Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра экономической математики, информатики и статистики (ЭМИС)

ОДНОНАПРАВЛЕННА ХЭШ ФУНКЦИЯ MD5

Отчет по лабораторной работе по дисциплине «Защита информации»

Студент гр. 590-1

\_\_\_\_\_\_\_\_\_\_\_\_/Г.К. Петров

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2023 г.

Доктор технических наук, профессор кафедры ЭМИС

\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_/ Спицын В.Г.

оценка подпись

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2023 г.

Томск 2023

**Лабораторная работа №5**

**Однонаправленная хэш функция MD5**

**Введение**

Однонаправленные хэш-функции представляют собой важный класс криптографических алгоритмов, используемых для преобразования произвольных данных в фиксированную строку байтов, известную как "хэш-значение" или "хэш". В контексте криптографии, однонаправленность означает, что процесс вычисления хэш-значения является легким, но обратное преобразование – восстановление исходных данных из хэш-значения – вычислительно сложным.

MD5 (Message Digest Algorithm 5) представляет собой одну из наиболее широко используемых однонаправленных хэш-функций. Разработанная Рональдом Ривестом в 1991 году, она была призвана обеспечить быстрый и надежный метод создания хэш-значений для целей цифровой подписи, проверки целостности данных и других задач криптографии.

**История создания**

MD5 (Message Digest Algorithm 5) был разработан Рональдом Л. Ривестом, выдающимся американским криптографом и профессором электротехники и информатики Массачусетского технологического института (MIT). Ривест представил MD5 в 1991 году как улучшенную версию своего предыдущего алгоритма MD4.

MD4, разработанный Ривестом в 1990 году, был предназначен для быстрого создания 128-битных хэш-значений. Тем не менее, уже в 1991 году были предложены атаки, демонстрирующие недостаточную стойкость MD4 к коллизиям (ситуации, когда двум разным наборам данных соответствует одно и то же хэш-значение). Это стало мотивацией для создания более надежного алгоритма – MD5.

Ривест предложил MD5 как улучшенную версию MD4, исправляя обнаруженные недостатки. MD5 создавал хэш-значения длиной 128 бит, используя серию логических и арифметических операций, а также нелинейные функции, чтобы обеспечить однонаправленность и стойкость к коллизиям.

MD5 быстро стала популярной хэш-функцией благодаря своей простоте и быстроте вычислений. Она получила широкое применение в цифровой подписи, проверке целостности данных и других криптографических задачах.

С течением времени стали появляться серьезные сомнения в стойкости MD5. В 2004 году исследователи продемонстрировали возможность создания коллизий для MD5 с использованием специально подобранных данных. Это привело к рекомендации отказаться от использования MD5 в криптографических целях.

Сегодня MD5 считается устаревшей и уязвимой к атакам, особенно к атакам с использованием коллизий. Криптографическое сообщество рекомендует применение более стойких алгоритмов, таких как SHA-256 и SHA-3, для обеспечения безопасности в современных условиях.

Хотя MD5 утратила свою первоначальную значимость в криптографических приложениях, изучение ее истории предоставляет важный урок о постоянной необходимости развития криптографических методов и алгоритмов с учетом появляющихся угроз и технологического прогресса.

**Применение алгоритмов MD5**

MD5 (Message Digest Algorithm 5) была широко применяемой однонаправленной хэш-функцией в прошлом, но сегодня её использование в криптографических целях не рекомендуется из-за обнаруженных уязвимостей. Однако, MD5 оставляет свой след в различных областях информационных технологий:

1. **Цифровая подпись:** В прошлом MD5 использовалась для создания цифровых подписей. Цифровая подпись с использованием MD5 позволяла проверять авторство и целостность электронных документов. Однако, с появлением атак на стойкость MD5, этот метод стал ненадежным.
2. **Проверка целостности данных:** MD5 применялась для создания хэш-значений, которые можно было использовать для проверки целостности данных. Например, при загрузке файлов из Интернета, можно было предоставить MD5-хэш для сравнения с хэшем полученного файла и убедиться в его целостности.
3. **Хэширование паролей:** В некоторых системах MD5 использовалась для хэширования паролей пользователей. Однако, такое использование стало нежелательным из-за возможности подбора паролей с использованием предварительно вычисленных хэш-значений (так называемые "рейнбоу-таблицы").
4. **Системы контроля версий:** MD5 иногда использовалась в системах контроля версий, чтобы быстро определить, изменились ли файлы. Например, при работе с Git или SVN, MD5 могла использоваться для создания хэшей для версий файлов.
5. **Криптографические атаки и тестирование безопасности:** В современных условиях MD5 часто используется в качестве тестового примера для иллюстрации уязвимости хэш-функций к коллизиям. Исследователи и хакеры могут использовать атаки на MD5 для демонстрации концепций криптографических слабостей.

