**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-309Б-22

Студент: О. С. Концебалов

Преподаватель: А. В. Борисов

Оценка:

Дата: 23.03.2025

Москва, 2025

**ОГЛАВЛЕНИЕ**

[1 Тема 3](#_Toc158983147)

[2 Задание 3](#_Toc158983148)

[3 Теория 4](#_Toc158983149)

[4 Ход лабораторной работы 5](#_Toc158983150)

[5 Выводы 6](#_Toc158983151)

# **Тема**

Разложение двух огромных (у которых число разрядов крайне велико) чисел на нетривиальные сомножители. Знакомство и работа с хэш-функцией ГОСТ Р 34.11-2012 «Стрибог»

# **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.

# **Теория**

Факторизация числа — это процесс разложения числа на произведение простых множителей. Простыми множителями называются простые числа, которые при умножении друг на друга дают исходное число. Факторизация является одной из ключевых задач в теории чисел и имеет важное значение в криптографии, алгоритмах и компьютерных науках.

Факторизация чисел играет ключевую роль в современной криптографии, особенно в асимметричных криптографических системах. Её сложность лежит в основе безопасности многих алгоритмов, таких как RSA, которые широко используются для защиты данных, цифровых подписей и шифрования.

Для небольших чисел факторизация выполняется быстро, но для больших чисел (например, произведений двух больших простых чисел) не существует известных эффективных алгоритмов. Это делает задачу факторизации сложной для современных компьютеров.

**Применение факторизации в криптографии**

**1. Криптосистема RSA**

* RSA (Rivest–Shamir–Adleman) — одна из самых известных асимметричных криптосистем, основанная на сложности факторизации больших чисел.
* **Принцип работы**:
  + Выбираются два больших простых числа *p* и *q*.
  + Вычисляется их произведение *n*=*p*×*q*, которое используется как модуль в криптосистеме.
  + Открытый ключ состоит из числа *n* и экспоненты *e*, а закрытый ключ — из числа *d*, которое вычисляется с использованием *p* и *q*.
  + Без знания *p* и *q* (т.е. без факторизации *n*) вычислить закрытый ключ крайне сложно.
* **Безопасность RSA**:
  + Основана на предположении, что факторизация больших чисел является вычислительно сложной задачей.
  + Если злоумышленник сможет эффективно факторизовать n*n*, он сможет вычислить закрытый ключ и расшифровать сообщения.

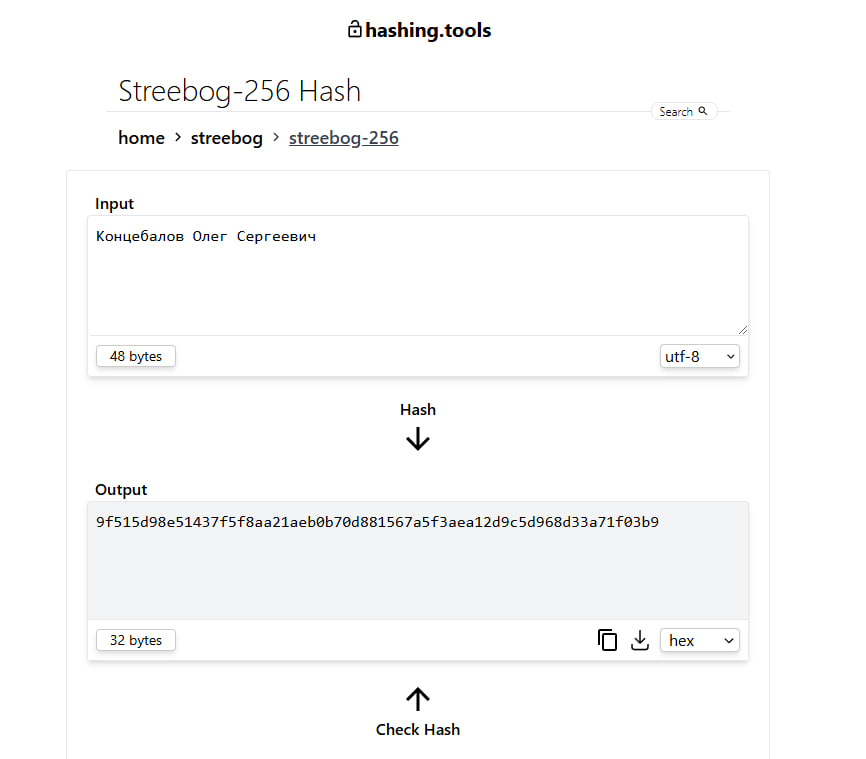
**2. Квантовые компьютеры и угрозы факторизации**

