1(-y) = 1. Теперь умножим обе ее части на x : (x(-y)(p-1)(q-1)+1)mod n = 1\*x = x.

Для того чтобы прочесть сообщение ci=((mi)e)mod n достаточно возвести его в степень d по модулю m : ((ci)d)mod n = ((mi)e\*d)mod n = mi.

Проясним использование алгоритма RSA на конкретном примере. Выбираем два простые числа p=7; q=17. В этом случае n = p\*q будет равно 119. Теперь необходимо выбрать e=5. Следующий шаг связан с формированием числа d так, чтобы (d\*e) mod [(p-1)(q-1)] =1. d=77 (использован расширенный алгоритм Эвклида). d – секретный ключ, а e и n характеризуют открытый ключ. Пусть текст, который нам нужно зашифровать представляется M=19. С = Memod n. Получаем зашифрованный текст C=66. Этот «текст» может быть послан соответствующему адресату. Получатель дешифрует полученное сообщение, используя М= Cdmod n и C=66. В результате получается M=19.

**ХОД РАБОТЫ**

При запуске программы, на экране появляется окно с полями ввода чисел p и q, кнопкой Generate Key Pair (Генерация пары ключей), пустыми полями ключей и кнопками шифрования (Encrypt File) и дешифрования (Decrypt File) файла (рисунок 1).

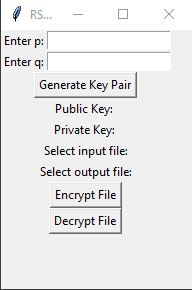


Рисунок 1 – Окно программы при запуске.

Вводим p,q и генерируем ключи. Пример работы программы представлен на рисунке 2.

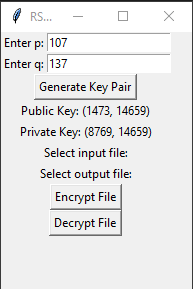


Рисунок 2 – Пример работы программы.

Нажимаем кнопку шифрования и выбираем файлы (рисунок 3).

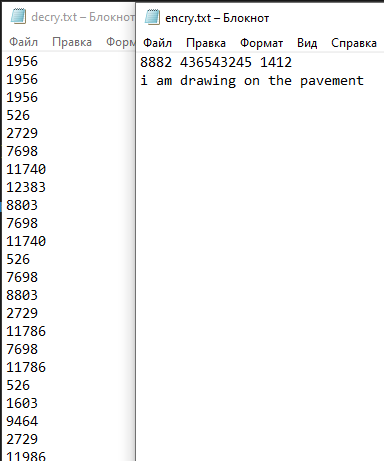


Рисунок 3 - Исходный и зашифрованный файлы

Нажимаем кнопку Decrypt и выбираем файлы для дешифрования (рисунок 4).

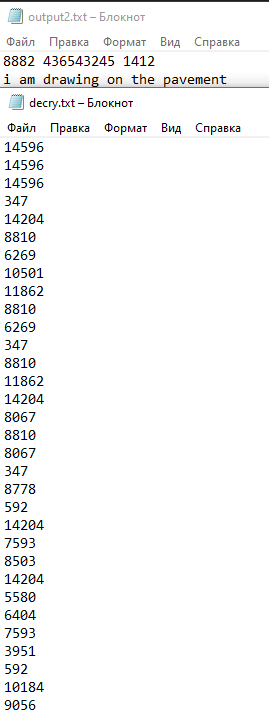


Рисунок 4 - Дешифрование файла

**ПРИЛОЖЕНИЕ А**

**Листинг программы**

import tkinter as tk

from tkinter import filedialog

import random

from functools import partial

import math

public\_key = None

private\_key = None

def gcd(a, b):

while b != 0:

a, b = b, a % b

return a

def multiplicative\_inverse(e, phi):

d = 0

x1, x2, y1, y2 = 0, 1, 1, 0

temp\_phi = phi

while e > 0:

temp1 = temp\_phi // e

temp2 = temp\_phi - temp1 \* e

temp\_phi = e

e = temp2

x = x2 - temp1 \* x1

y = y2 - temp1 \* y1

x2 = x1

x1 = x

y2 = y1

y1 = y

if temp\_phi == 1:

d = y2 + phi

return d

def generate\_keypair(p, q):

n = p \* q

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

e = random.randrange(1, phi)

g = gcd(e, phi)

while g != 1:

e = random.randrange(1, phi)

g = gcd(e, phi)

d = multiplicative\_inverse(e, phi)

