Министерство образования и науки Российской Федерации.
Санкт-Петербургский Государственный Морской
Технический Университет
Факультет корабельной энергетики и автоматики
Кафедра судовой автоматики и измерений

КУРСОВОЙ ПРОЕКТ

«Разработка комплекса лабораторных работ по курсу
Криптографические методы защиты информации
III курс 5 семестр»

по дисциплине
«Криптографические методы защиты информации»

Выполнил:
Студент гр. 2350

Соколов Г. А.

Проверил:
Доцент кафедры судовой автоматики и измерений

Шавинская С. К.

Дата сдачи отчёта:
27.12.2022

Санкт-Петербург

2022 г.

Оглавление

Введение

Целью данного курсового проекта является разработка программного комплекса лабораторных работ **«Crypto_2350»** по курсу «Криптографические Методы Защиты Информации» пятого семестра программы обучения по специальности **10.03.01** – «Информационная безопасность» на языке программирования **Python 3.8**.

Для достижения цели был прослушан курс лекций по криптографии, составлен конспект, разработана блок-схема реализации каждой лабораторной работы, составлен писок необходимых функций. После этого разрабатывался код, а также проводились тестирования с различными входными данными. В ходе работы понадобились навыки программирования на Python, знание IDE **Pycharm 2022.1**.

Итоговый Проект состоит из семи модулей: модуля криптографических функций (п.1), а также 6 лабораторных работ (п. 2-7), целью которых был разработка скриптов, эмулирующих различные протоколы обработки, передачи и шифрования данных. В данной работе, каждый пункт содержит:

- краткое описание программы
- таблицу используемых переменных
- пример работы программы в консоли
- код программы с комментариями

Проект **«Crypto_2350»** может быть использован в рамках учебного процесса на курсе «Криптографические Методы Защиты Информации» для проверки корректности решения задач в рамках освоения курса, а также служить наглядным пособием программной реализации различных криптографических алгоритмов.

В ходе работы над проектом прикладывались усилия для достижения максимальной гибкости (настраиваемости) используемых функций, возможности удобной коррекции кода (в одном месте, а не по всему проекту), расширяемости проекта (возможно дальнейшее расширения модуля криптографических функций, а также подключение новых модулей) и отсутствия дублирования кода (использование импортируемых функций).

Также побочной целью проекта **«Crypto_2350»** является размещение в публичном репозитории для возможности оперативного обновления проекта, а также для удобного доступа к нему.

Для получения необходимых теоретических сведений использовались материалы лекционных и практических занятий по Криптографии в СПбГМТУ за осенний семестр 2022 года, учебная литература (А. В. Черёмушкин – Лекции по арифметическим алгоритмам в криптографии), а также открытая информация в сети Интернет. Полный список используемой литературы указан на *странице 23*.

1. Модуль Используемых Криптографических Функций (prot_cryptofunctions)

1.1 Описание

Данный модуль был разработан как хранилище всех функций, используемых в каждой лабораторной работе (1-7).

Описание назначения каждой функции представлено в качестве комментария первой строчкой тела функции (*""" заключено в тройные кавычки """*).

Модуль импортируется и используется в каждой лабораторной работе.

Цели разработки модуля:

- Избегание дублирования кода в каждой лабораторной работе;
- Удобство вызова часто используемых функций;
- Возможность использовать изменяемые сообщения, задавая необходимый текст сообщения-приглашения («промпта», *prompt*), который помогает пользователю понять, какие данные вводятся в данный момент.

1.2 Код

```
# 2350 Соколов    |    +79217916237    |
M.L.Swgr@gmail.com
# Модуль для используемых функций
from math import *

# Функция от Жени Россамахина 2350
def slow_degree(number, degree, mod, lvl=2):
    """Быстрое возведение в степень по
модулю"""
    low_degree = degree // lvl
    count = 1
    if degree % 2 == 0:
        min_lvl = 0
    else:
        min_lvl = 1
    for i in range(lvl - min_lvl):
        count *= number ** low_degree % mod
    return count * (number ** (degree -
low_degree * (lvl - min_lvl)) % mod) % mod


def eul(n):
    """Считает функцию Эйлера"""
    import math
    q = 0
    for i in range(1, n):
        if (math.gcd(i, n) == 1):
            q += 1
        else:
            continue
    return q


def fermaeuler(a, m):
    """Теорема Ферма-Эйлера с исп. ф-и Эйлера
    Решает сравнение вида Ax=1(mod m)"""
    if mutprime(a, m):
        fi = eul(m)
        x = slow_degree(a, fi - 1, m, lvl=2)
        return x
    else:
        return False


def fermaeuler2(a, b, m):
    """Функция, решающая сравнение вида
Ax=b(mod m) с использованием теор. Ферма-Эйлера
    Решает сравнение вида Ax=B(mod m)"""
    x = slow_degree((fermaeuler(a, m) * b), 1,
m)
    return x


# Проверка на простое
def issimple(a):
    """Проверка на простоту перебором"""
    for i in range(2, (a // 2) + 1):
        if (a % i == 0):  # если остаток от
деления на какое-то число = 0
            return False  # -> число не простое
    return True


def mutprime(a, b):
    """Проверка на взаимную простоту"""
    import math
    if math.gcd(a, b) == 1:
        return True
    else:
        return False


def inbetween(x, a, b):
    """Простая проверка на нахождение в
промежутке"""
    if x <= a or x >= b:
        return False
    else:
        return True


# * * * ИНТЕРФЕЙСНЫЕ ФУНКЦИИ ВВОДА/ВЫВОДА * * *

def inputprimecheck(message=str,
message_fail=str):
    """функция ввода с клавиатуры с проверкой
на простое"""
    x = int(input(message))
    while (issimple(x) != True):
        x = int(input(message_fail))
    return x
```

```
def inputmutprimecheck(y, message=str,
message_fail=str):
    """функция ввода с клавиатуры с проверкой
на взаимную простоту с функцией Эйлера"""
    x = int(input(message))
    while (mutprime(x, eul(y)) != True):
        x = int(input(message_fail))
    return x

def inputmutprimecheck2(y, message=str,
message fail=str):
    """функция ввода с клавиатуры с проверкой
на взаимную простоту"""
    x = int(input(message))
    while (mutprime(x, y) != True):
        x = int(input(message_fail))
    return x


# 20.10
def input_inbetween_check(a, b, message=str,
message_fail=str):
    """Проверка нахождения вводимого числа в
промежутке"""
    x = int(input(message))
    while inbetween(x, a, b) == False:
        x = int(input(message fail))
    return x


def input_mutprime_inbetween(a, b, c,
message=str, message fail=str):
    """Проверка на нахождение в промежутке
     + на взаимную простоту с заданным
числом"""
    x = int(input(message))
    while mutprime(x, c) == False or
inbetween(x, a, b) == False:
        x = int(input(message fail))
    return x


def message_input(a, prompt=str,
message fail=str):
    """Ввод сообщения с проверкой длины"""
    m = int(input(prompt))
    while m >= a:
        m = int(input(message_fail))
    return m
```

```
# 1.12
def input_mutprime_full(spisok, prompt=str,
message_fail=str):
    """Функция ввода переменой m, чтобы она
была взаимно проста
    со всеми элементами списка spisok"""
    m = int(input(prompt))
    while rowcheck(spisok, m)!=True:
        m = int(input(message_fail))
    return m
```

```
def rowcheck(spisok,m):
    """Функция для проверки того
    что некая m взаимно проста
    со всеми элеметами списка spisok"""
    if len(spisok)>1:
        for i in range(len(spisok)):
            if gcd(m,spisok[i])!=1 or
m==spisok[i]:
                return False
    return True

# три функции для преобразования строку
сообщения в список цифр (коды символов)
# коды символов удобно передавать через
протоколы отдельно и получать не просто цифры а
текст
# ASCII имеет 127 символов (англ раскладка)


def text_crypt(message):
    """Функция для разбиение введённой строки
    на символы и сохранение кода каждого
символа в список.
    Возвращает список с кодами сомволов ввенной
строки
    Удобно для последовательной передачи
каждого кода по протоколу"""
    crypt messg = []
    for i in range(len(message)):
        crypt_messg.append(ord(message[i]))
    return crypt_messg


def text decrypt(crypt messg):
    """Функция преобразующая список кодов
символов обратно в текст
    Возвращает строку с символами"""
    decrypt messg = []
    for i in range(len(crypt messg)):

decrypt_messg.append(chr(crypt_messg[i]))
    return decrypt_messg


def text output(decrypt messg, info=str):
    """Функция вывода списка в виде строки
(текста) в консоль"""
    print(info, end=' ')
    print("".join((map(str, decrypt messg))))


def text_equalcheck(message=list,
message2=list):
    """Функция проверки совпадения
отправленного и расшифр. сообщ."""
    for i in range(len(message)):
        if message[i] != message2[i]:
            return False
    return True


def text equalcheck2(message=list,
message2=list):
    """Функция проверки совпадения
отправленного и расшифр. сообщ."""
    if message == message2:
        return True
    return False
```

