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

**ХЕШ-ФУНКЦИИ**

Выполнил: студент гр.253505 Ющук И.А.

Проверил: ассистент кафедры информатики Герчик А.В.

Минск 2025

**СОДЕРЖАНИЕ**

[Введение 3](#_Toc207721845)

[1 Ход работы 4](#_Toc207721846)

[Заключение 5](#_Toc207721847)

[Приложение А (обязательное) Листинг программного кода 6](#_Toc207721848)

# ВВЕДЕНИЕ

Современный этап развития информационных технологий характеризуется повсеместным обменом цифровыми данными и острой необходимостью обеспечения их целостности и подлинности. Под целостностью понимается гарантия того, что информация не была изменена несанкционированным образом с момента ее создания или передачи. Для решения этой фундаментальной задачи информационной безопасности используются криптографические хеш-функции.

В отличие от алгоритмов шифрования, которые предназначены для обеспечения конфиденциальности и являются обратимыми (с помощью ключа), хеш-функции представляют собой односторонние математические преобразования. Они преобразуют произвольный массив входных данных (сообщение) произвольной длины в уникальную битовую строку фиксированной длины, называемую хешем или дайджестом сообщения. Ключевыми свойствами криптографических хеш-функций являются стойкость к коллизиям (невозможность найти два разных сообщения с одинаковым хешем), необратимость (невозможность восстановить исходное сообщение по его хешу) и лавинный эффект (незначительное изменение во входных данных приводит к кардинальному изменению выходного хеша).

Такие свойства обеспечивают основу для создания механизмов контроля целостности, цифровых подписей, аутентификации сообщений и хранения паролей. Наиболее известными и широко применяемыми в мире алгоритмами являются алгоритмы семейства SHA (Secure Hash Algorithm), разработанные NSA и стандартизированные NIST. Однако существуют и национальные стандарты, отражающие криптографические практики отдельных стран.

Одним из таких стандартов является российский алгоритм ГОСТ Р 34.11-2012 («Стрибог»), который пришел на смену более старому ГОСТ Р 34.11-94. Его основное преимущество заключается в использовании современных криптографических преобразований и соответствии требованиям безопасности Российской Федерации. В то же время, алгоритм SHA-1, некогда повсеместно используемый, сейчас считается устаревшим и небезопасным из-за обнаруженных уязвимостей и теоретических атак на его стойкость. Его изучение, однако, остается важным с исторической и образовательной точек зрения для понимания эволюции криптографических стандартов.

Целью данной лабораторной работы является теоретическое и практическое изучение принципов работы криптографических хеш-функций и реализация программного средства контроля целостности сообщений. В рамках работы будет проведена сравнительная реализация и анализ двух алгоритмов: современного отечественного стандарта ГОСТ 34.11 и исторически значимого, но устаревшего алгоритма SHA-1. Это позволит на практике оценить их основные свойства, производительность и особенности реализации.

# **1 ХОД РАБОТЫ**

В рамках выполнения лабораторной работы были реализованы алгоритмы криптографических хеш-функций ГОСТ Р 34.11-94 и SHA-1 на языке Python.

На первом этапе проведено изучение математических основ и структуры данных алгоритмов. Для ГОСТ 34.11 особое внимание уделено изучению алгоритма шифрования ГОСТ 28147-89, используемого в качестве преобразующей функции, а также особенностям заполнения и обработки сообщений. Для SHA-1 изучены принципы итерационного сжатия, использования буферов и логических функций.

После освоения теоретического материала приступили к практической реализации. Были написаны функции для предобработки входных сообщений (дополнение до кратного длины, разбиение на блоки). Для ГОСТ 34.11 реализованы алгоритм шифрования ГОСТ 28147-89 и основные преобразования: итерация сжатия, суммирование с регистром. Для SHA-1 реализованы функции расширения сообщения, раундовые преобразования и логические операции.

Для проверки корректности работы алгоритмов проведено тестирование на стандартных тестовых векторах из документации. Реализована функция визуализации хеш-значений в шестнадцатеричном представлении.

Начальные данные представлены на рисунке 1.1.



Рисунок 1.1 – Входные данные

Программа была в результате появился захешированный текст. Процесс работы программы представлен на рисунке 1.2.

