MD5

The story of MD5 is more that of a failure than that of a cryptographic hash

function synthesized to protect data.

In spite of all the shortcomings of this algorithm it is still used extensively for

<small side purposes>.

This algorithm first came into existence when Professor Ronald Rivest of MIT

discovered that his previous algorithm, MD4, was found to be analytically insecure.

It was designed in the year 1991 to be a secure replacement for the previously popular

cryptographic hash function in the series of message digest algorithms.

MD5's spotlight was short lived when in 1993 Den Boer and Bosselaers published results

of pseudo-collisions of the MD5 function.

Since then many attempts have been made to find a practical collision for this

algorithm and finally in 2004 a group of Chinese researchers found a full MD5 collision which took only one hour on an IBM p690 cluster.

Currently MD5 is used to provide checksums for files downloaded over the internet to verify their integrity by comparing it to the original MD5 checksum of the file on the server. Although not a 100% secure method it is just used to check errors in the received files as it is not safe against tampering.

-Functions, constants used

word A: 01 23 45 67

word B: 89 ab cd ef

word C: fe dc ba 98

word D: 76 54 32 10

The following array is used in the computation of the MD5 hash

**for** i **from** 0 **to** 63  
 K[i] := floor(232 × abs(sin(i + 1)))

The following bitwise operations are used in the Functions

XOR, AND, OR and NOT.

The functions used are:

F(X,Y,Z) = XY v not(X) Z

G(X,Y,Z) = XZ v Y not(Z)

H(X,Y,Z) = X xor Y xor Z

I(X,Y,Z) = Y xor (X v not(Z))

-Algorithm

-Padding

Let the length of the messgae be L.

Append 1 to the message and add k amount of zeroes such that the total message length is equivalent to 448 (mod 512) i.e. L + 1 + k ≡ 448 (mod 512).Then append the 64-bit block that is equal to the number L expressed using a binary representation.

-Parsing

The padded message is then looped 512 bits at a time where the 512 bits are divided into 16 word of 32 bits each and stored in M[j], 0 < j <= 15.

-Algorithm

**var** *int* A := word A  
 **var** *int* B := word B  
 **var** *int* C := word C  
 **var** *int* D := word D  
//*Main loop:*  
 **for** i **from** 0 **to** 63  
 **var** *int* F, g  
 **if** 0 ≤ i ≤ 15 **then**  
 F := (B **and** C) **or** ((**not** B) **and** D)  
 g := i  
 **else if** 16 ≤ i ≤ 31  
 F := (D **and** B) **or** ((**not** D) **and** C)  
 g := (5×i + 1) **mod** 16  
 **else if** 32 ≤ i ≤ 47  
 F := B **xor** C **xor** D  
 g := (3×i + 5) **mod** 16  
 **else if** 48 ≤ i ≤ 63  
 F := C **xor** (B **or** (**not** D))  
 g := (7×i) **mod** 16  
//*Be wary of the below definitions of a,b,c,d*  
 F := F + A + K[i] + M[g]  
 A := D  
 D := C  
 C := B  
 B := B + **leftrotate**(F, s[i])  
 **end for**  
//*Add this chunk's hash to result so far:*  
 a0 := a0 + A  
 b0 := b0 + B  
 c0 := c0 + C  
 d0 := d0 + D

The message digest then becomes (a0 append b0 append c0 append d0).

-FALL

Collision vulnerability

As mentioned in the introduction the MD5 algorithm was was severely compromised due to the fact that many collisions were found of the MD5 algorithm and there were many instances of forgery where people forged fake digital certificates. In 2012, according to Microsoft, the authors of the Flame malware used an MD5 collision to forge a Windows code-signing certificate.

Pre-image vulnerability

Although pre-image vulnerabilities are rare in hashing functions as it is a one way function in April 2009, a preimage attack against MD5 was published that breaks MD5's preimage resistance. This attack is only theoretical, with a computational complexity of 2^123.4 for full preimage.

SHA 1

-Introduction

SHA-1 (Secure Hash Algorithm 1) is a cryptographic hash function which takes

an input and produces a 160-bit (20-byte) hash value known as a message

digest - typically rendered as a hexadecimal number, 40 digits long.

It was designed by the United States National Security Agency, and is a U.S. Federal

Information Processing Standard.