2. Протокол Шифрования без Передачи Ключа (prot_nokey)

2.1 Описание

Данная программа реализует эмуляцию протокола шифрования без передачи ключа. Для реализации протокола шифрования без передачи ключа два абонента должны сначала встретиться лично и договориться о каком-то простом числе p. Необходимость личной встречи является недостатком данного протокола, наряду с необходимостью передавать сообщение 4 раза (по 2 раза в обе стороны), что повышает риск перехвата сообщения злоумышленником.

2.2 Таблица переменных

Имя переменной	Тип	Назначение	Примечание
p	*int*	Простое число p, о котором должны заранее договориться абоненты	*Вводится пользователем*
a	*int*	Открытый ключ a	*Вводится пользователем*
b	*int*	Открытый ключ b	*Вводится пользователем*
m	*int*	Сообщение	*Вводится пользователем*
m1	*int*	Дубликат сообщения для сравнения корректности передачи	
alpha	*int*	Секретный ключ для a	
beta	*int*	Секретный ключ для b	
res	*int*	Переменная для хранения результата обработки входных данных	

2.3 Работа программы

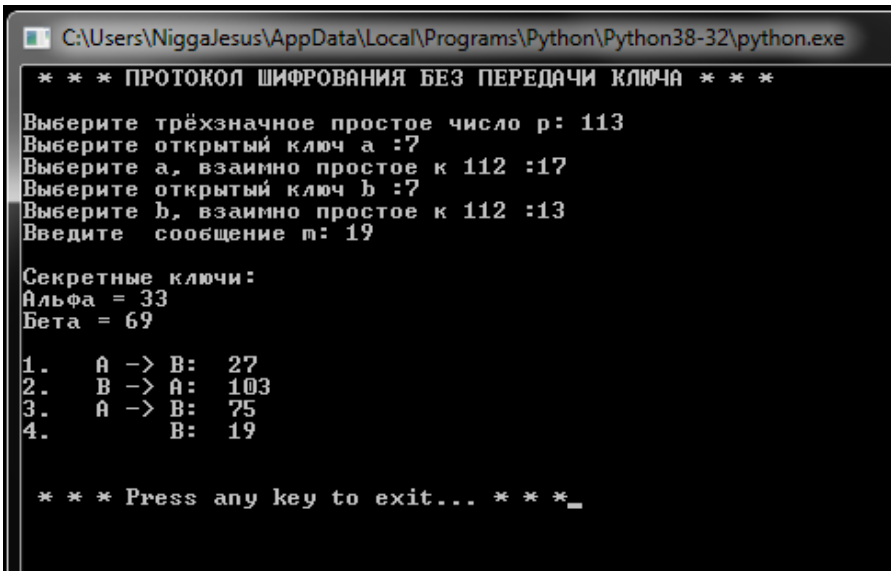

Рисунок 1

2.4 Код

```
import cryptofunctions[1]


# Функция протокола шифрования без передачи ключа
def prot_nokey():
    print(' * * * ПРОТОКОЛ ШИФРОВАНИЯ БЕЗ ПЕРЕДАЧИ КЛЮЧА * * *\n')
    # БЛОК ВВОДА ДАННЫХ
    # Выбор простого числа и открытых ключей
    p = cryptofunctions.inputprimecheck('Выберите трёхзначное простое число p: ',
'Выберите ПРОСТОЕ p: ')
    a = cryptofunctions.inputmutprimecheck(p, 'Выберите открытый ключ a :',
                                           f'Выберите a, взаимно простое к
{cryptofunctions.eul(p)} :')
    b = cryptofunctions.inputmutprimecheck(p, 'Выберите открытый ключ b :',
                                           f'Выберите b, взаимно простое к
{cryptofunctions.eul(p)} :')

    # Ввод сообщения с проверкой длины
    m = m1 = int(input('Введите  сообщение m: '))
    while m >= p:
        m = m1 = int(input('m должно быть меньше p. Введите m: '))

    # БЛОК ОБРАБОТКИ
    # непосредственно обработка сообщения
    alpha = cryptofunctions.fermaeuler(a, cryptofunctions.eul(p))  # функция эйлера от
простого p = (p-1)
    beta = cryptofunctions.fermaeuler(b, cryptofunctions.eul(p))
    print(f"\nСекретные ключи:\nАльфа = {alpha}\nБета = {beta}")
    res = cryptofunctions.slow_degree(m1, a, p, lvl=2)
    print("\n1.   A -> B: ", res)
    res = cryptofunctions.slow_degree(res, b, p, lvl=2)
    print("2.   B -> A: ", res)
    res = cryptofunctions.slow_degree(res, alpha, p, lvl=2)
    print("3.   A -> B: ", res)
    res = cryptofunctions.slow_degree(res, beta, p, lvl=2)
    print(f'4.        B:  {res}')
    return res


prot_nokey()  # в итоге получаем функцию протокола с возможностью ввода данных кот.
можно вызвать одной строкой
input('\n\n * * * Press any key to exit... * * *')
```

[1] Команда для импорта разработанного модуля Киптографических Функций (см. п.1)

3. Протокол Шифрования с Открытым Ключом (prot_RSA)

3.1 Описание

Протокол шифрования с открытым ключом не требует заранее договариваться о каком-либо числе или ключе. Данные об абоненте (его имя, открытый ключ и число r, являющееся произведением двух простых чисел) публикуются в телефонной книге. Пользуясь этими данными и имея собственный набор, любой человек может отправить зашифрованное сообщение любому абоненту из телефонной книги.

