12.06.2021

Warsaw

ECRYP Project Documentation

Project topic: **MD5 Hash Function implementation in python**

Author: Jan Radzimiński

Index number: 293052

Supervisor: dr hab. inż. Tomasz Adamski

Table of contents:

1. Theoretical Introduction

MD5 Hash Function Overview

MD5 is a message-digest algorithm is used as a hashed function that produces 128-bit value (hash) at the output. A hash function is a function that maps an data of arbitrary size into fixed size output. The main purpose of such functions is to produce totally different output for every input message, independently of its similarities. Hash functions are widely used in many computer science fields, for example:

- Hash tables
- Digital signatures
- Pseudo random number generation

and many more. The input of the hash function is usually called *message* and the output is usually called *digest* or *hash*. The function itself is noted as *h(m)* or *H(m).*

The MD5 algorithm was created by Ronald Rivest in 199. It was supposed to replace older hash function MD4.

One of the main assumptions of any cryptographic hash function is that it should be impossible or very difficult to find two distinct messages that have the same hash output value. MD5 algorithm fails this requirement catastrophically – the collisions can be found easily on any computer.

Security

As mentioned earlier, the MD5 hash function security has already been severely compromised. A collision attack for this function can be successfully run on it by most of the personal computers used by people around the world. There is also a chosen-prefix collision attack that can find collisions within seconds of its runtime.

In March od 2005, there was a demonstration of construction of two X.509 certificates with different public keys and the same MD5 hash value – a proof of MD5 collision.

Generally, the MD5 algorithm weaknesses and deprecation were well-documented by security experts. Nevertheless, it is still widely used for various purposes around the world.

2. Algorithm description

The MD5 hash function takes an arbitrary length data as an input and produces exactly 128bit output. The output hash is computed according to following algorithm:

1. Before the data is processed a single “1” bit is added at the end of the input
2. Then, the input is filled with “0” bits up to the point that the length of the message is exactly 64 bit fewer than the next multiple of 512
3. Next, the initial length modulo 2^{64} needs to be calculated and transform into form of 64 bits, then added to the end of the message. At that point, the length of a message is a multiple of 512
4. Then, the input message is split into fixed size 512 bit blocks that are processed one after the other
5. Each such block is split into sixteen 32-bit words. Lets call these blocks *M[i],* where *0 <= i < 16.*
6. Then, there are 4 stages, where everyone is in every block iteration. Each stage is composed of 16 similar operations based on a nonlinear function F, modular addition and left rotation. Following figure illustrates one operation within a stage:

$$F(B,C,D) = (B \wedge C) \vee (\neg B \wedge D)$$
$$G(B,C,D) = (B \wedge D) \vee (C \wedge \neg D)$$
$$H(B,C,D) = B \oplus C \oplus D$$
$$I(B,C,D) = C \oplus (B \vee \neg D)$$

$\oplus, \wedge, \vee, \neg$ denote the XOR, AND, OR and NOT operations respectively.

Figure 1 (https://en.wikipedia.org/wiki/MD5)

7. Full operation of algorithm is presented on the following figure

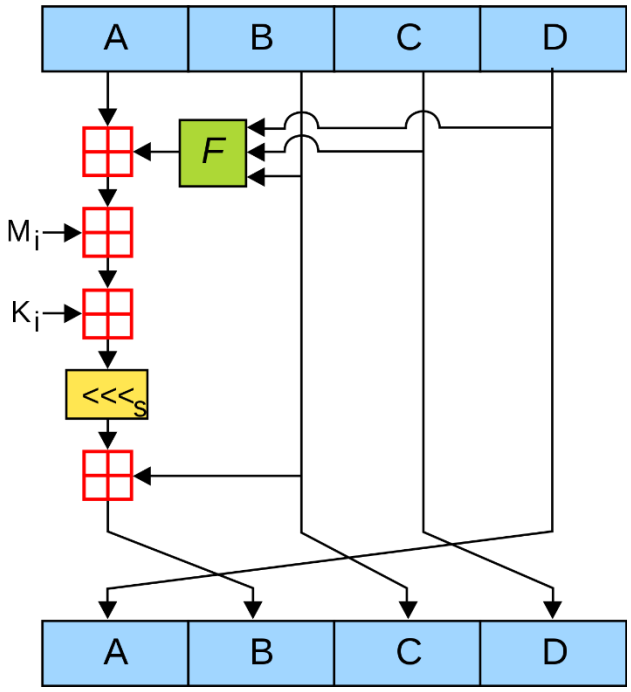

Figure 2 (https://en.wikipedia.org/wiki/MD5)

