УДК: 004.421:003.26

A Study of the MD5 Collisions

Vladimir Nasteski, Doc. D-r Toni Stojanovski
Faculty of Informatics, European University

***Abstract*—MD5 hash function was designed as an improvement and strengthened successor of the MD4 hash function by Ron Rivest [4]. It quickly gained popularity, but not long after its publication it was subjected to a number of attacks which discovered many weaknesses [1][2][3]. Klima's tunneling method is the fastest and the most understood method of finding collisions in MD5. In this paper we examine the possibilities for parallel and distributed implementation of Klima's tunneling method. We present theoretical comparisons of two parallel and distributed exhaustive search methods.**

***Key Words**: hash, MD5, collision, attack*

I. INTRODUCTION

Hash functions are famous primitive functions that are used in cryptography in many secure applications such as authentication protocols, digital signatures, etc.

The cryptographic value of a hash function relies on these notions of security: for a hash function *h*, with domain *D*, and range *R* we require the following three properties [7]:

1. For a given $y \in R$, it is computationally infeasible to find $x \in D$ such that $h(x) = y$.
2. For a given $x \in D$ it is computationally infeasible to find another $x' \in D$ such that $x \neq x'$ and $h(x) = h(x')$
3. It is computationally infeasible to find different $x, x' \in D$ such that $x \neq x'$ and $h(x) = h(x')$.

Message Digest 5 (MD5) function is a successor of the MD4 hash function [8], presented by Ron Rivest in 1992 [4]. It is well known 128-bit iterated hash function, used in many applications such as SSL/TLS, and IPSec. MD5 is also used in many distributed file systems, time-stamping mechanisms, and pseudo random-number generation.

In this paper we summarize the collision attacks on MD5 from [2] and [3]. We briefly explain the "multi-message modification" technique introduced by Wang [2] and the tunneling method by Klima [3]. We then use the source code for Klima's attack available on the web[1] to design and test two parallel and distributed implementation of this attack. We theoretically analyze the performance of these two exhaustive search techniques. These results are applicable not only to Klima's tunneling method but also to other exhaustive search methods.

This paper is organized as follows. In section II we describe the idea behind the MD5 algorithm short, followed by an explanation of the two collision attacks described by Wang [2] and Klima [3]. In section III we give theoretical results based on the Klima's method and we define our practical implementation of two methods for dividing the search space of random generated numbers, which are responsible for finding a collision.

[1] http://cryptography.hyperlink.cz

II. BACKGROUND

A. The MD5 Algorithm and compression function

MD5 is a hash function in the Merkle-Damgard paradigm, designed by Ron Rivest. The security of this hash function reduces to the security of its compression function. The compression function takes as an input a block *m* of 512 bits together with an initialization vector (intermediate hash value) *IHV=(a, b, c, d)* of 128 bits and outputs the new *IHV'* of 128 bits. The *IHV* for the first block is fixed in the algorithm, and it is called the MD5 initial value.

What is the difference between *collision* and *semi-collision?*

Suppose that a given hash function *h* based on some compression function *f* is given. A *collision of the compression function* consist of an initial value *IV* and two different inputs χ and $\overline{\chi}$, such that:

$$f(IV;\chi) = f(IV;\overline{\chi})$$

The term *pseudo-collision of the compression function* is used if two different initial values IV and $\overline{IV}$ and possibly identical inputs χ and $\overline{\chi}$ are given such that:

$$f(IV;\chi) = f(\overline{IV};\overline{\chi})$$

The finding of pseudo-collision shows that the compression function has a weakness.

Next, we give a brief overview of the compression function in MD5.

The steps in the compression process of MD5 are based upon these word operations (word denotes a 32-bit quantity) [1]:

- Bitwise Boolean operations
- Addition modulo 2^{32}
- Cyclic shifts

These operations are chosen because (i) they can be computed very fast on 32bit processors; and (ii) the mixture between Boolean operations and the addition is believed to be cryptographically strong.

