**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ**

**Федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Московский Авиационный Институт»**

**(Национальный Исследовательский Университет)**

**Институт: №8 «Информационные технологии   
и прикладная математика»   
Кафедра: 806 «Вычислительная математика   
и программирование»**

Лабораторная работа № 2  
по курсу «Криптография»

Группа: М8О-301Б-22

Студент(ка): А. В. Шляхтуров

Преподаватель: А. В. Борисов

Оценка:

Дата: 28.03.2025

Москва, 2024

**ОГЛАВЛЕНИЕ**

[**1** **Тема** 3](#_Toc193539473)

[**2** **Задание** 3](#_Toc193539474)

[**3** **Теория** 3](#_Toc193539475)

[**4** **Ход лабораторной работы** 5](#_Toc193539480)

[**5** **Выводы** 7](#_Toc193539485)

[**6** **Список используемой литературы** 8](#_Toc193539489)

# **Тема**

Работа с хеш-функцией ГОСТ Р 34.11-2012 «Стрибог». Разложение двух чисел на нетривиальные сомножители.

# **Задание**

Строку, в которой записано своё ФИО подать на вход в качестве аргумента хеш-функции ГОСТ Р 34.11-2012 (Стрибог). Младшие 8 бит выхода интерпретировать как число, которое в дальнейшем будет номером варианта от 0 до 255. В отчёт включить снимок экрана с выбором номера варианта, а также описать шаги решения задачи.

Задача: разложить каждое из чисел `a` и `b` на нетривиальные сомножители. Ниже представлены 256 вариантов.

# **Теория**

Хеш-функция — это алгоритм, который преобразует произвольные данные фиксированной длины в битовую строку фиксированной длины, называемую хеш-значением. Хеш-функции широко используются в криптографии для обеспечения целостности данных, аутентификации и цифровых подписей.

ГОСТ Р 34.11-2012 (Стрибог) — это российский стандарт хеш-функции, принятый в 2012 году.

Разложение числа на множители (факторизация) — это процесс представления натурального числа в виде произведения простых чисел. Факторизация является одной из важнейших задач в теории чисел и криптографии, так как многие криптографические алгоритмы (например, RSA) основаны на сложности факторизации больших чисел.

Для больших чисел факторизация является вычислительно сложной задачей, что делает её привлекательной для использования в криптографических системах.

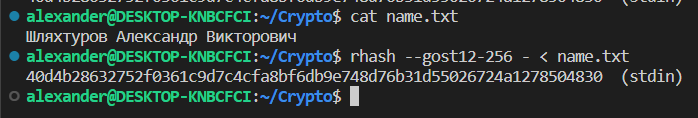
Ро-алгоритм Полларда (Pollard's Rho algorithm) — это вероятностный алгоритм для факторизации целых чисел. Он был предложен Джоном Поллардом в 1975 году. Алгоритм эффективен для нахождения небольших простых делителей больших чисел.

Алгоритм основан на поиске циклов в последовательности чисел, генерируемой с помощью полиномиальной функции. Если в последовательности обнаруживается цикл, это позволяет найти нетривиальный делитель числа.

Хеш-функция ГОСТ Р 34.11-2012 (Стрибог) является важным инструментом в криптографии, обеспечивая высокий уровень безопасности и целостности данных. Разложение числа на множители и алгоритмы факторизации, такие как ро-алгоритм Полларда, играют ключевую роль в криптоанализе и разработке криптографических систем. Понимание этих алгоритмов позволяет глубже изучить основы криптографии и методы защиты информации.

# **Ход лабораторной работы**

## Определение варианта:



Младшие 8 бит выхода – две последние цифры – **30 вариант**

a[30]=112085497440388511364493266926808359258182229984346544114566622069335233815379 b[30]=3231700607131100730071487668866995196044410266971548403213034542752465513886789089319720141152291346368871796092189801949411955915049092109508815460367633011615775193898822438924144460625440530195221757278453207898458480890541247028005668822752981570891458528568294724106538237972970534050989635694474401020183378096732153297648894573843297760503212930739078479213765655706380346401992547055284784120132722139951533541085294131700123044439726731201116192376533867726372660670573533503631694955755234683702272304947722710987286242495369195451210077395191785947268063712817772891541022491483 2056397349155953757360361457

## Поиск множителей

### Первое число

Найдем множители с помощью библиотеки sympy и функции factorint

from sympy import factorint, isprime

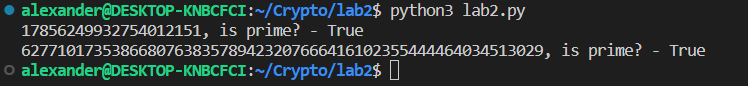
a = 112085497440388511364493266926808359258182229984346544114566622069335233815379

factors = factorint(a)

keys = list(factors.keys())

for key in keys:

    print(f"{key}, is prime? - {isprime(key)}")



Таким образом разложение выглядит:

**112085497440388511364493266926808359258182229984346544114566622069335233815379**

== **17856249932754012151**\* **6277101735386680763835789423207666416102355444464034513029**

### Второе число

Второе число слишком велико для всех известных алгоритмов факторизации. От старших коллег была получена информация, что числа b из разных имеют общие делители, поэтому было достаточно посчитать НОД числа из моего варианта со всеми остальными числами.

