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-308Б-22

Студентка: К. А. Былькова

Преподаватель: А. В. Борисов

Оценка:

Дата: 06.04.2025

Москва, 2025

ОГЛАВЛЕНИЕ

1 Тема

Хеширование с помощью ГОСТ Р 34.11-2012 (Стрибог) и факторизация больших чисел.

2 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.

3 Теория

«Стрибог» — криптографический алгоритм вычисления хеш-функции с размером блока входных данных 512 бит и размером хеш-кода 256 или 512 бит.

Концепции построения хэш-функции «Стрибог»:

- у новой хеш-функции не должно быть свойств, которые позволяли бы применить известные атаки;
- в хеш-функции должны использоваться изученные конструкции и преобразования;
- вычисление хеш-функции должно быть эффективным, занимать мало времени;
- не должно быть лишних преобразований, усложняющих конструкцию хеш-функции. Причем каждое используемое в хеш-функции преобразование должно отвечать за определённые криптографические свойства.

В основу хеш-функции положена итерационная конструкция Меркла — Дамгора с использованием MD-усиления. Под MD-усилением понимается дополнение неполного блока при вычислении хеш-функции до полного путём добавления вектора (0 … 01) такой длины, чтобы получился полный блок. Из дополнительных элементов нужно отметить следующие:

- завершающее преобразование, которое заключается в том, что функция сжатия применяется к контрольной сумме всех блоков сообщения по модулю 2^{512};
- при вычислении хеш-кода на каждой итерации применяются разные функции сжатия. Можно сказать, что функция сжатия зависит от номера итерации.

В хеш-функции важным элементом является функция сжатия. В ГОСТ Р 34.11-2012 функция сжатия основана на конструкции Миагути — Пренеля.

Кратко описание хеш-функции ГОСТ Р 34.11-2012 можно представить следующим образом. На вход хеш-функции подается сообщение произвольного размера. Далее сообщение разбивается на блоки по 512 бит, если размер сообщения не кратен 512, то оно дополняется необходимым количеством бит. Потом итерационно используется функция сжатия, в результате действия которой обновляется внутреннее состояние хеш-

функции. Также вычисляется контрольная сумма блоков и число обработанных бит. Когда обработаны все блоки исходного сообщения, производятся ещё два вычисления, которые завершают вычисление хеш-функции:

- обработка функцией сжатия блока с общей длиной сообщения;
- обработка функцией сжатия блока с контрольной суммой.

Факторизацией натурального числа называется его разложение в произведение простых множителей.

В отличие от задачи распознавания простоты числа, факторизация предположительно является вычислительно сложной задачей. В настоящее время неизвестно, существует ли эффективный не квантовый алгоритм факторизации целых чисел. Однако доказательства того, что не существует решения этой задачи за полиномиальное время, также нет.

Предполагаемая большая вычислительная сложность задачи факторизации лежит в основе криптостойкости некоторых алгоритмов шифрования с открытым ключом, таких как RSA. Более того, если известен хотя бы один из параметров ключей RSA, то система взламывается однозначно, кроме того, существует множество алгоритмов восстановления всех ключей в системе, обладая какими-то данными.

Как правило, алгоритм факторизации ищет первый простой делитель, после чего, при необходимости, можно запустить алгоритм заново для дальнейшей факторизации. Также, прежде чем начинать факторизацию большого числа, следует убедиться в том, что оно не простое. Существует множество алгоритмов факторизации. Вот некоторые из них:

- Перебор возможных делителей;
- Метод факторизации Ферма;
- ρ-алгоритм Полларда;

4 Ход лабораторной работы

Для определения номера варианта был написан скрипт на Python с использованием библиотеки hashlib. Для начала конвертируем строку с ФИО в байтовый формат и создаём объект для вычисления SHA-256 хеша от полученных байтов. Далее вычисляем хеш и возвращаем его как байтовую строку. Затем получаем хеш в виде шестнадцатеричной строки, извлекаем последние 8 бит, которые и представляют нужный номер варианта.

Код:

```python
import hashlib

message = "Былькова Кристина Алексеевна"

hash_object = hashlib.sha256(message.encode())
hash_bytes = hash_object.digest()  # Получаем хеш в виде байтов
hash_hex = hash_object.hexdigest()  # Получаем хеш в hex-формате
variant_number = hash_bytes[-1]

print("SHA-256 хеш:", hash_hex)
print("Вариант:", variant_number)
```

Результат:

```
Admin@kr1st1na0 MINGW64 ~/MINE/Python_rep/crypto
$ python variant.py
SHA-256 хеш: 48d25deb86f102307657c25cd1d9d1a351885f143882fa5e65b716853bd13a82
Вариант: 130
(myenv)
```

Таким образом, получили номер варианта. Числа a и b:

a[130]=74824937442907644295447904672755907842966347185456200207511741852189014011791

b[130]=32317006071311007300714876688669951960444102669715484032130345427524655138867890893197201411522913463688717960921898019494119559150490921095088153086249419559718052131324767711886673879844433992185285976186206340313183340137864755107892990120576580413399428088195869109036509736457212250441125434177525335001187530188218311657731855730123657446894274587405502726816767124723352811243072600 2

