Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Методы защиты информации

ОТЧЁТ

к лабораторной работе №5

на тему

на тему «Хеш-функции»

Выполнил: Е.А. Киселева

Проверил: А. В. Герчик

Минск 2024

**СОДЕРЖАНИЕ**

[1 Постановка задачи 3](#_Toc181565713)

[2 Краткие теоретические сведения 4](#_Toc181565714)

[2.1 Генерация ключа 4](#_Toc181565714)

[2.2 Шифрование 4](#_Toc181565714)

[2.3 Расшифрование 4](#_Toc181565714)

[3 Результаты выполнения лабораторной работы 5](#_Toc181565715)

[Выводы 6](#_Toc181565716)

[Приложение А (обязательное) Листинг программного кода 7](#_Toc181565717)

# **ПОСТАНОВКА ЗАДАЧИ**

Целью выполнения данной лабораторной работы является изучение теоретических сведений и реализация без использования готовых библиотек и функций программного средства контроля целостности сообщений с помощью вычисления хеш-функции и алгоритма ГОСТ 34.11 и SHA 1.

# **КРАТКИЕ ТЕОРЕТИЧЕСКИЕ** **СВЕДЕНИЯ**

1. **Общие сведения**

Хеш-функции получили широкое распространение в разнообразных алгоритмах быстрого поиска информации.

Однако с появлением криптографии у них появилась вторая, ничуть не меньшая, область применения.

Хеш-функцией (англ, hash — мелко измельчать и перемешивать) называется необратимое преобразование данных, обладающее следующими свойствами:

1 На вход алгоритма преобразования может поступать двоичный блок данных произвольной длины.

2 На выходе алгоритма получается двоичный блок данных фиксированной длины.

3 Значения на выходе алгоритма распределяются по равномерному закону по всему диапазону возможных результатов;

4 При изменении хотя бы одного бита на входе алгоритма его выход значительно меняется: в идеальном случае инвертируется произвольная половина бит.

Основное, но не единственное, предназначение хеш-функций в криптографии – вычисление "неподделываемых" контрольных сумм документов.

Действительно, если для алгоритма хеширования выполняются требования невозможности подобрать иной документ с той же хеш-суммой и невозможности подобрать два документа с произвольной одинаковой хеш-суммой, то хеш-сумма становится уникальной характеристикой документа.

1. **Алгоритм вычисления хеш-функции ГОСТ 34.11**

В алгоритме ГОСТ 34.11 используются следующие преобразования:

1 X-преобразование. На вход функции X подаются две последовательности длиной 512 бит каждая, выходом функции является XOR этих последовательностей.

2 S-преобразование. Функция S является обычной функцией подстановки. Каждый байт из 512-битной входной последовательности заменяется соответствующим байтом из таблицы подстановок π.

3 P-преобразование. Функция перестановки. Для каждой пары байт из входной последовательности происходит замена одного байта другим.

4 L-преобразование. Представляет собой умножение 64-битного входного вектора на бинарную матрицу A размерами 64x64.

Для любого входного сообщения M:

1 Присвоить начальные значения внутренних переменных.

2 Проверить условие длина сообщения M<512. Если условие выполняется, то произвести дополнение сообщения M до длины в 512 бит. В противном случае выполнить последовательность вычислений и обрезать М, убрав последние 512 бит.

3 Повторять шаг 2 до выполнения условия на длину сообщения.

Для хеш-функции с длиной выхода в 512 бит возвращаем h в качестве результата. Для функции с длиной выхода 256 бит возвращаем MSB 256 (h).

1. **Алгоритм вычисления хеш-функции SHA-1**

Алгоритм SHA-1 (Secure Hash Algorithm) предложен Институтом Стандартизации США NIST как стандарт хеширования в гражданской криптографии. Этот алгоритм был призван дать еще больший запас прочности к криптоатакам.

SHA-1 реализует [хеш-функцию](https://ru.wikipedia.org/wiki/%D0%A5%D0%B5%D1%88%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), построенную на идее функции сжатия. Входами функции сжатия являются блок сообщения длиной 512 бит и выход предыдущего блока сообщения.

Выход представляет собой значение всех хеш-блоков до этого момента. Иными словами хеш блока Mi равен hi = f (Mi , hi−1 ). Хеш-значением всего сообщения является выход последнего блока.

