THE INTERNATIONAL JOURNAL OF SCIENCE & TECHNOLOGY

Dictionary Attack On Md5 Hashed Key Password

Shubham Goyal
Mtech, Department of Computer Science & Engineering, Yit, Jaipur, India
Monotosh Manna
Asst. Professor, Department of Computer Science & Engineering, Yit, Jaipur, India
Ankur Goyal
Asst. Professor, Department of Computer Science & Engineering, Yit, Jaipur, India

Abstract: *There exists many cryptographic one-way hash function which takes arbitrary length of a message as a input and produces fixed length message as output called as "fingerprint" or "message digest" of the input . The popular algorithm MD5 also takes aribitary length message as a input and produces 128 bit as output. The MD5 algorithm is intended for digital signature applications, where a large file must be "compressed" in a secure manner before being encrypted with a private (secret) key under a public-key cryptosystem and is commonly used to generates hash of passwords that is an encrypted form of password to be sent over insecure network. In this paper, we have primarily presented MD5 algorithm including simulation result and then several password cracking techniques like brute-forse attack, chosen-prefix collision attackand most important dictionary attacks [2] based on the MD5 algorithm.*

1.Introduction

Many cryptographic hash algorithm like SHA-1, SHA-2, SHA256, SHA-512, MD5 [1] exists in the literature and all these algorithms has one-way property, it means that no one can find the input from the given output or message digest. These algorithm including MD5 has great application in digital signature, key exchange protocol etc. There are basically two groups of hash function, keyed based hash function (MAC) and unkeyed based hash function. The MAC ensures the source of a message and integrity. In key based hash function, one key is needed to compute the hash like y = h(m,k) where m is the message and k is the key. Therefore, to compute y from a known message, the key k is required otherwise infeasible in polynomial time. A subclass of unkeyed hash functions are Modification Detection Codes MDCs. Contrary to MACs, we do not need a special key to compute the hash value y = h (m). Nevertheless, it should be computationally infeasible to find two messages (m, m*) such that h (m) = h (m*, k) .

A hash function with n bits output yields in 2^n possible output hash values. Due to the birthday paradox, we only have to compute $2^n/2$ different input messages to find a collision with a certain probability. If we are able to find a collision in less than $2^n/2$ the hash function is considered to be broken. Most hash functions used in practice are designed as iterated hash functions. The arbitrary input message m is split into k fixed length blocks mi (i=0 to k). Then, the compression function f is applied to every single message block mi and the result of the previous message block mi−1. For the application of the compression function to the first message block m0, we use a constant predefined initial value IV.

$$hi= f(mi, hi-1)^{k}_{i=0} \text{ with } ho=IV \text{ and } h(m) = h_k$$

1.1. Related Works

Vlastimil Klima et.al [3] showed how the conditions Wang presented in could be satisfied with several different ways of message modification. With this methods he decreases the complexity for finding a first message block pair M0−M0 ′ to $O(2^{39})$. This was 1000-2000 times faster than Wang's algorithm [4]. For the second message block he used a new message modification method which allows him to find M1 and M1′ with a complexity of $O(2^{39})$ or even $O(2^{24})$. Although finding this second message blocks is (2-80) times slower than in the entire algorithm for finding two colliding messages is (3-6) times faster. Therefore, a collision with arbitrary initial values can be found on a home computer within 8 hours. Sasaki et al. presented (independently from Vlastimil Klima) a collision attack in 2005 . Their work was based on the conditions of Wang algorithm [4], but adds some "extra conditions" for the second round of the compression function. With Wang's [4] message modification they satisfied 6 conditions for the 2nd round and left 37 uncorrected. With the help of these "extra conditions" Sasaki et al. [6] was now able to correct 14 conditions for the 2nd round – 29 conditions left. They experimentally found out that the probability for satisfying these conditions is 1/2 and therefore the complexity for finding two colliding messages is $O(2^{30})$. In 2006, John Black et al. [5] presented a paper where they described new ways of single message modification. For the multi message modification they used the algorithm of Klima [3] and the complexity for the entire collision algorithm was now $O(2^{30}) + O(2^{24})$. Further, they presented

possible new approaches to find collisions faster than existing approaches: The updates of the step variables in the round functions are not very random.

2. Overview of MD5 and Algorithm

Message Digest 5 (MD5) hash was developed by Rivest as an update to his previous MD4[7] hash and published in 1992. MD5, like other cryptographic hash algorithms, takes a message of arbitrary size and produces an output of fixed size (128 bits). In the following, we have presented flow diagram of MD5 algorithm where input messages is divided into 512 bits chunk and at the end padding is necessary and finally we get 128 bits as a output of the corresponding input message.