SHA-1 produces a message digest based on principles similar to those used in

the design of the MD4 and MD5 message digest algorithms, but has a more conservative design.

The original specification of the algorithm was published in 1993 under the title Secure Hash Standard.

This version is now often named SHA-0. It was withdrawn by the NSA shortly after

publication and was superseded by the revised version, published in 1995 and commonly designated SHA-1.

SHA-1 differs from SHA-0 only by a single bitwise rotation in the message

schedule of its compression function. According to the NSA, this was done to

correct a flaw in the original algorithm which reduced its cryptographic security,

but they did not provide any further explanation.

The algorithm has also been used on Nintendo's Wii gaming console for signature

verification when booting, but a significant flaw in the first implementations

of the firmware allowed for an attacker to bypass the system's security scheme.

Revision control systems such as Git, Mercurial, and Monotone use SHA-1 not for

security but to identify revisions and to ensure that the data has not changed due to accidental corruption.

Examples:

These are examples of SHA-1 message digests in hexadecimal:

SHA1("The quick brown fox jumps over the lazy dog")

gives hexadecimal: 2fd4e1c67a2d28fced849ee1bb76e7391b93eb12

SHA1("The quick brown fox jumps over the lazy cog")

gives hexadecimal: de9f2c7fd25e1b3afad3e85a0bd17d9b100db4b3

-Functions, constants used

Word A: 5a827999

Word B: 6ed9eba1

Word C: 8f1bbcdc

Word D: ca62c1d6

Five working variables labeled as : a,b,c,d & e.

Initial Hash Values: H0,H1,H2,H3 & H4.

H 0 ( 0 ) = 67452301

H 1 ( 0 ) = efcdab89

H 2 ( 0 ) = 98badcfe

H 3 ( 0 ) = 10325476

H 4 ( 0 ) = c3d2e1f0

The following bitwise operations are used in the Functions

XOR, AND, OR and NOT.

-Algorithm:

-Padding:

The purpose of this padding is to ensure that the padded message is a multiple of 512 bits.

Suppose that the length of the message, M, is L bits. Append the bit “1” to the end of the

message, followed by k zero bits, where k is the smallest, non-negative solution to the equation

L + 1 + k ≡ 448 mod 512 . Then append the 64-bit block that is equal to the number L expressed

using a binary representation. //01100001 01100010 01100011 1 00...00 00...011000

-Parsing:

The message and its padding are parsed into N 512-bit

blocks, M (1) , M (2) ,..., M (N) . Since the 512 bits of the input block may be expressed as sixteen 32- bit words, the first 32 bits of message block i are denoted M 0 ( i ) , the next 32 bits are M 1 ( i ) , and so on up to M 15 ( i ) .

-Setting the Initial hash Values:

For SHA-1, the initial hash value, H (0) , shall consist of the following five 32-bit words, in hex:

H 0 ( 0 ) = 67452301

H 1 ( 0 ) = efcdab89

H 2 ( 0 ) = 98badcfe

H 3 ( 0 ) = 10325476

H 4 ( 0 ) = c3d2e1f0

-Algorithm:

Each message block, M (1) , M (2) , ..., M (N) , is processed in order, using the following steps:

For i=1 to N:

{

1. Prepare the message schedule, {W t }:

M t (i ) 0 ≤ t ≤ 15

ROTL 1 ( W t − 3 ⊕ W t − 8 ⊕ W t − 14 ⊕ W t − 16 ) 16 ≤ t ≤ 79

W t =

2. Initialize the five working variables, a, b, c, d, and e, with the (i-1) st hash value:

a = H 0 ( i − 1 )

b = H 1 ( i − 1 )

c = H 2 ( i − 1 )

d = H 3 ( i − 1 )

e = H 4 ( i − 1 )

3. For t=0 to 79:

{

T = ROTL 5 ( a ) + f t ( b , c , d ) + e + K t + W t

e = d

d = c

c = ROTL 30 ( b )

b = a

a = T

}

4. Compute the i th intermediate hash value H (i) :

H 0 ( i ) = a + H 0 ( i − 1 )

H 1 ( i ) = b + H 1 ( i − 1 )

H 2 ( i ) = c + H 2 ( i − 1 )

H 3 ( i ) = d + H 3 ( i − 1 )

H 4 ( i ) = e + H 4 ( i − 1 )

}

After repeating steps one through four a total of N times (i.e., after processing M (N) ), the resulting

160-bit message digest of the message, M, is

H 0 ( N ) || H 1 ( N ) || H 2 ( N ) || H 3 ( N ) || H 4 ( N )

-Fall of SHA1:

Since 2005 SHA-1 has not been considered secure against well-funded opponents.

