МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«**Вятский государственный университет**»

**(«ВятГУ»)**

Факультет автоматики и вычислительной техники

Кафедра электронных вычислительных машин

Отчет по лабораторной работе №3

по дисциплине «Защита информации»

Выполнил студент группы ИВТб-41\_\_\_\_\_\_\_\_\_\_/Жеребцов К. А./

Проверил преподаватель кафедры ЭВМ\_\_\_\_\_\_\_\_\_/Караваева О. В./

Киров 2024

**Задание 1.** Напишите дешифратор для <<ШИФРА А1Я33>> и расшифруйте предоставленную строку:

-19-14-6-18-20-30-15-6-8-5-6-20-10-8-10-9-15-30-8-5-1-20-30-15-6-5-16-13-8-15-1

**Описание алгоритма:**

**Алгоритм шифрования:**

1. **Инициализация шифра:**
   * Создать таблицу замен, которая будет соответствовать каждому символу из алфавита (как кириллического, так и латинского) некоторому другому символу.
   * Это может быть любая таблица замен, но в случае ШИФРА А1Я33 используется определенная таблица.
2. **Шифрование:**
   * Для каждого символа в открытом тексте:
     + Используйте таблицу замен, чтобы найти соответствующий зашифрованный символ.
     + Замените символ в открытом тексте на его зашифрованный эквивалент.
3. **Вывод зашифрованного текста.**

**Алгоритм дешифрования:**

1. **Инициализация дешифра:**
   * Используйте ту же таблицу замен, чтобы создать таблицу для дешифровки.
2. **Дешифрование:**
   * Для каждого символа в зашифрованном тексте:
     + Используйте таблицу дешифровки, чтобы найти соответствующий символ в открытом тексте.
     + Замените символ в зашифрованном тексте на его дешифрованный эквивалент.
3. **Вывод дешифрованного текста.**