Исходное число: b[30]=3231700607131100730071487668866995196044410266971548403213034542752465513886789089319720141152291346368871796092189801949411955915049092109508815460367633011615775193898822438924144460625440530195221757278453207898458480890541247028005668822752981570891458528568294724106538237972970534050989635694474401020183378096732153297648894573843297760503212930739078479213765655706380346401992547055284784120132722139951533541085294131700123044439726731201116192376533867726372660670573533503631694955755234683702272304947722710987286242495369195451210077395191785947268063712817772891541022491483 2056397349155953757360361457

Для нахождения первого множителя был написан код на питоне, который бы перебирал числа b из других вариантов и искал бы наибольший общий делитель с моим числом. Условие остановки – НОД должен быть не равен единице и вариант, для которого нашелся такой НОД, не равен моему варианту, то есть тридцати. Пришлось перевести PDF документ в TXT и распарсить с помощью регулярных выражений.

Второй множитель был найден делением числа b на первое число. Дополнительно проверим, что оба эти числа простые, чтобы убедиться в правильности факторизации.

Вот код программы на Python для поиска разложения второго числа.

my\_num = ('3231700607131100730071487668866995196044410266971548403213034542752465513886789'

    '0893197201411522913463688717960921898019494119559150490921095088154603676330116157751'

    '9389882243892414446062544053019522175727845320789845848089054124702800566882275298157'

    '0891458528568294724106538237972970534050989635694474401020183378096732153297648894573'

    '8432977605032129307390784792137656557063803464019925470552847841201327221399515335410'

    '8529413170012304443972673120111619237653386772637266067057353350363169495575523468370'

    '2272304947722710987286242495369195451210077395191785947268063712817772891541022491483'

    '2056397349155953757360361457')

def extract\_b\_numbers(file\_path):

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

        content = file.read()

    pattern = r'b\[(\d+)\]\s\*=\s\*([\d\s]+)(?=\s\*b\[\d+\]|$)'

    matches = re.findall(pattern, content, re.MULTILINE)

    matches.sort(key=lambda x: int(x[0]))

    b\_numbers = [match[1].replace('\n', '').replace(' ', '').replace('\r', '') for match in matches]

    return b\_numbers

file\_path = 'Лабораторная-2.txt'

b\_numbers = extract\_b\_numbers(file\_path)

for i, b in enumerate(b\_numbers):

    gcd\_val = gcd(int(my\_num), int(re.sub(r"[^\d]", "", b)))

    if gcd\_val != 1 and i != 30:

        print('ВАРИАНТ  ПЕРВОЕ\_ЧИСЛО   is\_prime')

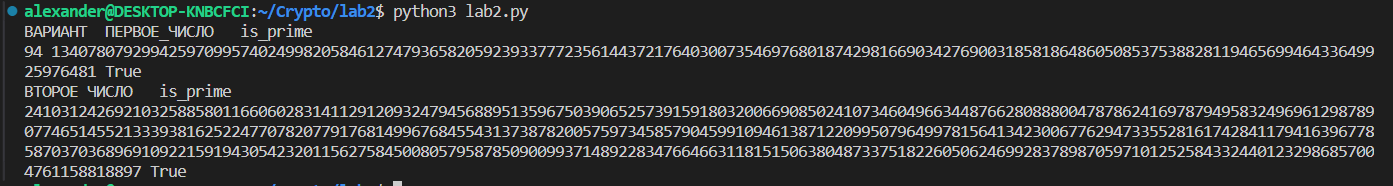
        print(i, gcd\_val, isprime(gcd\_val))

        print("ВТОРОЕ ЧИСЛО   is\_prime")

        print(int(my\_num)/gcd\_val, isprime(int(my\_num)/gcd\_val))

        break

Вывод из терминала



В выводе я прописал, с каким вариантом был найден НОД, не равный единице. В моем случае это 94 вариант. Так же выведены оба числа с пометкой, что они простые – True для обоих чисел.

Первый множитель:

13407807929942597099574024998205846127479365820592393377723561443721764030073546976801874298166903427690031858186486050853753882811946569946433649925976481

Второй множитель: 2410312426921032588580116606028314112912093247945688951359675039065257391591803200669085024107346049663448766280888004787862416978794958324969612987890774651455213339381625224770782077917681499676845543137387820057597345857904599109461387122099507964997815641342300677629473355281617428411794163967785870370368969109221591943054232011562758450080579587850900993714892283476646631181515063804873375182260506246992837898705971012525843324401232986857004761158818897

# **Выводы**

В ходе выполнения лабораторной работы были изучены ключевые аспекты криптографии, связанные с хеш-функциями и алгоритмами факторизации.

Изучен вероятностный алгоритм факторизации, предложенный Джоном Полландом. Алгоритм эффективен для нахождения небольших простых делителей больших чисел и основан на поиске циклов в последовательностях, генерируемых полиномиальными функциями.

# **Список используемой литературы**

* <https://ru.wikipedia.org/wiki/Факторизация_целых_чисел#Экспоненциальные_алгоритмы>
* <https://ru.wikipedia.org/wiki/Ро-алгоритм_Полларда#Особенности_реализации>