MD5 Encryption Algorithm Improvements

Sirui Li  
*School of Advanced Technology*   
*Xi’an JiaoTong-Liverpool University*Suzhou, China  
Sirui.Li19@student.xjtlu.edu.cn

Xinyi Huang  
*School of Advanced Technology*  
*Xi’an JiaoTong-Liverpool University*Suzhou, China  
Xinyi.Huang19@student.xjtlu.edu.cn Zizhe Wang  
*School of Advanced Technology*   
*Xi’an JiaoTong-Liverpool University*Suzhou, China  
Zizhe.Wang19@student.xjtlu.edu.cn

Lincheng Shi  
*School of Advanced Technology*   
*Xi’an JiaoTong-Liverpool University*Suzhou, China  
Lincheng.Shi19@student.xjtlu.edu.cn Kaize Yao  
*School of Advanced Technology   
Xi’an JiaoTong-Liverpool University*Suzhou, China  
Kaize.Yao19@student.xjtlu.edu.cn

Ziyu Xiong   
*School of Advanced Technology*   
*Xi’an JiaoTong-Liverpool University*Suzhou, China  
Ziyu.Xiong19@student.xjtlu.edu.cn

*Abstract*—Despite some drawbacks of MD5, its speed, stability, and irreversibility make it still widely used in the field of encryption protection for ordinary data. As a classical hash function, it is worth studying. In this study, we proposed some methods to optimize the performance of MD5. We modified the formula, number of iterations, the array of loop left shifts to improve the collision resistance, the efficiency of encryption and the security of MD5. Besides, we implemented MD5 in Python by ourselves and verified the reliability. Finally, we designed some indicators to test the performance of MD5 and showed some results in this report.

Keywords—MD5, collision resistance, efficiency of encryption, security

# Introduction

In the information age, where technology is booming, the importance of computer security is increasingly valued by people. Usually, we use algorithms for encryption to ensure data privacy and security [1]. MD5 is a widely used hash function to judge the integrity and consistency of data, which can enhance the security and reliability of network data transmission. It is mainly used in three fields: digital signature, consistency verification, and secure access authentication. In 2004, Professor Wang Xiaoyun pointed out in a report that algorithms such as MD5 and SHA-1 are prone to the risk of hash conflicts [2]. In theory, there are an infinite number of original messages with the same hash value, but it is generally hard to achieve such collisions due to computational power and time constraints, and finding valuable collisions is also more difficult. Wang Xiaoyun's work is a breakthrough in the first step of quickly discovering collisions, but there is still a long way to go to find valuable collisions. Therefore, MD5 still has broad application space and research value in today's society, especially in the industrial field. Therefore, our team conducted extensive research and found that the most effective way to improve the security of MD5 is to improve the step function and message component calculation method of the MD5 algorithm. This can not only increase the difficulty of discovering differential paths to effectively resist differential attacks proposed by Xiaoyun Wang's team but also greatly improve the stability and security of the MD5 encryption algorithm.

# Literature Review

The hash algorithm, also known as the message digest algorithm, is one of the core algorithms of modern cryptography. And MD5 is one of the most commonly used and widely applied hash algorithms. The MD5 algorithm is mainly used to ensure the integrity and consistency of information transmission. This always algorithm generates a fixed-length message digest of 128 bits after receiving plaintext of any length and be calculated by particular method and iteration rounds [1]. The MD5 algorithm has compressibility since it can input plaintext of any length, and the length of the resulting message digest must be 128 bits. It also has anti-collision properties, changing just one byte will result in a significantly different ciphertext, which makes it extremely difficult to collide [3]. Due to these characteristics of the MD5 algorithm, it is often used for file consistency verification and digital signature. For example, before downloading a file, a user can obtain an MD5 value and can obtain an MD5 value of the downloaded file after downloading, which can be compared to the original MD5 value to determine if the file has been tampered with. Furthermore, the MD5 algorithm can also be used to prevent file tampering when sending files. And this algorithm is widely used in password authentication.