**Экранные формы:**

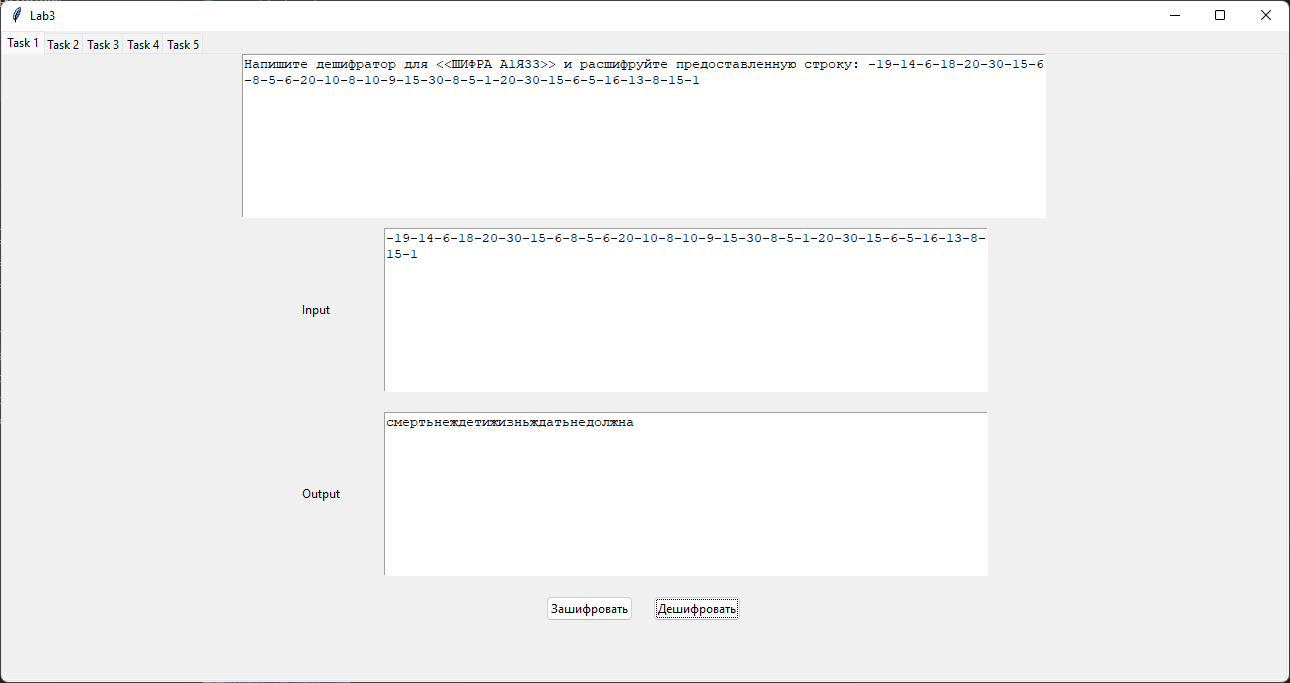
****

Рисунок 1 – Экранная форма выполнения задания 1

**Код программы:**

import tkinter as tk

letters = 'абвгдеёжзийклмнопрстуфхцчшщъыьэюя'

def encrypt\_text(input\_entry, output\_entry):

    input\_text = input\_entry.get("1.0", "end-1c")

    result\_string = '-'.join(str(letters.index(char) + 1) for char in input\_text)

    output\_entry.delete("1.0", tk.END)

    output\_entry.insert("1.0", result\_string)

def decrypt\_text(input\_entry, output\_entry):

    input\_text = input\_entry.get("1.0", "end-1c")

    numbers = list(map(int, filter(None, input\_text.split('-'))))

    result\_string = ''

    for i in range(len(numbers)):

        result\_string += letters[numbers[i] - 1]

    output\_entry.delete("1.0", tk.END)

    output\_entry.insert("1.0", result\_string)

**Задание 2:**

Алгоритм шифрования ГОСТ 28147 89

Зашифруйте строку: <<Я хочу хоть с одним человеком обо всём говорить, как с собой.>>

Алгоритмом шифрования ГОСТ 28147 89 в режиме гаммирование.

Ключ для шифрования: 0825163986280077626186437455897039834754944246885404358645718834

Используйте узлы замены:

11 11 19 10 14 19 13 16

16 20 20 14 12 13 19 10

13 19 17 11 13 17 15 19

14 15 16 19 10 16 10 13

12 20 10 12 12 13 15 16

15 10 20 15 16 17 13 19

10 17 16 11 16 15 19 14

20 12 10 16 13 11 17 19

17 11 17 17 19 12 11 16

19 13 17 15 12 16 17 12

14 20 11 15 11 19 16 17

18 19 13 16 17 19 19 13

12 16 18 13 10 14 10 19

20 13 17 13 18 14 19 13

15 13 18 18 17 17 10 19

19 19 14 18 17 10 14 19

**Описание алгоритма:**

1. **Инициализация:**
   * Задать ключ шифрования длиной 256 бит (32 байта).
   * Сгенерировать случайное начальное значение (IV) длиной 64 бита (8 байт).
2. **Разбиение данных:**
   * Разбить открытый текст на блоки по 64 бита (8 байт).
3. **Шифрование:**
   * Для каждого блока открытого текста выполнить следующие шаги:
     1. Выполнить операцию XOR между блоком открытого текста и предыдущим зашифрованным блоком (или IV для первого блока).
     2. Зашифровать получившийся результат с использованием ключа шифрования ГОСТ 28147-89.
     3. Полученный шифротекст является зашифрованным блоком.
     4. Передать полученный зашифрованный блок на следующий шаг.
4. **Формирование шифротекста:**
   * Собрать все зашифрованные блоки вместе.
5. **Вывод шифротекста.**

