Министерство образования Республики Беларусь

Учреждение образования

«Брестский Государственный технический университет»

Кафедра ИИТ

Лабораторная работа №2

По дисциплине «КМЗИ»

Тема: “Криптографические хэш-функции”

Выполнил:

Студент 3 курса

Группы ИИ-21

Романко Н.А.

Проверила:

Хацкевич А.С.

Брест 2023

Цель:Изучить существующие алгоритмы вычисления дайджестов сообщений и написать программу, реализующую заданный алгоритм хэширования.

1. Методы: RIPEMD-256

Реализовать алгоритм хэширования сообщения. Сообщение подается через текстовый файл

1. Исследовать лавинный эффект для заданной хэш-функции

Код программы:

import matplotlib.pyplot as plt

import numpy as np

avalanche\_effect = []

#-----------CONSTANTS---------------

#номера выбираемых из сообщения 32-битных слов

R1 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,\

      7, 4, 13, 1, 10, 6, 15, 3, 12, 0, 9, 5, 2, 14, 11, 8, \

      3, 10, 14, 4, 9, 15, 8, 1, 2, 7, 0, 6, 13, 11, 5, 12, \

      1, 9, 11, 10, 0, 8, 12, 4, 13, 3, 7, 15, 14, 5, 6, 2, \

      4, 0, 5, 9, 7, 12, 2, 10, 14, 1, 3, 8, 11, 6, 15, 13 ]

R2 = [5, 14, 7, 0, 9, 2, 11, 4, 13, 6, 15, 8, 1, 10, 3, 12, \

      6, 11, 3, 7, 0, 13, 5, 10, 14, 15, 8, 12, 4, 9, 1, 2, \

      15, 5, 1, 3, 7, 14, 6, 9, 11, 8, 12, 2, 10, 0, 4, 13, \

      8, 6, 4, 1, 3, 11, 15, 0, 5, 12, 2, 13, 9, 7, 10, 14, \

      12, 15, 10, 4, 1, 5, 8, 7, 6, 2, 13, 14, 0, 3, 9, 11]

#количество бит, на которое будут осуществляться сдвиги

S1 = [11, 14, 15, 12,  5,  8,  7,  9, 11, 13, 14, 15,  6,  7,  9,  8, \

      7, 6, 8, 13, 11, 9, 7, 15, 7, 12, 15, 9, 11, 7, 13, 12, \

      11, 13, 6, 7, 14, 9, 13, 15, 14, 8, 13, 6, 5, 12, 7, 5, \

      11, 12, 14, 15, 14, 15, 9, 8, 9, 14, 5, 6, 8, 6, 5, 12, \

      9, 15, 5, 11, 6, 8, 13, 12, 5, 12, 13, 14, 11, 8, 5, 6]

S2 = [8, 9, 9, 11, 13, 15, 15, 5, 7, 7, 8, 11, 14, 14, 12, 6, \

      9, 13, 15, 7, 12, 8, 9, 11, 7, 7, 12, 7, 6, 15, 13, 11, \

      9, 7, 15, 11, 8, 6, 6, 14, 12, 13, 5, 14, 13, 13, 7, 5, \

      15, 5, 8, 11, 14, 14, 6, 14, 6, 9, 12, 9, 12, 5, 15, 8, \

      8, 5, 12, 9, 12, 5, 14, 6, 8, 13, 6, 5, 15, 13, 11, 11]

#начальное значение хэш-функции

h0 = 0x67452301

h1 = 0xEFCDAB89

h2 = 0x98BADCFE

h3 = 0x10325476

h4 = 0xC3D2E1F0

h5 = 0x76543210

h6 = 0xFEDCBA98

h7 = 0x89ABCDEF

h8 = 0x01234567

h9 = 0x3C2D1E0F

#-----------CONSTANTS---------------

def f(j, x, y, z):

    #print(type(x))

    # x\_str = hex(x)[2:]

    # y\_str = hex(y)[2:]

    # z\_str = hex(z)[2:]

    int\_x = int(str(x), 16) & 0xFFFFFFFF

    int\_y = int(str(y), 16) & 0xFFFFFFFF

    int\_z = int(str(z), 16) & 0xFFFFFFFF

    if 0 <= j <= 15:

        result = int\_x ^ int\_y ^ int\_z

        if result <= 0:

            binary\_representation = bin(result & 0xFFFFFFFF)

            binary\_representation = binary\_representation[2:].zfill(32)

            hex\_representation = ""

            for i in range(0, 32, 4):

                hex\_digit = hex(int(binary\_representation[i:i+4], 2))[2:]

                hex\_representation += hex\_digit

            return hex\_representation

        #print(hex(result)[2:].upper())