* С появлением квантовых компьютеров классические криптосистемы, такие как RSA, могут стать уязвимыми.
* Алгоритм Шора, разработанный для квантовых компьютеров, позволяет факторизовать большие числа за полиномиальное время, что делает RSA небезопасным в эпоху квантовых вычислений.
* Это стимулирует разработку постквантовой криптографии, устойчивой к атакам с использованием квантовых компьютеров.

# **Ход лабораторной работы**

1) Открыл условие лабораторной работы, прочитал про факторизацию и подумал, что сейчас сделаю за 10 минут. Увидел числа и ужаснулся – числа крайне большие и на стандартные алгоритмы вряд ли дадут успех. Если для первого числа – a еще можно найти библиотеки, то для числа b задача кажется невозможной. Подсчитал количество разрядов числа b – более 500.

2) Вычислил свой вариант с помощью хэш-функции «Стрибог». Для этого воспользовался онлайн версией на сайте <https://hashing.tools/streebog/streebog-256>. Получил следующий вывод для своих ФИО



9f515d98e51437f5f8aa21aeb0b70d881567a5f3aea12d9c5d968d33a71f03b9 -> последний байт b9 -> в десятичном формате это 185. Получил свой вариант – 185.

3) Нашел свои числа. Решил попробовать для них банальный алгоритм перебора до квадрата. Запустил код на своем ноутбуке и ушел на работу. Вернувшись вечером ответа так и не было... Товарищи подсказали, что у старших курсов числа в соседних вариантах делились друг на друга, попробовал разделить свое число b на числа a из вариантов 184 и 186. Попытка успехом не увенчалась. Решил попробовать перебрать вообще все числа a из файла и попробовать разделить на них. Для этого был написан следующий скрипт на Python:

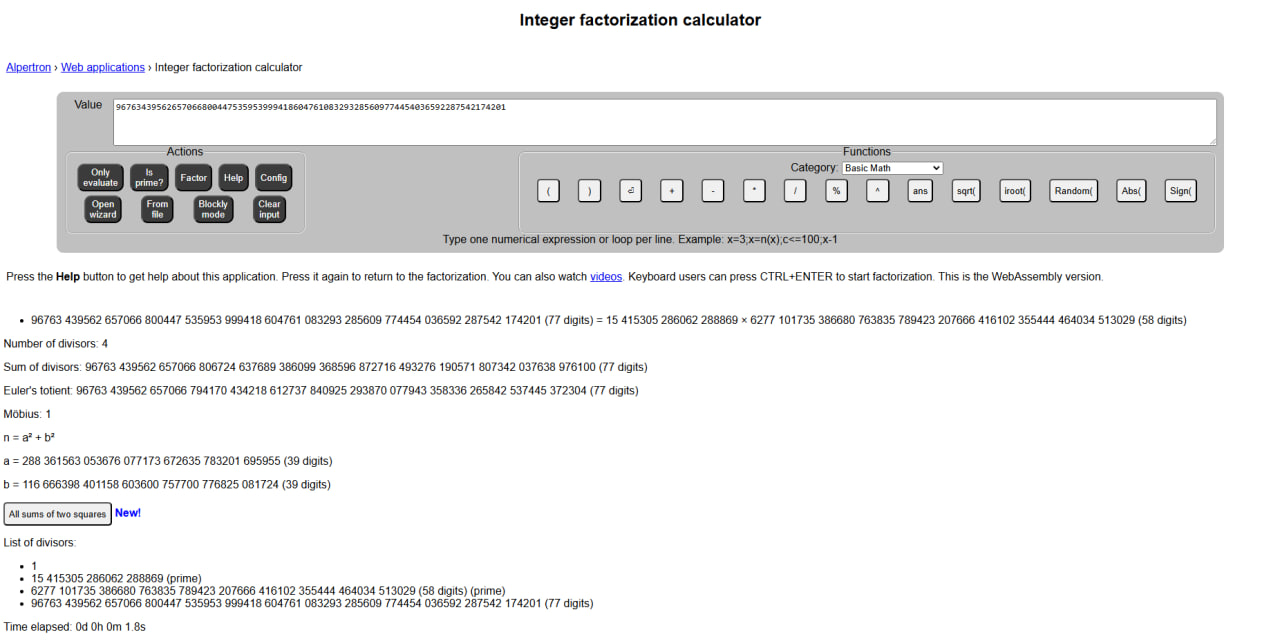
data\_parser.py

import re  
  
  
def parse\_numbers(input\_filename="data.txt"):  
 with open(input\_filename, "r") as f:  
 text = f.read()  
  
 text = text.replace("\f", "\n")  
 pattern = re.compile(  
 r'a\[(\d+)\]\s\*=\s\*([0-9\s]+?)\s\*b\[\1\]\s\*=\s\*([0-9\s]+)',  
 re.DOTALL  
 )  
  
 matches = pattern.findall(text)  
  
 a\_numbers = []  
 b\_numbers = []  
  
 for \_, a\_str, b\_str in matches:  
 a\_num = "".join(a\_str.split())  
 b\_num = "".join(b\_str.split())  
  
 a\_numbers.append(a\_num)  
 b\_numbers.append(b\_num)  
  
 return a\_numbers, b\_numbers  
  
  
def split\_numbers(a\_numbers, b\_numbers, a\_filename="a.txt", b\_filename="b.txt"):  
 with open(a\_filename, "w") as file\_a:  
 for num in a\_numbers:  
 file\_a.write(num + "\n")  
  
 with open(b\_filename, "w") as file\_b:  
 for num in b\_numbers:  
 file\_b.write(num + "\n")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 print("INFO: Starts processing numbers")  
  
 try:  
 a\_list, b\_list = parse\_numbers()  
 split\_numbers(a\_list, b\_list)  
 except Exception as error:  
 print(f"ERROR: {error}")  
 exit(1)  
  
 print("INFO: Processing numbers finished successfully")

Он разделял числа на два файла – в одном только числа a, а в другом только числа b

Попытка разделить свое число b на любое из чисел a тоже не привела к успеху. Грустный пошел спать

4) Нашел сайт (<https://www.alpertron.com.ar/ECM.HTM>), для факторизации чисел. Число b он потянуть не смог, но с числом a справился очень бодро (за 2 секунды). В результате получил разложение для числа a



Исходное число a:

96763439562657066800447535953999418604761083293285609774454036592287542174201

Первый множитель:

15 415305 286062 288869

Второй множитель:

6277 101735 386680 763835 789423 207666 416102 355444 464034 513029

Так же для числа a использовал алгоритм Ро-Полларда. Для этого написал скрипт

factorize.py

import math  
import random  
import sympy  
  
  
def read\_data():  
 with open("a.txt", "r") as a\_file:  
 a\_values = a\_file.readlines()  
  
 with open("b.txt", "r") as b\_file:  
 b\_values = b\_file.readlines()  
  
 a\_values = [int(x.strip()) for x in a\_values]  
 b\_values = [int(x.strip()) for x in b\_values]  
  
 return a\_values, b\_values  
  
  
def pollard\_rho(n):  
 if n % 2 == 0:  
 return 2  
  
 x = random.randint(2, n - 1)  
 y = x  
 c = random.randint(1, n - 1)  
 d = 1  
  
 while d == 1:  
 x = (x \* x + c) % n  
 y = (y \* y + c) % n  
 y = (y \* y + c) % n  
 d = math.gcd(abs(x - y), n)  
  
 return d if d != n else None  
  
  
def factorize(n):  
 if n <= 1:  
 return []  
 if n % 2 == 0:  
 return [2] + factorize(n // 2)  
  
 factors = []  
 stack = [n]  
  