С учетом возможности коллизий и недостаточной стойкости MD5, рекомендуется заменять её более современными алгоритмами, такими как SHA-256 и SHA-3, для обеспечения надежной защиты данных и систем.

**Принцип работы алгоритма хеширования MD5**

MD5 (Message Digest Algorithm 5) - это однонаправленная хэш-функция, предназначенная для создания фиксированного по длине хэш-значения из произвольных данных. Вот основные шаги и принципы работы алгоритма MD5:

1. **Инициализация переменных:** MD5 начинает свою работу с инициализации четырех 32-битных переменных (A, B, C, D), которые будут использоваться в процессе обработки данных. Эти переменные инициализируются определенными константами. На рисунке 1 предствален пример инициализации переменных на языке Python.

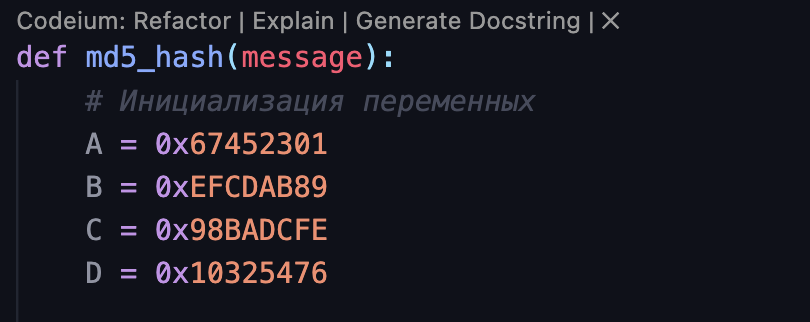


Рисунок 1 – Инициализация переменных

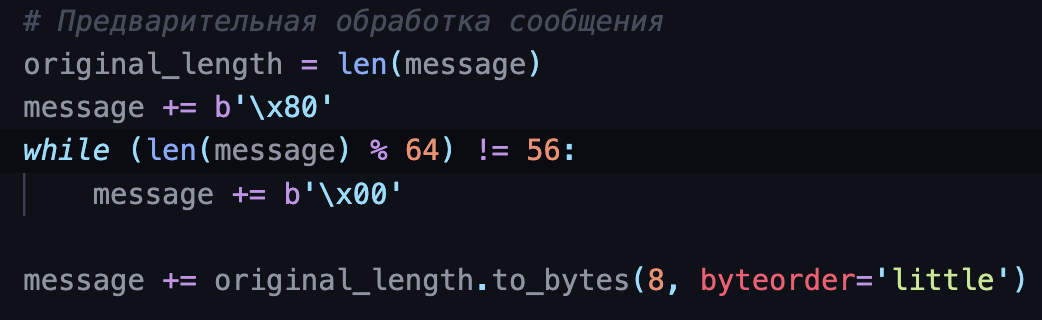
1. **Дополнение данных:** Входные данные дополняются так, чтобы их длина стала кратной 512 битам (64 байтам). Если данные не кратны 512 битам, то к ним добавляются биты так, чтобы длина данных стала правильной. На рисунке 2 приведени пример дополнения данных на языке Python.

Рисунок 2– Дополнение данных

1. **Разбиение данных на блоки:** Дополненные данные разбиваются на блоки по 512 бит. Каждый блок подается на вход функции сжатия. Пример разбиения данных представлен на рисунке 3.

