МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА № 42

ОТЧЁТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

| старший преподаватель |  |  |  | С.Ю. Гуков |
| --- | --- | --- | --- | --- |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

| ОТЧЁТ О ЛАБОРАТОРНОЙ РАБОТЕ №9 |
| --- |
| Асимметричный алгоритм шифрования RSA |
| по курсу: |
| АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ |
|  |

РАБОТУ ВЫПОЛНИЛ

| СТУДЕНТ гр. № | 4329 |  |  |  | Д.С. Шаповалова |
| --- | --- | --- | --- | --- | --- |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2024

ОГЛАВЛЕНИЕ

[Цель работы 3](#_30j0zll)

[Постановка задачи 3](#_1fob9te)

[Схема алгоритма решения 4](#_3znysh7)

[Полное описание реализованной функции 4](#_gqc1nhfmq022)

[Листинг программы 5](#_2et92p0)

[Результат выполнения программы. 5](#_tyjcwt)

[ВЫВОДЫ 6](#_3dy6vkm)

# Цель работы

Программно реализовать на любом языке программирования ассиметричный алгоритм шифрования RSA.

# Постановка задачи

Задание: Написать программу, реализующую ассиметричный алгоритм шифрования RSA. Продемонстрировать на примерах его работу по зашифровыванию и расшифровыванию текста. Разрешается реализовать как версию с пользовательским интерфейсом, так и консольную версию с дружелюбным командно-текстовым интерфейсом. Текст задания приведён в таблице 1.

Таблица 1. Индивидуальное задание

| Текст задания |
| --- |
| Последовательность шагов алгоритма RSA:   * выбрать два больших простых числа р и q; * вычислить: n=р\*q и фи(n) = (р-1)\*(q-1); * выбрать случайное число e, взаимно простое с фи(n) ; * определить такое число d, для которого является истинным выражение: (e\*d)mod(фи(n))=1; * числа е и n - это открытый ключ, а числа d и n - это закрытый ключ; На практике это означает следующее: открытым ключом зашифровывают сообщение, а закрытым - расшифровывают. Пара чисел закрытого ключа держится в секрете. * разбить шифруемый текст на блоки, каждый из которых может быть представлен в виде числа M (i); Обычно блок берут равным одному символу и представляют этот символ в виду числа - его номера в алфавите или кода в таблице символов (например ASCII или Unicode). * шифрование алгоритмом RSA производится по формуле: C(i) = (M(i)^e)mod n; * расшифровка сообщения производится по формуле: M(i)=(C(i)^d)mod n.   mod – операция взятия остатка от деления.  взаимно простыми называются такие числа, которые не имеют между собой ни  одного общего делителя, кроме единицы. |

# 

# Схема алгоритма решения

1. Установка ключей (генерация/ввод)
2. Обработка сообщения (шифрование/дешифрование)

# Полное описание реализованной функции

**Функция ввода команд (дружелюбный консольный интерфейс):**

1. Сгенерировать ключи
   1. Вводится длина простых чисел в битах
   2. Генерируется 2 простых числа - p и q
   3. На основе них генерируется пара ключей (ключ - пара чисел (связанных математически)) - открытый и закрытый (см. функцию генерации ключей (generate\_keypair))
   4. Ключи выводятся в консоль
2. Ввести открытый ключ
   1. Вводятся числа e и n
3. Ввести закрытый ключ
   1. Вводятся числа d и n
4. Зашифровать сообщение
   1. Проверяется наличие открытого ключа
   2. Передача сообщения в функцию шифрования (encrypt)
   3. Вывод зашифрованного
5. Расшифровать сообщение
   1. Проверяется наличие закрытого ключа
   2. Передача сообщения в функцию шифрования (decrypt)
   3. Вывод расшифрованного

