МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КУБГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**Отчет**

**по практическому заданию №11**

**по курсу**

**«КРИПТОГРАФИЧЕСКИЕ ПРОТОКОЛЫ»**

Работу выполнил

Студенты 46 группы

Нагалевский А.М.

Преподаватель:

Крамаренко А.А.

Краснодар 2024

**Постановка задачи.**

Реализовать программный продукт построения SHA-1 для введенного текста.

SHA-1 (Secure Hash Algorithm 1) — это криптографический алгоритм хеширования, который принимает на вход сообщение произвольной длины и выдает хеш-значение фиксированной длины, равной 160 битам (20 байтам). SHA-1 был разработан Национальным институтом стандартов и технологий США (NIST) и был опубликован в 1993 году.

Рассмотрим основные характеристики и особенности алгоритма:

1. **Длина хеш-значения**: 160 бит (20 байт).
2. **Размер блока**: 512 бит (64 байта).
3. **Составляющие функции**: SHA-1 использует комбинацию операций И, ИЛИ, ИСКЛЮЧАЮЩЕЕ ИЛИ и циклических сдвигов для обработки данных.
4. **Итеративный процесс**: Сообщение делится на блоки по 512 бит, а затем каждый блок обрабатывается пошагово для генерации окончательного хеш-значения.
5. **Константы и инициализационные значения**: Алгоритм использует некоторые константы и начальные значения, которые определяются стандартом.
6. **Дополнение сообщения**: Для того, чтобы длина сообщения стала кратной 512 битам, оно дополняется нулевыми битами и дополнительной информацией о длине сообщения.
7. **Раунды**: Процесс хеширования включает в себя 80 раундов обработки каждого блока сообщения.
8. **Устойчивость к коллизиям**: В начале 2000-х годов были найдены некоторые коллизии, и в настоящее время SHA-1 считается устаревшим в качестве криптографической хеш-функции для целей безопасности.

SHA-1 широко использовался в различных криптографических протоколах и приложениях, включая цифровые подписи, аутентификацию, SSL/TLS и другие. Однако с появлением коллизий, возможность подделки хеш-значений стала угрозой. В настоящее время рекомендуется использовать более безопасные алгоритмы хеширования, такие как SHA-256, SHA-384 или SHA-512, для новых приложений.

Логика алгоритма SHA-1 включает в себя несколько шагов:

1. **Подготовка сообщения**: Исходное сообщение разбивается на блоки фиксированного размера. Каждый блок должен быть представлен в виде последовательности байтов.
2. **Дополнение сообщения**: После разбиения сообщения на блоки, к последнему блоку добавляется дополнение. Это делается так, чтобы длина сообщения была кратной 512 битам (64 байта). Дополнение обычно состоит из единицы, за которой следует необходимое количество нулей, а также дополнительной информации о длине исходного сообщения.
3. **Инициализация переменных**: Задаются начальные значения хеш-значений. Для SHA-1 используется 5 переменных, каждая по 32 бита.
4. **Обработка блоков сообщения**: Каждый блок сообщения обрабатывается по отдельности. В процессе обработки каждый блок преобразуется в хеш-значение, которое затем объединяется с предыдущими хеш-значениями.
5. **Функции раунда**: Для обработки блоков используются функции раунда, включая циклические сдвиги, побитовые операции и нелинейные функции. Эти функции обеспечивают высокий уровень случайности и равномерного распределения бит в итоговом хеш-значении.
6. **Обновление хеш-значений:** После обработки каждого блока переменные, представляющие хеш-значения, обновляются в соответствии с результатами обработки блока.
7. **Формирование окончательного хеш-значения**: После обработки всех блоков исходного сообщения получается окончательное хеш-значение, которое представляет собой уникальный идентификатор для этого сообщения.

**Текст программы:**

**Файл task1.py:**

def sha1(message):

    # Инициализация переменных

    h0 = 0x67452301

    h1 = 0xEFCDAB89

    h2 = 0x98BADCFE

    h3 = 0x10325476

    h4 = 0xC3D2E1F0

    # Преобразование сообщения в байты

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

    # Дополнение сообщения до кратности 512 битам

    original\_length\_bits = (8 \* len(message)) & 0xFFFFFFFFFFFFFFFF

    message.append(0x80)

    while len(message) % 64 != 56:

        message.append(0)

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

    # Функция циклического сдвига влево

    def left\_rotate(n, b):

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

    # Функция преобразования блока

    def process\_chunk(chunk, h0, h1, h2, h3, h4):

        w = [0] \* 80

        # Разбиение блока на 16 слов по 32 бита

        for i in range(16):

            w[i] = int.from\_bytes(chunk[i\*4:i\*4+4], byteorder='big')

        # Расширение блока до 80 слов

        for i in range(16, 80):

            w[i] = left\_rotate(w[i-3] ^ w[i-8] ^ w[i-14] ^ w[i-16], 1)

        # Инициализация переменных

        a = h0

        b = h1

        c = h2

        d = h3

        e = h4

        # Основной цикл

        for i in range(80):

            if i <= 19:

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

                k = 0x5A827999

            elif i <= 39:

                f = b ^ c ^ d

                k = 0x6ED9EBA1

            elif i <= 59:

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

                k = 0x8F1BBCDC

            else:

                f = b ^ c ^ d

                k = 0xCA62C1D6

            temp = (left\_rotate(a, 5) + f + e + k + w[i]) & 0xFFFFFFFF

            e = d

            d = c

            c = left\_rotate(b, 30)

            b = a

            a = temp

        # Обновление хеш-значений

        h0 = (h0 + a) & 0xFFFFFFFF

        h1 = (h1 + b) & 0xFFFFFFFF

        h2 = (h2 + c) & 0xFFFFFFFF

        h3 = (h3 + d) & 0xFFFFFFFF

        h4 = (h4 + e) & 0xFFFFFFFF

        return h0, h1, h2, h3, h4

    # Обработка сообщения блоками

    for i in range(0, len(message), 64):

        chunk = message[i:i+64]

        h0, h1, h2, h3, h4 = process\_chunk(chunk, h0, h1, h2, h3, h4)

    # Формирование хеш-значения

    hash\_result = (h0 << 128) | (h1 << 96) | (h2 << 64) | (h3 << 32) | h4

    return '%040x' % hash\_result

# Пример использования

message = input("Введите текст для хеширования: ")

hashed\_message = sha1(message)

print("Хеш SHA-1:", hashed\_message)