МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

******

Лабораторна робота №2

З дисципліни

“Технології захисту інформації”

*Виконав :*

*Студент групи КН-314*

*Ляшеник Остап*

*Прийняв:*

*Яковина В.С.*

*Львів - 2023*

**Тема**: СТВОРЕННЯ ПРОГРАМНОГО ЗАСОБУ ДЛЯ ЗАБЕЗПЕЧЕННЯ ЦІЛІСНОСТІ ІНФОРМАЦІЇ

**Мета роботи:** ознайомитись з методами криптографічного забезпечення цілісності інформації, навчитись створювати програмні засоби для забезпечення цілісності інформації з використанням алгоритмів хешування.

**Хід роботи**

1. Написання основної функції “md5”

def md5\_padding(message):  
 original\_byte\_len = len(message)  
 message += b'\x80'  
 message += b'\x00' \* ((56 - (original\_byte\_len + 1) % 64) % 64)  
 message += struct.pack('<Q', original\_byte\_len \* 8)  
 return message  
  
  
def md5(message):  
 def left\_rotate(val, r\_bits, max\_bits):  
 return (val << r\_bits % max\_bits) & (2 \*\* max\_bits - 1) | \  
 ((val & (2 \*\* max\_bits - 1)) >> (max\_bits - (r\_bits % max\_bits)))  
  
 # Initialize the chaining variables  
 a0 = 0x67452301  
 b0 = 0xEFCDAB89  
 c0 = 0x98BADCFE  
 d0 = 0x10325476  
  
 # Precompute the sine table  
 T = [int(2 \*\* 32 \* abs(math.sin(i + 1))) & 0xFFFFFFFF for i in range(64)]  
  
 # Pad the message to a multiple of 64 bytes  
 message = md5\_padding(message)  
  
 # Constants used for left rotations  
 s = [7, 12, 17, 22] \* 4 + [5, 9, 14, 20] \* 4 + [4, 11, 16, 23] \* 4 + [6, 10, 15, 21] \* 4  
  
 # Iterate over the message in chunks of 64 bytes  
 for i in range(0, len(message), 64):  
 chunk = message[i:i + 64]  
  
 # Initialize the round variables  
 A, B, C, D = a0, b0, c0, d0  
  
 # Perform the 64 rounds of the MD5 algorithm  
 for j in range(64):  
 if 0 <= j <= 15:  
 F = (B & C) | ((~B) & D)  
 g = j  
 elif 16 <= j <= 31:  
 F = (D & B) | ((~D) & C)  
 g = (5 \* j + 1) % 16  
 elif 32 <= j <= 47:  
 F = B ^ C ^ D  
 g = (3 \* j + 5) % 16  
 elif 48 <= j <= 63:  
 F = C ^ (B | (~D))  
 g = (7 \* j) % 16  
  
 dTemp = D  
 D = C  
 C = B  
 B = (B + left\_rotate((A + F + T[j] + struct.unpack('<I', chunk[4 \* g:4 \* g + 4])[0]), s[j],  
 32)) & 0xFFFFFFFF  
 A = dTemp  
  
 # Update the chaining variables  
 a0 = (a0 + A) & 0xFFFFFFFF  
 b0 = (b0 + B) & 0xFFFFFFFF  
 c0 = (c0 + C) & 0xFFFFFFFF  
 d0 = (d0 + D) & 0xFFFFFFFF  
  
 # Format the result as a 32-character hexadecimal string with the correct byte order  
 result = struct.pack('<I', a0) + struct.pack('<I', b0) + struct.pack('<I', c0) + struct.pack('<I', d0)  
 result\_hex = result.hex()  
 return result\_hex