Although the convenience and irreversibility of MD5 have made it a relatively secure encryption algorithm and be widely used for many years. The research of the MD5 algorithm collision cracking result published by Professor Wang Xiaoyun in 2004 revealed some security risks in this algorithm. In Professor Wang's research, the difference method was used to find collision plaintext pairs, whose corresponding MD5 values are the same. Moreover, this method can quickly find collision plaintext pairs, taking only 15 minutes, and for the less secure MD4 algorithm, it only takes a few seconds [2]. This study greatly weakened the collision resistance and security of the MD5 algorithm. Although Wang Xiaoyun found out the collision method, but still did not find a meaningful collision plaintext, MD5 still has wide improvement space.

According to the research of Yi Mao (2012), in order to resist difference attacks, limiting the modification of messages and difference paths can significantly increase the difficulty of difference analysis and improve the collision resistance of the MD5 algorithm [4]. It is certain that the MD5 algorithm must occurs collisions, since the main calculation processes of each round are the same modular addition and bitwise operations. Therefore, to increase the overflow of the high bits, pre-processing modification of plaintext can accelerate the extension of the difference and enhance the avalanche effect results. Yi Mao states that the plaintext in different segments was added together to make the connections between the plaintext segments link more closely [4]. However, multiplication operations are more likely to cause overflow of high bits and make reverse operations more difficult than addition, which can greatly increase the collision resistance of the algorithm.

MD5 algorithm has high complexity, and encryption speed is a little bit low, which greatly reduces the efficiency of the plaintext-processed MD5 algorithm. To improve the performance of MD5, Liang Hu suggested reducing the number of algorithm iterations can increase the speed of calculation and reduce memory usage. In this study, the calculation speed was increased around 1.6 times, and the algorithm can cause faster avalanche effect due to the discontinuous indexing, so that the algorithm can maintain security [5].

Security specialist Lake [6] mentioned that the circular left-shift operation can confuse different parts of the message together, making it more difficult for an attacker to crack the hash. In the second, third, and fourth rounds of the MD5 algorithm, each round involves a circular left-shift operation on a different part of the message. However, the circular left-shift array in the implementation of the MD5 algorithm is a pseudo-random array, which is not very random. As a result, there is still a lack of strong research in improving the randomness of circular left-shift arrays.

In previous research, we have found that plaintext pre-processing, the number of algorithm iterations and circular left shift arrays have a significant impact on the efficiency and security of the MD5 algorithm. In this paper, we will analyze these three components and find ways to improve them.

# Problem Identification

## Weak collision resistance

With the emergence of big data, as a traditional encryption algorithm, MD5 is no longer as secure as before. Collisions are inevitable because now many oversized files need to be encrypted into a 128-bit message digest. Several algorithms, like original image attack algorithm, birthday attack algorithm, and differential attack algorithm (currently the most effective attack algorithm), can all find the plaintext pair that can produce the same hashing result within a certain time. Today, MD5 is still widely used in many scenarios, such as file verification and account comparison. If the anti-collision resistance is not strengthened, the transmission of malicious ciphertext may not be detected and our network security may be threatened. Besides. it is possible that we avoid collisions by increasing the length of the plaintext. However, passwords on many sites are limited in length, and, for sites with a large user base, database memory may not be enough. Therefore, the inadequacy of collision resistance is a big challenge for MD5 algorithm. One important reason for MD5 hash function collisions is that the results of the calculation lack a high-level calculation, so that they cannot be sufficiently confused [7]. Thus, we first decide to improve it from this perspective.

## Some variables might not be used and relatively low time efficiency

The encryption loop inside MD5 uses a lot of variables, and these are all stored in registers [5]. A standard CPU has only seven registers [5], and we were concerned that as we make the algorithm more complex (last step), the increase in variables might cause some variables to be ignored by the CPU, which would also increase the computation time. Hence, we want to cut the variables used and improve time efficiency.

## Not strong randomness

We found that the array of shifting to the left used by the MD5 is an arithmetic array. As is shown in Figure x, every four numbers in every row are the same. We think that if the left shift number follows a certain rule, the encryption of the algorithm is not random enough, which may benefit the back-push operation of message modification technology. As a result, we plan to modify its shifting array to strengthen its randomness.

