Modern Technologies for Information Security

MD5: Design and Collision Attack

Mocanu Alexandru
University of Bucharest, Security and Applied Logics, Group 510

December 2021

Abstract

This paper offers an overview of one of the most used hash function, MD5. The presentation aims towards two targets. First comes showcasing the design of the function, which is based on a common construction called Merkle-Damgard. To materialize this, a Python implementation is also provided. The second part is dedicated to illustrate issues that should be taken into account when using MD5, the focus being on revealing the most important ideas that have led to the construction of an attack that can generate collisions in a matter of seconds.

1 Introduction

Hash functions are cryptographic primitives of great importance in the security world of today, since they are used as building blocks for many complex systems. Among the most important use cases, we can note the storage of passwords, the integrity of files, the digital signatures and the digital certificates in a public key infrastructure system. With that in mind, it is easy to understand that a vulnerability discovered for a hash function would result in multiple failure points from within a complex system.

There are three cryptographic properties that should be analyzed for any given hash function (H) before using it in any security system:

- collision resistance (how hard can we find x, x' such that H(x) = H(x')?),
- second preimage resistance (how hard can we find x such that H(x) = H(x') for a given x'?),
- first preimage resistance (how hard can we find x such that H(x) = y for a given y?).

Apart from these, particular usages for a hash function may require additional properties. For example, for a hash that is used to store passwords, we want that function to be slow, in order to make a brute-force attack more difficult.

One of the most widely known hash function is called MD5 which was designed by Ronald Rivest in 1992 [8]. Cryptanalysts focused on understanding and breaking this function since its beginning, and now many methods for creating collisions for MD5 are known. Another important mention, even though it is not the main focus of the presentation, is that MD5 is also not recommended for storing passwords because it has a significantly higher speed in execution which facilitates brute-force-like attacks on a password. Considering all these issues, recommendations have been made against the use of MD5, but reports show that many systems are still using it, thus its relevance has not diminished yet.

In section 2, we aim to illustrate one of the general frameworks for building hash functions, called Merkle-Damgard [7], but also discuss the specific compression function that is used for MD5. In section 3, we briefly present the Python code which implements the MD5 hash function. In section 4, we give an overview of the ideas used to built a multiple-block collision in MD5. In the last section, we summarize the ideas and draw some conclusions regarding the use of MD5.

2 MD5

Generally, we want a hash function to accept an input of arbitrary length ($\{0,1\}^*$) and compute an output of fixed length, which in the case of MD5 is on 128-bits ($\{0,1\}^{128}$). In order to achieve such a construction, many hash function start with a compression function which is able to transform a fixed length input to a smaller fixed length output. In particular for MD5, we have a function $C : \{0,1\}^{512} \to \{0,1\}^{128}$.

Before describing how the function C is designed, let us assume it exists, and show how the MD5 function can be obtained. Let M be the input of the MD5 function. First, we have to make sure the length of M is a multiple of 512 and to do that we use padding. We compute M_{pad} as follows: add a bit of 1 to M followed by bits of 0 until the length of the result is 448 when computing modulo 512. That means that we are left with the last 64 bits from the last block, which will be completed with the initial length of M. After M_{pad} is obtained we split it in blocks of 512 bits ($M_{pad} = M_0|M_1|M_2|M_3|...|M_n$).

Before being able to understand the construction of MD5, we should also note that the compression function C has an initialization vector of length 128 bits as input, which will be denoted as $C(M, \text{IV})$, where M is the input of 512 bits and IV is the initialization vector. Now, we can present the Merkle-Damgard construction as follows: let IV_0 be the initialization vector for MD5 (it is constant and the exact value for this is given when building the compression function). Compute $\text{IV}_1 = C(M_0, \text{IV}_0)$, $\text{IV}_2 = C(M_1, \text{IV}_1)$, ..., $\text{IV}_{n+1} = C(M_n, \text{IV}_n)$. In the end, we denote $\text{MD5}(M) = \text{IV}_{n+1}$.