3.2 Таблица переменных

Имя переменной	Тип	Назначение	Примечание
p1	*int*	Простое число 1	*Вводится пользователем*
p2	*int*	Простое число 2	*Вводится пользователем*
r	*int*	Число явл. произведением p1 * p2	
openkey	*int*	Открытый ключ	*Вводится пользователем*
secretkey	*int*	Закрытый ключ	
l	*list*	Список данных абонента	
m	*int*	Сообщение	*Вводится пользователем*
m1	*int*	Дубликат сообщения для сравнения корректности передачи	

3.3 Работа программы

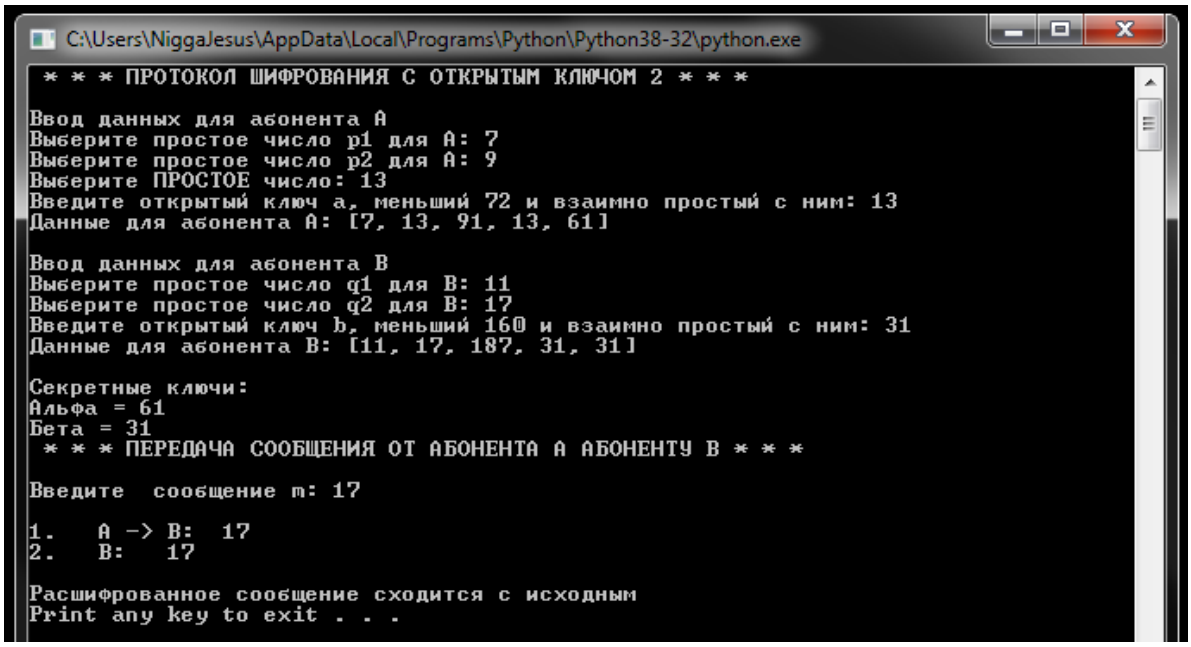

Рисунок 2

3.4 Код

```
import cryptofunctions

def rsa_input(x, y, abonent, a):
    print(f'\nВвод данных для абонента {abonent}')
    p1 = cryptofunctions.inputprimecheck(f'Выберите простое число {x} для {abonent}:
', 'Выберите ПРОСТОЕ число: ')
    p2 = cryptofunctions.inputprimecheck(f'Выберите простое число {y} для {abonent}:
', 'Выберите ПРОСТОЕ число: ')
    r = p1 * p2
    openkey = cryptofunctions.inputmutprimecheck(r,
                                                 f'Введите открытый ключ {a}, меньший
{cryptofunctions.eul(r)} и взаимно простый с ним: ',
                                                 f'Неудача. Введите a, которое МЕНЬШЕ и
взаимно просто с {cryptofunctions.eul(r)}: ')
    secretkey = cryptofunctions.fermaeuler(openkey, cryptofunctions.eul(r))
    l=[p1,p2,r,openkey,secretkey]
    print(f'Данные для абонента {abonent}: {l}')
    return l

def message_input(r,a,b):
    print(f' * * * ПЕРЕДАЧА СООБЩЕНИЯ ОТ АБОНЕНТА {a} АБОНЕНТУ {b} * * *\n')
    m = m1 = int(input('Введите  сообщение m: '))
    while m >= r:
        m = m1 = int(input(f'm должно быть меньше {cryptofunctions.eul(r)}. Введите m:
'))
    return m,m1

def rsa_process(message, r, openkey, secretkey):    # Непосредственно сама обработка
    message=cryptofunctions.slow_degree(message,openkey,r,lvl=2)
    print(f'\n1.   A -> B:  {message}')
    message=cryptofunctions.slow_degree(message,secretkey,r,lvl=2)
    print(f'2.   B:   {message}')
    return message

def rsa_check(m,m1):
    if m1==m:
        check=True
        print('\nРасшифрованное сообщение сходится с исходным')
    else:
        print('\nРасшифрованное сообщение НЕ сходится с исходным')
        check=False
    return check

def RSA(): # Rivest Shamir Adelman
    print(' * * * ПРОТОКОЛ ШИФРОВАНИЯ С ОТКРЫТЫМ КЛЮЧОМ 2 * * *')
    l1=rsa_input('p1', 'p2', 'A', 'a')
    l2=rsa_input('q1', 'q2', 'B', 'b')
    print(f"\nСекретные ключи:\nАльфа = {l1[4]}\nБета = {l2[4]}")

    m,m1=message_input(l2[2],'A','B')
    rsa_process(m1, l2[2], l2[3], l2[4])
    rsa_check(m,m1)
    input('Print any key to exit . . .')

RSA() # вызов функции одной строкой
```

4. Схема Шифрования Эль-Гамаля (prot_elgamal)

4.1 Описание

Схема Эль-Гамаля — криптосистема с открытым ключом, основанная *на трудности вычисления дискретных логарифмов в конечном поле*. Криптосистема включает в себя алгоритм шифрования и алгоритм цифровой подписи. Схема Эль-Гамаля лежит в основе бывших стандартов электронной цифровой подписи в США (DSA) и России (ГОСТ Р 34.10-94).

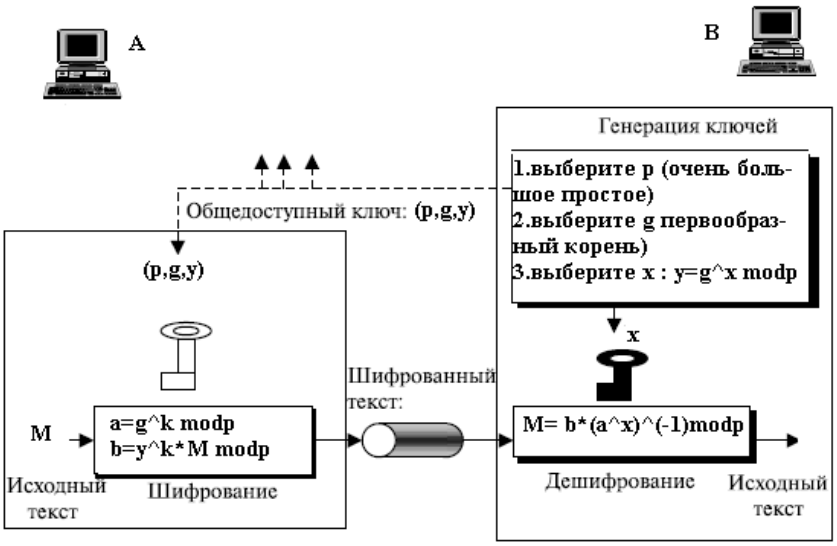

Рисунок 3 – схема работы протокола шифрования Эль-Гамаля

4.2 Таблица переменных

Имя переменной	Тип	Назначение	Примечание
p	*int*	Простое число p	*Вводится пользователем*
g	*int*	Число G < p	*Вводится пользователем*
x	*int*	Секретный ключ	*Вводится пользователем*
k	*int*	Целое k :: 1 < k < (p-1)	*Вводится пользователем*
abonent_info	*list*	Список для хранения информации об абоненте	
openkey	*int*	Открытый ключ	
a	*int*	Первое из пары чисел, получаемых адресатом на первом шаге	
b	*int*	Второе из пары чисел, получаемых адресатом на первом шаге	
res	*int*	Переменная для зранения результата обработки	
text	*str*	Сообщение - текст	*Вводится пользователем*
text2	*str*	Дубликат текстового сообщения для проверки корректности передачи сравнением	