8. Additionally, after each block, for calculation of F (as well as A, B, C, D) the sine values (in Radians) are used. Also for each stage the custom number of round is defined. These variables may be precomputed before the operation of algorithm (since they are constant).

3. Python Implementation

Note: *All of the source files used in the project are attached to this document and can be browsed independently. For running the script, please use "python main.py" or "python3 main.py" command in the project folder.*

Overview

In my implementation, I used different functions for different parts of the MD5 function. All of the most important functions connected to the MD4 algorithm are placed in the "md5.py" file. The "constants.py" file consist all of the constant values used across the scripts. The "utils.py" consist of helper functions used for proper conversion and display of the values. The "test.py" function consist of test function, which reads the input message from file, hashes it using my algorithm and the MD5 algorithm from hashlib and compares the result. Finally, "main.py" run the test function for each of the input file.

Python functions

As described before, the input message must be firstly pre-processed (padded). I implemented this padding in following python function:

```
73  def md5_message_padding(bytes_obj: bytes):
74      bytes_arr = bytearray(bytes_obj)
75      init_message_length_bits = len_in_bits(bytes_obj)
76
77      # Appending 1 bit at the end of the message (with 7 additional 0 to fill the byte)
78      bytes_arr.append(SINGLE_1_BIT_WITH_ZEROS)
79
80      # Filling bits with 0 till the message has only 64 bits left till be a multiple of 512
81      while (len(bytes_arr) % (MESSAGE_CHUNK_LENGTH / BITS_IN_BYTE) != ((MESSAGE_CHUNK_LENGTH - REMAINING_BITS_NUM) / BITS_IN_BYTE)):
82          bytes_arr.append(EMPTY_BYTE)
83
84      # Adding the length of input mod 2^64 at the end (as 64 bits)
85      remaining_bits = init_message_length_bits % (2 ** REMAINING_BITS_NUM)
86      remaining_bits = remaining_bits.to_bytes(
87          int(REMAINING_BITS_NUM / BITS_IN_BYTE), byteorder=DEFAULT_BYTE_ORDER)
88
89      bytes_arr += remaining_bits
90
91      return bytes_arr
```

The main md5 function uses also several helper functions (following DRY and single-responsibility programming principle):

```python
def break_into_chunks(byte_arr: bytearray, chunk_size_bits: int = 512):
    chunks = []
    chunk_size_in_bytes = int(chunk_size_bits / BITS_IN_BYTE)

    for i in range(0, int(len(byte_arr) / chunk_size_in_bytes), 1):
        chunks.append(byte_arr[(chunk_size_in_bytes * i):(chunk_size_in_bytes * (i + 1))])

    return chunks


def xor(a, b):
    return (a & ~b) | (~a & b)


def rotate_left(x, amount):
    x &= 0xFFFFFFFF
    return ((x << amount) | (x >> (32 - amount)))