2. Загальне оформлення роботи

def md5\_padding(message):  
 original\_byte\_len = len(message)  
 message += b'\x80'  
 message += b'\x00' \* ((56 - (original\_byte\_len + 1) % 64) % 64)  
 message += struct.pack('<Q', original\_byte\_len \* 8)  
 return message  
  
  
def md5(message):  
 def left\_rotate(val, r\_bits, max\_bits):  
 return (val << r\_bits % max\_bits) & (2 \*\* max\_bits - 1) | \  
 ((val & (2 \*\* max\_bits - 1)) >> (max\_bits - (r\_bits % max\_bits)))  
  
 # Initialize the chaining variables  
 a0 = 0x67452301  
 b0 = 0xEFCDAB89  
 c0 = 0x98BADCFE  
 d0 = 0x10325476  
  
 # Precompute the sine table  
 T = [int(2 \*\* 32 \* abs(math.sin(i + 1))) & 0xFFFFFFFF for i in range(64)]  
  
 # Pad the message to a multiple of 64 bytes  
 message = md5\_padding(message)  
  
 # Constants used for left rotations  
 s = [7, 12, 17, 22] \* 4 + [5, 9, 14, 20] \* 4 + [4, 11, 16, 23] \* 4 + [6, 10, 15, 21] \* 4  
  
 # Iterate over the message in chunks of 64 bytes  
 for i in range(0, len(message), 64):  
 chunk = message[i:i + 64]  
  
 # Initialize the round variables  
 A, B, C, D = a0, b0, c0, d0  
  
 # Perform the 64 rounds of the MD5 algorithm  
 for j in range(64):  
 if 0 <= j <= 15:  
 F = (B & C) | ((~B) & D)  
 g = j  
 elif 16 <= j <= 31:  
 F = (D & B) | ((~D) & C)  
 g = (5 \* j + 1) % 16  
 elif 32 <= j <= 47:  
 F = B ^ C ^ D  
 g = (3 \* j + 5) % 16  
 elif 48 <= j <= 63:  
 F = C ^ (B | (~D))  
 g = (7 \* j) % 16  
  
 dTemp = D  
 D = C  
 C = B  
 B = (B + left\_rotate((A + F + T[j] + struct.unpack('<I', chunk[4 \* g:4 \* g + 4])[0]), s[j],  
 32)) & 0xFFFFFFFF  
 A = dTemp  
  
 # Update the chaining variables  
 a0 = (a0 + A) & 0xFFFFFFFF  
 b0 = (b0 + B) & 0xFFFFFFFF  
 c0 = (c0 + C) & 0xFFFFFFFF  
 d0 = (d0 + D) & 0xFFFFFFFF  
  
 # Format the result as a 32-character hexadecimal string with the correct byte order  
 result = struct.pack('<I', a0) + struct.pack('<I', b0) + struct.pack('<I', c0) + struct.pack('<I', d0)  
 result\_hex = result.hex()  
 return result\_hex  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 test\_cases = {  
 "": "d41d8cd98f00b204e9800998ecf8427e",  
 "a": "0cc175b9c0f1b6a831c399e269772661",  
 "abc": "900150983cd24fb0d6963f7d28e17f72",  
 "message digest": "f96b697d7cb7938d525a2f31aaf161d0",  
 "abcdefghijklmnopqrstuvwxyz": "c3fcd3d76192e4007dfb496cca67e13b",  
 "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789": "d174ab98d277d9f5a5611c2c9f419d9f",  
 "12345678901234567890123456789012345678901234567890123456789012345678901234567890": "57edf4a22be3c955ac49da2e2107b67a"  
 }  
  
 for test\_input, expected\_hash in test\_cases.items():  
 calculated\_hash = md5(test\_input.encode('utf-8'))  
 print(f"String: '{test\_input}'")  
 print(f"Expected hash: {expected\_hash}")  
 print(f"Calculated hash: {calculated\_hash}")  
 print(f"Result: {'Matches' if calculated\_hash == expected\_hash else 'Does not match'}")