4.3 Работа программы

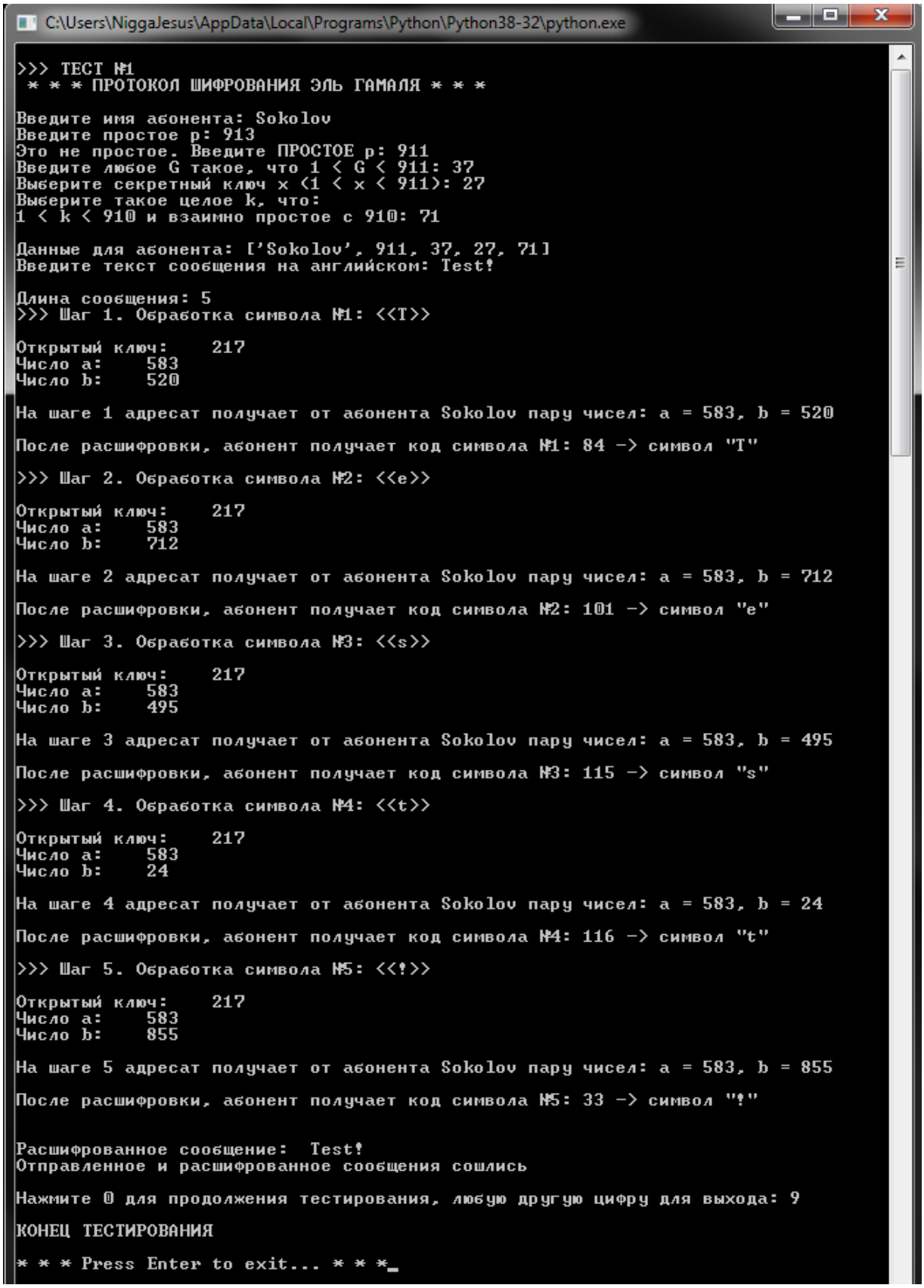

Рисунок 4

4.4 Код

```
import cryptofunctions as crf

def elgamal input(abonent):
    """Эльгамаль :: Функция для ввода данных с клавиатуры"""
    p = crf.inputprimecheck(f'Введите простое p: ', f'Это не простое. Введите ПРОСТОЕ p: ')
    g = crf.input inbetween check(1, p, f'Введите любое G такое, что 1 < G < {p}: ',
                                  f'Не подходит. Введите любое G такое, что 1 < G < {p}: ')
    x = crf.input inbetween check(1, p, f'Выберите секретный ключ x (1 < x < {p}): ',
                                  f'Выберите другой секретный ключ 1 < x < {p}: ')
    k = crf.input_mutprime_inbetween(1, p - 1, p - 1,
                                     f'Выберите такое целое k, что:\n1 < k < {p - 1} и взаимно
простое с {p - 1}: ',
                                     f'Не подходит, выберите 1 < k < {p - 1} и взаимно простое с
{p - 1}: ')

    abonent_info = [abonent, p, g, x, k]
    print(f'\nДанные для абонента: {abonent_info}')
    return abonent info


def elgamal_process(ab_info, letter, no):
    """Эльгамаль :: Функция обрабатывающая введённые данные"""
    openkey = pow(ab info[2], ab info[3], ab info[1])
    a = pow(ab info[2], ab info[4], ab info[1])
    b = crf.slow degree((openkey ** ab info[4]) * letter, 1, ab info[1])

    print(f'\nОткрытый ключ:    {openkey}\nЧисло a:    {a}\nЧисло b:    {b}\n')
    print(f'На шаге {no} адресат получает от абонента {ab_info[0]} пару чисел: a = {a}, b =
{b}\n')
    res = crf.fermaeuler2(a ** ab info[3], b, ab info[1])
    print(f'После расшифровки, абонент получает код символа №{no}: {res} -> символ
"{chr(res)}"\n')
    return res


def elgamal():
    """Эльгамаль :: функция полного протокола"""
    print(' * * * ПРОТОКОЛ ШИФРОВАНИЯ ЭЛЬ ГАМАЛЯ * * *\n')
    ab1 = elgamal_input(input(f'Введите имя абонента: '))

    text = crf.text crypt(input("Введите текст сообщения на английском: "))
    text2 = []
    print(f'\nДлина сообщения: {len(text)}')
    for i in range(len(text)):
        print(f'>>> Шаг {i + 1}. Обработка символа №{i + 1}: <<{chr(text[i])}>>')
        res = elgamal process(ab1, text[i], i + 1)
        text2.append(res)
    crf.text_output(crf.text_decrypt(text2), '\nРасшифрованное сообщение: ')  # вывод на экран


    #поправить проверку совпадения сообщений. можно проще сделать.
    if crf.text equalcheck2(text, text2):
        print("Отправленное и расшифрованное сообщения сошлись")
    else:
        print("Отправленное и расшифрованное сообщения НЕ сошлись")


a = 0
i = 1
while a == 0:  # Цикл для зацикливания тестирования
    print(f'\n>>> ТЕСТ №{i}')
    elgamal()  # получаем функцию, которую можно вызвать в 1 строчку без передачи аргументов на
вход
    a = int(input('\nНажмите 0 для продолжения тестирования, любую другую цифру для выхода: '))
    i += 1
input('\nКОНЕЦ ТЕСТИРОВАНИЯ\n\n* * * Press Enter to exit... * * *')
```

5. Китайская Теорема об Остатках (chinesetheorem)

5.1 Описание

Эта теорема в её арифметической формулировке была описана в трактате китайского математика Сунь Цзы «Сунь Цзы Суань Цзин» (кит. упр. 孙子算经, пиньинь *sunzi suanjing*), предположительно датируемом III веком н. э. и затем была обобщена Цинь Цзюшао в его книге «Математические рассуждения в 9 главах», датируемой 1247 годом, где было приведено точное решение.