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

def encrypt(pk, plaintext):

key, n = pk

cipher\_blocks = []

for char in plaintext:

mi = ord(char)

ci = pow(mi, key, n)

cipher\_blocks.append(ci)

return cipher\_blocks

def decrypt(pk, ciphertext):

key, n = pk

plain\_blocks = []

for char in ciphertext:

mi = pow(char, key, n)

plain\_blocks.append(chr(mi))

return ''.join(plain\_blocks)

def browse\_file(label):

filename = filedialog.askopenfilename()

label.config(text="Selected file: " + filename)

return filename

def encrypt\_file(public\_key, input\_file, output\_file):

with open(input\_file, 'r',encoding='utf-8') as file:

plaintext = file.read()

encrypted\_text = encrypt(public\_key, plaintext)

with open(output\_file, 'w',encoding='utf-8') as file:

for char in encrypted\_text:

file.write(str(char) + '\n')

print("Encryption completed.")

def decrypt\_file(private\_key, input\_file, output\_file):

with open(input\_file, 'r',encoding='utf-8') as file:

encrypted\_text = [int(line.strip()) for line in file]

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

with open(output\_file, 'w',encoding='utf-8') as file:

file.write(decrypted\_text)

print("Decryption completed.")

def generate\_key\_pair(p\_entry, q\_entry):

global public\_key, private\_key

p = int(p\_entry.get())

q = int(q\_entry.get())

public\_key, private\_key = generate\_keypair(p, q)

public\_key\_label.config(text="Public Key: " + str(public\_key))

private\_key\_label.config(text="Private Key: " + str(private\_key))

def encrypt\_selected\_file():

global public\_key

if public\_key is None:

print("Public key not generated")

return

input\_file = browse\_file(input\_label)

output\_file = browse\_file(output\_label)

encrypt\_file(public\_key, input\_file, output\_file)

def decrypt\_selected\_file():

global private\_key

if private\_key is None:

print("Private key not generated")

return

input\_file = browse\_file(input\_label)

output\_file = browse\_file(output\_label)

decrypt\_file(private\_key, input\_file, output\_file)

root = tk.Tk()

root.title("RSA File Encryption")

p\_label = tk.Label(root, text="Enter p:")

p\_label.grid(row=0, column=0)

p\_entry = tk.Entry(root)

p\_entry.grid(row=0, column=1)

q\_label = tk.Label(root, text="Enter q:")

q\_label.grid(row=1, column=0)

q\_entry = tk.Entry(root)

q\_entry.grid(row=1, column=1)

generate\_button = tk.Button(root, text="Generate Key Pair", command=partial(generate\_key\_pair, p\_entry, q\_entry))

generate\_button.grid(row=2, column=0, columnspan=2)

public\_key\_label = tk.Label(root, text="Public Key: ")

public\_key\_label.grid(row=3, column=0, columnspan=2)

private\_key\_label = tk.Label(root, text="Private Key: ")

private\_key\_label.grid(row=4, column=0, columnspan=2)

input\_label = tk.Label(root, text="Select input file:")

input\_label.grid(row=5, column=0, columnspan=2)

output\_label = tk.Label(root, text="Select output file:")

output\_label.grid(row=6, column=0, columnspan=2)

encrypt\_button = tk.Button(root, text="Encrypt File", command=lambda: encrypt\_selected\_file())

encrypt\_button.grid(row=7, column=0, columnspan=2)

decrypt\_button = tk.Button(root, text="Decrypt File", command=lambda: decrypt\_selected\_file())

decrypt\_button.grid(row=8, column=0, columnspan=2)

root.mainloop()