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
2. Algorithm description
3. Python implementation
4. Testing results, correctness and speed
5. **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.

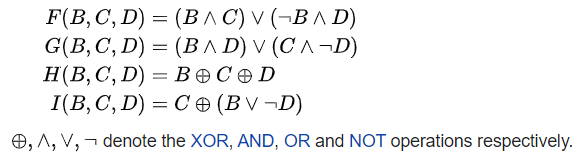
In March od 2005, there was a demonstration of construction of two X.509 certificates with different public keys and the same MD5 hash value – a proof of MD5 collision.

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.

1. **Algorithm description**

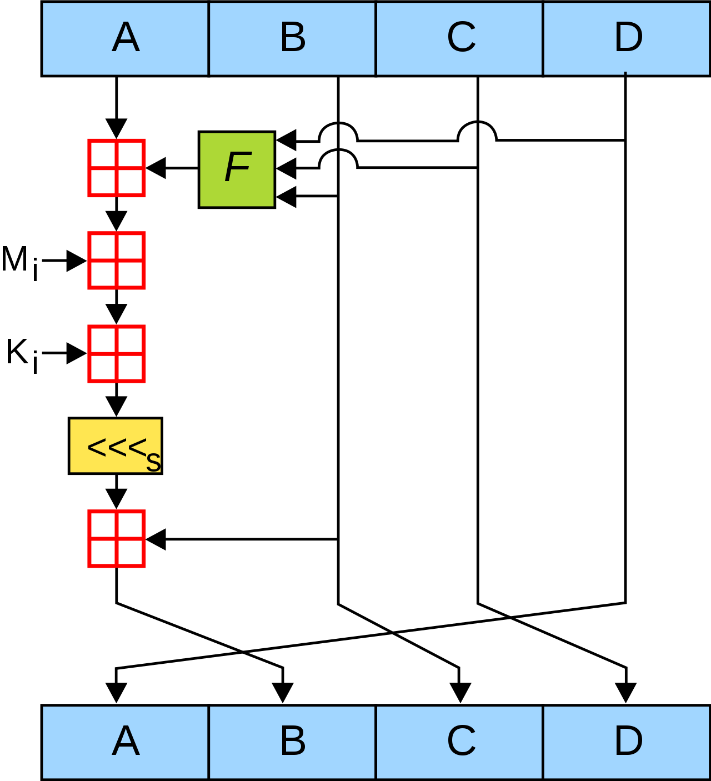
The MD5 hash function takes an arbitrary length data as an input and produces exactly 128bit output. The output hash is computed according to following algorithm:

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 264 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:



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

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



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

1. 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).
2. **Python Implementation**

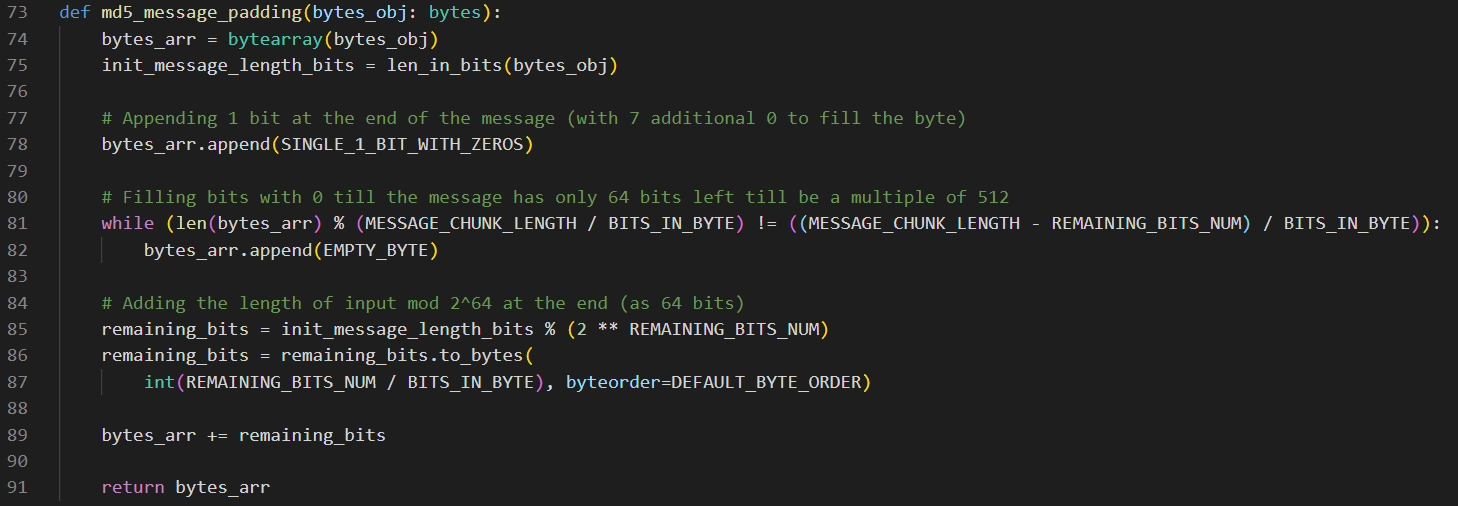
**Note:** *All of the source files used in the project are attached to this document and can be browsed independently. For running the script, please use “python main.py” or “python3 main.py” command in the project folder.*