**Функция генерации ключей (**generate\_keypair**):**

1. Генерируем 2 простых числа заданной длины(битности) - отдельная функция - перебираем рандомные (random.getrandbits) нечётные числа, пока не найдём простое (сверяемся с таблицей простых чисел (sympy : isprime))
2. Перемножаем p и q = n = модуль для RSA шифрования. Вычисляем функцию Эйлера = фи(n) = (p - 1) \* (q - 1)
3. Выбираем такое е, которое взаимно простое для фи(n) - их НОД(gcd) = 1, обычно е = 65537 (для открытого ключа)
4. Определяем d (для закрытого ключа) - обратный элемент для e по модулю фи(n). Вычисляется по формуле - остаток от деления на n от результат возведения e в -1 степень.
5. Возвращаем пары пар чисел - (e, n), (d, n)

**Функция шифрования (**encrypt**):**

1. Преобразует каждый символ строки в его код ASCII (ф-я ord) => получаем массив последовательных чисел
2. Возвращаем остаток от деления на n результата возведения символа-числа в степень e = (num^e) mod n
3. Возвращаем теперь непонятные числа

**Функция дешифрования (**decrypt**):**

1. Перебираем числа - от каждого числа берём остаток от деления на n результата возведения числа в степень d (обратный элемент для e по модулю фи(n))
2. Получаем снова символы-числа в ASCII => с помощью ф-и chr преобразуем символ-число в символ-букву => объединяем в строку.
3. Возвращаем понятные буквы

# Листинг программы

import random

from math import gcd

from sympy import isprime

# Генерация простого числа заданной битности

def generate\_large\_prime(length):

while True:

candidate = random.getrandbits(length) | 1 # Генерируем нечетное число

if isprime(candidate):

return candidate

def generate\_keypair(p, q):

# p и q - простые числа (генерация выше)

n = p \* q # модуль rsa

# Вычисление функции Эйлера

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

# Проверка на взаимную простоту

def is\_coprime(e, phi):

return gcd(e, phi) == 1 # НОД

e = 65537 # Обычно выбирается простое число 65537

if not is\_coprime(e, phi):

raise ValueError("e не является взаимно простым с phi(n)")

# Вычисление закрытой экспоненты d

d = pow(e, -1, phi)

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

'''

# Вычисление обратного элемента по модулю

def modinv(a, m):

m0, x0, x1 = m, 0, 1

if m == 1:

return 0

while a > 1:

q = a // m

m, a = a % m, m

x0, x1 = x1 - q \* x0, x0

if x1 < 0:

x1 += m0

return x1

'''

# Шифрование сообщения на основе открытого ключа

def encrypt(pk, plaintext):

numbers = [ord(char) for char in plaintext]

ciphertext = [pow(num, pk[0], pk[1]) for num in numbers]

return ciphertext

# Расшифровка сообщения на основе закрытого ключа

def decrypt(sk, ciphertext):

plaintext\_numbers = [pow(c, sk[0], sk[1]) for c in ciphertext]

message = ''.join(chr(num) for num in plaintext\_numbers)

return message

def main():

print("Добро пожаловать в программу шифрования RSA! 😊")

public\_key, private\_key = None, None

while True:

print("\nВыберите опцию:")

print("1. Сгенерировать ключи")

print("2. Ввести открытый ключ вручную")

print("3. Ввести закрытый ключ вручную")

print("4. Зашифровать сообщение")

print("5. Расшифровать сообщение")

print("6. Выход")

choice = input("Ваш выбор: ")

if choice == '1':

length = int(input("Введите длину простых чисел в битах: "))

p = generate\_large\_prime(length)

q = generate\_large\_prime(length)

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

print("Открытый ключ:", public\_key)

print("Закрытый ключ:", private\_key)

elif choice == '2':

e = int(input("Введите значение e: "))

n = int(input("Введите значение n: "))

public\_key = (e, n)

print("Открытый ключ установлен:", public\_key)

elif choice == '3':

d = int(input("Введите значение d: "))

n = int(input("Введите значение n: "))

private\_key = (d, n)

print("Закрытый ключ установлен:", private\_key)

elif choice == '4':

if public\_key is None:

print("Сначала сгенерируйте или введите открытый ключ! 🔑")

continue

message = input("Введите сообщение для шифрования: ")