RL(x, n) and *RR(x, n)* denote left and right rotation of the word *x* over *n* bits. Each block *m* is split into 16 words $m_0, \ldots, m_{15}$ (4 rounds) and is expanded into a series of 64 words W_t:

$$W_t = \begin{cases} m_t, & 0 \le t < 16 \\ m_{1+5t \bmod 16}, & 16 \le t < 32 \\ m_{5+3t \bmod 16}, & 32 \le t < 48 \\ m_{7t \bmod 16,} & 48 \le t < 64 \end{cases}$$

This compression function has a working state of Q_t, Q_{t-1}, Q_{t-2}, Q_{t-3}, which is initialized to the *IHV* split into four words $IHV_0, \ldots, IHV_3$ in this order:

$$Q_0 = IHV_1,\ Q_{-1} = IHV_2,\ Q_{-2} = IHV_3,\ Q_{-3} = IHV_0$$

The 64 steps in this compression function are executed in the following order:

$$f_t(X,Y,Z)=\begin{cases}(X\wedge Y)\oplus(\overline{X}\wedge Z), & 0\le t<16\\ (Z\wedge X)\oplus(\overline{Z}\wedge Y), & 16\le t<32\\ X\oplus Y\oplus Z, & 32\le t<48\\ Y\oplus(X\vee\overline{Z}), & 48\le t<64\end{cases}$$

For *t=0, ..., 63* the compression function maintains a working register with four state words Q_t, Q_{t-1}, Q_{t-2}, and Q_{t-3}. These are initialized as $(Q_0, Q_{-1}, Q_{-2}, Q_{-3}) = (b, c, d, a)$ and for *t=0,...63* follows:

$F_t = f_t(Q_t, Q_{t-1}, Q_{t-2})$;

$T_t = F_t + Q_{t-3} + AC_t + W_t$;

$R_t = RL(T_t, RC_t)$;

$Q_{t+1} = Q_t + R_t$

AC_t and RC_t denote the addition and rotation constants.

After all steps are computed, the resulting state words are added to the *IHV* and returned as output:

$$MD5\_Cf(IHV, B) = (a + Q_{61}, b + Q_{64}, c + Q_{63}, d + Q_{62})$$

where *MD5_Cf* denotes the MD5 Compression Function, and *B* is a message block.

B. Wang's and Klima's Attacks

The first attack of the MD5 hash was the so called semi collision attack, presented by Hans Dobbertin [1], in which a collision is found only in the first block of the message, as a weakness of the compression function that is an essential part of creating the hash output. A collision is found with a chosen IV different from the MD5's IV. Years later, a powerful attack was presented by Wang et al. in their paper [2] using the multi-message modification method. That was the first full attack of the MD5 hash function, that is, it managed to find collisions in two blocks. Their work was murky, with not many details, containing only some confusing details of the collisions and branch of results. Wang's attack was able to find collision on the first block in about 15 minutes on a super computer. Their method was subsequently used by many researchers. Klima [3] was motivated by the published data in the unclearly written paper of Wang, fully described his own method, and has generated collisions. Then, he announces a better algorithm [3], improving the time of searching the collisions on a single Notebook PC [3]. Klima's method was far quicker than the Wang's one: the generation of the first block of collision of the message was 1000-2000 times faster than the Chinese one. The method is known as tunneling, in which many tunnels are found to design appropriate differential schemes. That method works for any initialization vector (IV), and can find a collision on a simple PC notebook in about a minute. With this method, it is faster to find collisions for the first block than for the second block. Klima publishes the source code of his method on his site, only for educational purposes, in contrast to the Chinese team [2] whose algorithm is not yet completely know. The program exhaustively searches the space of input blocks starting from a random position. It then calculates a different input which generates the same output. Many papers, including [5] and [6], helped us understand the functions and the mathematics behind the MD5 collisions, including the Wang's method.

Wang and Yu presented a powerful attack on MD5 in their paper [2]. Their attack is based on a combined additive and XOR method. Using these differentials methods, they've constructed two differential paths for the *MD5_Cf*, which can be used to generate a collision of MD5 itself. Their paths describe how differences between two pairs (*IHV, B)* and (*IHV', B'*) of an intermediate hash value and an accompanying message block, propagate through the *MD5_Cf.* Using their collision finding algorithm they search a collision consisting of two consecutive pairs of blocks (B_0, B_0') and (B_1, B_1'), satisfying the two differential paths which starts from arbitrary *IHV = IHV'.* Their attack can be used to create two messages *M* and *M'* with the same hash that only differ slightly in two subsequent blocks.