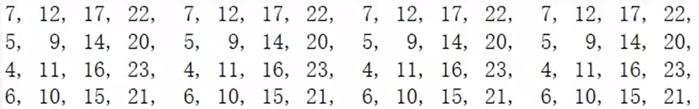


Figure 3.1 The left shifting table of MD5 algorithm

# Proposed Solution and Novelty

## Formula modification

We have updated the plaintext block selection in the original formula to improve its security. Our approach involves replacing the original Mi with the product of two 32-bit plaintext blocks Mi and Mj as a new parameter.

original formula

modification 1

This modification enhances resistance against differential attack algorithms used by Wang [2], as the product operation is more prone to high-order overflow compared to the XOR operation. However, during implementation and testing, we observed that the structure where two plaintext blocks are multiplied is susceptible to situations where the product is zero. This may increase the probability of collisions occurring. To mitigate this issue and increase the difficulty of message modification, we introduced a sliding window with an initial value of Mx instead of 0.

modification 2

Moreover, due to the greater modification of the data by the power operation, we chose to use the power operation of Mx instead of the sliding window Mx. This further increases the confusion level of the processed plaintext block parameters, making it more difficult for collisions to occur.

final version

## Reducing the number of iterations

As mentioned earlier, too many variables we added may take up the memory of CPU registers, consequently reduce effectiveness of the algorithm. We reduced the number of main iterations by 4 times as suggested by Lu [5]. The result is that its calculation speed did improve, whereas in our test, contrasting with the original MD5 algorithm, its average letter repetition rate (the calculation method is shown in section VI) is increased and the average avalanche effect degree is decreased to a large extent, as shown in the figure below:



Figure 4.1 The letter repetition rate of MD5 with 64 main iterations



Figure 4.2 The avalanche effect degree of MD5 with 64 main iterations



Figure 4.3 The letter repetition rate of MD5 with 16 main iterations



Figure 4.4 The avalanche effect degree of MD5 with 16 main iterations

So, we abandoned this modification and tried to reduce it by a smaller number. After many experiments, we found that when we cut the number of main cycles in half, both letter repetition rate and avalanche effect degree are better (shown in figure 4.5 and figure 4.6).



Figure 4.5 The letter repetition rate of MD5 with 32 cycles



Figure 4.6 The avalanche effect degree of MD5 with 32 cycles

To find the causes, we list the 32 steps of the main loop, in which each function is followed by the number of steps corresponding to our new algorithm (A, B, C, D, Mi, Mj, Mx, S<<):

1)FF (a, b, c, d, M0, M1, M0, 6)

2)FF (d, a, b, c, M1, M4, M1, 3)

3)FF (c, d, a, b, M2, M7, M2, 19)

4)FF (b, c, d, a, M3, M10, M3, 10)

5)FF (a, b, c, d, M4, M13, M4, 19)

6)FF (d, a, b, c, M5, M0, M5, 4)

7)FF (c, d, a, b, M6, M3, M6, 6)

8)FF (b, c, d, a, M7, M6, M7, 20)

9)GG (a, b, c, d, M13, M15, M7, 20)

10)GG (d, a, b, c, M0, M2, M9, 1)

11)GG (c, d, a, b, M3, M5, M10, 23)

12)GG (b, c, d, a, M6, M8, M11, 22)

13)GG (a, b, c, d, M9, M11, M12, 12)

14)GG (d, a, b, c, M12, M14, M13, 5)

15)GG (c, d, a, b, M15, M1, M14, 11)

16)GG (b, c, d, a, M2, M4, M15, 17)

17)HH (a, b, c, d, M1, M5, M0, 12)

18)HH (d, a, b, c, M6, M12, M1, 9)

19)HH (c, d, a, b, M12, M13, M2, 16)

20)HH (b, c, d, a, M2, M10, M3, 5)

21)HH (a, b, c, d, M8, M1, M4, 12)

22)HH (d, a, b, c, M0, M8, M5, 15)

23)HH (c, d, a, b, M6, M15, M6, 3)

24)HH (b, c, d, a, M12, M16, M7, 3)

25)II (a, b, c, d, M11, M15, M8, 16)

26)II (d, a, b, c, M2, M6, M9, 10)

27)II (c, d, a, b, M9, M13, M10, 21)

28)II (b, c, d, a, M0, M4, M11, 10)

29)II (a, b, c, d, M7, M11, M12, 7)

30)II (d, a, b, c, M14, M2, M13, 16)

31)II (c, d, a, b, M5, M9, M14, 21)

32)II (b, c, d, a, M12, M0, M15, 20)

We see that the M subscripts i and j are discontinuous and varies greatly, which makes the constants k even less similar to each other. At the same time, our left shifting number is completely different from the old version and that is why our avalanche effect is accelerated.

