Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**ЭКЗАМЕНАЦИОННЫЙ ПРОЕКТ**

**Дисциплина: Криптографические протоколы**

Работу выполнил: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_К.В.Стасюк

Д..Баева

Направление подготовки: 02.03.02 Фундаментальная информатика и информационные технологии

Преподаватель: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_О.В.Руденко

А.А. Крамаренко

**Тема проекта:** Система контроля целостности данных с помощью хеш-функций

**Ход работы:**

В своем проекте мы рассматривали контроль целостности файлов. Для получения хеша файла мы использовали алгоритм sha-256.

Для начала рассмотрим алгоритм :

Шаг 1 — Предварительная работа

Преобразуем входное сообщение в двоичный код. Добавим 1 в конец.

Дополняем код нулями, пока данные не станут равны 512 бит, минус 64 бита (в результате 448 бит).

Добавляем 64 бита в конец в виде целого числа с порядком байтов от старшего к младшему (big-endian), представляющего длину входного сообщения в двоичном формате.

Теперь у нас есть ввод, который будет делиться на 512 без остатка.

Шаг 2 — Инициализируем значения хэша (h)

Теперь мы создаем 8 хэш-значений. Это жестко запрограммированные константы, которые представляют собой первые 32 бита дробных частей квадратных корней из первых восьми простых чисел: 2, 3, 5, 7, 11, 13, 17, 19

Шаг 3 — Инициализация округленных констант (k)

Как и в предыдущем шаге, мы создадим еще несколько констант. На этот раз их будет 64. Каждое значение (0—63) представляет собой первые 32 бита дробных частей кубических корней первых 64 простых чисел (2—311).  
  
Шаг 4 — Цикл фрагментов

Следующие шаги будут выполняться для каждого 512-битного «фрагмента» из наших входных данных. В каждой итерации цикла мы будем изменять хэш-значения h0-h7, что приведет нас к конечному результату.

Шаг 5 — Создание расписания сообщений (w)

Скопируйте входные данные из шага 1 в новый массив, где каждая запись представляет собой 32-битное слово:  
  
Добавьте еще 48 слов, инициализированных нулем, чтобы у нас получился массив w [0… 63]  
  
  
Измените обнуленные индексы в конце массива, используя следующий алгоритм:  
Для **i** из w[16…63]:

* s0 = (w[i-15] rightrotate 7) xor (w[i-15] rightrotate 18) xor (w[i-15] rightshift 3)
* s1 = (w[i- 2] rightrotate 17) xor (w[i- 2] rightrotate 19) xor (w[i- 2] rightshift 10)
* w[i] = w[i-16] + s0 + w[i-7] + s1

Шаг 6 — Сжатие

Инициализируем переменные **a, b, c, d, e, f, g, h** и установите их равными текущим значениям хэш-функции соответственно **h0, h1, h2, h3, h4, h5, h6, h7**.  
  
Запустите цикл сжатия, который изменит значения **a… h**. Выглядит он следующим образом:  
  
Для **i** от 0 до 63

* S1 = (e rightrotate 6) xor (e rightrotate 11) xor (e rightrotate 25)
* ch = (e and f) xor ((not e) and g)
* temp1 = h + S1 + ch + k[i] + w[i]
* S0 = (a rightrotate 2) xor (a rightrotate 13) xor (a rightrotate 22)
* maj = (a and b) xor (a and c) xor (b and c)
* temp2 := S0 + maj
* h = g
* g = f
* e = d + temp1
* d = c
* c = b
* b = a
* a = temp1 + temp2

Все вычисления выполняются еще 63 раза, меняя переменные a-h.

Шаг 7 — Изменение окончательных значений

После цикла сжатия, во время цикла фрагментов, мы изменяем хеш-значения, добавляя к ним соответствующие переменные a-h. Как и ранее, все сложение производится по модулю 2 ^ 32:

h0 = h0 + a

h1 = h1 + b

h2 = h2 + c

h3 = h3 + d

h4 = h4 + e

h5 = h5 + f

h6 = h6 + g

h7 = h7 + h

Шаг 8 — Финальный хэш

Наконец, соединяем все вместе.

digest = h0 append h1 append h2 append h3 append h4 append h5 append h6 append h7

**Программная реализация:**

1. **Вычисление хеша SHA-256**: В функции **calculate\_sha256(data)** происходит вычисление хеша SHA-256 для переданных данных. Этот процесс включает в себя инициализацию начальных значений хеша, предварительную обработку данных, разбиение данных на блоки, цикл хеширования для каждого блока и обновление значения хеша.
2. **Проверка целостности файлов в папке**: Функция **verify\_folder\_integrity(folder\_path)** проверяет целостность файлов в указанной папке. Для этого она просматривает все файлы в указанной папке и её подпапках. Для каждого файла она вычисляет его хеш SHA-256 и сравнивает его с предыдущим сохраненным хешем (если таковой имеется). Результаты проверки сохраняются в списке **integrity\_results**.
3. **Выбор папки для проверки целостности**: Функция **choose\_folder()** открывает диалоговое окно проводника для выбора папки, содержащей файлы, которые нужно проверить на целостность. После выбора папки она вызывает функцию **verify\_folder\_integrity(folder\_path)** для выполнения проверки и вывода результатов.
4. **Отображение результатов в графическом окне**: Функция **display\_result(result)** создает графическое окно с текстовым полем, в котором отображаются результаты проверки целостности файлов. После отображения результатов пользователь может закрыть окно с помощью кнопки "OK".