3.Введення можливості хешування файлу та збереження результату у файл

def hash\_file():  
 file\_path = input("Path: ") or "text.txt"  
  
 if not os.path.exists(file\_path):  
 print("File is not existing.")  
 return None  
  
 with open(file\_path, 'rb') as file:  
 file\_data = file.read()  
 print(file\_data)  
 result = md5(file\_data)  
 print(f'File\'s hash "{file\_path}": {result}')  
 return result  
  
  
if input("Do you want to hash file? y/n") == 'y' or ' y':  
 result = hash\_file()  
  
  
if input("Should the result be saved into the another file? y/n") == 'y' or ' y':  
 with open("result.txt", 'w') as file:  
 file.write(result)  
 print("Saved succesfully")

Результат виведення коду

String: ''

Expected hash: d41d8cd98f00b204e9800998ecf8427e

Calculated hash: d41d8cd98f00b204e9800998ecf8427e

Result: Matches

String: 'a'

Expected hash: 0cc175b9c0f1b6a831c399e269772661

Calculated hash: 0cc175b9c0f1b6a831c399e269772661

Result: Matches

String: 'abc'

Expected hash: 900150983cd24fb0d6963f7d28e17f72

Calculated hash: 900150983cd24fb0d6963f7d28e17f72

Result: Matches

String: 'message digest'

Expected hash: f96b697d7cb7938d525a2f31aaf161d0

Calculated hash: f96b697d7cb7938d525a2f31aaf161d0

Result: Matches

String: 'abcdefghijklmnopqrstuvwxyz'

Expected hash: c3fcd3d76192e4007dfb496cca67e13b

Calculated hash: c3fcd3d76192e4007dfb496cca67e13b

Result: Matches

String: 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789'

Expected hash: d174ab98d277d9f5a5611c2c9f419d9f

Calculated hash: d174ab98d277d9f5a5611c2c9f419d9f

Result: Matches

String: '12345678901234567890123456789012345678901234567890123456789012345678901234567890'

Expected hash: 57edf4a22be3c955ac49da2e2107b67a

Calculated hash: 57edf4a22be3c955ac49da2e2107b67a

Result: Matches

Do you want to hash file? y/ny

Path: text.txt

b''

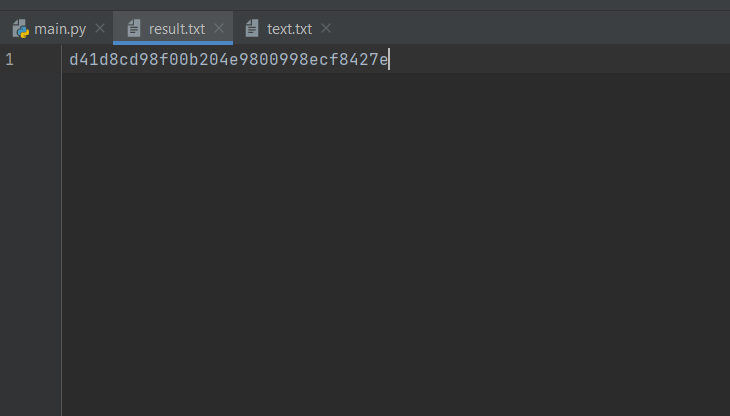
File's hash "text.txt": d41d8cd98f00b204e9800998ecf8427e

Should the result be saved into the another file? y/ny

Saved succesfully

Process finished with exit code 0

Результати записані у файл:

Було зчитано пустий файл 

**Висновок**

Під час лабораторної роботи було створено програму для обчислення MD5 хешу для різних текстових вхідних повідомлень. Програма використовує MD5 алгоритм для генерації 32-символьних шістнадцяткових хеш-кодів для вхідних даних. Також було вирішено помилку в коді, яка призводила до некоректних результатів, та було виправлено порядок байтів у фінальному хеші. Результати тестових вхідних даних були порівняні з очікуваними хешами для перевірки правильності роботи програми.