Furthermore, our issue is alleviated, with each register having half of the original number of iterations of 8 iterations per register, and time efficiency is also improved.

## Modifying the array of loop left shifts

Previously we mentioned that the number of moves in the MD5 algorithm is random, but we found that the circular left shift array in the original MD5 is a pseudo-random number and is highly repetitive. Therefore, in order to solve the problem that the random array of circular left-shift operations in the original MD5 was not very random, we decided to replace it with a new random array.

First, we generated several random arrays using a random number generation program and replaced them individually with the array from the original MD5. After testing, the overall performance of MD5 was improved, as evidenced by an increased degree of avalanche effect and a reduced letter repetition rate. However, since random arrays are less stable, we chose the one with the most stable performance for replacement after several tests. Please see the subsequent section for the exact test procedure. The new random array is shown in Figure 4.7.

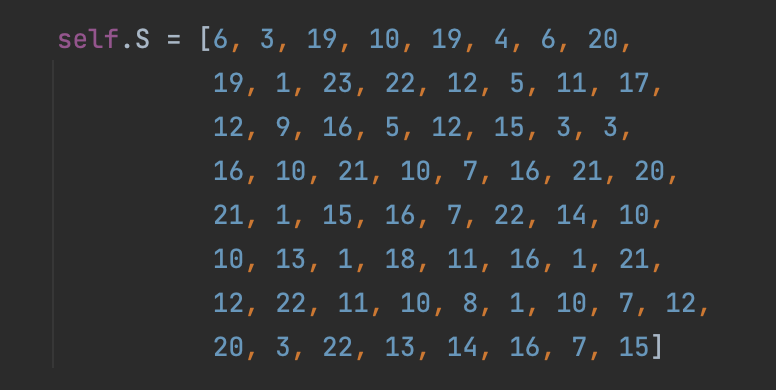


Figure 4.7 The new random array

# Implementation

## Implement the original MD5 algorithm

After conducting thorough research on Rivest’s paper [3] on the MD5 algorithm, we managed to implement the original MD5 algorithm using Python. The code is encapsulated into MD5.py. Further improvements to the MD5 algorithm can easily be applied with our implementation. Moreover, this practical implementation facilitates a deeper understanding of the algorithm itself and the coding methodology. The following are the core code blocks of MD5 implementation.

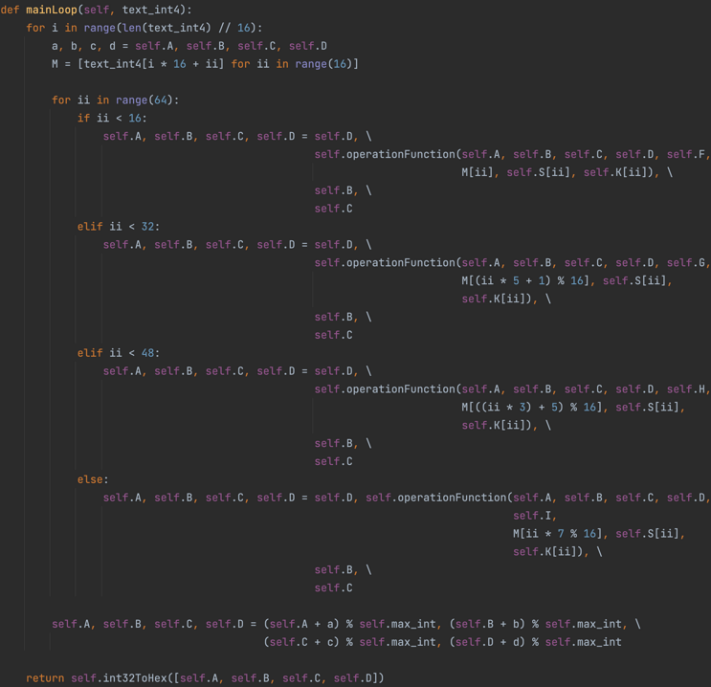


Figure 5.1 Core code blocks of MD5

## Test the correctness of the MD5 implementation

We tested the correctness of our MD5 implementation using Python's built-in package hashlib for result comparison. We randomly generated 100 strings for each testing loop, with 10 loops in total. Core functions and results are as follows.

