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

Пучинский А.А.

Проверила:

Хацкевич А.С.

Брест 2023

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

1. Методы: SHA-384

Реализовать алгоритм хэширования сообщения. Сообщение подается через текстовый файл

1. Исследовать лавинный эффект для заданной хэш-функции

Код программы:

import tkinter as tk

import hashlib

import matplotlib.pyplot as plt

import random

# Функция для вычисления хэш-функции SHA-384

def calculate\_sha384\_hash(text):

    sha384\_hash = hashlib.sha384(text.encode()).hexdigest()

    return sha384\_hash

# Функция для исследования лавинного эффекта и построения графика

def analyze\_avalanche\_effect(text, rounds):

    original\_hash = calculate\_sha384\_hash(text)

    changed\_hashes = []

    for round in range(rounds):

        position = random.randint(0, len(text) - 1)  # Выбираем случайную позицию для изменения бита

        text = change\_bit(text, position)

        changed\_hash = calculate\_sha384\_hash(text)

        changed\_bits = sum(1 for a, b in zip(original\_hash, changed\_hash) if a != b)

        changed\_hashes.append(changed\_bits)

    plt.plot(range(rounds), changed\_hashes)

    plt.xlabel('Раунды')

    plt.ylabel('Измененные биты в хэше')

    plt.title('Зависимость числа измененных бит от раунда')

    plt.grid(True)

    plt.show()

# Функция для изменения бита в тексте

def change\_bit(text, position):

    text\_list = list(text)

    text\_list[position] = '1' if text\_list[position] == '0' else '0'

    return ''.join(text\_list)

# Создание графического интерфейса с использованием Tkinter

root = tk.Tk()

root.title('SHA-384 Hash and Avalanche Effect Analyzer')

# Фрейм для текстового ввода и меток

input\_frame = tk.Frame(root)

input\_frame.pack(pady=10)

# Поле для ввода текста сообщения

text\_label = tk.Label(input\_frame, text='Текст сообщения:')

text\_label.pack(side='left', padx=5)

text\_entry = tk.Entry(input\_frame, width=40)

text\_entry.pack(side='left')

# Фрейм для кнопок

button\_frame = tk.Frame(root)

button\_frame.pack()

# Функция для вычисления и отображения хэш-функции

def calculate\_and\_display\_hash():

    input\_text = text\_entry.get()

    if input\_text:

        hash\_result = calculate\_sha384\_hash(input\_text)

        hash\_output.config(text=f'SHA-384 хэш: {hash\_result}')

hash\_button = tk.Button(button\_frame, text='Вычислить SHA-384 хэш', command=calculate\_and\_display\_hash)

hash\_button.pack(side='left', padx=5)

# Поле для вывода хеш-функции

hash\_output = tk.Label(root, text='', wraplength=400)

hash\_output.pack()

# Функция для анализа лавинного эффекта

def analyze\_effect():

    input\_text = text\_entry.get()

    rounds = 15  # Количество раундов для анализа

    analyze\_avalanche\_effect(input\_text, rounds)

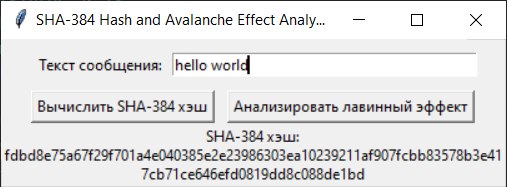
analyze\_button = tk.Button(button\_frame, text='Анализировать лавинный эффект', command=analyze\_effect)

analyze\_button.pack(side='left', padx=5)

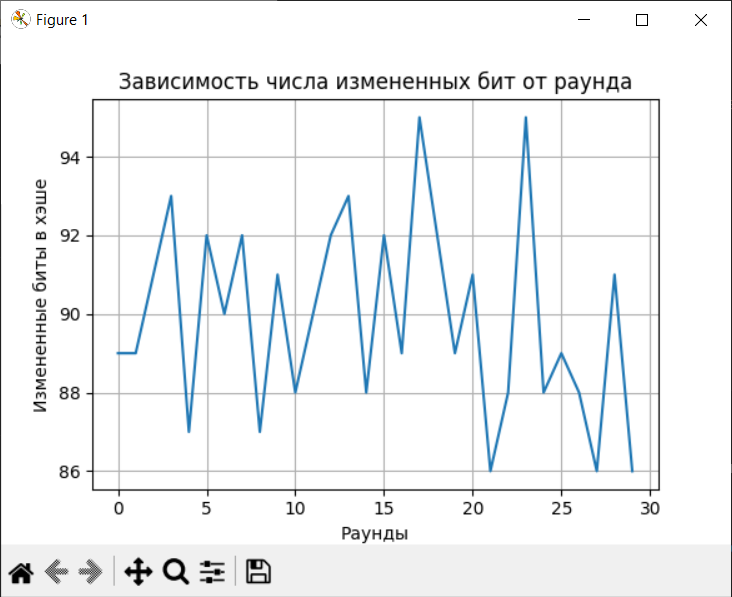
root.mainloop()

Вывод программы:

Хэширование сообщения:



Лавинный эффект:



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