**Экранные формы:**

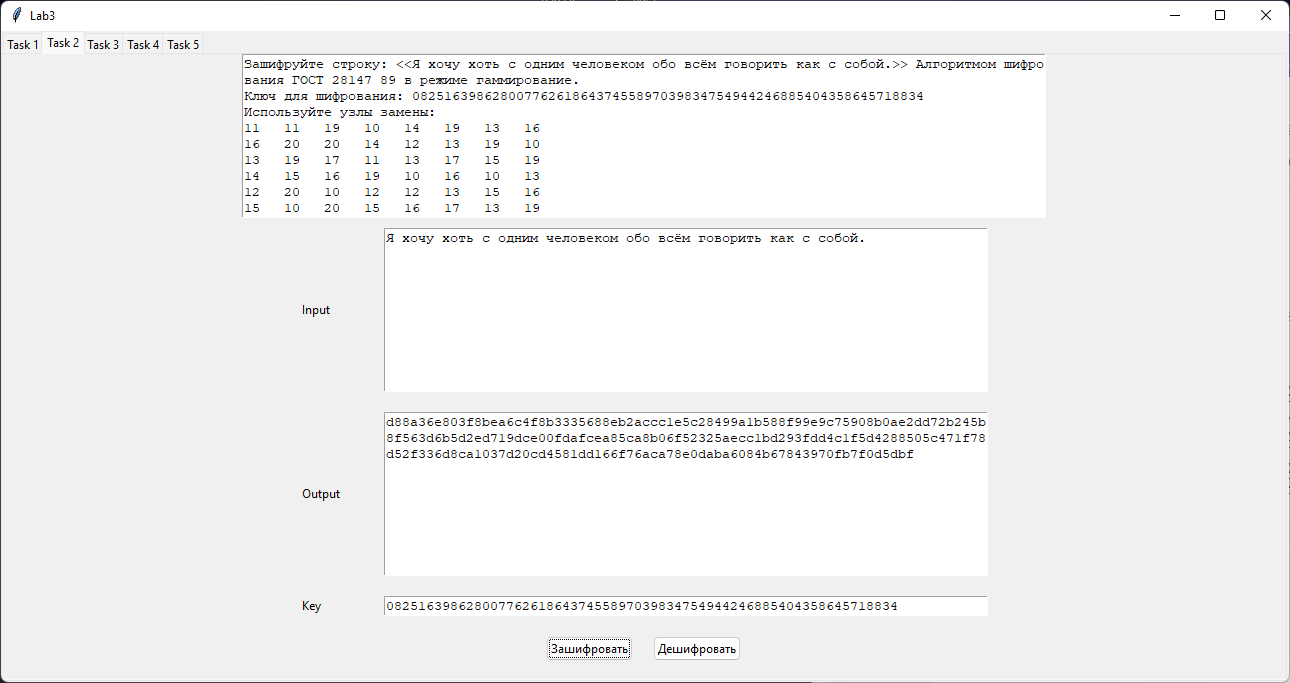


Рисунок 2 - Экранная форма выполнения задания 2

**Код программы:**

import tkinter as tk

# Таблица замены

substitution\_table = [

    11, 11, 19, 10, 14, 19, 13, 16,

    16, 20, 20, 14, 12, 13, 19, 10,

    13, 19, 17, 11, 13, 17, 15, 19,

    14, 15, 16, 19, 10, 16, 10, 13,

    12, 20, 10, 12, 12, 13, 15, 16,

    15, 10, 20, 15, 16, 17, 13, 19,

    10, 17, 16, 11, 16, 15, 19, 14,

    20, 12, 10, 16, 13, 11, 17, 19,

    17, 11, 17, 17, 19, 12, 11, 16,

    19, 13, 17, 15, 12, 16, 17, 12,

    14, 20, 11, 15, 11, 19, 16, 17,

    18, 19, 13, 16, 17, 19, 19, 13,

    12, 16, 18, 13, 10, 14, 10, 19,

    20, 13, 17, 13, 18, 14, 19, 13,

    15, 13, 18, 18, 17, 17, 10, 19,

    19, 19, 14, 18, 17, 10, 14, 19

]

# Функция для выполнения операции гаммирования (XOR)

def xor\_bytes(a, b):

    return bytes(x ^ y for x, y in zip(a, b))