String generator:

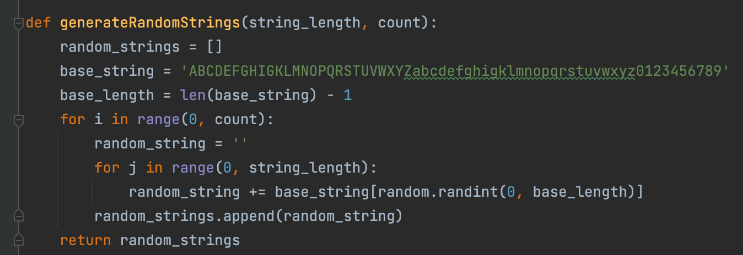


Figure 5.2 Code of string generator

Check correctness:

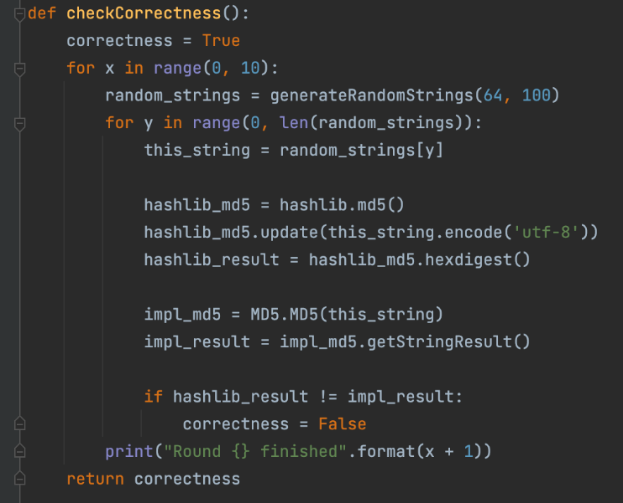


Figure 5.3 Code of checking correctness

Result:

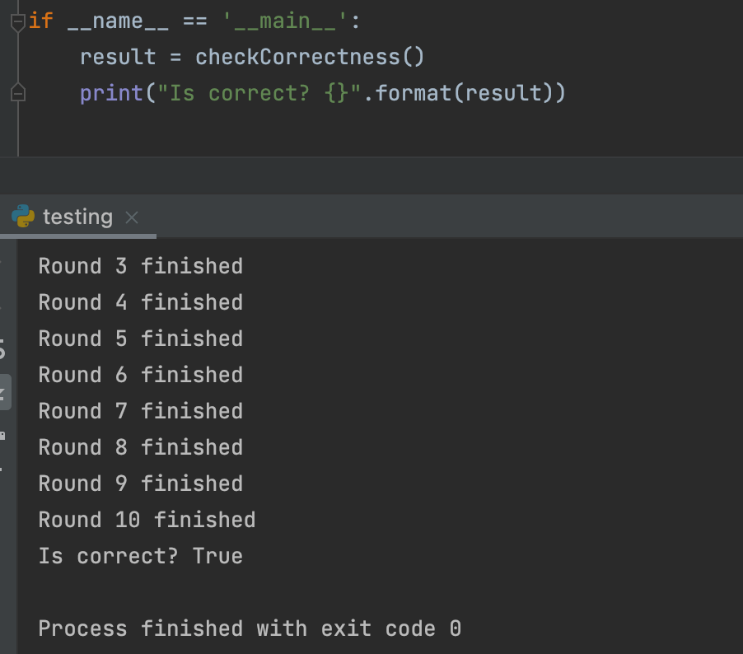


Figure 5.4 Result of correctness

The comparison results demonstrate the correctness of our implementation.

## Apply improvements

A new class with our improvements is implemented. The following are the comparisons of modification.