As differences, the combination of both integer modular substraction and XOR is used. Their combination gives more information than each by themselves. This combination also gives the integer differences (-1, 0, or +1) between each pair of bits *X[i]* and *X'[i]* for $0 \le i \le 31$.

On the other hand, the differential paths for both blocks are constructed specifically to create a collision. The differential paths describe precisely for each of the 64 steps of MD5 what the differences are in the working state and how these differences pass through the Boolean function and the rotation. This attack finds MD5 collisions in about 15 minutes up to an hour on a super computer (IBM P690), with a cost of about 2^{39} compressions for the first message block. Since their paper, many improvements were made. Based on this technique and on these differential paths, collision for MD5 can be found in several seconds using a single Notebook PC, a technique called *tunnels* [3], which was a great breakthrough for the MD5 hash function. Also as an improvement in 2006 was presented a technique called *controlling rotations in the first round* in [6].

All of the results presented in [1] and [2] inspired Klima to do some improvements of the algorithms. Klima uses the sufficient conditions, defined by Wang, from the beginning of the set to the point. He didn't change the conditions, but he verifies the remaining sufficient conditions, what he calls it a "point of verification" or POV. In MD5, using these sufficient conditions, there are 29 conditions, so Klima need to obtain 2^{29} POVs. As he says in [3], one of them will randomly fulfill the remaining conditions and will lead to a collision.

The method tunneling begins in the POV. It is enough only one POV, because it will continue to create series of POVs by one or several tunnels. He uses a tunnel with strength of 24 to generate 2^{24} new POVs. Klima's tunnels sped up the collision search by a factor of about 1000-2000 times compared to Wang's method. Klima claims that finding a tunnel is very difficult, because a sufficient condition in a given differential scheme blocks it up. Then he artificially creates a differential scheme, which contains tunnels, which led it to a collision. This method works with any *IV*. Klima suppose that he and his

team will create differential schemes that can be used not only for MD5, but also for SHA-1 and SHA-2, which will involve useful tunnels.
Algorithm's source code is published on Klima's Web site[2]; it's free and can be used for analysis and improvements of the algorithm. We use this code for an educational and experimental purpose only. To be more precise, we will take a look at the generation of random numbers.

III. Parallel and Distributed Implementation of Klima's Attack

Next, we propose a parallel and distributed search for collisions using Klima's method. We compare two methods for division of the search space. We derive the probability distribution function for the searching time for both methods. Theoretically and experimentally we prove that the first method yields better results. We argue that the first method yields better results for exhaustive search attacks against other algorithms and hash functions.

A. Theoretical results

Threemethods for partition of the search space are explained next:

1. *Random subspaces*: Number of search agents is not known in advance. One directional communication exists from search agents to central server, which is used by a search agent to communicate to the central server about a found solution. New search agents join at run time. Each search agent randomly chooses its search subspace. Each search agent starts from a randomly chosen starting point. Therefore, the size of the subspace searched by an agent can vary between 0 and the size of the entire search space.
2. *Semi-equal subspaces*: Number of search agents is not known in advance, and two directional communication exists between central server and search agents. New search agents can register at run time. Search subspace is allocated and communicated to each search agent by the central server as it registers. Search subspace depends on the current number of search agents. Let [0, N-1] denote the entire search space. Then the first agent will start its search from point 0. Second agent will start the search from point $N/2$. Third agent will start from point $N/4$, and the next agent will start from the point $3N/4$. The next agents will start from points $N/8$, $3N/8$, $5N/8$, $7N/8$ etc.
3. *Equal subspaces*: Number of search agents is known in advance, and two directional communication exists between central server and search agents. Search subspace is allocated and communicated to each search agent in advance. All search subspaces are of equal size.

Following figure gives the average size of the subspaces allocated to search agents for the three methods. Average search time is proportional to the average size of the subspaces.