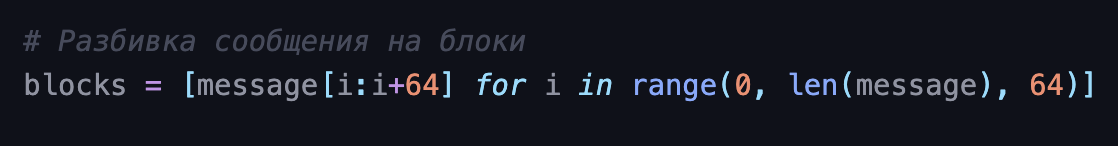


Рисунок 3– Разбиение сообщения на блоки

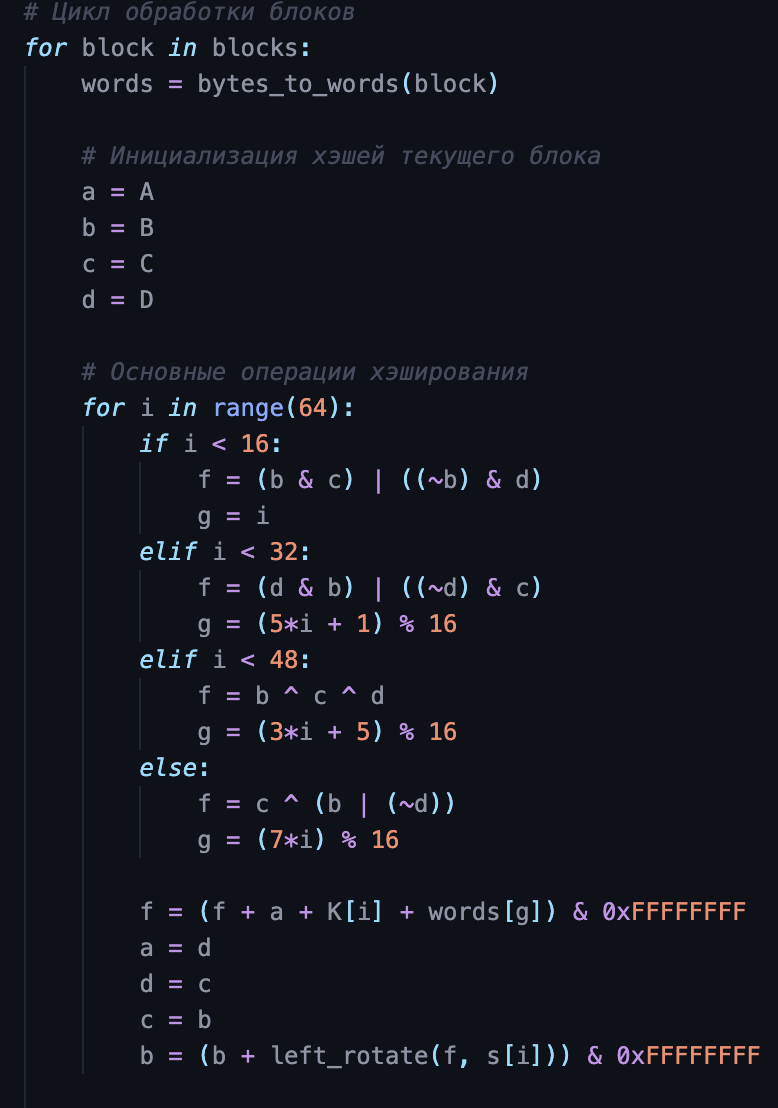
1. **Функция сжатия:** Функция сжатия MD5 принимает на вход 512-битный блок данных и текущее состояние переменных (A, B, C, D). В процессе обработки блока происходят четыре раунда (Round 1, Round 2, Round 3, Round 4), в каждом из которых используются различные нелинейные операции и таблицы констант.

Рисунок 4 – Функция сжатия

1. **Операции в каждом раунде:**
   * **Round 1:** Функции F(B, C, D) = (B & C) | ((~B) & D)
   * **Round 2:** Функции G(B, C, D) = (B & D) | (C & (~D))
   * **Round 3:** Функции H(B, C, D) = B ^ C ^ D
   * **Round 4:** Функции I(B, C, D) = C ^ (B | (~D))
2. **Обновление переменных:** Результаты каждого раунда служат для обновления текущего состояния переменных (A, B, C, D). Каждый результат прибавляется к соответствующей переменной с учетом определенных весов.