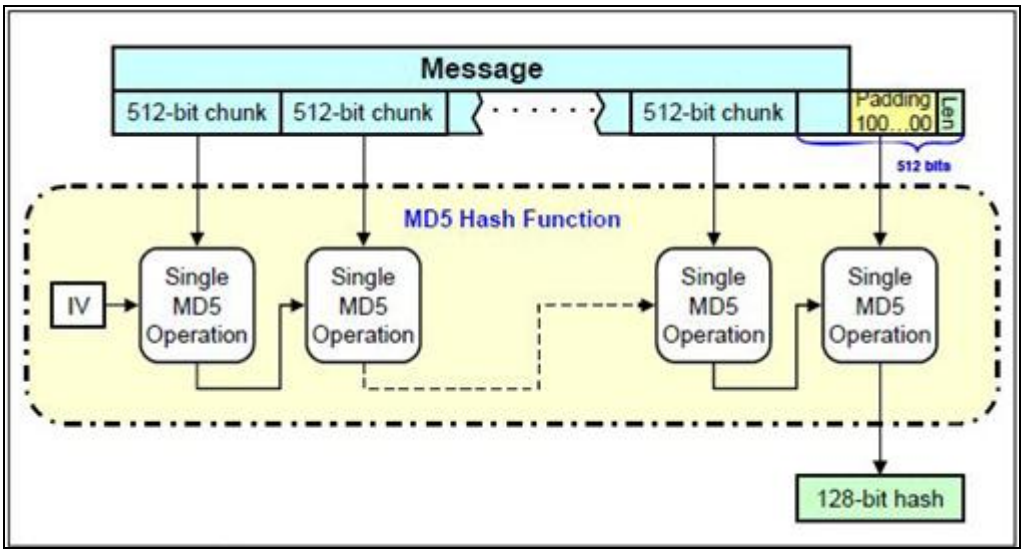

Figure 1: Illustration of MD5 Hash Function

2.1 MD5 Hash Algorithm

Input: m0,m1,m2,m3…….ms-1
Output: 128 bits

1. Construct M=M[0]M[1]…M[N-1]
2. A←67452301
3. B←EFCDAB89
4. C←98BADCFE
5. D←10325476
6. For i=0 to N/16 -1 Do
7. For j=0 to 15 do
8. X[j]=M[i*16+j]
9. A′ ← A
10. B′← B
11. C'← C
12. D′← D
13. Round1(Algo1)
14. Round1(Algo2)
15. Round1(Algo3)
16. Round1(Algo4)
17. A← A+ A'
18. B← B+ B′
19. C← C+ C'
20. D← D+ D′
21. end for;
22. end for;
23. h(m)= A OR B OR C OR D
24. end

2.2 Pseudo Code for MD5

```
1 MD5(message )
2 {
3 // definition of two arrays with constant s
4 r [ 6 4 ] = { 7 , 12 , 17 , 22 , 7 , 12 , 17 , 22 , 7 , 12 , 17 , 22
, 7 , 12 , 17 , 22 ,
5 5 , 9 , 14 , 20 , 5 , 9 , 14 , 20 , 5 , 9 , 14 , 20 , 5 , 9 , 14 , 20
,
6 4 , 11 , 16 , 23 , 4 , 11 , 16 , 23 , 4 , 11 , 16 , 23 , 4 , 11 , 16
, 23 ,
7 6 , 10 , 15 , 21 , 6 , 10 , 15 , 21 , 6 , 10 , 15 , 21 , 6 , 10 , 15
, 2 1};
8 for it = 0 . . . 6 3
9 k [ it ] = floor ( abs ( sin ( it+ 1 ) ) * 2 ^ 3 2 ) ;
10
11 // definition of initial values IV
12 IV1 = 0x67452301 ;
13 IV2 = 0xefcdab89 ;
14 IV3 = 0x98badcfe ;
15 IV4 = 0x10325476 ;
16
17 A = IV1 ; B = IV2 ; C = IV3 ; D = IV3 ;
18
19 // preprocess the message
20 append1BitToMessage (message ) ;
21 while (message .length != (448 mod 512) )
22 append 0 Bit To The Message (message ) ;
23 append 64Bit Message Length To The Message (message
) ;
24 M[ i ] = split message In 512 Bit Blocks (message ) ;
25
```

```
26 for every M[ i]
27 {
28 Q[−4] = A; Q[−3] = B; Q[−2] = C; Q[−1] = D; //define
step variables
29 m[ g ] = split Message Block In16Words ( ) ; //16 x
32−bit words
20
30
31 for step = 0 . . . 6 3
32 {
33 if step == 0 . . . 1 5 //round 1
34 phi = (Q[ step −1] & Q[ step −2]) | (˜Q[ step −2] & Q[
step −3 ] ) ;
35 g = step ;
36 if step == 1 6 . . . 3 1 //round 2
37 phi = (Q[ step −1] & Q[ step −3]) or (˜Q[ step −2] & Q[
step −3 ] ) ;
38 g = (5 *step + 1) mod 1 6 ;
39 if step == 3 2 . . . 4 7 //round 3
40 phi = Q[ step −1] XOR Q[ step −2] XOR Q[ step −3] ;
41 g = (5 *step + 1) mod 1 6 ;
42 if step == 4 8 . . . 6 3 //round 4
43 phi = Q[ step −2] XOR (Q[ step −1] | ˜Q[ step −3 ] ) ;
44 g = (7 *step + 1) mod 1 6 ;
45
46 // update step variable
47 Q[ step ] = ( (Q[ step −4] + phi + k [step ] + m[ g ] ) << r
[ step ] ) +Q[ step −1] ;
48 }
49
50 // update chaining variables
51 A = A + Q[ 6 0 ] ;
52 B = B + Q[ 6 3 ] ;
53 C = C + Q[ 6 2 ] ;
54 D = D + Q[ 6 1 ] ;
55 }
56 hash =concatenate (A, B, C, D) ; // hash value of the
entire input message
57 }
```