Circular shifting array:

New:

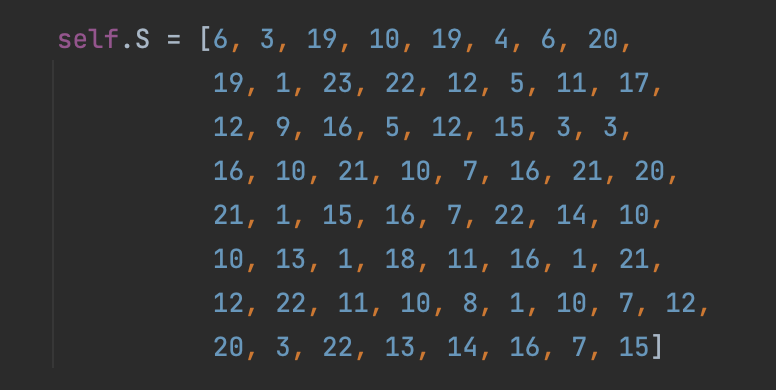


Figure 5.5 New shifting array

Old:

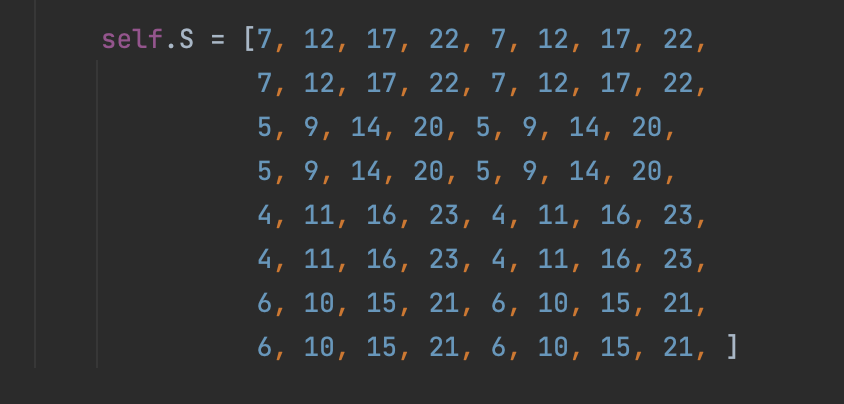


Figure 5.6 Old shifting array

Mi text in operation function:

New:

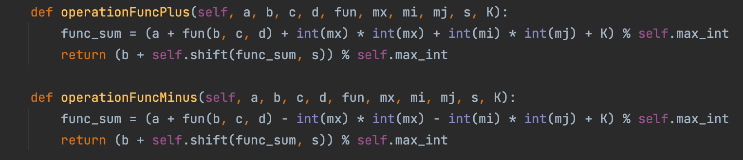


Figure 5.7 New Mi

Old:

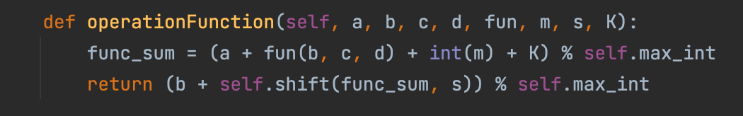


Figure 5.8 Old Mi

Cycles:

New:

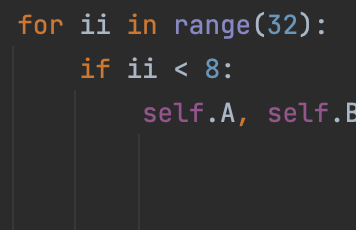


Figure 5.9 New cycles

Old:

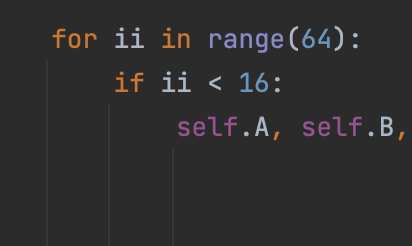


Figure 5.10 Old cycles

# Testing

## Ideas of Testing

We measured the performance changes of MD5 from the collision resistance, the efficiency of encryption and the security. In our code, three sections were used for testing. They are the folder 'collision\_files', the file 'testing2.py' and the file 'main.py'.