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

print("Зашифрованное сообщение:", encrypted\_message)

elif choice == '5':

if private\_key is None:

print("Сначала сгенерируйте или введите закрытый ключ! 🔑")

continue

ciphertext\_input = input("Введите зашифрованное сообщение (список чисел, разделенных запятыми): ")

ciphertext = list(map(int, ciphertext\_input.split(',')))

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

print("Расшифрованное сообщение:", decrypted\_message)

elif choice == '6':

print("Выход из программы. Всего хорошего! 👋")

break

else:

print("Неверный выбор, попробуйте снова.")

if \_\_name\_\_ == "\_\_main\_\_":

main()

# 

# Результат выполнения программы.

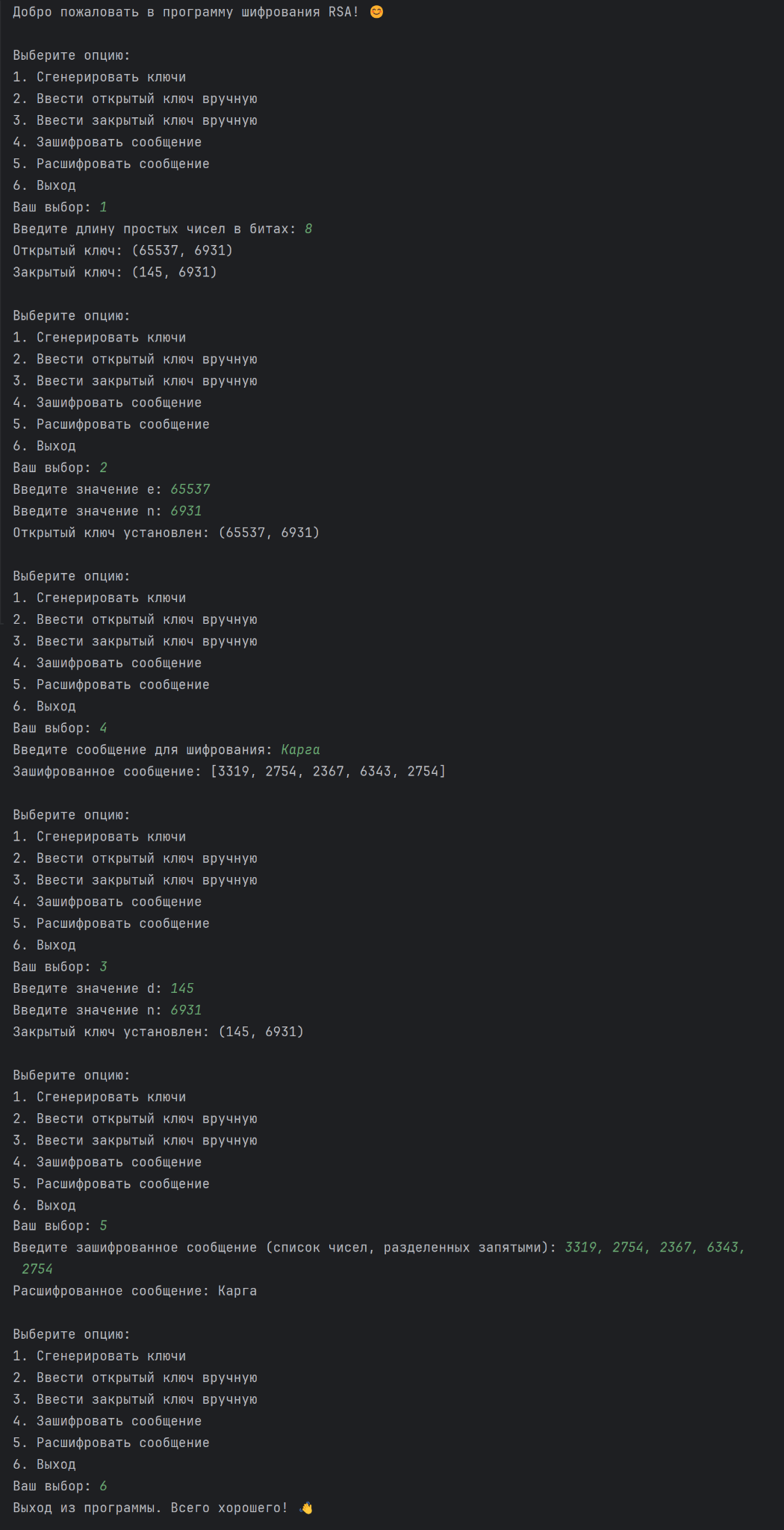


Рисунок 2.1 - Работа программы кратко (8 бит)

