SC205 Project

SHA

[Secure Hash Algorithm]

Jinesh Y. Kanjiya : 201901412

May 2020

Assigned by : Prof. Manish K. Gupta

1 Introduction

⇒ A hash function is any function that can be used to map data of arbitrary size to fixed-size values. The values returned by a hash function are called hash values, hash codes, digests, or simply hashes. The values are used to index a fixed-size table called a hash table. Use of a hash function to index a hash table is called hashing or scatter storage addressing.

Secure Hash Algorithms, also known as SHA, are a family of cryptographic functions designed to keep data secured. It works by transforming the data using a hash function: an algorithm that consists of bitwise operations, modular additions, and compression functions.They are published by the National Institute of Standards and Technology (NIST) as a U.S. Federal Information Processing Standard (FIPS). [1]

SHA-0, SHA-1, SHA-2, SHA-3, MD4, MD5, RipeMD-150, RipeMD-256, SHA-256, SHA-512 and CRC32 are different types of hash generators with different algorithms and different digest length.

Figure 1 shows how the digest changes , with the small change in input.

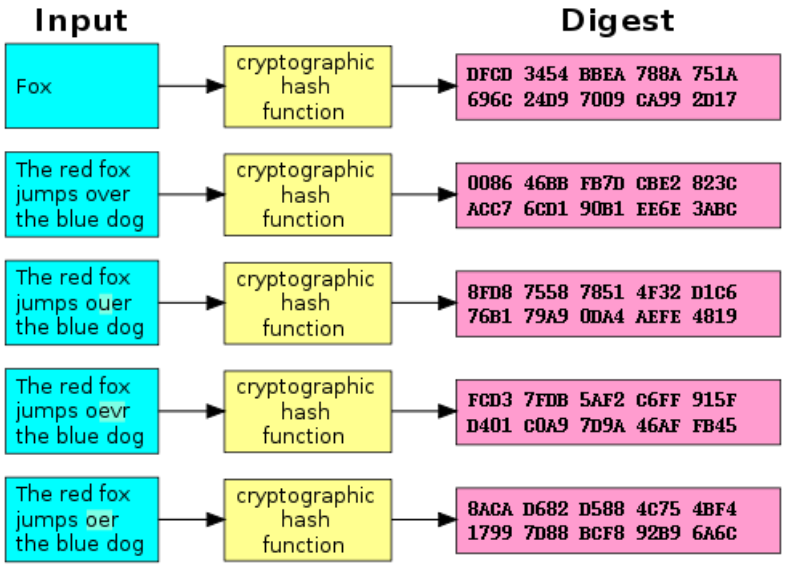

Figure 1: Input & Digest

Figure 2 shows the size of digest for different Hash Algorithms, message block size and the presence of collision.

Table 2. Comparison of Multiple Hash Functions

Algorithm	The Size of the Message Digest (bit)	Message Block Size	Collision
MD2	128	128	Yes
MD4	128	512	Almost
MD5	128	512	Yes
RIPEMD	128	512	Yes
RIPEMD-128/256	128/256	512	No
RIPEMD-160/320	160/320	512	No
SHA-0	160	512	Yes
SHA-1	160	512	There is a Disability
SHA-256/224	256/224	512	No
SHA-512/384	512/384	1024	No
WHIRPOOL	512	512	No

Figure 2: Comparison of multiple hash fun.

Collision means when two or more digests are same for the different input.Let y=h(x) ,where x is input, h() is the hash function and y is the output(digest). If for two different x_1 and x_2, if $h(x_1) = h(x_2)$, then there will be collision.

[NOTE : Here we are focusing on the Hash Function Generation. We are not discussing about the collision on hash.]

As shown in the figure, MD2, MD4, MD5, SHA-0 have reported collision. In SHA-1, collision is not found yet. While collisions can be avoided in SHA-256, SHA-224, SHA-512, and WHIRPOOL algorithms.

Figure 3 shows the digest made out of different Hash Algorithms when the input is 'abc123_##'.