# Функция для выполнения одного шага алгоритма ГОСТ

def gost\_step(key, data):

    result = []

    key\_len = len(key)

    for i in range(8):

        x = int.from\_bytes(data[i\*4:(i+1)\*4], byteorder='little')

        k = int.from\_bytes(key[i\*4 % key\_len:(i\*4 % key\_len)+4], byteorder='little')

        y = (x + k) % (2\*\*32)

        s = (y >> 24) & 0xFF

        t = (y >> 16) & 0xFF

        u = (y >> 8) & 0xFF

        v = y & 0xFF

        #w = (((s ^ t) + u) ^ v) & 0xFF

        result.extend([s, t, u, v])

    return bytes(result)

def string\_to\_bytes(text):

    return text.encode('utf-8')

def bytes\_to\_string(data):

    return data.decode('utf-8')

def gost\_encrypt(key, plaintext):

    ciphertext = b''

    for i in range(0, len(plaintext), 8):

        block = plaintext[i:i+8]

        block = xor\_bytes(block, key)

        key = gost\_step(key, substitution\_table)

        ciphertext += block

    return ciphertext

def gost\_decrypt(key, ciphertext):

    plaintext = b''

    for i in range(0, len(ciphertext), 8):

        block = ciphertext[i:i+8]

        block = xor\_bytes(block, key)

        key = gost\_step(key, substitution\_table)

        plaintext += block

    return plaintext

def encrypt\_text(input\_entry, output\_entry, key\_entry):

    input\_text = input\_entry.get("1.0", "end-1c")

    plaintext\_bytes = string\_to\_bytes(input\_text)

    key = bytes.fromhex(key\_entry.get("1.0", "end-1c"))

    encrypted\_text = gost\_encrypt(key, plaintext\_bytes)

    output\_entry.delete("1.0", tk.END)

    output\_entry.insert("1.0", encrypted\_text.hex())

def decrypt\_text(input\_entry, output\_entry, key\_entry):

    input\_text = input\_entry.get("1.0", "end-1c")

    byte\_string = bytes.fromhex(input\_text)

    key = bytes.fromhex(key\_entry.get("1.0", "end-1c"))

    decrypted\_text = gost\_decrypt(key, byte\_string)

    print(decrypted\_text)

    output\_entry.delete("1.0", tk.END)

    output\_entry.insert("1.0", bytes\_to\_string(decrypted\_text))

**Задание 3:**

Алгоритм шифрования rsa.

Сгенерируйте открытый и закрытый ключи в алгоритме шифрования RSA, выбрав простые числа p = 577 и q = 457.

Зашифруйте сообщение: Но никто не знает, что самое большое счастье — в понимании.

**Описание алгоритма:**

Берется 2 простых числа p и q. Определяется результат их перемножения n. Выбирается случайное число e, которое взаимно простое с (p-1)\*(q-1). Определяется такое число d, для которого является истинным следующее выражение (e\*d) mod (p-1)\*(q-1) = 1.

Открытым ключом являются числа e и n, закрытым d и n.

Шифрование выполняется следующими действиями:

1. Шифруемый текст разбивается на блоки, каждый из которых может быть представлен в виде числа M(i)=0,1,2..., n-1
2. Зашифрованный текст, рассматриваемый как последовательность чисел M(i) по формуле C(i)=(M(I)^e) mod n.