 while stack:  
 num = stack.pop()  
 if num == 1:  
 continue  
 if math.isqrt(num) \*\* 2 == num:  
 factors.append(math.isqrt(num))  
 factors.append(math.isqrt(num))  
 continue  
  
 divisor = pollard\_rho(num)  
  
 if divisor is None or divisor == num:  
 factors.append(num)  
 else:  
 stack.append(divisor)  
 stack.append(num // divisor)  
  
 return sorted(factors)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 # Концебалов Олег Сергеевич -> 9f515d98e51437f5f8aa21aeb0b70d881567a5f3aea12d9c5d968d33a71f03b9  
 MY\_VARIANT = int("b9", 16)  
 print(f"Variant number is {MY\_VARIANT}")  
  
 a\_vals, b\_vals = read\_data()  
  
 MY\_A = a\_vals[MY\_VARIANT]  
 a\_factors = factorize(MY\_A)  
 print(f"Number a factorization: {a\_factors}")

Оставил скрипт работать на ночь. Утром увидел такой же результат, как и на сайте. Понадеялся на удачу и оставил скрипт работать для числа b и ушел на работу. Вечером меня ждало разочарование...

5) От товарищей узнал, что числа b в файле имеют одинаковые делители. Получается, что достаточно только посчитать НОД моего числа и всех остальных! Для этого написал еще один скрипт

MY\_B = b\_vals[MY\_VARIANT]  
b\_factors = []  
  
for i in range(len(a\_vals)):  
 gcd\_b = math.gcd((b\_vals[i]), MY\_B)  
 if gcd\_b != 1 and gcd\_b != MY\_B:  
 if sympy.isprime(gcd\_b) and sympy.isprime(MY\_B // gcd\_b):  
 b\_factors = sorted([gcd\_b, MY\_B // gcd\_b])  
 print(f"GCD найден с b[{i}]")  
  
print(f"Number b factorization: {b\_factors}")

В результате получил следующие результаты. НОД нашелся с числами b из 3 и 49 вариантов

Исходное число:

32317006071311007300714876688669951960444102669715484032130345427524655138867890893197201411522913463688717960921898019494119559150490921095088153935978547413407106034705318724269290875132649705524027308196782217989880784005790132180056965730197368023864717088639627241255553299019067586986868632977354402821340072020314663874025281187486036659345853007883923236487363329861102452452808315330018955525281943958860867452493239200105086177288050089836253088669398873784435546028798732831479426756490842626877346265450786467263900337609425102679070272699986331890774807663717993368009646261419233410406005610916587986197

Первый множитель:

13407807929942597099574024998205846127479365820592393377723561443721764030073546976801874298166903427690031858186486050853753882811946569946433649648959371

Второй множитель:

2410312426921032588580116606028314112912093247945688951359675039065257391591803200669085024107346049663448766280888004787862416978794958324969612987890774651455213339381625224770782077917681499676845543137387820057597345857904599109461387122099507964997815641342300677629473355281617428411794163967785870370368969109221591943054232011562758450080579587850900993714892283476646631181515063804873375182260506246992837898705971012525843324401232986857004761095088607

# **Выводы**

В ходе лабораторной работы я познакомился с хэш функцией ГОСТ Р 34.11-2012 «Стрибог». Была рассмотрена задача факторизации, которая является одной из основополагающих в криптографии. Установлено, что сложность факторизации больших чисел лежит в основе безопасности многих криптографических алгоритмов, таких как RSA. И на собственном опыте получилось убедиться, насколько сложно факторизовать столь огромные числа, особенно если это делать без каких-либо подсказок (НОД чисел из других вариантов). Также 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. На практике было показано, как с помощью ро-алгоритма можно находить нетривиальные делители числа, что подтверждает его применимость для решения задач криптоанализа.

# **Список используемой литературы**

* <https://ru.wikipedia.org/wiki/Факторизация_целых_чисел#Экспоненциальные_алгоритмы>
* <https://ru.wikipedia.org/wiki/Ро-алгоритм_Полларда#Особенности_реализации>
* <https://www.alpertron.com.ar/ECM.HTM>
* <https://hashing.tools/streebog/streebog-256>