NTLM	FD44EC6F4B66468210115F673C60048B	MD2	10b9c738761fea8d076d4d09120f4504
MD4	fe19f1b0c25009375c0970dadf0591bd	MD5	17e4ba521f79693f8330ff9d5ba57210
MD6-128	beba4bfaee14f47805edd3e27f5593c3	MD6-256	6c8a570cf82ab999e4984726614181d0f355ca493a
MD6-512	db85152ce1f5426c4a1dca5d7dc7d35c3e0a0e873a	RipeMD-128	0ecfcd50b880ea5e1fa5a494f6a24929
RipeMD-160	ba3dbb8f0d311e18b207b49ea61724f562236636	RipeMD-256	5c0eca303404880266d184747521bdcad1a0cbf861
RipeMD-320	454eb100298fe1236d63a89ff5ceab4d2f0602aa443	SHA1	a56fbf03831953b35919177412d82acdc05e7ebb
SHA3-224	22144fda7a4571fccf099976bc237c2e64d0b24b22c	SHA3-256	ffb169802bbfdab60437b486d5d0994498bb365c59
SHA3-384	dbecdb9f437659b5a463a5eceb08315436cfdef64b	SHA3-512	9ae2254d06a959e5aabee76509e9b4d5c3c845d82
SHA-224	625536a0d32727a52ae062001706b7193a7f7dab5	SHA-256	ef426f6ad63fe292b1d1656d643bb40ceb8efce61fd
SHA-384	8edb93794bd455aff684267787235e9ec07206d292	SHA-512	2ee92f2c31fc3d3f8483ac4498ca3c59aaaa76e7828
CRC16	3f2b	CRC32	8f677c8
Adler32	0da90262	Whirlpool	9b1590e14492266ab051082f74610df966754445a

Figure 3: Digest from diff. Hash Fun.

You can check this in the following link : https://www.browserling.com/tools/all-hasheslink

→ Properties of hash algorithm :-

- It is deterministic, meaning that the same message always results in the same hash.
- It is irreversible. Given a hash value h, it should be difficult to find any message x such that h = hash(x). This concept is related to that of **a one-way function** .
- Given an input x_1, it should be difficult to find a different input x_2 such that $hash(x_1) = hash(x_2)$.

2 SHA-1 Algorithm

[Formulate the Mathematics]

⇒ Here we specifically discuss the algorithm of SHA-1 [2]. It 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.The steps to obtain digest is as follows...

1. First we convert the input in an array of ASCII values of the characters. Let us assume there are $\mathcal{N}$ characters in input.

2. Now, make a new array (named 'messagebits') converting ASCII values in binary with total 8 bits each, so the length of array messagebits would be $\mathcal{N}$x8. [Prepend 0's if it is not 8 bits long]

3. Append 1 at the end of messagebits-array.

4. Append 0's at the end of messagebits array untill length(messagebits) % 512 = 448. So, the new length of messagebits would be in the form of $512(k) + 448.(k \in \mathbf{W})$

5. Now,make a new array (named 'ml') which is of 64 bits. It contains the length of messagebits array in step 2 (i.e., $\mathcal{N}$x8) in binary form. Prepend 0's in-ml array untill length(ml) % 64 = 0 .

6. Now, append ml-array to the end of messagebits-array. Because of this, the length of messagebits-array is now in the multiple of 512.

   Now the main part of the algorithm begins...

7. Initialize variables : Variables are representing here in hexadecimal form.
   A=[6 7 4 5 2 3 0 1]
   B=[e f c d a b 8 9]
   C=[9 8 b a d c f e]
   D=[1 0 3 2 5 4 7 6]
   E=[c 3 d 2 e 1 f 0]
   Converting it in binary form (i.e., 4 bits for each digit) , the size of A, B, C, D, E is 32.

8. Divide messagebits into chunks of 512-bits.
   For each chunk, repeate steps a, b, c, and d.

(a) Initialize :

a = A , b = B , c = C , d = D , e = E

(b) Break chunk into 16 word (w[i]) of 32-bits , $1 \leq i \leq 16$. Extend 16 32-bit words into 80 32-bit words.

w[i] = (w[i-3] **xor** w[i-8] **xor** w[i-14] **xor** w[i-16]) **leftrotate** 1 $[17 \leq i \leq 80]$

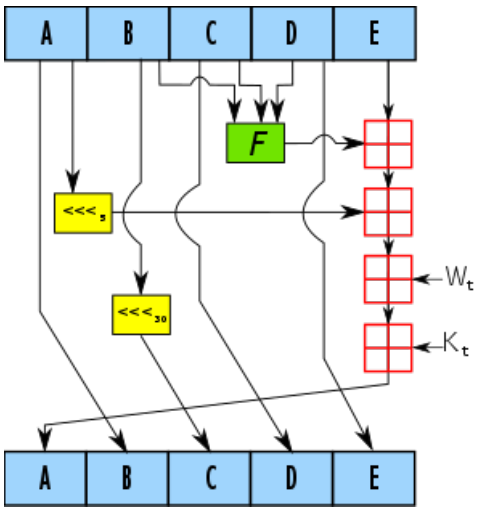

Figure 4: SHA-1

(c) **Main loop...(80 iteration for each chunk)**

for $1 \leq i \leq 80$

- if $1 \leq i \leq 20$,
  → F = (b **and** c) **or** ((**not**b) **and** d)
  → k = [5 a 8 2 7 9 9 9]
- elseif $21 \leq i \leq 40$,
  → F = b **xor** c **xor** d
  → k = [6 e d 9 e b a 1]
- elseif $41 \leq i \leq 60$,
  → F = (b **and** c) **or** (b **and** d) **or** (c **and** d)
  → k = [8 f 1 b b c d c]
- elseif $61 \leq i \leq 80$,
  → F = b **xor** c **xor** d
  → k = [c a 6 2 c 1 d 6]