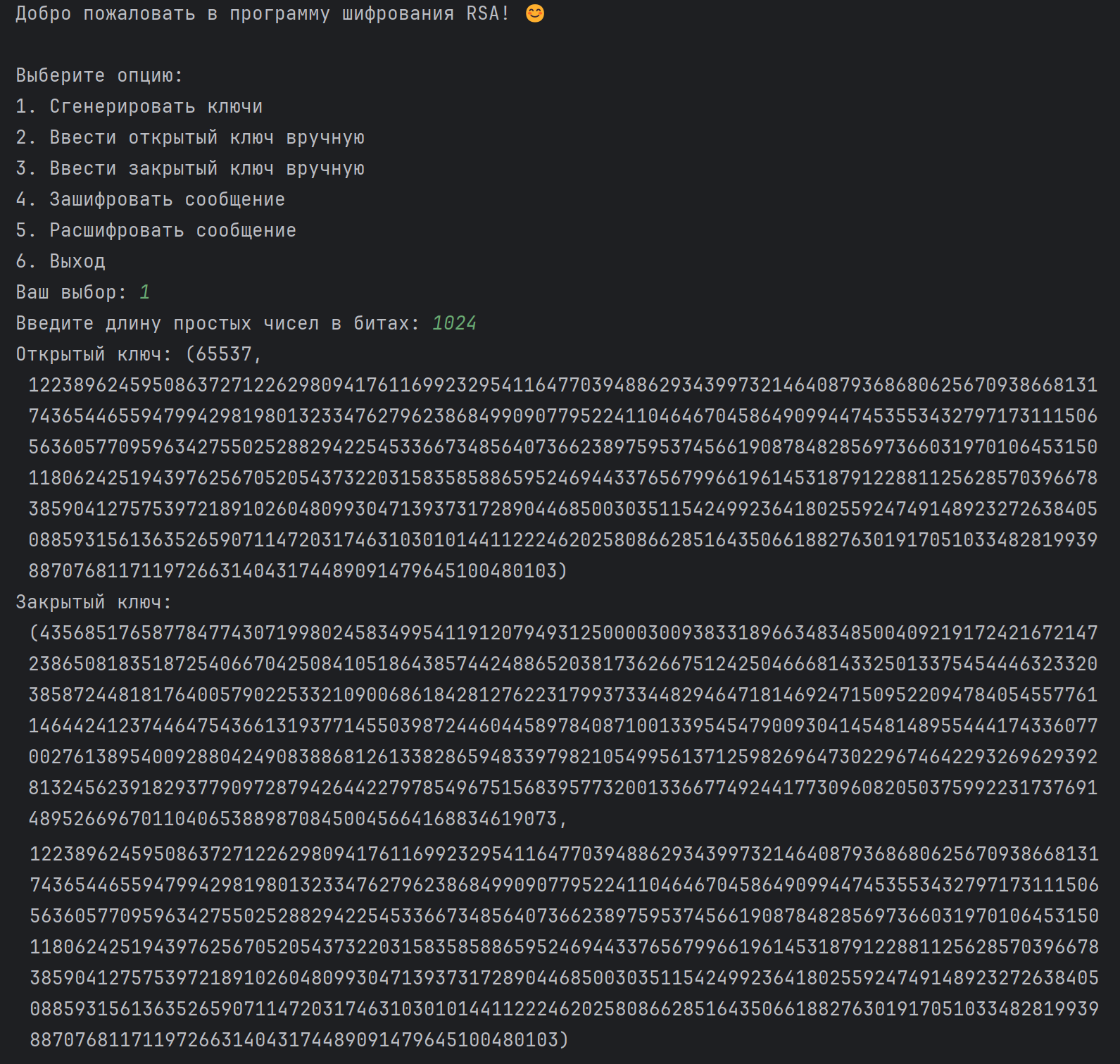


Рисунок 2.1 - Генерация пары ключей (1024 бит)