        return hex(result)[2:].upper()

    elif 16 <= j <= 31:

        result = (int\_x & int\_y) | (~int\_x & int\_z)

        if result <= 0:

            binary\_representation = bin(result & 0xFFFFFFFF)

            binary\_representation = binary\_representation[2:].zfill(32)

            hex\_representation = ""

            for i in range(0, 32, 4):

                hex\_digit = hex(int(binary\_representation[i:i+4], 2))[2:]

                hex\_representation += hex\_digit

            return hex\_representation

        #print(hex(result)[2:].upper())

        return hex(result)[2:].upper()

    elif 32 <= j <= 47:

        result = (int\_x | ~int\_y) ^ int\_z

        if result <= 0:

            binary\_representation = bin(result & 0xFFFFFFFF)

            binary\_representation = binary\_representation[2:].zfill(32)

            hex\_representation = ""

            for i in range(0, 32, 4):

                hex\_digit = hex(int(binary\_representation[i:i+4], 2))[2:]

                hex\_representation += hex\_digit

            return hex\_representation

        #print(result)

        #print(hex(result)[2:].upper())

        return hex(result)[2:].upper()

    elif 48 <= j <= 63:

        result = (int\_x & int\_z) | (int\_y & ~int\_z)

        if result <= 0:

            binary\_representation = bin(result & 0xFFFFFFFF)

            binary\_representation = binary\_representation[2:].zfill(32)

            hex\_representation = ""

            for i in range(0, 32, 4):

                hex\_digit = hex(int(binary\_representation[i:i+4], 2))[2:]

                hex\_representation += hex\_digit

            return hex\_representation

        #print(hex(result)[2:].upper())

        return hex(result)[2:].upper()

    elif 64 <= j <= 79:

        result = int\_x ^ (int\_y | ~int\_z)

        if result <= 0:

            binary\_representation = bin(result & 0xFFFFFFFF)

            binary\_representation = binary\_representation[2:].zfill(32)

            hex\_representation = ""

            for i in range(0, 32, 4):

                hex\_digit = hex(int(binary\_representation[i:i+4], 2))[2:]

                hex\_representation += hex\_digit

            return hex\_representation

        #print(hex(result)[2:].upper())

        return hex(result)[2:].upper()

def K1(j):

    if 0 <= j <= 15:

        return 0x00000000

    elif 16 <= j <= 31:

        return 0x5A827999

    elif 32 <= j <= 47:

        return 0x6ED9EBA1

    elif 48 <= j <= 63:

        return 0x8F1BBCDC

    elif 64 <= j <= 79:

        return 0xA953FD4E

def K2(j):

    if 0 <= j <= 15:

        return 0x50A28BE6

    elif 16 <= j <= 31:

        return 0x5C4DD124

    elif 32 <= j <= 47:

        return 0x6D703EF3

    elif 48 <= j <= 63:

        return 0x7A6D76E9

    elif 64 <= j <= 79:

        return 0x00000000

def change\_bit(word, position):

    if position < 0 or position >= len(word):

        raise ValueError("Позиция выходит за пределы слова")

    word\_list = list(word)

    current\_bit = word\_list[position]

    new\_bit = "0" if current\_bit == "1" else "1"

    word\_list[position] = new\_bit

    new\_word = ''.join(word\_list)

    return new\_word

def count\_diff(word1, word2):

    if len(word1) != len(word2):

        raise ValueError("Строки должны иметь одинаковую длину")

    # Инициализируем счетчик отличающихся букв

    count = 0

    # Проходим по символам в обеих строках и сравниваем их

    for i in range(len(word1)):

        if word1[i] != word2[i]:

            count += 1

    return count

def encode\_string(string):

    print("Полученная строка: ", string)

    byte\_string = string.encode('utf-8')

    string\_length\_bits = len(byte\_string) \* 8

    padding\_length = (448 - (string\_length\_bits + 1) % 512) % 512  # Длина дополнения

    padded\_string = byte\_string + bytes([0x80])   # Добавляем "1"

    padded\_string += bytes([0] \* (padding\_length // 8))      # Добавляем нули

    length\_bytes = string\_length\_bits.to\_bytes(8, 'big')      #Добавляем младшие 64 бита длины сообщения

    padded\_string += length\_bytes

    block\_size = 64

    blocks = [padded\_string[i:i+block\_size] for i in range(0, len(padded\_string), block\_size)]