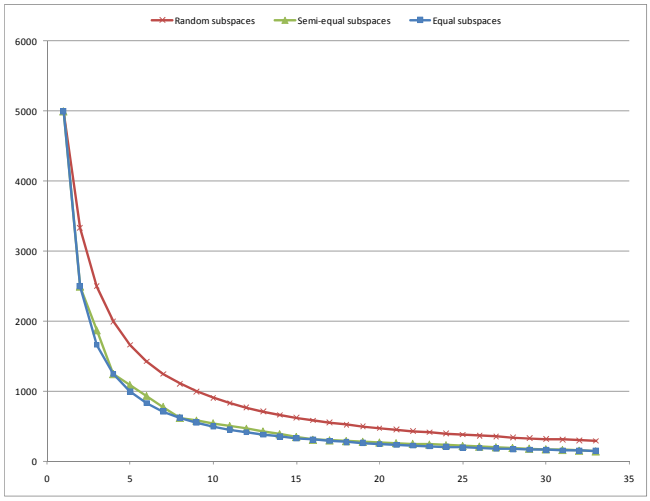

Figure 1: Comparison of performances of the three searching methods.

As expected *Equal subspaces* method produces the best performance, i.e., the shortest average search time. *Semi-equal subspaces* method produces performance which are close to the ones produced by the *Equal subspaces* method. *Random subspaces* method results in significantly slower search times. For example, 20 search agents using the *random subspaces* method will produce same performance as 12 search agents using the *semi-equal subspaces* method

B. Practical implementation

As a practical implementation, we have created an application for experimental purposes only. The application is created in Adobe Flash CS3. Experimentally we compare only the two methods: *Semi-equal subspaces* method and *Random subspaces* method.
In the *Semi-equal subspaces* method, the searching subspace is shrinking as the number of search agents is growing. Thus, if there is only one search agent in the system, then it searches the whole range. When there is a second search agent in the system, then its searching subspace is the second half of the whole space. If a random number is located in the second half of the range, then the second search agent will find it faster than the first search agent. With this, the space and time of searching is divided by half, for each next searching agent. Each time a new search agent joins the search, it is given to search the second half of the currently largest search subspace. On Figure 2 we give an example of 8 search agents, with their allocated search subspaces.

[2] http://cryptography.hyperlink.cz/MD5_collisions.html

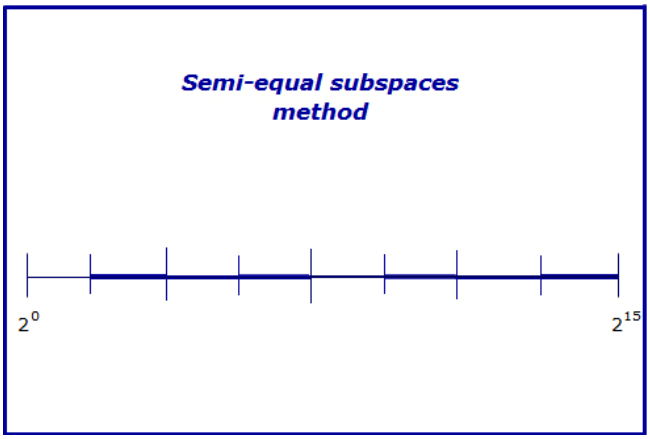

Figure 2: The semi-equal subspaces method

The range for generating a random number in the Klima's MD5 application is from $0 - 2^{15}$, what presents the range of the RANDMAX operation in C++. Our application divides that range by half for each new search agent, and with that the time of searching a random number is also divided by half.

In the *Random-subspaces* method the starting point of the searching subspace for each search agent is randomly chosen.

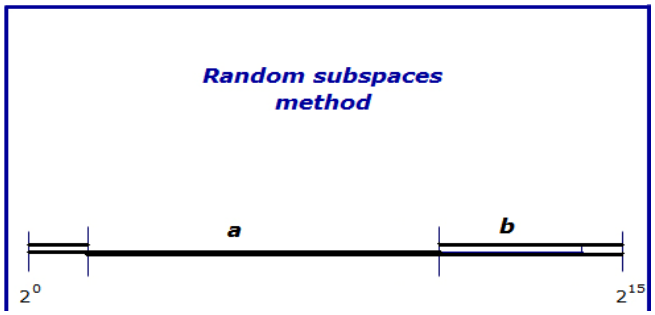

Figure 3: The Random-subspaces method

In Figure 3 we present the Random-subspaces method, with two search agents in the systems. As show, the first search agent searches the random number in the range *a,* and the second search agent searches in the range *b.* Both agents have random range, and appear randomly on the scale.