In 'collision\_files', there are two pairs of text files for testing collision resistance. They are all generated by a ready-made collision generator. The link of the collision generator is ‘https://www.cnblogs.com/zz-w/p/17337083.html’. On the original MD5, the ciphertexts obtained between paired files are the same.

In 'testing2.py', we defined many functions to calculate and compare relevant indicators. For the collision resistance, we used paired files in the folder 'collision\_files' to obtain their ciphertexts through the original MD5 and improved MD5 respectively. Then we compared whether the ciphertexts generated by the improved MD5 are the same. If they are different, it indicates that the collision resistance of MD5 has been improved. For the efficiency of encryption, we used the built-in functions of Python from the 'time' library to calculate it.

For the security, we used two ways to measure it. One is the avalanche effect, and the other is the letter repetition rate. The purpose of both ways is to know how different ciphertext generated by similar plaintext is. Both of these ways are calculated through the two different generated ciphertexts. The formula of avalanche effect degree value is as follows:

value = 1 - N1 / T,

where N1 is the number of identical letters in the corresponding position, T is the total number of letters. Since the avalanche effect of the original MD5 is good enough, we designed the new way which is to calculate letter repetition rate to measure the security more clearly. The formula of the letter repetition rate is as follows:

rate = N2 / T,

where N2 is the number of occurrences of the same letter, T is the total number of letters. In these two formulas, if the value increases and the rate decreases, this is the best situation. This also indicates that the security of MD5 has been improved.

In order to make the final results universal, we defined a function which can randomly generated 100 pairs of strings to calculate the average of the time, value and rate. Among these 100 pairs of strings, there are 20 pairs each with the length of 20, 40, 60, 80 and 100. The length of the string is not randomly generated to ensure that our results are valid for both long and short strings. In this file, we can not only get the performance of the original MD5 but also the performance of the improved MD5. Apart from getting the average of the value and rate of 100-pair random strings, we can also get the value and rate of any strings we wanted.

In 'main.py', the final performance results can be printed. We can change the testing files by changing 'file\_1' and 'file\_2'. In addition to 'txt' files, other formats such as images can also be used as input files. We can also change the testing strings by changing 'string\_1' and 'string\_2'.

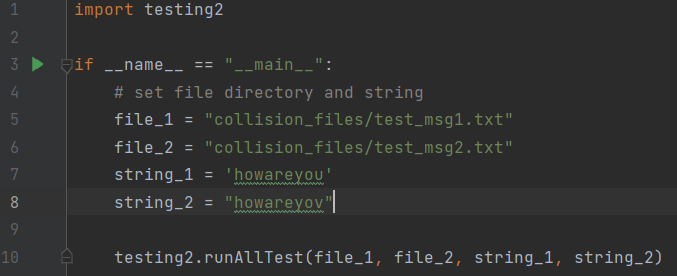


Figure 6.1 Code of ‘main.py’

## Results of Testing

The selected file pairs and string pairs in our default code are 'test\_msg1.txt', 'test\_msg2.txt' and 'howareyou', 'howareyov' respectively. After running the code, we got the following results.