In order to have a complete definition, all that is left now is to describe the function C. The input for C is a 512-bits M (part of the message being hashed) and a 128-bits IV (the initialization vector). Both values will be split in 32-bits words: $M = m_0|m_1|...|m_{15}$ and $\text{IV} = a_0|b_0|c_0|d_0$.

For the first iteration, the values used for IV_0 are
$a_0 = 0X67452301$,
$b_0 = 0XEFCDAB89$,
$c_0 = 0X98BADCFE$,
$d_0 = 0X10325476$.

Let q_i be the sequence that denotes the states of C. For initialization we have:
$q_{-4} = a_0$,
$q_{-3} = d_0$,
$q_{-2} = c_0$,
$q_{-1} = b_0$.

After that, the following step is repeated 64 times:
$q_i = q_{i-1} + (q_{i-4} + f_i(q_{i-1}, q_{i-2}, q_{i-3}) + w_i + t_i) <<< s_i$, where $+$ is the modular addition on 32-bits and $<<<$ is the cyclic left rotation with the given offset of bits.

To give complete meaning to the previous statement we have to define f, w, t and s:

- t is meant to be a constant for each step, defined as $t_i = \lfloor |sin(i+1)| * 2^{32} \rfloor$.
- f is a function that changes after every 16 steps:

$$f_i(b,c,d) = \begin{cases} F(b,c,d) = (b \wedge c) \vee (\neg b \wedge d), i \in \{0,1,...,15\} \\ G(b,c,d) = (b \wedge d) \vee (c \wedge \neg d), i \in \{16,17,...,31\} \\ H(b,c,d) = b \bigoplus c \bigoplus d, i \in \{32,33,...,47\} \\ I(b,c,d) = c \bigoplus (b \vee \neg d), i \in \{48,49,...,63\} \end{cases}$$

  where $\wedge$ is the and bitwise operator, $\vee$ is the or bitwise operator, $\neg$ is the 32-bit complement and $\bigoplus$ is the xor bitwise operator.

- s is represent the number of bits for the rotation and changes every step according to the following table:

i div 16	i mod 4			
	0	1	2	3
0	7	12	17	22
1	5	9	14	20
2	4	11	16	23
3	6	10	15	21

- last remaining is w, which describes what part of the input is involved in the state computation and is defined as:

$$w_i = \begin{cases} m_i, i \in \{0,1,...,15\} \\ m_{(5*i+1)mod16}, i \in \{16,17,...,31\} \\ m_{(3*i+5)mod16}, i \in \{32,33,...,47\} \\ m_{(7*i)mod16}, i \in \{48,49,...,63\} \end{cases}$$

Looking at the previous definition, some similarities between the groups of 16 steps can certainly be seen. For that reason many refer to this algorithm as having 4 rounds, each round consisting of 16 steps. In the end, the output for the function C will be

$\text{IV}_f = (a_0 + q_{60}, b_0 + q_{63}, c_0 + q_{62}, d_0 + q_{61})$, where $+$ is the modular addition.

3 Implementation

As seen in the previous section, it is straightforward that the MD5 hash function can be implemented and used as-is. There already are Python modules like hashlib [4] that implement the MD5 algorithm.

For didactic purposes, a Python 3.8 implementation of MD5 from scratch was provided. The script uses only the built-in math module to improve accessibility.

The main components of the implementation are the functions compression_function and MD5. Apart for those, only auxiliary functions are implemented to provide the functionality required by the compression function such as the left rotation, the functions F, G, H, I, the rest of parameters that changed from one step to another (s and t). Referring to the previous section, the compression function is an implementation for the function C with the mention that for an easier implementation

instead of indexing the sequence q_i from -4 as described in theory, we started with index 0 in the algorithm itself. Related to the MD5 function, it is responsible for 2 tasks: pad the message received as a byte sequence and, afterwards, iteratively call the compression function on each block.