Microsoft, Google, Apple and Mozilla have all announced that their respective browsers will stop accepting SHA-1 SSL certificates by 2017.

In 2017 CWI Amsterdam and Google announced they had performed a collision attack

against SHA-1, publishing two dissimilar PDF files which produced the same SHA-1 hash.

For a hash function for which L is the number of bits in the message digest, finding

a message that corresponds to a given message digest can always be done using a brute

force search in approximately 2^L evaluations. This is called a preimage attack and

may or may not be practical depending on L and the particular computing environment.

However, a collision, consisting of finding two different messages that produce the

same message digest, requires on average only about 1.2 × 2^(L/2) evaluations using

a birthday attack. Thus the strength of a hash function is usually compared to a

symmetric cipher of half the message digest length. SHA-1, which has a 160-bit

message digest, was originally thought to have 80-bit strength.

In 2005, cryptographers Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu produced collision

pairs for SHA-0 and have found algorithms that should produce SHA-1 collisions in far

fewer than the originally expected 2^80 evaluations.

In February 2005, an attack by Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu was

announced. The attacks can find collisions in the full version of SHA-1, requiring fewer than 2^69 operations

SHA-2

-Introduction

SHA-2 is the cryptographic hashing standard that all software and hardware should be using now, at least for the next few years. SHA-2 is often called the SHA-2 family of hashes because it contains many different-size hashes, including 224-, 256-, 384-, and 512-bit digests.

Although SHA-2 shares some of the same math characteristics as SHA-1 and minor weaknesses have been discovered, in crypto-speak it's still considered "strong” for the foreseeable future. Without question, it's way better than SHA-1, and any critical SHA-1 enabled certificates, applications, and hardware devices using SHA-1 should be moved to SHA-2.

SHA-256 partakes in the process of authenticating Debian software packages.

Several cryptocurrencies like Bitcoin use SHA-256 for verifying transactions and calculating proof-of-work or proof-of-stake.

The SHA-2 functions were not quickly adopted initially, despite better security than SHA-1. Reasons might include lack of support for SHA-2 on systems running Windows XP SP2 or olderand a lack of perceived urgency since SHA-1 collisions had not yet been found.

-Functions and constants:

SHA-224 and SHA-256 use the same sequence of sixty-four constant 32-bit words,

{ 256 }

. These words represent the first thirty-two bits of the fractional parts of

K 0 { 256 } , K 1 { 256 } , , K 63

the cube roots of the first sixty-four prime numbers. In hex, these constant words are (from left

to right)

0x428a2f98, 0x71374491, 0xb5c0fbcf, 0xe9b5dba5, 0x3956c25b, 0x59f111f1, 0x923f82a4, 0xab1c5ed5,  
 0xd807aa98, 0x12835b01, 0x243185be, 0x550c7dc3, 0x72be5d74, 0x80deb1fe, 0x9bdc06a7, 0xc19bf174,  
 0xe49b69c1, 0xefbe4786, 0x0fc19dc6, 0x240ca1cc, 0x2de92c6f, 0x4a7484aa, 0x5cb0a9dc, 0x76f988da,  
 0x983e5152, 0xa831c66d, 0xb00327c8, 0xbf597fc7, 0xc6e00bf3, 0xd5a79147, 0x06ca6351, 0x14292967,  
 0x27b70a85, 0x2e1b2138, 0x4d2c6dfc, 0x53380d13, 0x650a7354, 0x766a0abb, 0x81c2c92e, 0x92722c85,  
 0xa2bfe8a1, 0xa81a664b, 0xc24b8b70, 0xc76c51a3, 0xd192e819, 0xd6990624, 0xf40e3585, 0x106aa070,  
 0x19a4c116, 0x1e376c08, 0x2748774c, 0x34b0bcb5, 0x391c0cb3, 0x4ed8aa4a, 0x5b9cca4f, 0x682e6ff3,  
 0x748f82ee, 0x78a5636f, 0x84c87814, 0x8cc70208, 0x90befffa, 0xa4506ceb, 0xbef9a3f7, 0xc67178f2