**Overview**

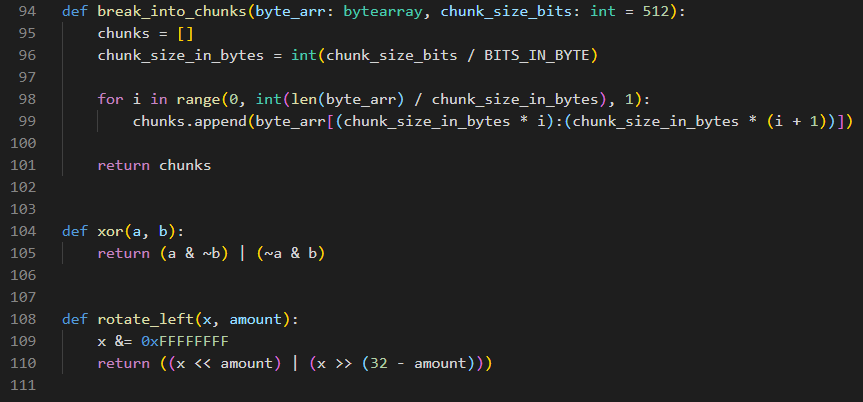
In my implementation, I used different functions for different parts of the MD5 function. All of the most important functions connected to the MD4 algorithm are placed in the “md5.py” file. The “constants.py” file consist all of the constant values used across the scripts. The “utils.py” consist of helper 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**

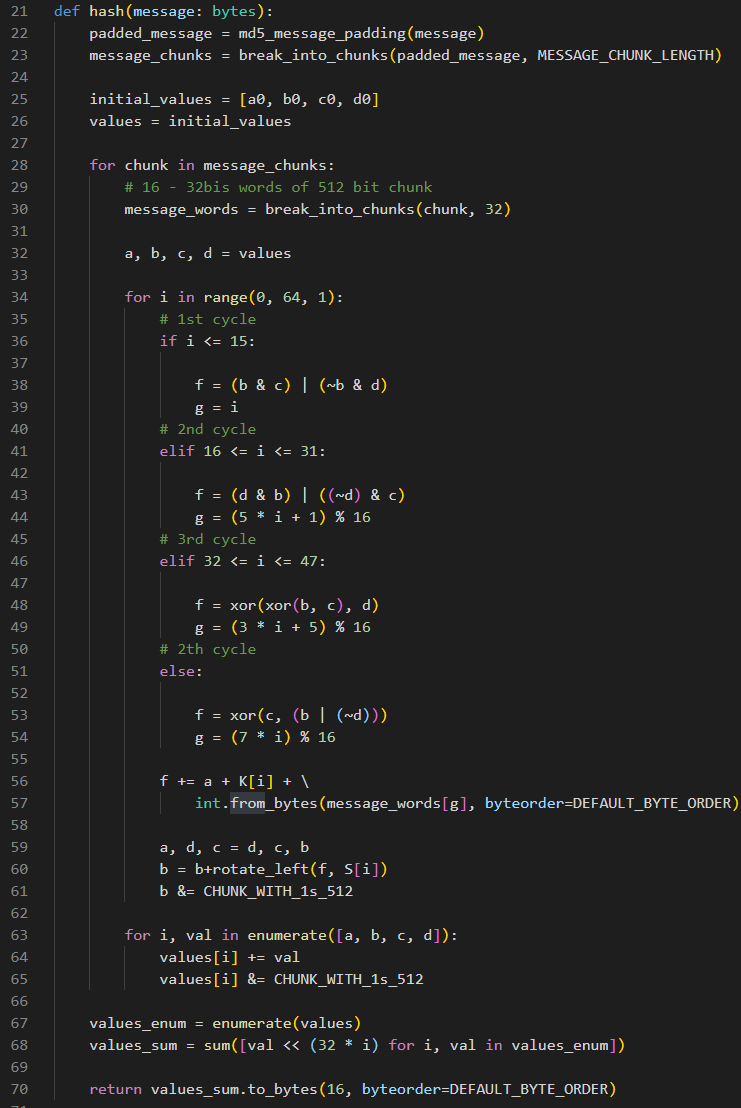
As described before, the input message must be firstly pre-processed (padded). I implemented this padding in following python function:



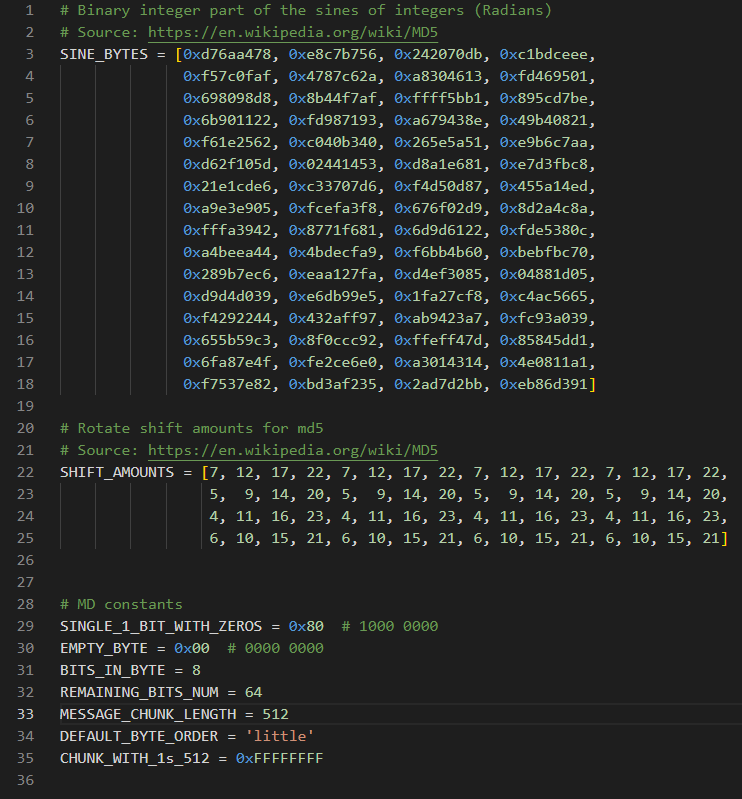
The main md5 function uses also several helper functions (following DRY and single-responsibility programming principle):



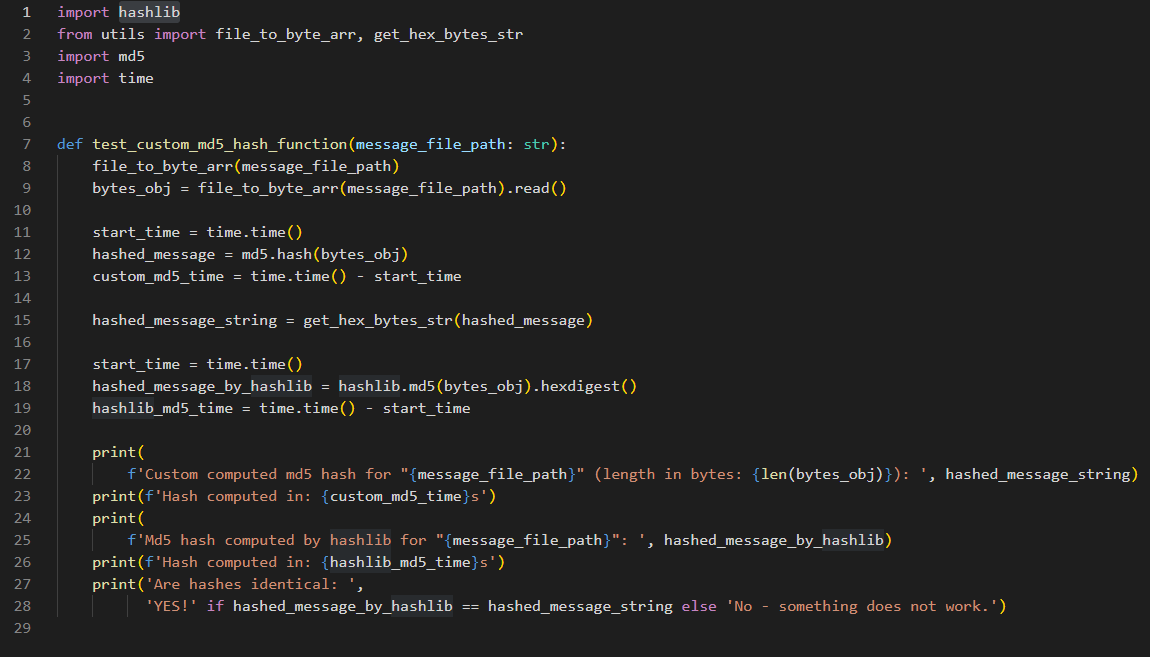
Finally, the main md5 hashing function:



