

Міністерство освіти і науки України  
Національний технічний університет «ХПІ»  
Навчально-науковий інститут комп'ютерних наук та інформаційних  
технологій  
Кафедра комп'ютерної інженерії та програмування

## **ЗВІТ**

з лабораторної роботи № 1  
з дисципліни «Сучасні технології безпечного програмування»  
**«ХЕШУВАННЯ»**

Виконав:  
студент гр. КН-Н9226  
Кулик Д.І.

Перевірив:  
Бульба С. С.

Харків – 2022

**Мета роботи:** Дослідити принципи роботи хешування.

### **Індивідуальне завдання**

Дослідити існуючі механізми хешування. Реалізувати алгоритм хешування SHA (будь-якої версії). Реалізацію інших алгоритмів хешування слід обговорити з викладачем.

Довести коректність роботи реалізованого алгоритму шляхом порівняння результатів з існуючими реалізаціями (напр. утилітою sha1sum).

### **Хід роботи**

SHA-2 (Secure Hash Algorithm 2), частиною якого є SHA-256, є одним із найпопулярніших алгоритмів хешування. Криптографічний хеш, який також часто називають «дайджест», «відбиток пальця» або «підпис», — це майже ідеально унікальний рядок символів, який генерується з окремого фрагмента введеного тексту. SHA-256 генерує 256-бітний (32-байтний) підпис.

Три основні цілі хеш-функцій:

- Детерміновано шифрувати дані (такий вид шифрування завжди створює одне й те саме зашифроване значення для того самого текстового значення);
- Приймати введення будь-якої довжини, а виводити результат фіксованої довжини;
- Змінювати дані не можна. Введення не можна отримати з висновку.

**КРОКИ:**

1. Попередня робота
2. Ініціалізація значення хеша (h)
3. Ініціалізація округлених констант (k)
4. Цикл фрагментів
5. Створення розкладу повідомлень (w)
6. Стиснення
7. Зміна остаточних значень
8. Фінальний хеш

## Важливі фрагменти програми

```
# Заповнювання
length = len(message) * 8 # len(message) - це кількість байтів

message.append(0x80)

# перетворене повідомлення заповнити нулями доки дані не стануть кратними 512 без останніх 64 біт
while (len(message) * 8 + 64) % 512 != 0:
    message.append(0x00)

message += length.to_bytes(8, 'big') # доповнення до 8 байтів або 64 бітів

assert (len(message) * 8) % 512 == 0, "Padding did not complete properly!"
```

Рисунок 1 – Заповнення

---

```
# Prepare message schedule
message_schedule = []
for t in range(0, 64):
    if t <= 15:
        # adds the t'th 32 bit word of the block,
        # starting from leftmost word
        # 4 bytes at a time
        message_schedule.append(bytes(message_block[t*4:(t*4)+4]))
        # print(message_schedule)
    else:
        term1 = _sigma1(int.from_bytes(message_schedule[t-2], 'big'))
        term2 = int.from_bytes(message_schedule[t-7], 'big')
        term3 = _sigma0(int.from_bytes(message_schedule[t-15], 'big'))
        term4 = int.from_bytes(message_schedule[t-16], 'big')

        # append a 4-byte byte object
        schedule = ((term1 + term2 + term3 + term4) %
                     2**32).to_bytes(4, 'big')
        message_schedule.append(schedule)

assert len(message_schedule) == 64
```

Рисунок 2 – Створення розкладу повідомлень

```

# Initialize working variables
a = h0
b = h1
c = h2
d = h3
e = h4
f = h5
g = h6
h = h7

# Iterate for t=0 to 63
for t in range(64):
    t1 = ((h + _capsigma1(e) + _ch(e, f, g) + K[t] +
           int.from_bytes(message_schedule[t], 'big'))) % 2**32

    t2 = (_capsigma0(a) + _maj(a, b, c)) % 2**32

    h = g
    g = f
    f = e
    e = (d + t1) % 2**32
    d = c
    c = b
    b = a
    a = (t1 + t2) % 2**32

```

**Рисунок 3 - Стиснення**

```

# Compute intermediate hash value
h0 = (h0 + a) % 2**32
h1 = (h1 + b) % 2**32
h2 = (h2 + c) % 2**32
h3 = (h3 + d) % 2**32
h4 = (h4 + e) % 2**32
h5 = (h5 + f) % 2**32
h6 = (h6 + g) % 2**32
h7 = (h7 + h) % 2**32

```

**Рисунок 4 – Зміна значень хешу**

## Результати роботи програми

```

C:\Users\Daniil\PycharmProjects\stbp\Scripts\python.exe C:/Users/Daniil/PycharmProjects/stbp/LABS/kulyk01/main.py
Введений текст: Daniil Kulyk
Розроблений SHA256 = 88d44f1bf7c5886b9f0eb28963037d7d9fff9e99a7c24a869124fede16bc46b8
HashLib SHA256    = 88d44f1bf7c5886b9f0eb28963037d7d9fff9e99a7c24a869124fede16bc46b8

```

**Рисунок 5 – Результат виконання програми**

# SHA256

SHA256 online hash function

Daniil Kulyk

Input type Text ▾

Hash

☒ Auto Update

88d44f1bf7c5886b9f0eb28963037d7d9fff9e99a7c24a869124fede16bc46b8

**Рисунок 6** – Результат хешування за допомогою ресурсу  
<https://emn178.github.io/online-tools/sha256.html>

При порівнянні можемо побачити що результат виконання реалізацій алгоритму однаковий в усіх випадках.

**Висновки:** в результаті виконання лабораторної роботи було досліджено принципи роботи хешування. В результаті порівняння власної реалізації алгоритму з вже реалізованими була виявлена ідентичність роботи, що доводить коректність першого.