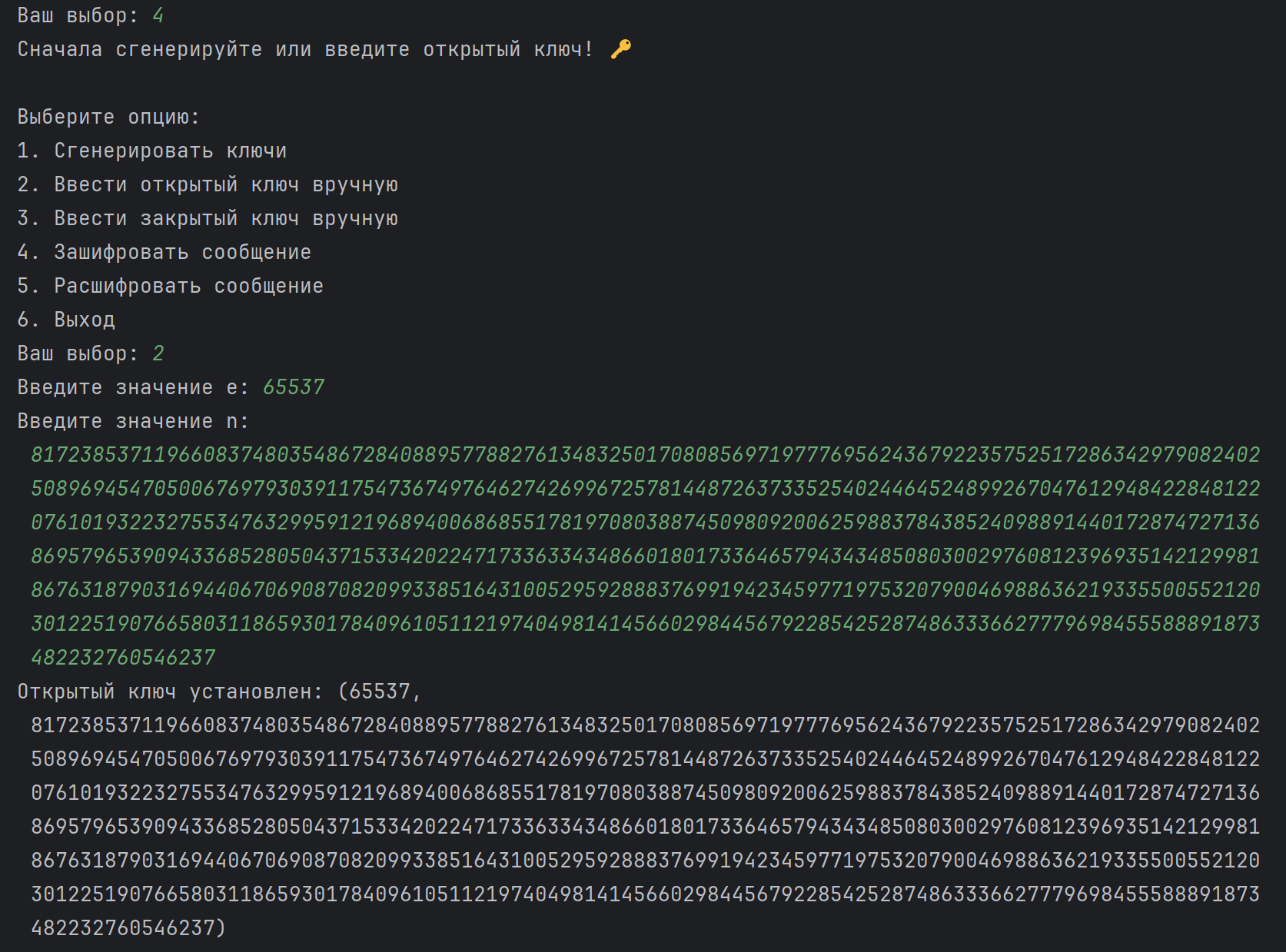


Рисунок 2.2 - Установка открытого ключа (1024 бит)