Рисунок 5 – Обновление переменных

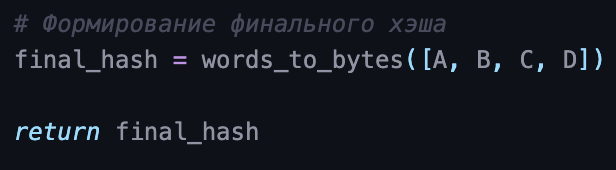
1. **Итоговое хэш-значение:** После обработки всех блоков данных получается 128-битное итоговое хэш-значение. Оно представляет собой объединение значений переменных A, B, C, D в определенном порядке.
2. **Фиксация и возврат хэша:** Итоговое хэш-значение считается окончательным и возвращается в качестве результата работы алгоритма MD5.

Рисунок 6 – Итоговое хэш значение

Важно отметить, что хотя MD5 была широко использована в прошлом, сегодня она считается устаревшей и уязвимой к атакам, таким как коллизии. Рекомендуется использовать более современные алгоритмы хэширования, такие как SHA-256 или SHA-3, для обеспечения надежности и безопасности данных.

Пример работы программы представлен на рисунке 7.

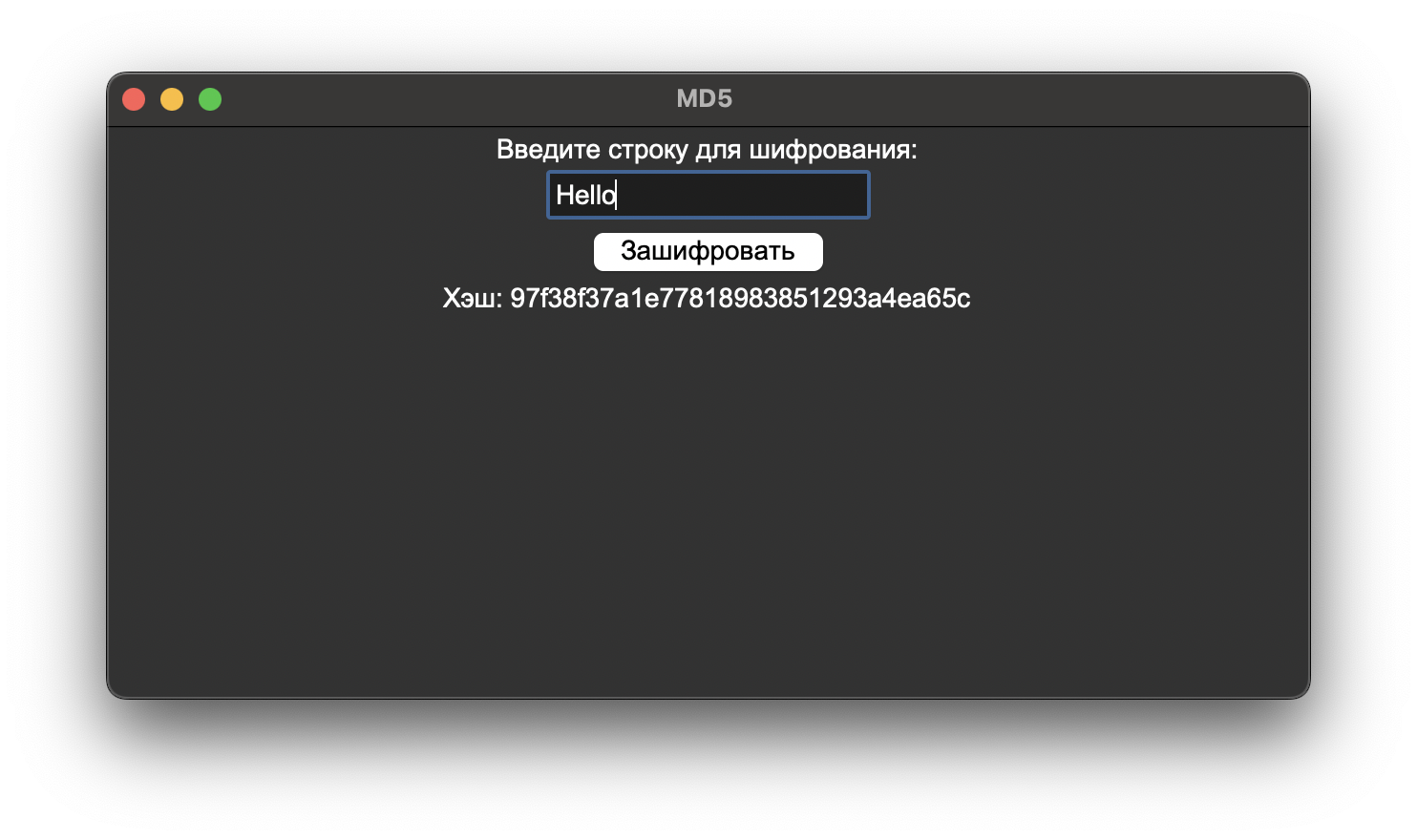


Рисунок 7 – Пример работы программы

**Вывод**

В заключении можно отметить, что хотя однонаправленная хэш-функция MD5 играла важную роль в истории криптографии и информационной безопасности, сегодня её использование не рекомендуется в связи с выявленными уязвимостями.

Приложение А

(обязательное)

Код для выполнения задания на языке Python

Код приложения с переводом строк в хэш MD5:

*import* math

*import* tkinter *as* tk

*# Инициализация констант*

K = [int(abs(math.sin(i+1)) \* 2\*\*32) & 0xFFFFFFFF *for* i *in* range(64)]

s = [

7, 12, 17, 22,

5, 9, 14, 20,

4, 11, 16, 23,

6, 10, 15, 21,

7, 12, 17, 22,

5, 9, 14, 20,

4, 11, 16, 23,

6, 10, 15, 21,

7, 12, 17, 22,

5, 9, 14, 20,

4, 11, 16, 23,

6, 10, 15, 21,

7, 12, 17, 22,

5, 9, 14, 20,

4, 11, 16, 23,

6, 10, 15, 21

]

*# Функции для преобразования байтов и слов*

def bytes\_to\_words(byte\_array):

*return* [int.from\_bytes(byte\_array[i:i+4], byteorder='little') *for* i *in* range(0, len(byte\_array), 4)]

def words\_to\_bytes(word\_array):

*return* b''.join([word.to\_bytes(4, byteorder='little') *for* word *in* word\_array])

*# Вспомогательные функции*

def left\_rotate(x, n):