**Экранные формы:**

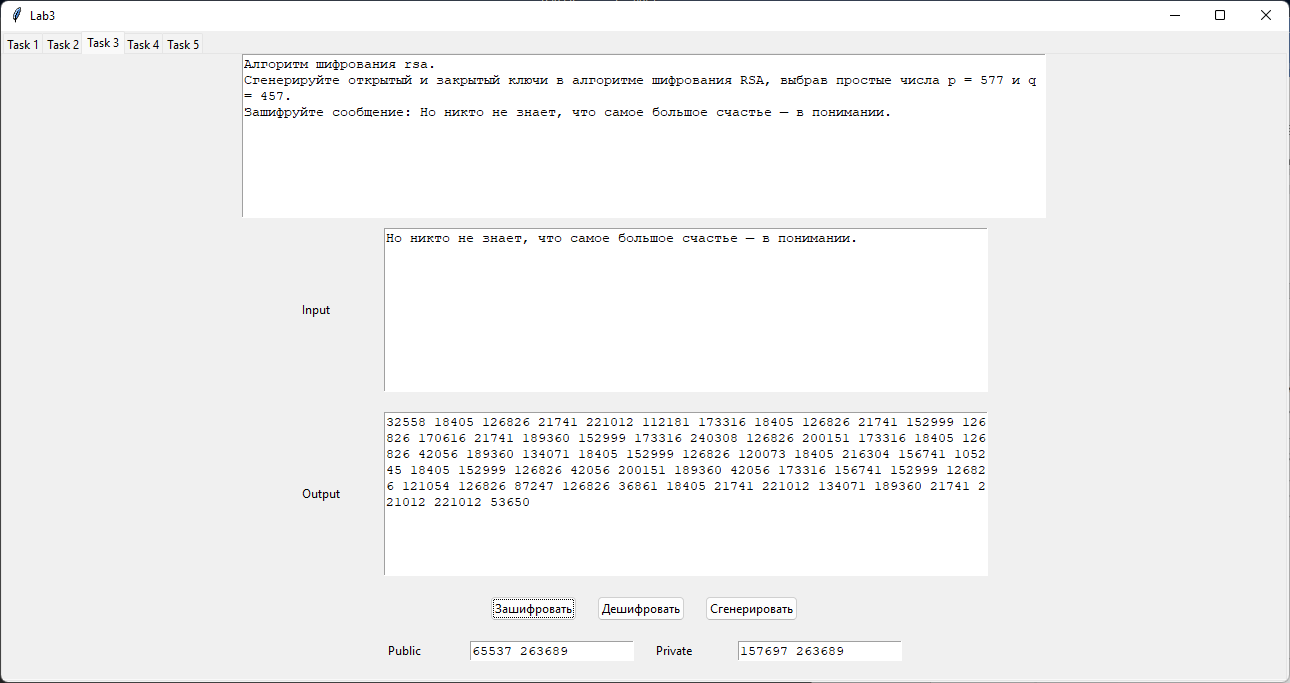


Рисунок 3 - Экранная форма выполнения задания 3

**Код программы:**

import tkinter as tk

def gcd(a, b):

    while b != 0:

        a, b = b, a % b

    return a

def modinv(a, m):

    m0, x0, x1 = m, 0, 1

    while a > 1:

        q = a // m

        m, a = a % m, m

        x0, x1 = x1 - q \* x0, x0

    return x1 + m0 if x1 < 0 else x1

def generate\_keys(p, q):

    n = p \* q

    phi\_n = (p - 1) \* (q - 1)

    # Выбираем открытую экспоненту e

    e = 65537

    # Находим закрытую экспоненту d

    d = modinv(e, phi\_n)

    return (e, n), (d, n)

def encrypt(message, public\_key):

    e, n = public\_key

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

    return encrypted\_message

def decrypt(encrypted\_message, private\_key):

    d, n = private\_key

    decrypted\_message = ''.join([chr(pow(char, d, n)) for char in encrypted\_message])

    return decrypted\_message

def encrypt\_sig(message, private\_key):

    d, n = private\_key

    encrypted\_message = [pow(byte, d, n) for byte in message]

    return encrypted\_message

def decrypt\_sig(encrypted\_message, public\_key):

    e, n = public\_key

    decrypted\_message = [pow(byte, e, n) for byte in encrypted\_message]

    return decrypted\_message

p = 577

q = 457

def generate(public, private):

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

    public.delete("1.0", tk.END)

    public.insert("1.0", public\_key)

    private.delete("1.0", tk.END)

    private.insert("1.0", private\_key)

def encrypt\_text(input\_entry, output\_entry, public\_entry):

    input\_text = input\_entry.get("1.0", "end-1c")

    public\_key = tuple(map(int, public\_entry.get("1.0", "end-1c").split()))