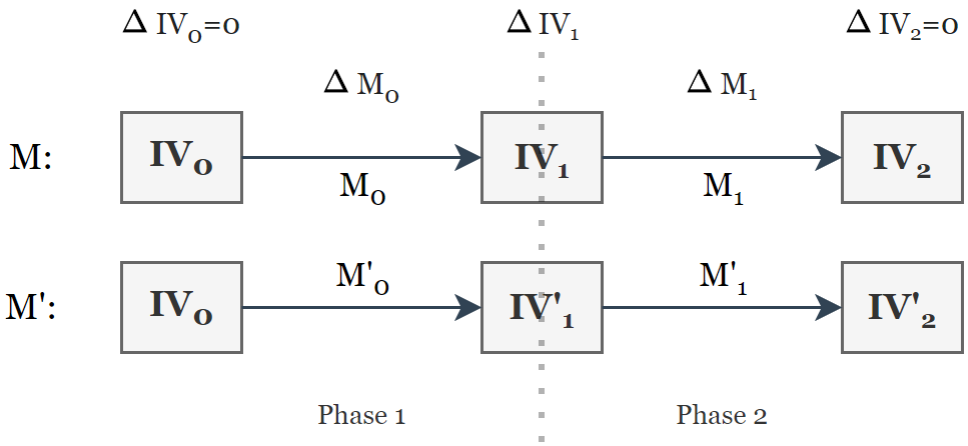
4 The collision attack

As mentioned in the introduction, one of the important cryptographic properties of hash functions is the resistance to collisions. Since the size of the input space is larger than the size of the output space, it should be trivial to understand that collisions exists for any given hash function. Thus, the analysis of this property consists of proving that finding collisions is hard (the probability of finding a collision is negligible [1]). To have a reference for what a small probability represent we can consider as a baseline the attack based on the birthday paradox [10], which shows that we need to hash around $\sqrt{2^b} = 2^{b/2}$ values (where b is the length of the output for the hash function) to find a collision with 50% probability. In the case of MD5, whose output is on 128 bits, we need to hash around 2^{64} values.

We are going to have a look at the ideas used to build a two-blocks collision attacks [9][6][5]. For that, we need to find two 1024-bits message M and M'. It is important to note that the padding mechanism is not applied, instead it is considered that these are the messages that are split and used as input for the compression function. This assumption does not invalidate the attack, instead it makes it more general (for example, we can use the fact that if $\text{MD5}(M) = \text{MD5}(M')$, then $\text{MD5}(M|P) = \text{MD5}(M'|P)$, for any given M, M' with length multiple of 512 bits and for any P). So, we can write that $M = M_0|M_1$ and $M' = M'_0|M'_1$. With the new notations, we can say that finding a collisions means that $C(M_1, C(M_0, \text{IV}_0)) = C(M'_1, C(M'_0, \text{IV}_0))$.

A very useful cryptanalysis technique used for hash functions as well as for encryption algorithms is called differential attack [2]. For hash functions, the aim is to analyze how changing the input impacts the output. The oldest method to track these differences was using the XOR function, but in the case of MD5, this was not enough. That is due to the fact that using the xor of two values, we cannot say if bit j is set or not in one of those values. To be able to also monitor these differences, it was introduced the idea of modular differential [3] which is computed as the signed difference between two values. Using these two concepts, we can define a differential path for MD5. Let us select M_0 and M'_0 for which we know that $M_0 \bigoplus M'_0 = d_{xor}$ and $M'_0 - M_0 = d_{sub}$. A differential path for MD5 will show the differences between the two computations for each step from the compression function depending on d_{xor} and d_{sub}. Ideally we want to find a differential path that is simple enough to be able to track the differentials, but we also want the path to be general enough to increase the probability of finding collisions. An example of differential path for MD5 can be seen in Table 3 from [9].

Moving to the attack itself, we will see that it is comprised of tWo phases. In each phase we will determine a pair of blocks for the collision (i.e. at phase 1 determine M_0, M'_0 and at phase 2 determine M_1, M'_1).