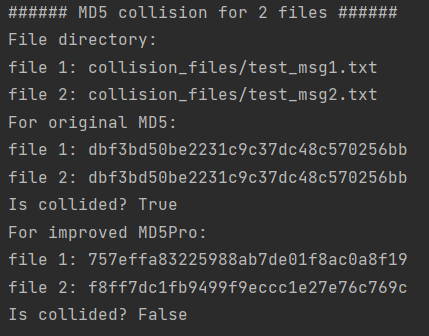


Figure 6.2 Collision results for 2 files

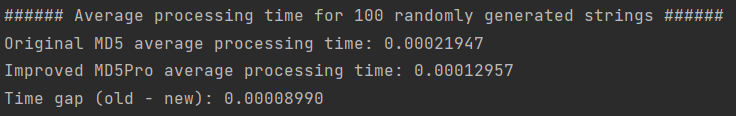


Figure 6.3 Time results for randomly strings

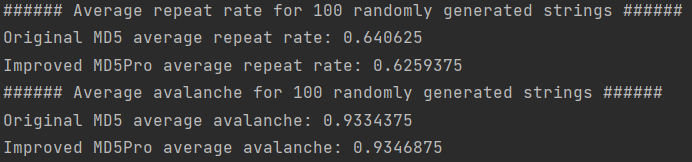


Figure 6.4 The average repeat rate and avalanche effect of randomly strings

Figure 6.2 shows the two files that would have collided no longer collide. It means the collision resistance of MD5 has been improved. Figure 6.3 shows the average encryption time has been reduced. It indicates the efficiency of encryption has been improved. Figure 6.4 shows the average letter repetition rate has been decreased and the average avalanche effect degree value has been increased. This indicates the security of the MD5 has been improved. For Figure 6.4, only one scenario is shown here because 100 pairs of strings are random. Under our multiple tests, there were slight fluctuations in these two indicators. However, in most cases, the avalanche effect degree value of the improved MD5 is higher than that of the original MD5, and the letter repetition rate of the improved MD5 is lower than that of the original MD5.

Figure 6.5 can be seen as an example of the specific input string pair 'howareyou' and 'howareyov'. It not only shows the performance results but also shows the ciphertexts generated by the original MD5 and the improved MD5. It can be clearly seen from figure 6.5 that the results of this example meet our expectations.

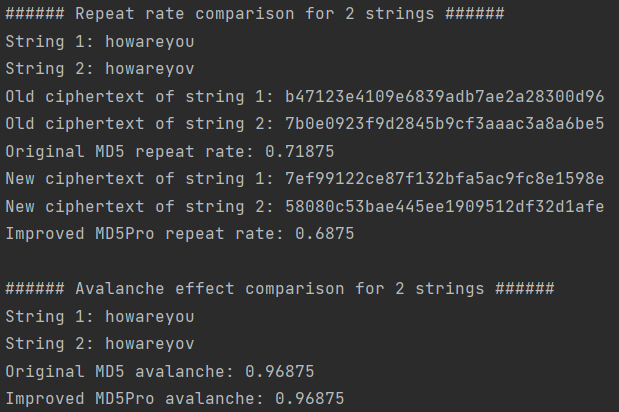


Figure 6.5 The repeat rate and avalanche effect of an example

# Conclusion and Future Work

The aim of this study is to find effective improvements to enhance the efficiency of the MD5 algorithm as well as to address its shortcomings. An analysis of the MD5 algorithm as a whole and of previous research shows that plaintext processing, iteration rounds and circular left shift arrays all have a significant impact on the efficiency and security of the algorithm. Finally, we enhance and improve the problems that occur in the above sessions by changing the plaintext processing function, reducing the number of iteration rounds and replacing the random array of the circular left shift. The results show that the security and operational efficiency of the MD5 algorithm have been effectively improved. Based on these findings, practitioners should consider further measures to enhance the security and other shortcomings of the MD5 algorithm in the future.

# Members Contributions

Every member of the team tried their best in developing the algorithm, and all of them are of equally 16.7% contribution.

##### References

1. C. Yonghui and J. Lainxing, "Research on MD5 algorithm," in Simulation technology and its application (17th CCSSTA 2016), 2016, pp.225-227.
2. X. Wang, D. Feng, X. Lai, et al., "Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD," [Online]. Available: http://eprint.iacr.org/2004/199. [Accessed: May 4, 2023].
3. F. Rivest, "The MD5 message-digest algorithm," RFC 1321[S], Internet Activities Board, Internet Privacy Task Force, 1992.
4. M. Yi and C. Na, "Research and Improvement of MD5 Algorithm," Computer Engineering, vol. 38, no. 24, pp. 262-264, Dec. 2012.
5. L. Lu, "The technical research and performance optimization of MD4 algorithm," [Online]. Available: https://www.docin.com/p-214206915.html. [Accessed: May 4, 2023].
6. J. Lake, "MD5 algorithm with examples," Comparitech, Aug. 20, 2021. [Online]. Available: https://www.comparitech.com/blog/information-security/md5-algorithm-with-examples/. [Accessed: May 4, 2023].
7. X. Zhang and X. Tang, "MD5 algorithm and its role in file system integrity protection," Computer Application, vol. 23, no. 12Z, pp. 430-432, Dec. 2003.