*return* ((x << n) | (x >> (32 - n))) & 0xFFFFFFFF

*# Основная функция хэширования MD5*

def md5\_hash():

message = input\_entry.get()

message = bytes(message, 'utf-8')

*# Инициализация переменных*

A = 0x67452301

B = 0xEFCDAB89

C = 0x98BADCFE

D = 0x10325476

*# Предварительная обработка сообщения*

original\_length = len(message)

message += b'\x80'

*while* (len(message) % 64) != 56:

message += b'\x00'

message += original\_length.to\_bytes(8, byteorder='little')

*# Разбивка сообщения на блоки*

blocks = [message[i:i+64] *for* i *in* range(0, len(message), 64)]

*# Цикл обработки блоков*

*for* block *in* blocks:

words = bytes\_to\_words(block)

*# Инициализация хэшей текущего блока*

a = A

b = B

c = C

d = D

*# Основные операции хэширования*

*for* i *in* range(64):

*if* i < 16:

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

g = i

*elif* i < 32:

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

g = (5\*i + 1) % 16

*elif* i < 48:

f = b ^ c ^ d

g = (3\*i + 5) % 16

*else*:

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

g = (7\*i) % 16

f = (f + a + K[i] + words[g]) & 0xFFFFFFFF

a = d

d = c

c = b

b = (b + left\_rotate(f, s[i])) & 0xFFFFFFFF

*# Обновление хэшей*

A = (A + a) & 0xFFFFFFFF

B = (B + b) & 0xFFFFFFFF

C = (C + c) & 0xFFFFFFFF

D = (D + d) & 0xFFFFFFFF

*# Формирование финального хэша*

final\_hash = words\_to\_bytes([A, B, C, D])

encrypted\_label.config(text="Хэш: " + final\_hash.hex())

window = tk.Tk()

window.title("MD5 ")

input\_label = tk.Label(window, text="Введите строку для шифрования:", font=("Arial", 14))

input\_label.pack()

input\_entry = tk.Entry(window, font=("Arial", 14))

input\_entry.pack()

encrypt\_button = tk.Button(window, text="Зашифровать", command=md5\_hash, font=("Arial", 14))

encrypt\_button.pack()

encrypted\_label = tk.Label(window, text="Хэш строки:", font=("Arial", 14))

encrypted\_label.pack()

window.geometry("630x300+700+350")

window.mainloop()

print(hash.hex())