-Padding:

Same as that of SHA 1

-Parsing:

Same as that of SHA 1

-Setting the initial Hash value:

For SHA-256, the initial hash value, H (0) , shall consist of the following eight 32-bit words, in

hex:

H 0 ( 0 ) = 6a09e667

H 1 ( 0 ) = bb67ae85

H 2 ( 0 ) = 3c6ef372

H 3 ( 0 ) = a54ff53a

H 4 ( 0 ) = 510e527f

H 5 ( 0 ) = 9b05688c

H 6 ( 0 ) = 1f83d9ab

H 7 ( 0 ) = 5be0cd19

These words were obtained by taking the first thirty-two bits of the fractional parts of the square

roots of the first eight prime numbers.

-Hash Function:

Each message block, M (1) , M (2) , ..., M (N) , is processed in order, using the following steps:

For i=1 to N:

{

1. Prepare the message schedule, {W t }:

M t (i ) 0 ≤ t ≤ 15

σ 1 { 256 } ( W t − 2 ) + W t − 7 + σ 0 { 256 } ( W t − 15 ) + W t − 16 16 ≤ t ≤ 63

W t =

2. Initialize the eight working variables, a, b, c, d, e, f, g, and h, with the (i-1) st hash

value:

a = H 0 ( i − 1 )

b = H 1 ( i − 1 )

c = H 2 ( i − 1 )

d = H 3 ( i − 1 )

e = H 4 ( i − 1 )

f = H 5 ( i − 1 )

g = H 6 ( i − 1 )

h = H 7 ( i − 1 )

3. For t=0 to 63:

{

T 1 = h + ∑ 1

{ 256 }

T 2 = ∑ 0

{ 256 }

( e ) + Ch ( e , f , g ) + K t { 256 } + W t

( a ) + Maj ( a , b , c )

h = g

g = f

f = e

e = d + T 1

d = c

c = b

b = a

a = T 1 + T 2

}

4. Compute the i th intermediate hash value H (i) :

H 0 ( i ) = a + H 0 ( i − 1 )

H 1 ( i ) = b + H 1 ( i − 1 )

H 2 ( i ) = c + H 2 ( i − 1 )

H 3 ( i ) = d + H 3 ( i − 1 )

H 4 ( i ) = e + H 4 ( i − 1 )

H 5 ( i ) = f + H 5 ( i − 1 )

H 6 ( i ) = g + H 6 ( i − 1 )

H 7 ( i ) = h + H 7 ( i − 1 )

}

After repeating steps one through four a total of N times (i.e., after processing M (N) ), the resulting

256-bit message digest of the message, M, is

H 0 ( N ) H 1 ( N ) H 2 ( N ) H 3 ( N ) H 4 ( N ) H 5 ( N ) H 6 ( N ) H 7 ( N )

-Comparison:

All times below are per 1 000 000 calculations:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Hash** | **36 char length String (ms)** | **49 char Length String (ms)** | **72 Char Length String (ms)** | **85 Char Length String (ms)** |
| MD5 | 627.4 | 765.6 | 839 | 1029.4 |
| SHA-1 | 604 | 748.2 | 916.8 | 1009.6 |
| SHA-256 | 737.8 | 851 | 1168.2 | 1260 |
| SHA-512 | 1056.4 | 1158.8 | 1118.4 | 1227.4 |

Ref: <https://automationrhapsody.com/md5-sha-1-sha-256-sha-512-speed-performance/>

## 

Inference

Some conclusions of the results based on two cases with short string (36 and 49 chars) and longer string (72 and 85 chars).

* SHA-256 is faster with 31% than SHA-512 only when hashing small strings. When string are longer SHA-512 is faster with 2.9%.
* Time to get system time stamp is ~121.6 ms per 1M iterations.
* SHA-1 is fastest hashing function with ~587.9 ms per 1M operations for short strings and 881.7 ms per 1M for longer strings.
* MD5 is 7.6% slower than SHA-1 for short strings and 1.3% for longer strings.
* SHA-256 is 15.5% slower than SHA-1 for short strings and 23.4% for longer strings.
* SHA-512 is 51.7% slower that SHA-1 for short strings and 20% for longer.