In phase 1, we start with a constant value for

$$\Delta M_0 = M_0' - M_0 = (0, 0, 0, 0, 2^{31}, 0, 0, 0, 0, 0, 0, 2^{15}, 0, 0, 2^{31}, 0)$$

and afterwards a differential path is defined with the goal to obtain a given value for

$$\Delta \mathrm{IV}_1 = \mathrm{IV}_1' - \mathrm{IV}_1 = (2^{31}, 2^{31} + 2^{25}, 2^{31} + 2^{25}, 2^{31} + 2^{25})$$

The issue that must be overcome to complete the attack is that not any arbitrary value for M_0 follows the defined differential path. To obtain that, multiple conditions are imposed on the bits of the intermediary states from the compression function calculus. All those sufficient conditions can be found in Table 4 from [6]. We can observe looking at the steps described in section 2, that in the first 16 of them, each part of the message M_0 only appear once, that mean that most of the conditions described in the table can transform into conditions on the message M_0 as highlighted in the Table 2 from [5]. Therefore, we can start by randomly generating a message M_0 which respects most of the conditions and check the remaining ones. In case of failure, we can return to the random generation of the message and try with a new value.

For phase 2, we start from the value from phase 1 for $\Delta \mathrm{IV}_1 = \mathrm{IV}_1' - \mathrm{IV}_1$ and attempt to obtain

$$\Delta \mathrm{IV}_2 = \mathrm{IV}_2' - \mathrm{IV}_2 = (0, 0, 0, 0)$$

which will mean that we obtained a collision. To be able to build a differential path we also define

$$\Delta M_1 = M_1' - M_1 = (0, 0, 0, 0, 2^{31}, 0, 0, 0, 0, 0, 0, -2^{15}, 0, 0, 2^{31}, 0)$$

Afterwards, similarly to phase 1, we define the differential path as well as the conditions that need to apply to the intermediary states, on which we will apply the same algorithm.

Regarding the performance of this attack, the authors of [9] computed the probability of success for phase 1 as 2^{-37} and for phase 2 as 2^{-30}. The total time complexity resulting in 2^{39} MD5 computations.

5 Conclusion

To conclude, we presented an overview for the hash function MD5, looking both at its design and at an attack that aims to generate collisions. Even if the scope of this paper is to focus on the basic attack, it is important to highlight that many other collision generating techniques have derived from it, making the MD5 function insecure in many contexts. Beside that, its output can also be considered a vulnerability considering the computing power of today.

Considering all that, the MD5 function should be replaced as much as possible with newer variants of hash algorithms. Nonetheless, the study of MD5 design and cryptanalysis is still relevant because it offers ideas about bad practices when building a hash function as well as attacking techniques that cryptographers should take into account when designing a new hash algorithm.

References

[1] Mihir Bellare. "A Note on Negligible Functions." In: *Journal of Cryptology* 15.4 (2002).

[2] Eli Biham and Adi Shamir. "Differential cryptanalysis of DES-like cryptosystems". In: *Journal of CRYPTOLOGY* 4.1 (1991), pp. 3–72.

[3] S Cotini et al. *Security of the RC6 TM Block Cipher*. 1998.

[4] *Hashlib - Python Documentation*. URL: https://docs.python.org/3/library/hashlib.html.

[5] Vlastimil Klima. "Tunnels in Hash Functions: MD5 Collisions Within a Minute." In: *IACR Cryptol. ePrint Arch.* 2006 (2006), p. 105.

[6] Jie Liang and Xue-Jia Lai. "Improved collision attack on hash function MD5". In: *Journal of Computer Science and Technology* 22.1 (2007), pp. 79–87.

[7] Ralph C. Merkle. *Secrecy, authentication, and public key systems*. Stanford Ph.D. 1979. URL: http://www.merkle.com/papers/Thesis1979.pdf.

[8] Ronald L. Rivest. *The MD5 Message-Digest Algorithm*. RFC 1321. Apr. 1992. DOI: 10.17487/RFC1321. URL: https://rfc-editor.org/rfc/rfc1321.txt.

[9] Xiaoyun Wang and Hongbo Yu. "How to break MD5 and other hash functions". In: *Annual international conference on the theory and applications of cryptographic techniques*. Springer. 2005, pp. 19–35.

[10] Wikipedia. *Birthday problem*. [Online; accessed 22-December-2021]. URL: https://en.wikipedia.org/wiki/Birthday_problem.