```

Finally, the main md5 hashing function:

```python
def hash(message: bytes):
    padded_message = md5_message_padding(message)
    message_chunks = break_into_chunks(padded_message, MESSAGE_CHUNK_LENGTH)

    initial_values = [a0, b0, c0, d0]
    values = initial_values

    for chunk in message_chunks:
        # 16 - 32bis words of 512 bit chunk
        message_words = break_into_chunks(chunk, 32)

        a, b, c, d = values

        for i in range(0, 64, 1):
            # 1st cycle
            if i <= 15:

                f = (b & c) | (~b & d)
                g = i
            # 2nd cycle
            elif 16 <= i <= 31:

                f = (d & b) | ((~d) & c)
                g = (5 * i + 1) % 16
            # 3rd cycle
            elif 32 <= i <= 47:

                f = xor(xor(b, c), d)
                g = (3 * i + 5) % 16
            # 2th cycle
            else:

                f = xor(c, (b | (~d)))
                g = (7 * i) % 16

            f += a + K[i] + \
                int.from_bytes(message_words[g], byteorder=DEFAULT_BYTE_ORDER)

            a, d, c = d, c, b
            b = b+rotate_left(f, S[i])
            b &= CHUNK_WITH_1s_512

        for i, val in enumerate([a, b, c, d]):
            values[i] += val
            values[i] &= CHUNK_WITH_1s_512

    values_enum = enumerate(values)
    values_sum = sum([val << (32 * i) for i, val in values_enum])

    return values_sum.to_bytes(16, byteorder=DEFAULT_BYTE_ORDER)
```

And the constants used in it:

```python
# Binary integer part of the sines of integers (Radians)
# Source: https://en.wikipedia.org/wiki/MD5
SINE_BYTES = [0xd76aa478, 0xe8c7b756, 0x242070db, 0xc1bdceee,
              0xf57c0faf, 0x4787c62a, 0xa8304613, 0xfd469501,
              0x698098d8, 0x8b44f7af, 0xffff5bb1, 0x895cd7be,
              0x6b901122, 0xfd987193, 0xa679438e, 0x49b40821,
              0xf61e2562, 0xc040b340, 0x265e5a51, 0xe9b6c7aa,
              0xd62f105d, 0x02441453, 0xd8a1e681, 0xe7d3fbc8,
              0x21e1cde6, 0xc33707d6, 0xf4d50d87, 0x455a14ed,
              0xa9e3e905, 0xfcefa3f8, 0x676f02d9, 0x8d2a4c8a,
              0xfffa3942, 0x8771f681, 0x6d9d6122, 0xfde5380c,
              0xa4beea44, 0x4bdecfa9, 0xf6bb4b60, 0xbebfbc70,
              0x289b7ec6, 0xeaa127fa, 0xd4ef3085, 0x04881d05,
              0xd9d4d039, 0xe6db99e5, 0x1fa27cf8, 0xc4ac5665,
              0xf4292244, 0x432aff97, 0xab9423a7, 0xfc93a039,
              0x655b59c3, 0x8f0ccc92, 0xffeff47d, 0x85845dd1,
              0x6fa87e4f, 0xfe2ce6e0, 0xa3014314, 0x4e0811a1,
              0xf7537e82, 0xbd3af235, 0x2ad7d2bb, 0xeb86d391]

# Rotate shift amounts for md5
# Source: https://en.wikipedia.org/wiki/MD5
SHIFT_AMOUNTS = [7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22,
                 5,  9, 14, 20, 5,  9, 14, 20, 5,  9, 14, 20, 5,  9, 14, 20,
                 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23,
                 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21]


# MD constants
SINGLE_1_BIT_WITH_ZEROS = 0x80  # 1000 0000
EMPTY_BYTE = 0x00  # 0000 0000
BITS_IN_BYTE = 8
REMAINING_BITS_NUM = 64
MESSAGE_CHUNK_LENGTH = 512
DEFAULT_BYTE_ORDER = 'little'
CHUNK_WITH_1s_512 = 0xFFFFFFFF

```

Function used for algorithm testing:

```python
import hashlib
from utils import file_to_byte_arr, get_hex_bytes_str
import md5
import time


def test_custom_md5_hash_function(message_file_path: str):
    file_to_byte_arr(message_file_path)
    bytes_obj = file_to_byte_arr(message_file_path).read()

    start_time = time.time()
    hashed_message = md5.hash(bytes_obj)
    custom_md5_time = time.time() - start_time

    hashed_message_string = get_hex_bytes_str(hashed_message)

    start_time = time.time()
    hashed_message_by_hashlib = hashlib.md5(bytes_obj).hexdigest()
    hashlib_md5_time = time.time() - start_time

    print(
        f'Custom computed md5 hash for "{message_file_path}" (length in bytes: {len(bytes_obj)}): ', hashed_message_string)
    print(f'Hash computed in: {custom_md5_time}s')
    print(
        f'Md5 hash computed by hashlib for "{message_file_path}": ', hashed_message_by_hashlib)
    print(f'Hash computed in: {hashlib_md5_time}s')
    print('Are hashes identical: ',
          'YES!' if hashed_message_by_hashlib == hashed_message_string else 'No - something does not work.')