Пусть $n_1, n_2, \ldots, n_k$ — натуральные попарно взаимно простые числа, а $r_1, r_2, \ldots, r_k$ — некоторые целые числа, тогда существует такое целое число M, что оно будет решением системы сравнений:

$$\begin{cases} M \equiv r_1 \pmod{n_1}, \\ M \equiv r_2 \pmod{n_2}, \\ \ldots \\ M \equiv r_k \pmod{n_k}. \end{cases}$$

При этом для любых двух решений A и B этой системы справедливо

$$A \equiv B \pmod{n_1 n_2 \ldots n_k},$$

то есть решение системы сравнений существует и единственно по модулю $n_1 n_2 \ldots n_k$.

Рисунок 5

5.2 Таблица переменных

Имя переменной	Тип	Назначение	Примечание
q	*int*	Количество уравнений в системе	*Вводится пользователем*
a	*int*	Коэффициент справа от знака =, перед модулем	*Вводится пользователем*
m	*int*	Модуль. Все модули должны быть взаимно просты друг с другом!	*Вводится пользователем*
system	*list*	Двумерный список для хранения коэффициентов	
M	*int*	Переменная хранения произведения ряда показателей модуля	
Mi_list	*list*	Список для хранения $M_i = M/m_i$	
Ni_list	*list*	Список хранения $N_i :: M_i = 1 \pmod{m_i}$	
x	*int*	Список хранения суммы ряда ai * Ni * Mi (mod M) + переменная для хранения итогового ответа (рассчитанного значения x)	*Итоговый ответ*

5.3 Работа программы

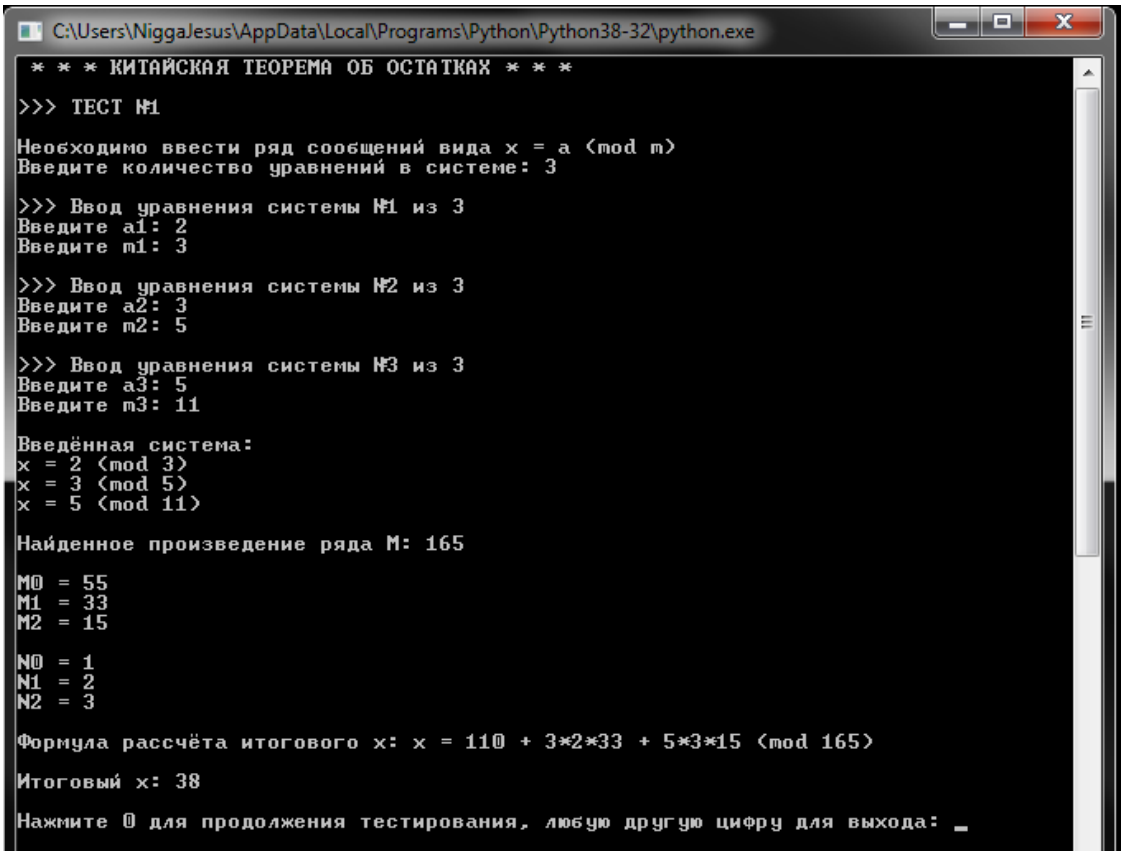

Рисунок 6

5.4 Код

```
from cryptofunctions import *

"""Китайская теорема об остатках (КТО)"""


def cht_input():
    """КТО :: функция ввода данных"""
    system = []
    q = int(input('\nНеобходимо ввести ряд сообщений вида x = a (mod m)\nВведите
количество уравнений в системе: '))
    a = []
    m = []
    for i in range(q):
        # Нужна проверка на простоту всего введнного
        a.append(int(input(f'\n>>> Ввод уравнения системы №{i + 1} из {q}\nВведите a{i
+ 1}: ')))
        m.append(
            int(input_mutprime_full(m, f'Введите m{i + 1}: ', f'Не взаимно с каким-то
из ранее введнных m, ещё раз: ')))
    system.append(a)
    system.append(m)

    # Вывод введенной системы на экран
    print(f'\nВведённая система:')
    for i in range(len(system[0])):
        print(f'x = {system[0][i]} (mod {system[1][i]})')

    return system
```

```
def cht_process(system):
    """КТО :: функция обработки данных"""
    # Определяем M = П(0,i)mi
    M = 1
    for i in range(len(system[0])):
        M *= system[1][i]
    print(f'\nНайденное произведение ряда M: {M}\n')

    # определяем каждую Mi = M/mi
    Mi_list = []
    for i in range(len(system[0])):
        Mi_list.append(M // system[1][i])
        print(f'M{i} = {Mi_list[i]}')  # вывод полученных Mi
    print('')

    # Находим все Ni, решая сравнения: Mi*Ni=1(mod mi) с помощью ферма эйлера
fermaeuler
    Ni_list = []
    for i in range(len(system[0])):
        Ni_list.append(fermaeuler(Mi_list[i], system[1][i]))
        print(f'N{i} = {Ni_list[i]}')  # вывод полученных Ni

    # Находим x = ROWSUM(ai*Ni*Mi(mod M))
    x = 0
    for i in range(len(system[0])):
        x += system[0][i] * Ni_list[i] * Mi_list[i]
    x = slow_degree(x, 1, M, lvl=2)

    # Эти три строчки просто для вывода формулы, моно удалить
    print(f'\nФормула рассчёта итогового x: x = {system[0][0] * Ni_list[0] *
Mi_list[0]}', end='')
    for i in range(1, len(system[0])):
        print(f' + {system[0][i]}*{Ni_list[i]}*{Mi_list[i]}', end='')
    print(f' (mod {M})')

    print(f'\nИтоговый x: {x}')


def cht():
    """КТО: общая функция обработки"""
    system = cht_input()
    cht_process(system)


a = 0
i = 1
while a == 0:  # Цикл для зацикливания тестирования
    print(' * * * КИТАЙСКАЯ ТЕОРЕМА ОБ ОСТАТКАХ * * *')
    print(f'\n>>> ТЕСТ №{i}')
    cht()  # получаем функцию, которую можно вызвать в 1 строчку без передачи
аргументов на вход
    a = int(input('\nНажмите 0 для продолжения тестирования, любую другую цифру для
выхода: '))
    i += 1
input('\nКОНЕЦ ТЕСТИРОВАНИЯ\n\n* * * Press Enter to exit... * * *')
```