    #Разделим каждый 512-битовый блок на 16 32-битных слов

    formatted\_blocks = []

    for block in blocks:

        words = []

        for i in range(0, len(block), 4):

            word\_bytes = block[i:i+4]  # Инвертируем порядок байтов

            word = int.from\_bytes(word\_bytes, 'big')

            words.append(word)

        formatted\_blocks.append(words)

    return formatted\_blocks, len(blocks)

with open('kmzi\LAB2\input.txt', 'r') as file:

    string = file.read()

def calculate\_hash(string):

    global h0,h1,h2,h3,h4,h5,h6,h7

    M, t = encode\_string(string)  #закодирование сообщения

    #вывод всех блоков/слов

    # for i, block in enumerate(M):

    #     print(f"Блок {i}:")

    #     for j, word in enumerate(block):

    #         print(f"Слово {j}: {word:08x}")

    for i in range(t):

        A1 = f"{h0:08x}";   B1 = f"{h1:08x}";   C1 = f"{h2:08x}";   D1 = f"{h3:08x}"

        A2 = f"{h4:08x}";   B2 = f"{h5:08x}";   C2 = f"{h6:08x}";   D2 = f"{h7:08x}"

        for j in range(64):

            print(j)

            T = (((int(str(A1),16) + int(f(j, B1, C1, D1),16) + int(f"{M[i][R1[j]]:08x}",16) + int(f"{K1(j):08x}",16))) << S1[j]) \

                | ((int(str(A1),16) + int(f(j, B1, C1, D1),16) + int(f"{M[i][R1[j]]:08x}",16) + int(f"{K1(j):08x}",16))) >> (32 - S1[j])

            T = T & 0xFFFFFFFF

            A1 = D1;   D1 = C1;   C1 = B1;   B1 = T

            T = (((int(str(A2),16) + int(f(63-j, B2, C2, D2),16) + int(f"{M[i][R2[j]]:08x}",16) + int(f"{K2(j):08x}",16))) << S2[j]) \

                | ((int(str(A2),16) + int(f(63-j, B2, C2, D2),16) + int(f"{M[i][R2[j]]:08x}",16) + int(f"{K2(j):08x}",16))) >> (32 - S2[j])

            T = T & 0xFFFFFFFF

            A2 = D2;   D2 = C2;   C2 = B2;   B2 = T

            if j == 15:

                T = A1; A1 = A2; A2 = T

            elif j == 31:

                T = B1;   B1 = B2;   B2 = T

            elif j == 47:

                T = C1;   C1 = C2;   C2 = T

            elif j == 63:

                T = D1;   D1 = D2;   D2 = T

        h0 = (h0 + A1) & 0xFFFFFFFF;   h1 = (h1 + B1) & 0xFFFFFFFF;   h2 = (h2 + C1) & 0xFFFFFFFF;   h3 = (h3 + D1) & 0xFFFFFFFF

        h4 = (h4 + A1) & 0xFFFFFFFF;   h5 = (h5 + B2) & 0xFFFFFFFF;   h6 = (h6 + C2) & 0xFFFFFFFF;   h7 = (h7 + D2) & 0xFFFFFFFF

    hash = f"{h0:08x}{h1:08x}{h2:08x}{h3:08x}{h4:08x}{h5:08x}{h6:08x}{h7:08x}"

    print("Результат: ", hash)

    #print(f"Результат: {h0:08x}, {h1:08x}, {h2:08x}, {h3:08x},{h4:08x},{h5:08x}, {h6:08x}, {h7:08x}")

    # with open('kmzi\LAB2\output.txt', 'w') as file:

    #     file.write("Результат: ", hash)

    return hash

def lavina(string):

    hashes\_diff = []

    round\_num = []

    original\_hash = calculate\_hash(string)

    position = int(input("Какой бит меняем? "))

    new\_word = change\_bit(string, position)

    rounds = int(input("Сколько раундов? "))

    for round in range(rounds):

        new\_hash = calculate\_hash(new\_word)

        diff = count\_diff(original\_hash, new\_hash)

        hashes\_diff.append(diff)

        round\_num.append(round)

    plt.plot(round\_num, hashes\_diff)

    plt.xlabel('Номер раунда')

    plt.ylabel('Разница в хэше')

    plt.title('График изменения хэша в зависимости от раунда')

    plt.show()

while True:

    choice = input("1-хэш, 2 - лавина: ")

    match choice:

        case '1':

            calculate\_hash(string)

        case '2':

            lavina(string)

        case 'e':

            break

        case \_:

            print("NNNAAAAHHH")

Вывод программы:

Хэширование сообщения:



Лавинный эффект:

Изображение выглядит как текст, снимок экрана, дисплей, Шрифт

Автоматически созданное описание

Вывод: изучил основные алгоритмы хэширования сообщений и исследовал лавинный эффект.