6582468217628355641680496123694130052068905454664210263890123415540099535817891590638287525556884955810416611583635799712529399455613784260009015365313458172187874376126418174574625101754478782064027180392242210735496287988054 1

Для того, чтобы разложить число на тривиальные сомножители (произвести факторизацию) в Python существует функция factorint из библиотеки sympy. Но перед этим необходимо убедиться, что число не простое (с помощью функции isprime из simpy). Поскольку число a относительно небольшое, его можно факторизовать с помощью данной функции. Число b же гораздо больше и такой способ не подойдёт. Была попытка использовать ρ-алгоритм Полларда, но программа работала долго и так и не нашла делители. Было предположено, что при наличии подходящих чисел, можно использовать их для решения поставленной задачи. Было решено при факторизации числа b использовать следующий подход: первый множитель находится как НОД с числом другого варианта, а второй — делением.

Описание алгоритма:

Все числа из других вариантов добавлены в файл, где каждая строка – число. Далее для входного числа n вычисляется НОД(n, m) с каждым числом m из файла. В функции factorize найденный делитель d и частное n/d рекурсивно разлагаются на множители. Если число простое (проверяется через isprime из библиотеки sympy), оно возвращается как конечный множитель.

В конце проводится проверка найденных делителей, перемножая их и сравнивая с исходным числом. Также для каждого числа производится замер времени выполнения алгоритма.

Для первого числа факторизация производилась двумя способами: используя factorint и используя поиск наибольшего общего делителя. Оценка времени выполнения каждого из способов показывает то, что нахождение делителей с помощью алгоритма, использующего НОД, гораздо быстрее и эффективнее.

Код:

```
import math
import time
from sympy import isprime, factorint

def read_from_file(filename):
    with open(filename, 'r') as file:
        numbers = [int(line.strip()) for line in file if line.strip()]
    return numbers

def gcd(n, numbers):
    for m in numbers:
        d = math.gcd(n, m)
        if 1 < d < n:
            return d
    return None

def factorize(n, numbers):
    if isprime(n):
        return [n]
    d = gcd(n, numbers)
    if d is None:
        raise ValueError("Делители не были найдены")
    return factorize(d, numbers) + factorize(n // d, numbers)

def print_results(factors, start_time):
    for i, factor in enumerate(factors, 1):
        print(f"делитель_{i} = {factor}")
        # print(isprime(factor))
    print(f"Затраченное время: {(time.time() - start_time):.6f}
секунд")

def main():
    a =
748249374429076442954479046727559078429663471854562002075117418521890 1
4011791
    print("Число a простое:", isprime(a))
    print("Факторизация числа a =", a)

    print("\n-С использованием factorint из sympy:")
    start_time = time.time()
    factors = factorint(a)
    print_results(factors, start_time)

    print("\n-С использованием НОД:")
    a_filename = "a_numbers.txt"
    a_numbers = read_from_file(a_filename)
    start_time = time.time()
    a_factors = factorize(a, a_numbers)
    print_results(a_factors, start_time)
    print("Проверка a == делитель_1 * делитель_2:", a == a_factors[0]
* a_factors[1])

    print("_____\n")

    b =
323170060713110073007148766886699519604441026697154840321303454275246 5
513886789089319720141152291346368871796092189801949411955915049092109 5
```

```
088153086249419559718052131324767711886673879844433992185285976186206340313183340137864755107892990120576580413399428088195869109036509736457212250441125434177525335001187530188218311657731855730123657446894274587405502726816767124723352811243072600265824682176283556416804961236941300520689054546642102638901234155400995358178915906382875255568849558104166115836357997125293994556137842600090153653134581721878743761264181745746251017544787820640271803922422107354962879880541
    print("Число b простое:", isprime(b))
    print("Факторизация числа b =", b, "\n")

    b_filename = "b_numbers.txt"
    b_numbers = read_from_file(b_filename)
    start_time = time.time()
    b_factors = factorize(b, b_numbers)
    print_results(b_factors, start_time)
    print("Проверка b == делитель_1 * делитель_2:", b == b_factors[0]
* b_factors[1])


if __name__ == '__main__':
        main()
```