6. Протокол с Нулевой Передачей Знаний (prot_zeroknowledge)

6.1 Описание

Широкое распространение смарт-карт (интеллектуальных карт, токенов и т.д.) для разнообразных коммерческих, гражданских и военных применений (кредитные карты, карты социального страхования, карты доступа в охраняемые помещения, компьютерные пароли и ключи и т.д.) потребовало обеспечение безопасности идентификации таких карт и их владельцев. Во многих приложениях главная проблема заключается в том, чтобы при предъявлении смарт-карты оперативно обнаружить обман и отказать обманщику в допуске, ответе и обслуживании.

Для безопасного использования смарт-карт разработаны протоколы идентификации с нулевой передачей знаний. Ключ (пароль) при этом *не передаётся в открытом виде*. Доказательство знания этого секретного ключа с нулевой передачей этого знания служит доказательством подлинности личности владельца карты.

Схему идентификации с нулевой передачей знаний предложили в 1986 г. **У.Фейге**, **А.Фиат** и **А.Шамир**. Она является наиболее известным доказательством идентичности с нулевой передачей конфиденциальной информации.

В данной лабораторной работе эмулируется **параллельная схема передачи** по протоколу с нулевой передачей знаний.

6.2 Таблица переменных

Имя переменной	Тип	Назначение	Примеч.
openkey	*list*	список открытых ключей	
openkey_reverse	*list*	список обратных к открытым ключам	
secretkey	*list*	список секретных ключей s	
mutprime_n	*list*	список квадратичных вычетов n, взаимно простых с n	
vychety	*list*	список квадратичных вычетов n	
ping	*list*	список возможных сигналов отправляемых на втором шаге: b = 0 v 1	*ping = [0, 1]*
p	*int*	простое число p	*Вводится пользователем*
q	*int*	простое число q	*Вводится пользователем*
n	*int*	модуль n = p * q	
d	*int*	переменная для промежуточного хранения данных в процессе обработки	
y	*int*	переменная для промежуточного хранения данных в процессе обработки	
kol	*int*	количество потоков передачи	*Вводится пользователем*
v	*int*	открытый ключ	*Вводится пользователем*
r	*int*	«Пароль»: то, знание чего мы пытаемся проверить	*Вводится пользователем*
b_list	*list*	список для хранения сигналов b = 0 v 1, передаваемых на шаге 2	
message	*int*	переменная для передачи «пароля» в зашифр. виде	
x	*bool*	булева переменная, показывающая результат аутентификации: **успех (True) / неудача (False)**	*Итоговый ответ*