Simulation Result: In the simulation result, we have taken a word "Simon" and all the output of all the permutation of the given word.

Input:	**Output:**
Simon	F17CC04885C802BC9B4226AD8262D418
simno	E7FD6715FFE3DD54000350120DB56083
siomn	01FC005A7D1E94E1F238882352C348EB
sionm	0B3DC94E0B1A94C18D0B83671EDA3711
sinmo	E3C8ACF1EAB51CD75D28E061920C38AA
sinom	B9EFB33D2F761A91E5B18AC0C1F302F8
smion	9FA3E146920D7DDDE19B4C5949EC47DB
smino	7890C01D4E495EAC6C76B00C970E4021
smoin	44B4F0233BD44986D66A3D73D75FF428
smoni	F4AC5EB3CD4758606EA777163504B8CF
smnio	C6F199FC294822DE0220B7A3C24610E6
smnoi	1FD0150E35EAE248731AD103DB22D537
soimn	273CABAB7DEE45EDF28D402E29C090A6
soinm	0537B05B59F143845E1555B1EDBCF0FF
somin	B2B59438B598D0C80B871F54FA2E5D0F
0469C23CA8169E44D2F55987224AE534	

3. Several Attacks on MD5

3.1. Brute Force Attack

The attacker tries every possible key on a piece of cipher text until an intelligible translation into plaintext is obtained. On average, half of all possible keys to be tried achieve success.

3.2. Chosen-Prefix Collisions Attack

We present a novel, automated way to find differential paths for MD5. As an application we have shown how, at an approximate expected cost of 229 calls to the MD5 compression function, for any two chosen message prefixes P and P' suffix s and S and S' can be constructed such that the concatenated values P OR S and P′OR S' collide under MD5. The practical attack potential of this construction of chosen-prefix collisions is of greater concern than the MD5-collisions that were published before. This is illustrated by a pair of MD5-based X.509 certificates one of which was signed by a commercial Certification Authority (CA) as a legitimate website certificate, while the other one is a certificate for a rogue CA that is entirely under our control. Given two arbitrarily chosen messages, we first apply padding to the shorter of the two, if any, to make their lengths equal.

3.3. Dictionary Attack

A dictionary attack refers to the general technique of trying to guess some secret by running through a list of likely possibilities, often a list of words from a dictionary . It contrasts to a brute force attack in which all possibilities are tried. The attack works because users often choose easy to guess passwords, even after being exhorted against doing so. Dictionary attacks either pre-compute hash values for a given dictionary file and then compare target password hashes to the pre-computed tables for a match, or words in a dictionary file are hashed and compared against a target password hash for a match.

4.Conclusion and Future Work

This paper mainly discusses about the cryptographic one-way hash function MD5 and then presented algorithm with psedo-code for MD5. After that, we have taken a word and shown all the output of all the permutation word of that input word and finally, we have discussed the cryptographic dictionary attcks resistance technique. Generally, an interner user always chooses weak entropy password for easy to remember for which dictionary attack may occur but in future we try to approach strong resistance technique for the dictionary attack.

5.Refrences

1. William Stallings, Cryptography and Network Security-Principles and Practice, Third edition, Prentice Hall publications 2004.
2. Ronald Rivest. The MD5 Message-Digest Algorithm. RFC 1321, MIT, RSA Data Security, April 1992.
3. Vlastimil Klima. Finding md5 collisions a toy for a notebook. Cryptology ePrint Archive, Report 2005/075, 2005. http://eprint.iacr.org/.
4. Xiaoyun Wang and Hongbo Yu. How to break md5 and other hash functions. In EUROCRYPT, pages 19–35, 2005.
5. M. Cochran J. Black and T. Highland. A study on the md5 attacks: Insights and improvements, 2006.
6. Y. Sasaki, Y. Naito, N. Kunihiro, and K. Ohta. Improved collision attack on md. citeseer.ist.psu.edu/728500.html.
7. Hans Dobbertin. Cryptanalysis of md4. In Fast Software Encryption, pages 53–69,1996. http://world.std.com/~franl/crypto.html