```

And finally, the main function used for running the script:

```python
# Custom Md5 Hash function implementation made by Jan Radzimiński
# Index number: 293052
# Md5 algorithm implementation based on algorithm from Wikipedia (https://en.wikipedia.org/wiki/MD5)

from test import test_custom_md5_hash_function

INPUT_FILES = ['test-message-1.txt',
               'test-message-2.txt', 'test-message-3.txt']

print('=====================================================')
print('Md5 Hash function implementation made by Jan Radzimiński')
print('Index number: 293052')
print('=====================================================')
print('')
print('Running custom md5 hash function test with input files: ', INPUT_FILES)
print('')

for file in INPUT_FILES:
    test_custom_md5_hash_function(file)
    print('')

```

4. Testing results, correctness and speed

For the needs of this project, I have created 4 different files consisting 4 different messages for testing the algorithm. The first ones are rather short, while the last ones are quite big (the biggest one is more than 100KiB in size). In that way I can make sure that my implementation works correctly for any input message.

For each file, the test function is run. The test function uses my custom MD5 implementation and compares it with the implementation from *hashlib* – python built-in library consisting of different hash functions. Then the function checks whether both results are the same, and evaluates the time needed for achieving each of the results. The result of the test function for each test file is as follows:

```
====================================================
Md5 Hash function implementation made by Jan Radzimiński
Index number: 293052
====================================================

Running custom md5 hash function test with input files:  ['test-message-1.txt', 'test-message-2.txt', 'test-message-3.txt', 'test-message-4.txt']

Custom computed md5 hash for "test-message-1.txt" (length in bytes: 18):  ac68595ab57815e5a363115c4475ac89
Hash computed in: 0.0s
Md5 hash computed by hashlib for "test-message-1.txt":  ac68595ab57815e5a363115c4475ac89
Hash computed in: 0.0s
Are hashes identical:  YES!

Custom computed md5 hash for "test-message-2.txt" (length in bytes: 857):  47acaecf69acb989a4714702421dae5b
Hash computed in: 0.001001596450805664s
Md5 hash computed by hashlib for "test-message-2.txt":  47acaecf69acb989a4714702421dae5b
Hash computed in: 0.0s
Are hashes identical:  YES!

Custom computed md5 hash for "test-message-3.txt" (length in bytes: 25362):  977ece017b077d4da5072a7fd2c9e6ed
Hash computed in: 0.0320127010345459s
Md5 hash computed by hashlib for "test-message-3.txt":  977ece017b077d4da5072a7fd2c9e6ed
Hash computed in: 0.0s
Are hashes identical:  YES!

Custom computed md5 hash for "test-message-4.txt" (length in bytes: 152172):  e638a6712b120bfaed553726b73ad2d8
Hash computed in: 0.18599557876586914s
Md5 hash computed by hashlib for "test-message-4.txt":  e638a6712b120bfaed553726b73ad2d8
Hash computed in: 0.0s
Are hashes identical:  YES!
```

As you can see, for all of the input files, the output is correct. The performance of the function is also quite good – for the biggest file the hash was calculated in less than a second (using my custom function). However, the hash computed by *hashlib* function was computed in (virtually) no time – so it performs slightly better.

This confirms that my implementation works as intended, with satisfactory performance for any input provided to it.

Bibliography:

1. https://en.wikipedia.org/wiki/MD5
2. https://www.geeksforgeeks.org/md5-hash-python/
3. https://pl.wikipedia.org/wiki/MD5