Результат:

```
Admin@kr1st1na0 MINGW64 ~/MINE/Python_rep/crypto
$ python main.py
Число a простое: False
Факторизация числа a =
748249374429076442954479046727559078429663471854562002075117418521890140117791

-С использованием factorint из sumpy:
делитель_1 = 11920300259128792579
делитель_2 =
627710173538668076383578942320766641610235544446403451302 9
Затраченное время: 7.865051 секунд

-С использованием НОД:
делитель_1 =
627710173538668076383578942320766641610235544446403451302 9
делитель_2 = 11920300259128792579
Затраченное время: 0.001000 секунд
Проверка a == делитель_1 * делитель_2: True
_____________________________________

Число b простое: False
Факторизация числа b =
323170060713110073007148766886699519604441026697154840321303454275246551388678908931972014115229134636887179609218980194941195591504909210950881530862494195597180521313247677118866738798444339921852859761862063403131833401378647551078929901205765804133994280881958691090365097364572122504411254341775253350011875301882183116577318557301236574468942745874055027268167671247233528112430726002658246821762835564168049612369413005206890545466421026389012341554009953581789159063828752555688495581041661158363579971252939945561378426000901536531345817218787437612641817457462510175447878206402718039224221073549628798805413
```

```
делитель_1 =
13407807929942597099574024998205846127479365820592393377723561443721764030073546976801874298166903427690031858186486050853753882811946569946433649296420373
делитель_2 =
24103124269210325885801166060283141129120932479456889513596750390652573915918032006690850241073460496634487662808880047878624169787949583249696129878907746514552133393816252247707820779176814996768455431373878200575973458579045991094613871220995079649978156413423006776294733552816174284117941639677858703703689691092215919430542320115627584500805795878509009937148922834766466311815150638048733751822605062469928378987059710125258433244012329868570047606846454175
Затраченное время: 0.044002 секунд
Проверка b == делитель_1 * делитель_2: True
(myenv)
```

Скриншоты:

```
Число a простое: False
Факторизация числа a = 748249374429076442954479046727559078429663471854562002075117418521890140117911

-С использованием factorint из sumpy:
делитель_1 = 119203002591287925791
делитель_2 = 627710173538668076383578942320766641610235544446403451302 9
Затраченное время: 7.865051 секунд

-С использованием НОД:
делитель_1 = 627710173538668076383578942320766641610235544446403451302 9
делитель_2 = 119203002591287925791
Затраченное время: 0.001000 секунд
Проверка a == делитель_1 * делитель_2: True
```

```
Число b простое: False
Факторизация числа b = 323170060713110073007148766886699519604441026697154840321303454275246551388678908931972014115229134636887179609218980194941195591504909210950881530862494195597180521313247677118866738798444339921852859761862063403131833401378647551078929901205765804133994280881958691090365097364572122504411254341775253350011875301882183116577318557301236574468942745874055027268167671247233528112430726002658246821762835564168049612369413005206890545466421026389012341554009953581789159063828752555688495581041661158363579971252939945561378426000901536531345817218787437612641817457462510175447878206402718039224221073549628798805410

делитель_1 = 134078079299425970995740249982058461274793658205923933777235614437217640300735469768018742981669034276900318581864860508537538828119465699464336492964203730
делитель_2 = 241031242692103258858011660602831411291209324794568895135967503906525739159180320066908502410734604966344876628088800478786241697879495832496961298789077465145521333938162522477078207791768149967684554313738782005759734585790459910946138712209950796499781564134230067762947335528161742841179416396778587037036896910922159194305423201156275845008057958785090099371489228347664663118151506380487337518226050624699283789870597101252584332440123298685700476068464541796
Затраченное время: 0.044002 секунд
Проверка b == делитель_1 * делитель_2: True
```

5 Выводы

Был написан скрипт на Python, который по ФИО определяет номер варианта, используя хеш-функцию ГОСТ Р 34.11-2012 (Стрибог). Также была разработана программа, раскладывающая большие числа на нетривиальные сомножители.

В ходе выполнения данной лабораторной работы были сделаны ценные теоретические и практические выводы, подтверждающие фундаментальные положения теории чисел и криптографии. Экспериментальным путем было установлено, что метод факторизации с использованием вычисления наибольшего общего делителя демонстрирует высокую эффективность при наличии в исходных данных чисел, имеющих нетривиальные общие делители с факторизуемым числом.

В результате выполнения данной лабораторной работы были приобретены навыки, которые будут полезны для выполнения других работ и курсовых проектов.

6 Список используемой литературы

1. Применко Э. А. Алгебраические основы криптографии — М.: Книжный дом “ЛИБРОКОМ”, 2013. — 289 с.
2. НАЦИОНАЛЬНЫЙ СТАНДАРТ РОССИЙСКОЙ ФЕДЕРАЦИИ, Информационная технология, КРИПТОГРАФИЧЕСКАЯ ЗАЩИТА ИНФОРМАЦИИ, Функция хэширования, ГОСТ Р 34.11-2012 — Стандартинформ, 2012. — 38 с.
3. Криптоанализ хэш-функции ГОСТ Р 34.11-2012 // Хабр — URL: https://habr.com/ru/articles/210684/ (дата обращения: 01.04.2025)
4. Streebog (GOST R 34.11-2012) in Python // SSOJet — URL: https://hashing.ssojet.com/streebog-gost-r-3411-2012-in-python/ (дата обращения: 01.04.2025)
5. RSA простыми словами и в картинках // Хабр — URL: https://habr.com/ru/articles/745820/ (дата обращения: 01.04.2025)
6. Pollard’s Rho Algorithm for Prime Factorization // GeeksForGeeks — URL: https://www.geeksforgeeks.org/pollards-rho-algorithm-prime-factorization/ (дата обращения: 01.04.2025)