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

    output\_entry.delete("1.0", tk.END)

    output\_entry.insert("1.0", encrypted\_text)

def decrypt\_text(input\_entry, output\_entry, private\_entry):

    input\_text = input\_entry.get("1.0", "end-1c")

    private\_key = tuple(map(int, private\_entry.get("1.0", "end-1c").split()))

    numbers\_list = list(map(int, input\_text.split()))

    decrypted\_text = decrypt(numbers\_list, private\_key)

    output\_entry.delete("1.0", tk.END)

    output\_entry.insert("1.0", decrypted\_text)

**Задание 4:**

Функция хеширования.

Реализовать алгоритм криптографической функции – adler32.

Зашифруйте сообщение: И нет величия там, где нет простоты, добра и правды.

**Описание алгоритма:**

1. **Инициализация:**
   * Установить две переменные **s1** в начальное значение 0 и **s2** в начальное значение 1.
2. **Хеширование:**
   * Разбить входные данные на блоки определенного размера (обычно 16 байт).
   * Для каждого байта **b** в блоке выполнить следующие действия:
     1. Увеличить **s1** на значение **b**.
     2. Увеличить **s2** на значение **s1**.
   * Если размер блока не кратен 16, выполнить дополнительные шаги для последних байтов.
3. **Финализация:**
   * Результатом является конкатенация значений **s2** и **s1**.
   * Если значение **s1** или **s2** выходит за пределы 16 бит (65535), выполнить модульное деление на 65521.
4. **Вывод хеша:**
   * Вывести полученное значение как контрольную сумму хеша.