Начало формы

**Результаты выполнения программы:**

Поскольку алгоритм хеширования SHA-256 применяется к байтам, составляющим содержимое файла, программа вычисляет хеш для любого типа файла: текстовых, изображений, аудио, видео и так далее.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 1 – Создание файлов.

После обработки файлов появляется консольное окно с результатом - хешем для каждого файла, после чего хеш сохраняется в json файл.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 2 – Вычисление хеша.

При повторном запуске - значения хеша для файлов уже сохранены. Программа сравнивает эти значения, они совпадают - целостность подтверждена.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 3 – Проверка хеша.

Изменим содержимое файлов - обрежем картинку, отредактируем текст.

Можно заметить, что значения хеша измененных файлов не совпадают с сохраненными - целостность не подтверждена.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 4 – Проверка хеша после изменения.

import os  
import json  
import tkinter as tk  
from tkinter import filedialog  
  
def calculate\_sha256(data):  
 *"""Вычисляет SHA-256 хеш."""* # Инициализация начальных значений хеша  
 h0 = 0x6a09e667  
 h1 = 0xbb67ae85  
 h2 = 0x3c6ef372  
 h3 = 0xa54ff53a  
 h4 = 0x510e527f  
 h5 = 0x9b05688c  
 h6 = 0x1f83d9ab  
 h7 = 0x5be0cd19  
  
 # Определение констант K  
 K = [  
 0x428a2f98, 0x71374491, 0xb5c0fbcf, 0xe9b5dba5,  
 0x3956c25b, 0x59f111f1, 0x923f82a4, 0xab1c5ed5,  
 0xd807aa98, 0x12835b01, 0x243185be, 0x550c7dc3,  
 0x72be5d74, 0x80deb1fe, 0x9bdc06a7, 0xc19bf174,  
 0xe49b69c1, 0xefbe4786, 0x0fc19dc6, 0x240ca1cc,  
 0x2de92c6f, 0x4a7484aa, 0x5cb0a9dc, 0x76f988da,  
 0x983e5152, 0xa831c66d, 0xb00327c8, 0xbf597fc7,  
 0xc6e00bf3, 0xd5a79147, 0x06ca6351, 0x14292967,  
 0x27b70a85, 0x2e1b2138, 0x4d2c6dfc, 0x53380d13,  
 0x650a7354, 0x766a0abb, 0x81c2c92e, 0x92722c85,  
 0xa2bfe8a1, 0xa81a664b, 0xc24b8b70, 0xc76c51a3,  
 0xd192e819, 0xd6990624, 0xf40e3585, 0x106aa070,  
 0x19a4c116, 0x1e376c08, 0x2748774c, 0x34b0bcb5,  
 0x391c0cb3, 0x4ed8aa4a, 0x5b9cca4f, 0x682e6ff3,  
 0x748f82ee, 0x78a5636f, 0x84c87814, 0x8cc70208,  
 0x90befffa, 0xa4506ceb, 0xbef9a3f7, 0xc67178f2  
 ]  
  
 # Предварительная обработка данных(переводим в байты)  
 bit\_len = len(data) \* 8  
 #добавить 1  
 data += b'\x80'  
 #пока не 448  
 while len(data) % 64 != 56:  
 #добавляем 0  
 data += b'\x00'  
 #добавляется длина слова в битах - 8 байт  
 data += bit\_len.to\_bytes(8, 'big')  
  
 # Разбивка данных на блоки - по 64 байта  
 blocks = [data[i:i+64] for i in range(0, len(data), 64)]  
  
 # Цикл хеширования для каждого блока  
 for block in blocks:  
 #разбиваем на слова по 4 байта  
 words = [int.from\_bytes(block[i:i+4], 'big') for i in range(0, 64, 4)]  
  
 # Расширение 16-ти 32-битных слов до 64-ти 32-битных слов  
 for i in range(16, 64):  
 s0 = (rotate\_right(words[i-15], 7) ^  
 rotate\_right(words[i-15], 18) ^  
 (words[i-15] >> 3))  
 s1 = (rotate\_right(words[i-2], 17) ^  
 rotate\_right(words[i-2], 19) ^  
 (words[i-2] >> 10))  
 words.append((words[i-16] + s0 + words[i-7] + s1) & 0xFFFFFFFF)  
  