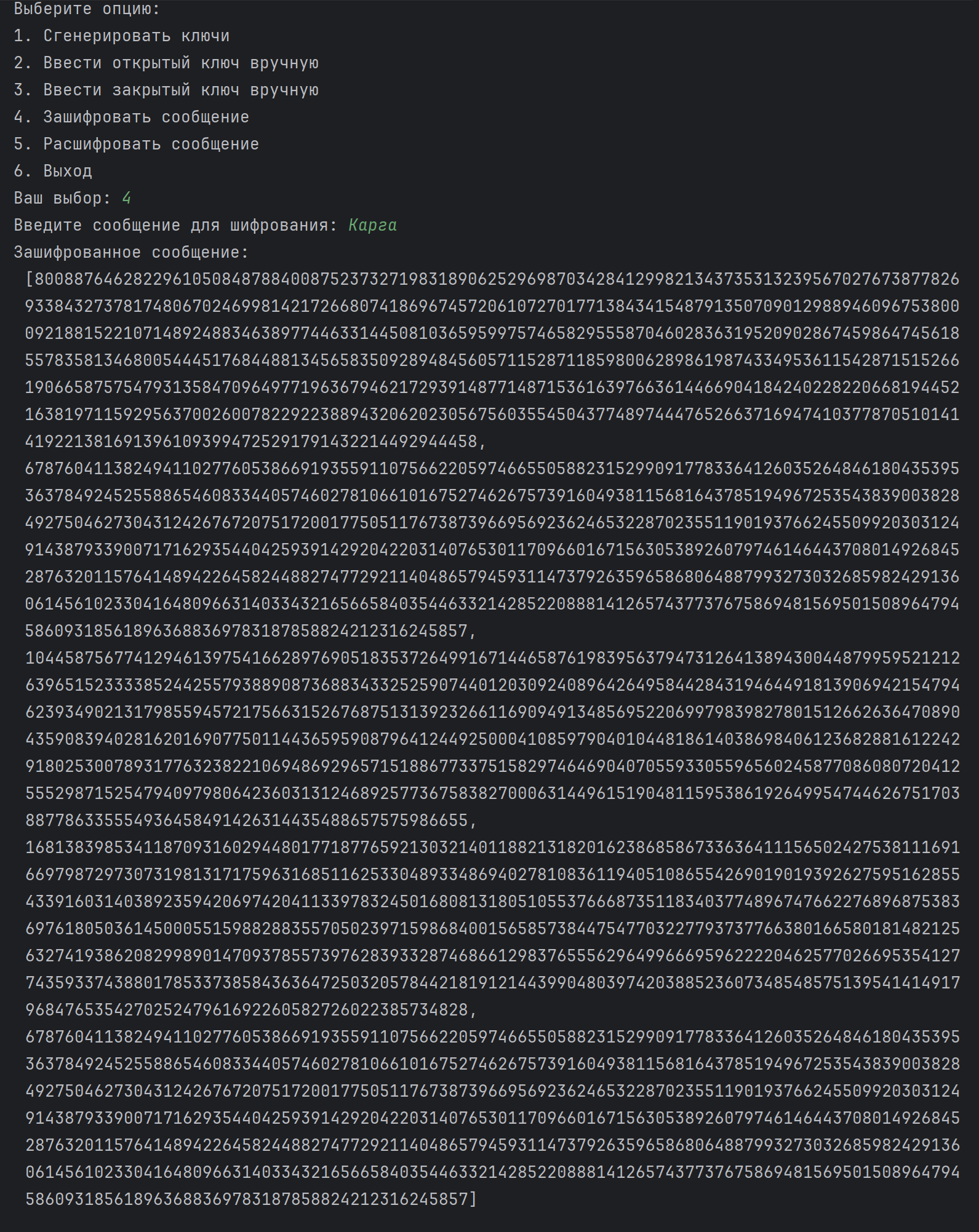


Рисунок 2.3 - Шифрование сообщения (1024 бит)