- temp = (a **leftrotate** 5) + F + e + k + w[i]
  e = d
  d = c
  c = b **leftrotate** 30
  b = a
  a = temp

(d) Add chunk's hash to result so far,
A = A + a
B = B + b
C = C + c
D = D + d
E = E + e

[NOTE : The value of 'k' is constant and is changes 4 times in 80 iterations. Here, every variable is mod 2^{32} e.g., temp variable may exceeds 32-bits, but we have to neglect the starting bits other than 32 bits. Similarly, in step (d).]

9. Merge all variables in single array.
hh = [A B C D E]
Thus, hh-array is of size 5 x 32-bits = 160-bits. Converting it into hexadecimal, we get 160/4 = 40 digits of hexadicimal string.

3 Solving The Mathematics

⇒ In this section we should solve the mathematics described above. Let’s take the input as ‘abc123_##’.

1. converting into the array of ASCII value,

   inp=[97 98 99 49 50 51 95 35 35]

2. messagebits =

   [01100001 01100010 01100011 00110001 00110010 00110011 01011111 00100011 00100011]

3. Appending 1 at the end, messagebits =

   [01100001 01100010 01100011 00110001 00110010 00110011 01011111 00100011 00100011 1]

4. Appending 0’s at the end,size of array becomes 448. messagebits =

   [01100001 01100010 01100011 00110001 00110011 01011111 00100011 00100011 10000000 00000000]

5. Here $\mathcal{N} = 9$,so converting $\mathcal{N}$ x $8 = 72$ in binary, prepending 0’s to the front and extending the length of ml-array to 64-bits long, ml =

   [00000000 00000000 00000000 00000000 00000000 00000000 00000000 01001000]

6. Appending messagebits-array with ml-array, messagebits-array =

   [01100001 01100010 01100011 00110001 00110011 01011111 00100011 00100011 10000000 01001000]

7. Initializing variables

   A = [0110 0111 0100 0101 0010 0011 0000 0001]

   B = [1110 1111 1100 1101 1010 1011 1000 1001]

   C = [1001 1000 1011 1010 1101 1100 1111 1110]

   D = [0001 0000 0011 0010 0101 0100 0111 0110]

   E = [1100 0011 1101 0010 1110 0001 1111 0000]

8. After completing the 8^{th} step, the new updated A, B, C, D, and E are

   A = [1010 0101 0110 1111 1011 1111 0000 0011]

   B = [1000 0011 0001 1001 0101 0011 1011 0011]

   C = [0101 1001 0001 1001 0001 0111 0111 0100]

   D = [0001 0010 1101 1000 0010 1010 1100 1101]

   E = [1100 0000 0101 1110 0111 1110 1011 1011]

9. Merging all arrays, hh =

   [1010 0101 0110 1111 1011 1111 0000 0011 1000 0011 0001 1001 0101 0011 1011 0011 0101 1001 0001 1001 0001 0111 0111 0100 0001 0010 1101 1000 0010 1010 1100 1101 1100 0000 0101 1110 0111 1110 1011 1011]

   Converting hh-array into hexadecimal, we get our 40-digit long digest as

   a56fbf03831953b35919177412d82acdc05e7ebb

4 Interpretation And Significance

⇒ SHA-1 has been deployed as an important component in various cryptographic schemes and protocols, such as user authentication, key agreement, and pseudorandom number generation. Consequently, SHA-1 has been widely implemented in almost all commercial security systems and products.

Since its publication, SHA-1 has been adopted by many government and industry security standards, in particular standards on digital signatures for which a collision-resistant hash function is required.

The SHA-1 hashing function was theoretically broken in 2005; however, the first successful collision attack in the real world was carried out in 2017. [3]

As of 2020, attacks against SHA-1 are as practical as against MD5; as such, it is recommended to remove SHA-1 from products as soon as possible and use instead SHA-256 or SHA-3. Replacing SHA-1 is urgent where it is used for signatures.

References

[1] Sha-1. https://en.wikipedia.org/wiki/SHA-1.

[2] Penny Pritzker. *Secure Hash Standard (SHS).* FEDERAL INFORMATION PROCESSING STANDARDS(FIPS) PUB 180-4, August 2015.

[3] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. *Finding Collisions in the Full SHA-1.*