Алгоритм получает на входе сообщение максимальной длины бит и создает в качестве выхода дайджест сообщения длиной 160 бит.

# **РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ**

В ходе выполнения лабораторной было реализовано криптостойкое программное средство шифрования и дешифрования текстовых файлов при помощи Криптосистемы Мак-Элиса.

Начальный текст находится в файле input.txt. После шифрования зашифрованная информация помещается в файл encrypted.txt, после чего она снова дешифруется и помещается в файл decrypted.txt. В консоль выводится информация о завершении шифрования и завершении дешифрования, а также в какие файлы сохраняются данные.

Результат выполнения лабораторной работы представлен на рисунке 3.1.

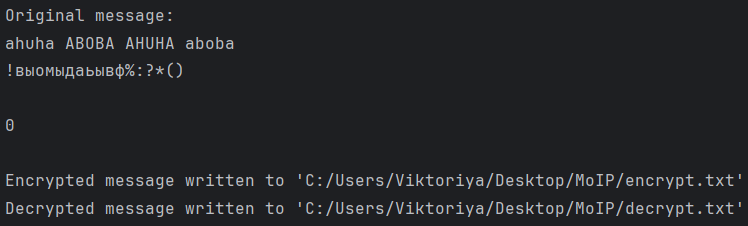


Рисунок 3.1 – Результат выполнения лабораторной работы

Содержимое файла encrypted.txt представлено на рисунке 3.2.

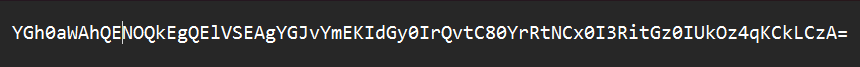


Рисунок 3.2 – Файл encrypted.txt

# **ВЫВОДЫ**

В ходе данной лабораторной работы были изучены теоретические сведения и реализовано без использования готовых библиотек и функций программное средство контроля целостности сообщений с помощью вычисления хеш-функции и алгоритма ГОСТ 34.11 и SHA 1.

## **ПРИЛОЖЕНИЕ А**

## **(обязательное)**

## **Листинг программного кода**

Листинг 1 – Программный код файла lab4.py

import os

import numpy as np

import base64

input\_file\_path = "C:/Users/Viktoriya/Desktop/MoIP/input.txt"

output\_file\_path = "C:/Users/Viktoriya/Desktop/MoIP/encrypt.txt"

decrypt\_file\_path = "C:/Users/Viktoriya/Desktop/MoIP/decrypt.txt"

if not os.path.exists(input\_file\_path):

print(f"Input file '{input\_file\_path}' not found.")

else:

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

original\_message = file.read()

print("Original message:\n" + original\_message)

def generate\_random\_matrix(rows, columns):

return np.random.randint(2, size=(rows, columns), dtype=np.uint8)

def encrypt(message, generator\_matrix):

message\_bytes = message.encode('utf-8')

encrypted\_bytes = bytearray(len(message\_bytes))

rows, columns = generator\_matrix.shape

for i in range(len(message\_bytes)):

encrypted\_bytes[i] = message\_bytes[i] ^ generator\_matrix[i % rows, i % columns]

return base64.b64encode(encrypted\_bytes).decode('utf-8')

def decrypt(encrypted\_message, generator\_matrix):

encrypted\_bytes = base64.b64decode(encrypted\_message)

decrypted\_bytes = bytearray(len(encrypted\_bytes))

rows, columns = generator\_matrix.shape

for i in range(len(encrypted\_bytes)):

decrypted\_bytes[i] = encrypted\_bytes[i] ^ generator\_matrix[i % rows, i % columns]

return decrypted\_bytes.decode('utf-8')

# Размеры матрицы

n = 512

k = 256

generator\_matrix = generate\_random\_matrix(k, n)

encrypted\_message = encrypt(original\_message, generator\_matrix)

with open(output\_file\_path, "w", encoding="utf-8") as file:

file.write(encrypted\_message)

print(f"\nEncrypted message written to '{output\_file\_path}'")

decrypted\_message = decrypt(encrypted\_message, generator\_matrix)

with open(decrypt\_file\_path, "w", encoding="utf-8") as file:

file.write(decrypted\_message)

print(f"\nDecrypted message written to '{decrypt\_file\_path}'")