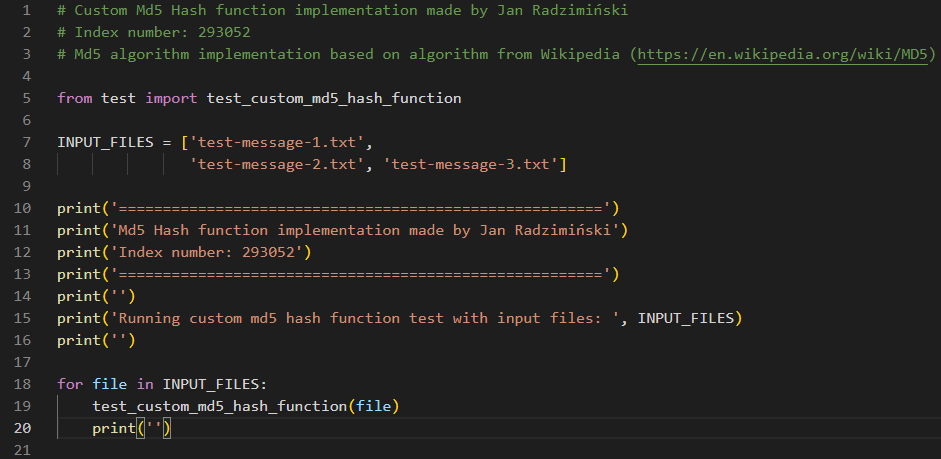
And the constants used in it:



Function used for algorithm testing:



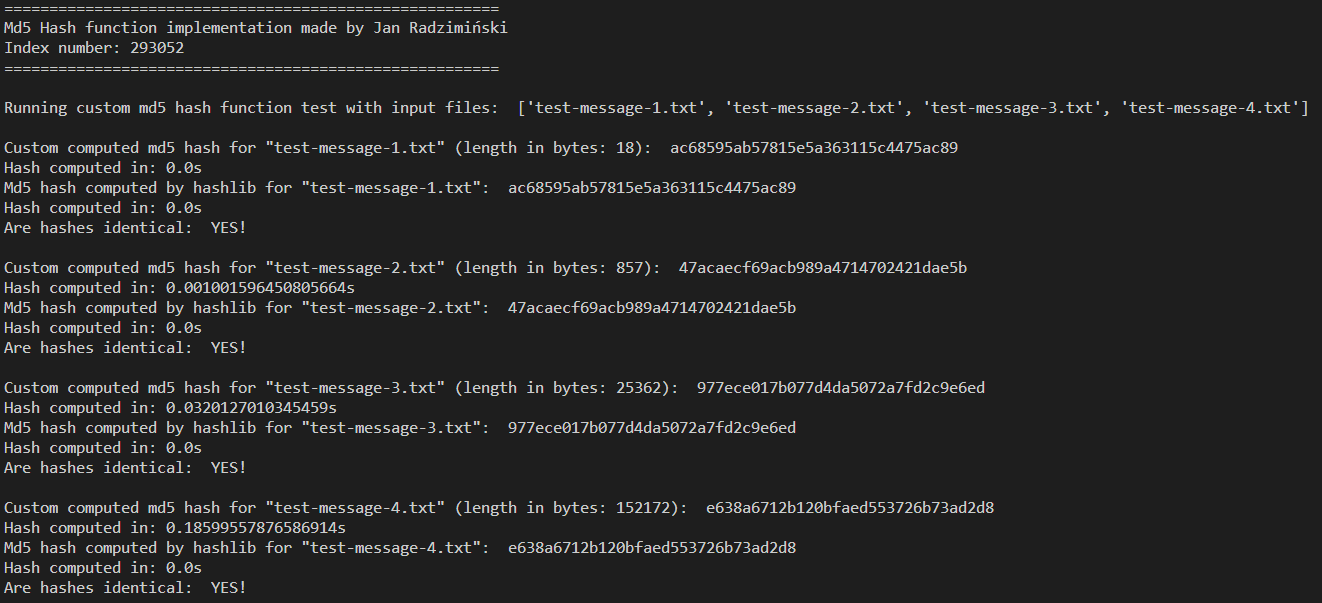
And finally, the main function used for running the script:



1. **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:



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>