6.3 Работа программы

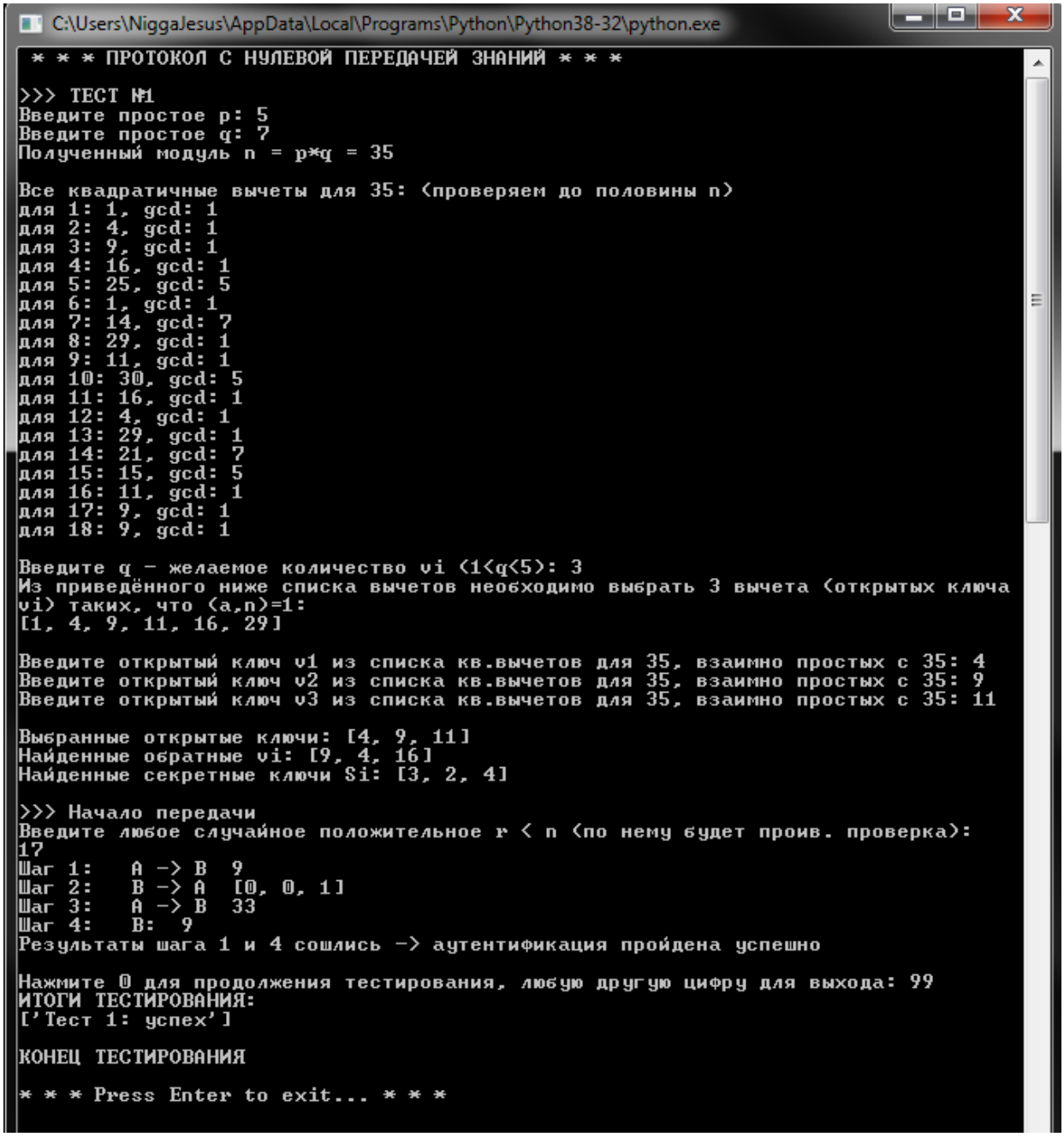

Рисунок 7

6.4 Код

```
import cryptofunctions as crf
import math
import random


# Протокол с нулевой передачей данных оформлен в одну длинную функцию без разбиения

def zero_knowledge():
    openkey = []  # список открытых ключей
    openkey_reverse = []  # обратные к откытым ключам
    secretkey = []  # список секретных ключей s
    mutprime_n = []  # список кв.вычетов n, взаимно простых с n
    vychety = []  # список кв.вычетов n
    ping = [0, 1]  # Список возможных сигналов отправляемыз на втором шаге

    p = crf.inputprimecheck('Введите простое p: ', 'НЕ ЯВЛЯЕТСЯ простым, введите
ПРОСТОЕ p: ')
    q = crf.inputprimecheck('Введите простое q: ', 'НЕ ЯВЛЯЕТСЯ простым, введите
ПРОСТОЕ q: ')
    n = p * q
    print(f'Полученный модуль n = p*q = {n}')
    print(f'\nВсе квадратичные вычеты для {n}: (проверяем до половины n)')

    # заполняем список кв.вычетов n, взаимно простых с n
    for i in range(1, n // 2 + 2):
        d = pow(i, 2, n)
        vychety.append(d)
        print(f'для {i}: {d}, gcd: {math.gcd(i, n)}')
        if math.gcd(d, n) == 1:
            mutprime_n.append(d)
    # удаляем из списка взаимно простых с n вычетов повторяющиеся:
    mutprime_n = list(set(mutprime_n))

    # выбираем количество vi
    try:
        kol = int(input('\nВведите q - желаемое количество vi (1<q<5): '))
    except:
        raise ValueError('НЕУДАЧА ОДНАКО ПОЧЕМУ-ТО.')

    # выбираем открытые ключи vi взаимно простые с n
    print(
        f'Из приведённого ниже списка вычетов необходимо выбрать {kol} вычета
(открытых ключа vi) таких, что (a,n)=1:\n{mutprime_n}\n')
    for i in range(kol):
        v = crf.inputmutprimecheck2(n,
                                    f'Введите открытый ключ v{i + 1} из списка
кв.вычетов для {n}, взаимно простых с {n}: ',
                                    f'Не взамно просты. Введите v{i} взаимно простое с
{n}: ')
        openkey.append(v)
    print(f'\nВыбранные открытые ключи: {openkey}')

    # ищем обратные ключи и заполняем список
    for i in range(kol):
        openkey_reverse.append(crf.fermaeuler(openkey[i], n))
    print(f'Найденные обратные vi: {openkey_reverse}')

    # ищем секретные ключи S
    for i in range(kol):
        for j in range(len(vychety)):
            if openkey_reverse[i] == vychety[j]:
                secretkey.append(j + 1)
                break
        else:
            print('какаято ошибка. НЕ получилось определить S')
```

```
    # выводим на экран введённые и полученные данные
    print(f'Найденные секретные ключи Si: {secretkey}\n')
    print(f'>>> Начало передачи')
    r = int(input('Введите любое случайное положительное r < n (по нему будет проив.
проверка):\n'))

    b_list = []  # список для хранения сигналов b = 0 v 1, передаваемых на шаге 2

    # шаг 1
    message = pow(r, 2, n)
    print(f'Шаг 1:   A -> B  {message}')

    # шаг 2
    for i in range(kol):
        b = random.choice(ping)
        b_list.append(b)
    print(f'Шаг 2:   B -> A  {b_list}')

    # шаг 3
    y = int(r)
    for i in range(kol):
        y *= secretkey[i] ** b_list[i]
    y = pow(y, 1, n)
    print(f'Шаг 3:   A -> B  {y}')

    # шаг 4
    d = y ** 2
    for i in range(kol):
        d *= openkey[i] ** b_list[i]
    d = pow(d, 1, n)
    print(f'Шаг 4:   B:  {d}')

    if crf.text_equalcheck2(message, d):
        print('Результаты шага 1 и 4 сошлись -> аутентификация пройдена успешно')
        return True
    else:
        print('Результаты шага 1 и 4 НЕ сошлись -> аутентификация НЕ пройдена!')
        return False


a = 0
i = 1
check = []
print(' * * * ПРОТОКОЛ С НУЛЕВОЙ ПЕРЕДАЧЕЙ ЗНАНИЙ * * *')

while a == 0:  # Цикл для зацикливания тестирования
    print(f'\n>>> ТЕСТ №{i}')

    x = zero_knowledge()  # получаем функцию, которую можно вызвать в 1 строчку без
передачи аргументов на вход

    if x:
        check.append(f'Тест {i}: успех')
    else:
        check.append(f'Тест {i}: неудача')
    a = int(input('\nНажмите 0 для продолжения тестирования, любую другую цифру для
выхода: '))
    i += 1

print(f'ИТОГИ ТЕСТИРОВАНИЯ:\n{check}')
input('\nКОНЕЦ ТЕСТИРОВАНИЯ\n\n* * * Press Enter to exit... * * *')
```

7. Хэш-Функция (hashfunction)

7.1 Описание

Хэш-Функция использует односторонние функции для необратимого преобразования текста в произвольную битовую строку фиксированной длины по определённому алгоритму. Результат преобразования называется **Хэшем (hash)**.

Хэш-функция, разработанная в данной лабораторной работе, основана на действии функции Рабина, которая является кандидатом в односторонние функции. Функция Рабина принимает два целых числа: x и N, где N — произведение двух простых p и q. На выходе — остаток от деления x^2 на N. Нахождение обратной функции требует вычисления квадратного корня по модулю N, то есть нахождения x, если известно y и то, что $x^2 \bmod N = y$. Можно показать, что последняя задача вычислительно так же сложна, как и разложение N на множители.

Криптосистема Рабина основывается на предположении, что функция Рабина является односторонней.

Хэш-функция должна отвечать следующим основным требованиям:

- Необратимость шифрования;
- Однозначность шифрования;
- Низкая вероятность коллизий[2];
- Фиксированная длина;
- Если исходные данные отличаются незначительно, то итоговые хэши будут кардинально разными.

7.2 Таблица переменных

Имя переменной	Тип	Назначение	Примечание
p	*int*	Простое число p	*Вводится пользователем*
q	*int*	Простое число p	*Вводится пользователем*
n		n = p * q, переменная, использующаяся в функции Рабина `(rabin())`	
x	*int*	Переменная для хранения символа при обработке функцией Рабина `(rabin())`	
y	*int*	Переменная для хранения результата обработки символа функцией Рабина `(rabin())`	
text	*list*	Текст для хэширования	*Вводится пользователем*
hash	*list*	Переменная для хранения хэшированного текста	*Итоговый ответ*
a	*int*	Переменная для зацикливания тестирования и выхода из программы	*Изначально a == 0; затем водится пользователем*
i	*int*	Счётчик цикла	