**Экранные формы:**

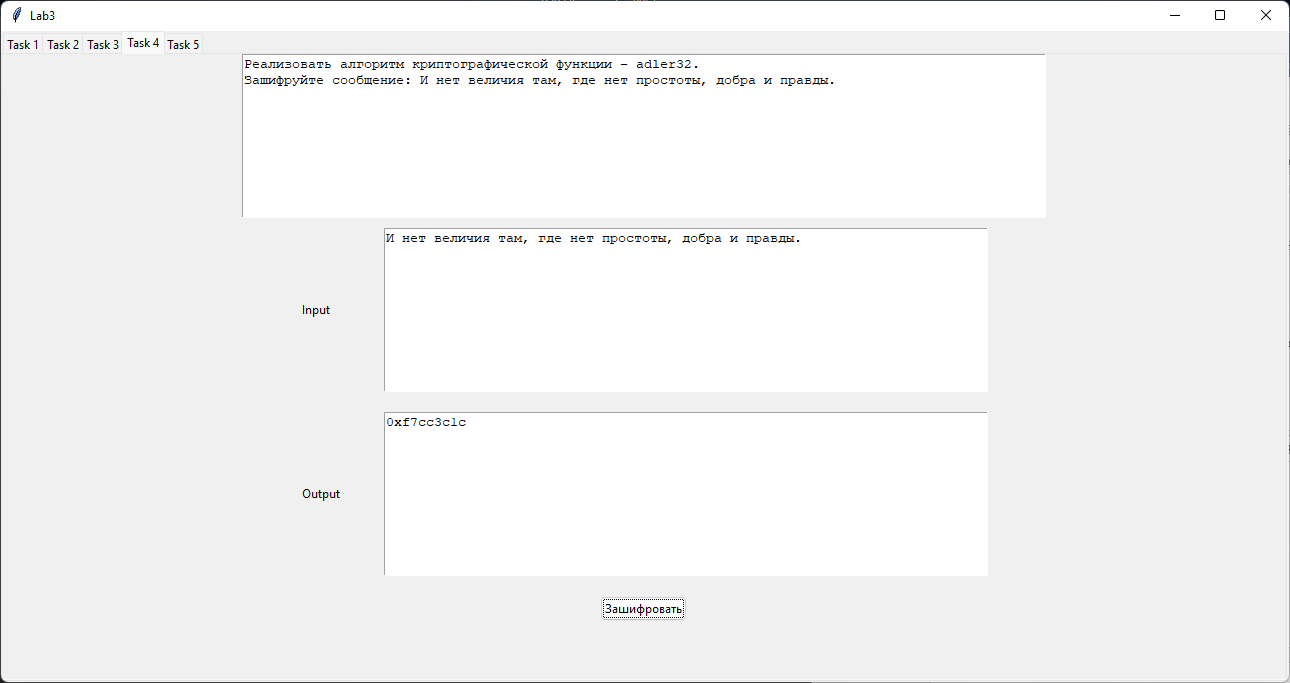


Рисунок 4 - Экранная форма выполнения задания 4

**Код программы:**

import tkinter as tk

def encrypt\_adler32(string):

    MOD\_ADLER = 65521

    s1 = 1

    s2 = 0

    for byte in string:

        s1 = (s1 + byte) % MOD\_ADLER

        s2 = (s2 + s1) % MOD\_ADLER

    return hex((s2 << 16) + s1)

def encrypt\_text(input\_entry, output\_entry):

    input\_text = input\_entry.get("1.0", "end-1c")

    encoded\_message = input\_text.encode('utf-8')

    result\_string = encrypt\_adler32(encoded\_message)

    output\_entry.delete("1.0", tk.END)

    output\_entry.insert("1.0", result\_string)

**Задание 5:**

Электронная цифровая подпись.

Используя хеш-образ строки: <<Скудость мысли порождает легионы единомышленников.>>

И вычислите электронную цифровую подпись по схеме RSA.

**Описание алгоритма:**

Электронная цифровая подпись сообщения, состоящего из Фамилии, вычисляется по правилу: s=hashd mod n, где hash – хэш образ сообщения, d и n – закрытый ключ.

Для проверки ЭЦП, используя открытый ключ (e, n)

H=se mod n. Если H=hash, то ЭЦП подлинная.