 # Инициализация рабочих переменных  
 a = h0  
 b = h1  
 c = h2  
 d = h3  
 e = h4  
 f = h5  
 g = h6  
 h = h7  
  
 # Основной цикл  
 for i in range(64):  
 S1 = (rotate\_right(e, 6) ^  
 rotate\_right(e, 11) ^  
 rotate\_right(e, 25))  
 ch = (e & f) ^ (~e & g)  
 temp1 = (h + S1 + ch + K[i] + words[i]) & 0xFFFFFFFF  
 S0 = (rotate\_right(a, 2) ^  
 rotate\_right(a, 13) ^  
 rotate\_right(a, 22))  
 maj = (a & b) ^ (a & c) ^ (b & c)  
 temp2 = (S0 + maj) & 0xFFFFFFFF  
  
 h = g  
 g = f  
 f = e  
 e = (d + temp1) & 0xFFFFFFFF  
 d = c  
 c = b  
 b = a  
 a = (temp1 + temp2) & 0xFFFFFFFF  
  
 # Обновление значения хеша  
 h0 = (h0 + a) & 0xFFFFFFFF  
 h1 = (h1 + b) & 0xFFFFFFFF  
 h2 = (h2 + c) & 0xFFFFFFFF  
 h3 = (h3 + d) & 0xFFFFFFFF  
 h4 = (h4 + e) & 0xFFFFFFFF  
 h5 = (h5 + f) & 0xFFFFFFFF  
 h6 = (h6 + g) & 0xFFFFFFFF  
 h7 = (h7 + h) & 0xFFFFFFFF  
  
 # Возвращаем хеш в виде строки 16теричной по 8 сим на хеш  
 return '{:08x}{:08x}{:08x}{:08x}{:08x}{:08x}{:08x}{:08x}'.format(h0, h1, h2, h3, h4, h5, h6, h7)  
  
def rotate\_right(n, b):  
 *"""Побитовый сдвиг вправо."""* return ((n >> b) | (n << (32 - b))) & 0xFFFFFFFF  
  
  
def verify\_folder\_integrity(folder\_path):  
 *"""Проверяет целостность файлов в папке."""* integrity\_results = []  
  
 # Инициализация пустого словаря для хранения сохраненных хешей  
 saved\_hashes = {}  
  
 # Имя файла, в который будут сохраняться хеши  
 hash\_file = "hashes.json"  
  
 # Загружаем сохраненные хэши (если есть)  
 if os.path.exists(hash\_file):  
 with open(hash\_file, "r") as f:  
 saved\_hashes = json.load(f)  
  
 # Проходим по всем файлам в указанной папке и ее подпапках  
 for root, \_, files in os.walk(folder\_path):  
 for file\_name in files:  
 file\_path = os.path.join(root, file\_name)  
  
 # Открываем файл в бинарном режиме и читаем его данные(в виде байт)  
 with open(file\_path, "rb") as file:  
 file\_data = file.read()  
  
 # Вычисляем ожидаемый хеш для файла  
 expected\_hash = calculate\_sha256(file\_data)  
  
 # Получаем сохраненный хеш для файла из словаря  
 saved\_hash = saved\_hashes.get(file\_path)  
  
 # Проверяем соответствие хешей  
 if saved\_hash:  
 if saved\_hash == expected\_hash:  
 integrity\_results.append(f"Файл: {file\_path} | Целостность подтверждена. Хэш совпадает с сохраненным значением.")  
 else:  
 integrity\_results.append(f"Файл: {file\_path} | Проверка целостности не пройдена. Хэш не совпадает с сохраненным значением.")  
 else:  
 integrity\_results.append(f"Файл: {file\_path} | Новый файл. Вычисленный хэш: {expected\_hash}")  
  
 # Сохраняем хеш файла в словарь  
 saved\_hashes[file\_path] = expected\_hash  
  
 # Сохраняем обновленные хеши в файл JSON  
 with open(hash\_file, "w") as f:  
 json.dump(saved\_hashes, f, indent=4)  
  
 return integrity\_results  
  
  
def choose\_folder():  
 *"""Открывает диалоговое окно проводника для выбора папки."""* root = tk.Tk()  
 root.withdraw() # Скрыть главное окно tkinter  
  
 folder\_path = filedialog.askdirectory() # Открываем диалоговое окно для выбора папки  
 if folder\_path:  
 result = verify\_folder\_integrity(folder\_path)  
 print(result)  
 display\_result(result)  
 root.mainloop()  
  
def display\_result(result):  
 *"""Отображает результат в графическом окне."""* result\_window = tk.Toplevel()  
 result\_window.title("Результат проверки целостности файлов")  
  
 # Создаем текстовое поле для отображения результата  
 result\_text = tk.Text(result\_window, wrap=tk.WORD)  
 result\_text.insert(tk.END, "\n".join(result))  
 result\_text.config(state=tk.DISABLED) # Запрещаем редактирование текста  
 result\_text.pack(fill=tk.BOTH, expand=True)  
  
 # Кнопка для закрытия окна и завершения программы  
 ok\_button = tk.Button(result\_window, text="OK", command=lambda: [result\_window.destroy(), result\_window.quit()])  
 ok\_button.pack()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 choose\_folder()