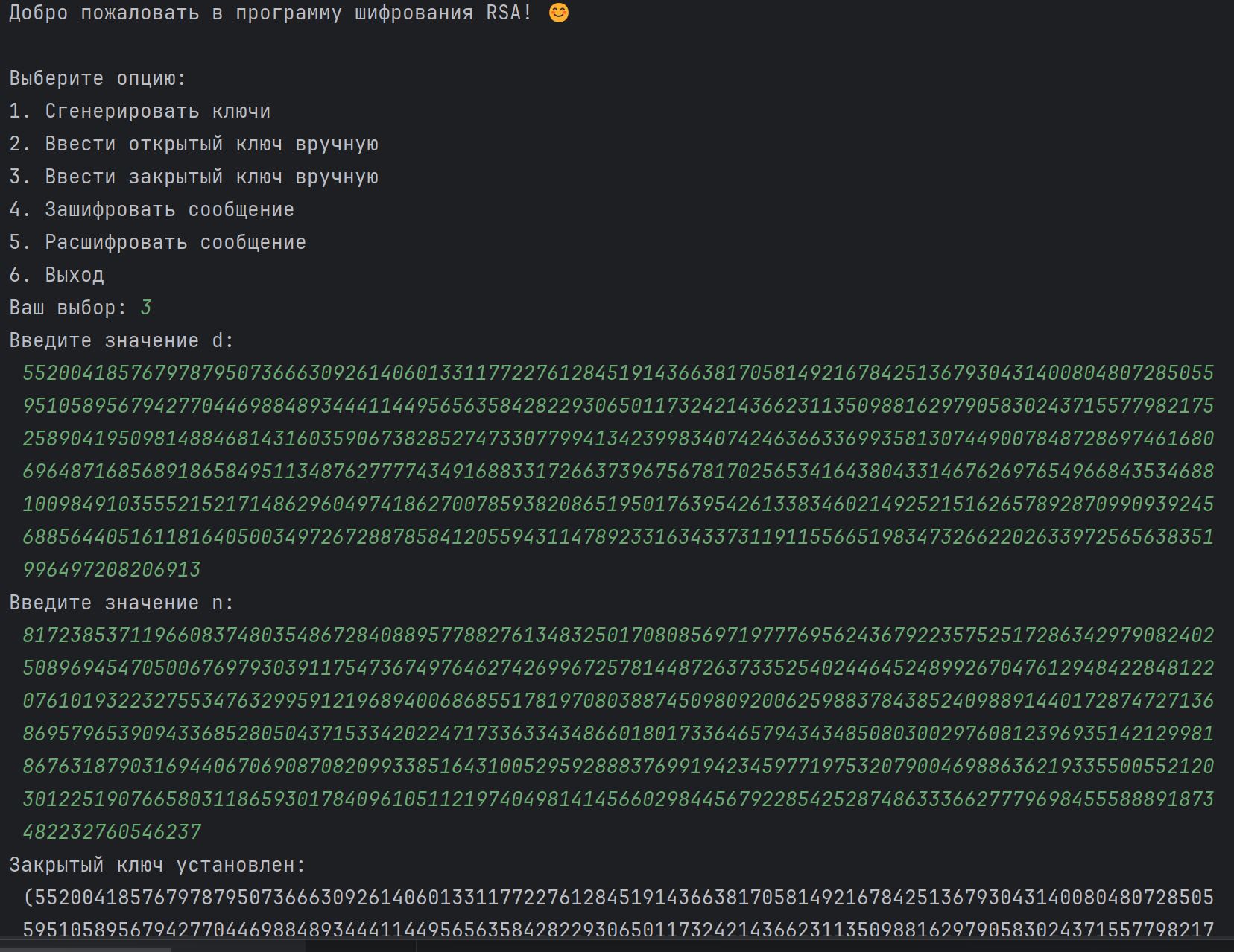


Рисунок 2.4 - Установка закрытого ключа (1024 бит)