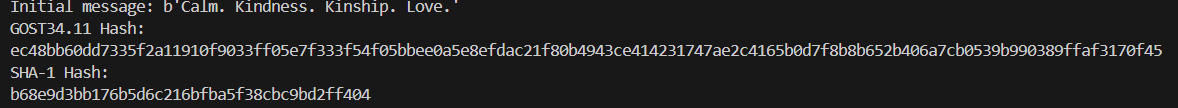


Рисунок 1.2 – Результат работы программы

# ЗАКЛЮЧЕНИЕ

В результате выполнения лабораторной работы были успешно реализованы криптографические хеш-функции ГОСТ Р 34.11-94 и SHA-1, основанные на различных принципах построения. Практическая реализация охватила все ключевые этапы алгоритмов: предобработку входных данных, дополнение сообщений, разбиение на блоки и итеративное вычисление хеш-значения. Основным достижением работы стало создание работоспособного программного средства контроля целостности, наглядно демонстрирующего принципы работы криптографических хеш-функций.

Реализация подтвердила теоретические положения алгоритмов, в частности, важность корректного выполнения всех этапов предобработки данных и преобразования блоков. Особое значение имела точная реализация алгоритма сжатия для обоих стандартов: с использованием шифрования ГОСТ 28147-89 в отечественном стандарте и системы раундовых преобразований в SHA-1. Разработанные алгоритмы продемонстрировали способность к生成 уникальных хеш-значений даже для минимальных изменений во входных данных, что подтверждает наличие лавинного эффекта.

Практическая ценность работы заключается в создании инструмента для сравнительного анализа различных криптографических хеш-функций. Реализованная система может использоваться для демонстрации принципов контроля целостности данных, изучения особенностей национального и международного стандартов хеширования, а также для понимания эволюции криптографических алгоритмов. В процессе реализации были приобретены навыки работы с битовыми операциями, освоены методы обработки данных блоками и отработаны приемы тестирования криптографических реализаций.

Перспективы развития работы включают оптимизацию алгоритмов для обработки больших объемов данных, реализацию более современных хеш-функций (таких как ГОСТ Р 34.11-2012 "Стрибог" или SHA-256), а также исследование стойкости реализованных алгоритмов к различным типам криптоатак, включая атаки нахождения коллизий.

Таким образом, реализованные хеш-функции представляют собой законченное решение, которое демонстрирует принципы построения криптографических средств контроля целостности и служит основой для дальнейшего изучения современных методов обеспечения информационной безопасности.

# ПРИЛОЖЕНИЕ А (обязательное) Листинг программного кода

@staticmethod

def hash(message\_bytes):

# Константы, инициализация

A = 0x67452301

B = 0xEFCDAB89

C = 0x98BADCFE

D = 0x10325476

E = 0xC3D2E1F0

# Дополнение сообщения

bytes\_ = bytearray(message\_bytes)

bytes\_.append(0x80) # Добавление 1 в начале

# Дополнение нулями до длины, кратной 512 битам (делается всегда)

while len(bytes\_) % 64 != 56:

bytes\_.append(0x00)

# Добавление длины иcходного сообщения (в битах) в конец сообщения

message\_length\_bits = len(message\_bytes) \* 8 #размер в байтах \* 8

bytes\_.extend(struct.pack('>Q', message\_length\_bits)) # Преобразуем к big-endian(старший байт впереди), если система little-endian

# Разделение сообщения на блоки по 512 бит (64 байта)

for i in range(0, len(bytes\_), 64):

# Создание массива из 80 слов по 32 бита

w = [0] \* 80

# Перенос блока в первые 16 слов

for j in range(16):

w[j] = struct.unpack('>I', bytes\_[i + j \* 4:i + j \* 4 + 4])[0]

# Дополнение массива до 80 слов

for j in range(16, 80):

w[j] = SHA1.rotate\_left( w[j - 16] ^ w[j - 14] ^ w[j - 8] ^ w[j - 3], 1)

# Инициализация переменных для текущего блока

a, b, c, d, e = A, B, C, D, E

# Основной цикл обработки

for j in range(80):

if j < 20:

f = (b & c) | (~b & d)

k = 0x5A827999

elif j < 40:

f = b ^ c ^ d

k = 0x6ED9EBA1

elif j < 60:

f = (b & c) | (b & d) | (c & d)

k = 0x8F1BBCDC

else:

f = b ^ c ^ d

k = 0xCA62C1D6

temp = (SHA1.rotate\_left(a, 5) + f + e + k + w[j]) & 0xFFFFFFFF