[2] Коллизия – совпадение итоговых хэшей при различных входных данных

7.3 Работа программы

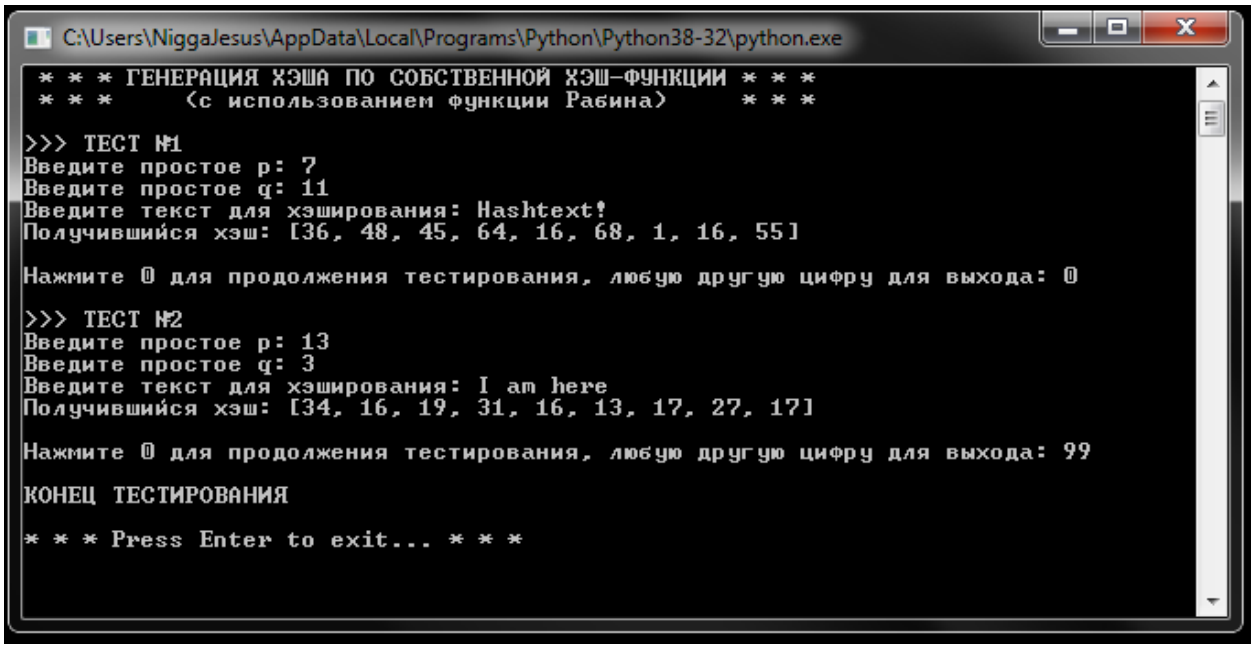

Рисунок 8

7.4 Код

```
import cryptofunctions as crf

#возведение в квадрат и извлечение корня по модулю (криптосистема Рабина)


def rabin(x,n):
    """функция Рабина - кандидат в односторонние функции
    На вход получаем число x и n=p*q, где p, q - простые"""
    y=(x**x)%n
    return y

def megahash(text,n):
    hash=[]
    """Посимвольная обработка исходного текста"""
    for i in range(len(text)):
        hash.append(rabin(text[i],n))
    return hash

def hashfunction():
    """Общая функция для получения хэша из текста
    Включает ввод, посимвольную обработку получение и вывод хэша
    Использует функцию cryptofunctions.text_crypt
    для кодирования текста из строки символов в строку чисел"""

    p=crf.inputprimecheck('Введите простое p: ','НЕ ЯВЛЯЕТСЯ простым, введите ПРОСТОЕ p: ')
    q=crf.inputprimecheck('Введите простое q: ','НЕ ЯВЛЯЕТСЯ простым, введите ПРОСТОЕ q: ')
    n=p*q

    text = crf.text_crypt(input('Введите текст для хэширования: '))
    hash = megahash(text, n)

    print(f'Получившийся хэш: {hash}')



a = 0
i = 1
print(' * * * ГЕНЕРАЦИЯ ХЭША ПО СОБСТВЕННОЙ ХЭШ-ФУНКЦИИ * * *\n * * * (с использованием функции Рабина) * * *')

while a == 0:  # Цикл для зацикливания тестирования
    print(f'\n>>> ТЕСТ №{i}')
    hashfunction()  # получаем функцию, которую можно вызвать в 1 строчку без передачи аргументов на вход
    a = int(input('\nНажмите 0 для продолжения тестирования, любую другую цифру для выхода: '))
    i += 1
input('\nКОНЕЦ ТЕСТИРОВАНИЯ\n\n* * * Press Enter to exit... * * *')
```

Заключение

В результате данного курсового проекта был разработан программный комплекс лабораторных работ **«Crypto_2350»** по курсу «Криптографические Методы Защиты Информации» пятого семестра программы обучения по специальности **10.03.01** – «Информационная безопасность» на языке программирования **Python 3.8**.

В ходе работы были улучшены навыки разработки концепции и структуры скриптов, реализации кода на Python, тестирования и дебаггинга, обращения с IDE **Pycharm 2022.1**, а также навыки поиска информации в открытых источниках.

Для того, чтобы проект был доступен, итоговый проект был загружен в публичный репозиторий на Github[3]: таким образом, в ходе дальнейшего расширения и обновления программы, а также в случае обнаружения и исправления каких-либо ошибок, пользователям будет доступна новейшая версия проекта. Также пользователи смогут дать обратную связь в случае обнаружения ошибок в работе программы.

Файлы проекта в формате **.py** предоставлены преподавателю как Приложение к данной курсовой работе.

Цель курсового проекта достигнута.

[3] https://github.com/HeadcrabFM/Crypto_2350

Список использованной литературы

1. Oded Goldreich. Volume 1, Basic Tools // Foundations of Cryptography. — Cambridge University Press, 2001. — ISBN 0-521-79172-3.
2. А. В. Черёмушкин // Лекции по арифметическим алгоритмам в криптографии
3. Глава 2.3. Односторонние функции // Введение в криптографию // Под ред. В. В. Ященко.
4. Глава 2.5.2 Односторонняя функция // Приложение теории детерминированного хаоса в криптографии // Птицын Н. — 2002. (недоступная ссылка
5. Саломаа А. // Криптография с открытым ключом. с. 74-75.
6. Вильям Столлингс // Криптография и защита сетей: принципы и практика // М.: Вильямс, 2001. ISBN 5-8459-0185-5.
7. Василенко О. Н. Теоретико-числовые алгоритмы в криптографии — М.: МЦНМО, 2003. — 328 с. — ISBN 978-5-94057-103-2.
8. Минеев М. П., Чубариков В. Н. // Об одном применении китайской теоремы об остатках к шифру Виженера. // М.: ДАН. — 2010. — Том: 430. — №1. — С. 21, 22.
9. Смарт Н. // Криптография. — 2005. — 528 с
10. Сонг Й. Ян. // Криптоанализ RSA. — М., Ижевск: «Регулярная и хаотическая динамика», «Институт компьютерных исследований», 2011. — 312 с. — ISBN 978-5-93972-873-7.
11. Elgamal T. A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms, A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms (англ.) // IEEE Trans. Inf. Theory / F. Kschischang — IEEE, 1985. — Vol. 31, Iss. 4. — P. 469—472. — ISSN 0018-9448; 1557-9654 — doi:10.1109/TIT.1985.1057074; doi:10.1007/3-540-39568-7_2
12. Menezes A. J., Oorschot P. V., Vanstone S. A. // The ElGamal signature scheme // Handbook of Applied Cryptography (англ.) — CRC Press, 1996. — 816 p. — (Discrete Mathematics and Its Applications) — ISBN 978-0-8493-8523-0
13. Брюс Шнайер // Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си. — М.: Триумф, 2002. — ISBN 5-89392-055-4.
14. Никлаус Вирт // Алгоритмы и структуры данных. — М.: «Мир», 1989. — ISBN 5-03-001045-9.
15. Хеш-функция // https://ru.wikipedia.org/wiki/%D0%A5%D0%B5%D1%88-%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F
16. Протоколы идентификации с нулевой передачей знаний // https://protect.htmlweb.ru/p32.htm