**Экранные формы:**

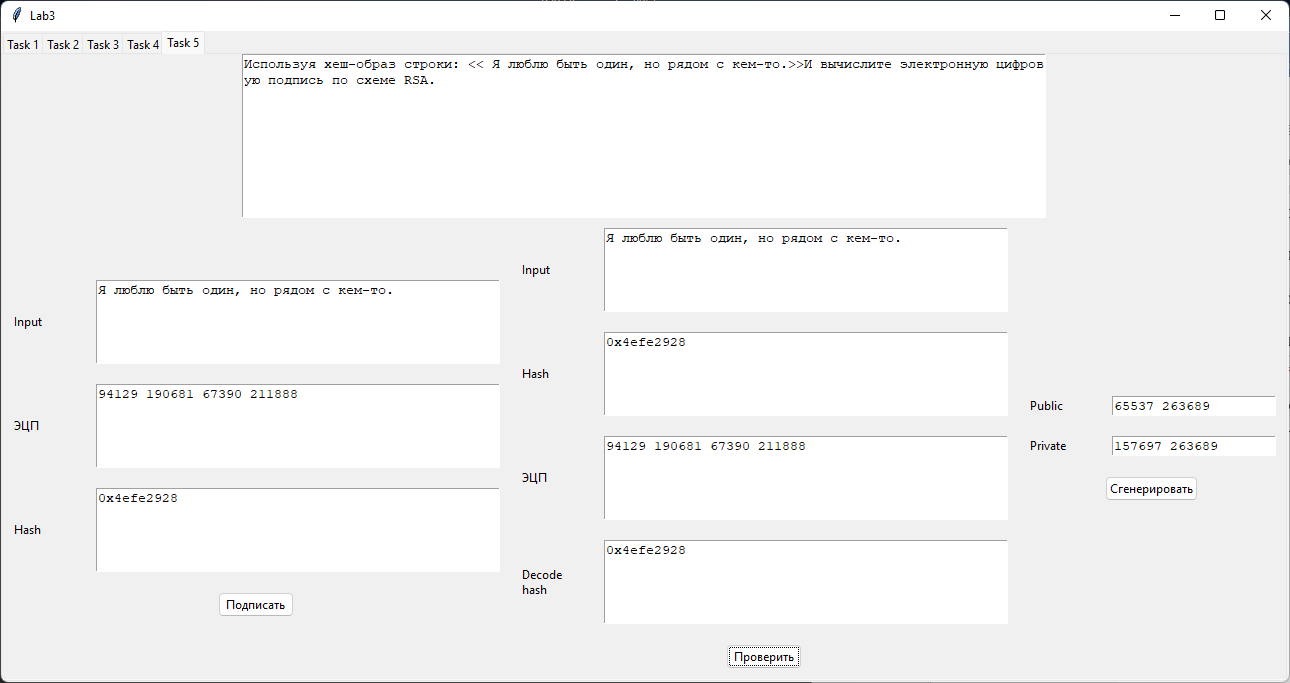


Рисунок 5 – Экранная форма выполнения задания 5

**Код программы:**

import tkinter as tk

import third\_task

import fourth\_task

def split\_hash(hash\_value):

    hash\_value = int(hash\_value, 16)

    byte1 = (hash\_value >> 24) & 0xFF

    byte2 = (hash\_value >> 16) & 0xFF

    byte3 = (hash\_value >> 8) & 0xFF

    byte4 = hash\_value & 0xFF

    return [byte1, byte2, byte3, byte4]

def bytes\_to\_hash(byte\_array):

    hash\_value = (byte\_array[0] << 24) | (byte\_array[1] << 16) | (byte\_array[2] << 8) | byte\_array[3]

    return hex(hash\_value)

def generate(public\_entry, private\_entry):

    public\_key, private\_key = third\_task.generate\_keys(third\_task.p, third\_task.q)

    public\_entry.delete("1.0", tk.END)

    public\_entry.insert("1.0", public\_key)

    private\_entry.delete("1.0", tk.END)

    private\_entry.insert("1.0", private\_key)

def encrypt\_text(input\_entry, output\_entry1, output\_entry11, private\_entry):

    input\_text = input\_entry.get("1.0", "end-1c")

    byte\_message = input\_text.encode('utf-8')

    hash\_value = fourth\_task.encrypt\_adler32(byte\_message)

    hash\_bytes = split\_hash(hash\_value)

    private\_key = tuple(map(int, private\_entry.get("1.0", "end-1c").split()))

    signature = third\_task.encrypt\_sig(hash\_bytes, private\_key)

    output\_entry1.delete("1.0", tk.END)

    output\_entry1.insert("1.0", signature)

    output\_entry11.delete("1.0", tk.END)

    output\_entry11.insert("1.0", hash\_value)

def decrypt\_text(input\_entry, output\_entry2, output\_entry22, output\_entry222, public\_entry):

    input\_text = input\_entry.get("1.0", "end-1c")

    byte\_message = input\_text.encode('utf-8')

    hash\_value = fourth\_task.encrypt\_adler32(byte\_message)

    input\_numbers = output\_entry22.get("1.0", "end-1c")

    numbers\_list = list(map(int, input\_numbers.split()))

    public\_key = tuple(map(int, public\_entry.get("1.0", "end-1c").split()))

    decode\_hash\_bytes = third\_task.decrypt\_sig(numbers\_list, public\_key)

    decode\_hash = bytes\_to\_hash(decode\_hash\_bytes)

    output\_entry2.delete("1.0", tk.END)

    output\_entry2.insert("1.0", hash\_value)

    output\_entry222.delete("1.0", tk.END)

    output\_entry222.insert("1.0", decode\_hash)

**Выводы:**

В результате лабораторной работы были изучены и реализованы

некоторые алгоритмы и способы шифрования данных. Реализован алгоритм

RSA и алгоритм цифровой подписи. Для хеширования использовался алгоритм adler32. Наиболее трудным в реализации оказался шифр

блочный ГОСТ.