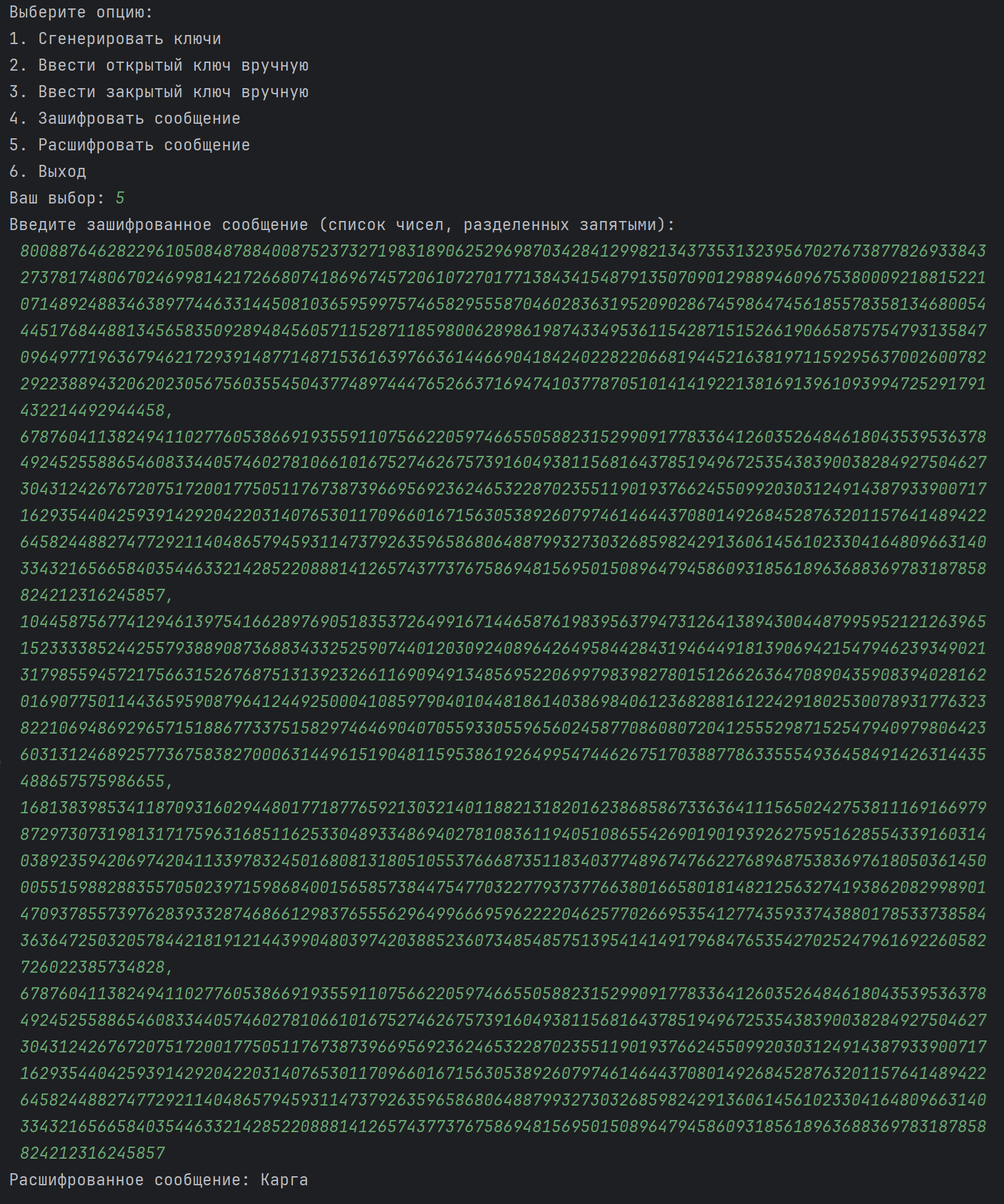


Рисунок 2.5 - Расшифровка сообщения (1024 бит)

# 

# Выводы

В ходе выполнения лабораторной работы мной были освоены и изучены: понятие шифрование; ассимметричный алгоритм шифрования RSA; дружелюбный консольный интерфейс. Написанная программа была протестирована, полученный результат соответствует значению в примере.