Figure 4 presents an excerption of the Klima's application. In a given range, depending on what method is used, it starts by finding a random big-endian value, and in a period of time it finishes when it finds a collision in the first block, and continues with finding a collision in the second block, if it finds it. If not, it starts with a different random big-endian value for finding a collision again in the first block, until accomplishing a full collision.

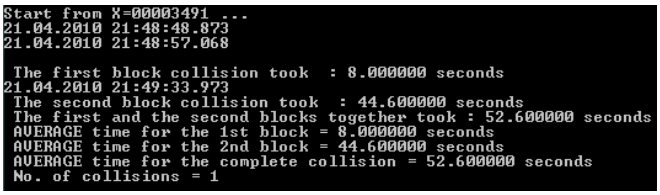

Figure 3: The MD5 application for finding collisions

IV. CONCLUSION

In this paper we show a survey of the MD5 hash function and many of its attacks. We demonstrate different attacks, beginning from the first attack after a few years of presenting the MD5 hash function, until the powerful attack of Wang and Klima. We give a brief explanation of their attacks, and inspired by Klima's application for finding a collision on a simple Notebook, we went far more beyond him. His generation of random numbers inspired us to take a step beyond his work. The division of the range of searching the numbers gives shorter and more specific time of searching. We show 2 methods of dividing the searching space, show which of them yields better results, and conclude the results of them.

As a future work we will work on an application that implements better methods for finding a random number in a given range, and that gives far better results of collisions in any hash function.

REFERENCES

[1] H. Dobbertin. *The Status of MD5 After a recent attack,* presented at the rump session of CryptoBytes *'96*. 1996

[2] X. Wang, H. Yu: *How to Break MD5 and Other Hash Functions*, Eurocrypt 2005, 2005.

[3] Vlastimil Klima: *Finding MD5 Collisions – a Toy For a Notebook*, Cryptology ePrint Archive, Report 2005/075.

[4] R. Rivest: *The MD5 Message Digest Algorithm*, RFC1321, Internet Activities Board, 1992.

[5] J. Black, M. Cochran, T. Highland. *A Study of the MD5 Attacks: Insights and Improvements*, Fast Software Encryption – FSE, Lecture Notes in Computer Science, Vol. 4047, Springer-Verlag, 2006.

[6] M. Stevens, *Fast Collision Attack on MD5*, Cryptography ePrint Archive, 2006

[7] Rogaway, P., Shrimpton T. *Cryptographic hash function basics: Definitions, implications and separations for preimage resistance, second-preimage resistance, and collision resistance*. Fast Software Encryption, Lecture Notes in Computer Science, 2004.

[8] R. Rivest: *The MD4 Message Digest Algorithm*, Proceedings of CRYPTO 1990, Springer Verlag 1991

Студија за MD5 колизии

Владимир Настески, Доц. Д-р Тони Стојановски
Факултет за информатика, ЕУРМ Скопје

Абстракт *– MD5 хаш функцијата е дизајнирана како подобрен наследник на MD4 хаш функцијата која е развиена од страна на Ron Rivest. Истата брзо достигна голема популарност, но кратко по нејзиното објавување, стана подложна на многу напади, преку кои што се согледа слабоста на функцијата. Методот на тунелирање од Klima стана најбрзиот и најразбирливиот метод за наоѓање на колизии на MD5 алгоритмот. Во оваа статија ги разгледуваме можностите за паралелна и дистрибуирана имплементација на методот на тунелирање на Klima. Исто така, во оваа статија теоретски споредуваме два паралелни и еден дистрибуиран метод на пребарување.*

Клучни зборови*: хаш, MD5, колизија, напад*