International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Impact Factor (2012): 3.358

Review Paper on Secure Hashing Algorithm and Its Variants

Priyanka Vadhera[1], Bhumika Lall[2]

[1, 2]Department of Computer Science B.S. Anangpuria Institute of Technology and Management, India

Abstract: ***SHA stands for "secure hash algorithm". The four SHA algorithms are structured differently and are named SHA-0, SHA-1, SHA-2, and SHA-3. Secure hashing algorithm is a method that produces a message digest based on principles similar to those used in the design of the MD4 and MD5message digest algorithms, but has a more conservative design.SHA appears to provide greater resistance to attacks, supporting the NSA's assertion that the change increased the security. This is a review paper which includes the comparisons between different secure hashing algorithms.***

Keywords: SHA-1, SHA-2, SHA-512, message digest, data integrity, message authentication

1. Introduction

Cryptographic hash functions have many information security applications, notably in digital signatures, message authentication codes (MACs), and other forms of authentication. They can also be used as ordinary hash functions, to index data in hash tables, for fingerprinting, to detect duplicate data or uniquely identify files, and as checksums to detect accidental data corruption. Indeed, in information security contexts, cryptographic hash values are sometimes called (digital) finger prints, checksums, or just hash values, even though all these terms stand for more general functions with rather different properties and purposes.

A cryptographic hash function is a hash function that takes an arbitrary block of data and returns a fixed-size bit string, the cryptographic hash value, such that any (accidental or intentional) change to the data will (with very high probability) change the hash value. The data to be encoded are often called the message, and the hash value is sometimes called the message digest or simply digests the methods resemble the block cipher modes of operation usually used for encryption. All well-known hash functions, including MD4, MD5, SHA-1 and SHA-2 are built from block-cipher-like components designed for the purpose, with feedback to ensure that the resulting function is not invertible. SHA-3 finalists included functions with block-cipher-like components though the function finally selected, was built on a cryptographic sponge instead.

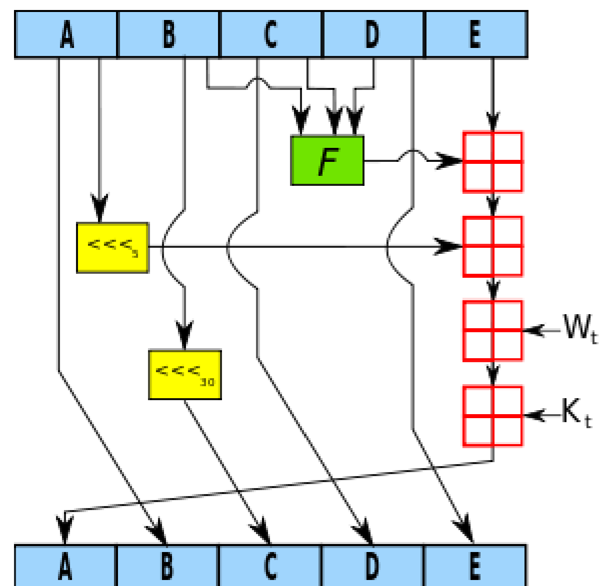
A standard block cipher such as AES can be used in place of these custom block ciphers; that might be useful when an embedded system needs to implement both encryption and hashing with minimal code size or hardware area. However, that approach can have costs in efficiency and security. The ciphers in hash functions are built for hashing: they use large keys and blocks, can efficiently change keys every block, and have been designed and vetted for resistance to related-key attacks. General-purpose ciphers tend to have different design goals. In particular, AES has key and block sizes that make it nontrivial to use to generate long hash values; AES encryption becomes less efficient when the key changes each block; and related-key attacks make it potentially less secure for use in a hash function than for encryption.

1.2 SHA-0

A retronym applied to the original version of the 160-bit hash function published in 1993 under the name "SHA". It was withdrawn shortly after publication due to an undisclosed "significant flaw" and replaced by the slightly revised version SHA-1.

1.3 SHA-1

SHA-0 is the original version of the 160-bit hash function SHA-1 is very similar to SHA-0, but alters the original SHA hash specification to correct alleged weaknesses. SHA-1[1,5] is the most widely used of the existing SHA hash functions, and is employed in several widely used applications and protocols.



SHA-1 produces a message digest based on principles similar to those used by Ronald L. Rivest of MIT in the design of the MD4 and MD5message digest algorithms, but has a more conservative design. SHA-1 differs from SHA-0 only by a single bitwise rotation in the message schedule of its compression function; this was done, according to the NSA, to correct a flaw in the original algorithm which reduced its cryptographic security. However, the NSA did

Licensed Under Creative Commons Attribution CC BY

not provide any further explanation or identify the flaw that was corrected. Weaknesses have subsequently been reported in both SHA-0 and SHA-1. SHA-1 appears to provide greater resistance to attacks, supporting the NSA's assertion that the change increased the security.

SHA-1 forms part of several widely used security applications and protocols, including TLS and SSL, PGP, SSH, S/MIME, and IPSec. Those applications can also use MD5; both MD5 and SHA-1 are descended from MD4. SHA-1 hashing is also used in distributed revision control systems like Git, Mercurial, and Monotone to identify revisions, and to detect data corruption or tampering. The algorithm has also been used on Nintendo's Wii gaming console for signature verification when booting, but a significant implementation flaw allows for an attacker to bypass the system's security scheme

Nobody has been able to break SHA-1, but the point is the SHA-1, as far as Git is concerned, isn't even a security feature. It's purely a consistency check. The security parts are elsewhere, so a lot of people assume that since Git uses SHA-1 and SHA-1 is used for cryptographically secure stuff, they think.

Due to the block and iterative structure of the algorithms and the absence of additional final steps, all SHA functions are vulnerable to length-extension and partial-message collision attacks.[15] These attacks allow an attacker to forge a message signed only by a keyed hash – SHA (message||key)or SHA(key||message) – by extending the message and recalculating the hash without knowing the key. The simplest improvement to prevent these attacks is to hash twice: SHA_d(message)=SHA(SHA(0^b||message)) (the length of 0^b, zero block, is equal to the block size of hash function).
These are examples of SHA-1 message digests in hexadecimal and in Base64 binary to ASCII text encoding.

Example:

SHA (“The quick brown fox jumps over the lazy dog”)

Gives hexadecimal:

2fd4e1c67a2d28fced849ee1bb76e7391b93eb12

Even a small change in the message will, with overwhelming probability, result in a completely different hash due to the avalanche effect.

SHA1 (“”) Gives hexadecimal:

da39a3ee5e6b4b0d3255bfef95601890afd80709

1.3.1 SHA-1 PSEUDOCODE

Pseudocode for the SHA-1 algorithm follows:

```
Step1:
initialize all the variables
ml = message length in bits (always a multiple of the number of bits in a character).

Step2:
Pre-processing:
append the bit '1' to the message i.e. by adding 0x80 if characters are 8 bits.
Step 3:
Process the message in successive 512-bit chunks:
break message into 512-bit chunks
for each chunk
   break chunk into sixteen 32-bit big-endian words w[i], 0 ≤ i ≤ 15
Step 4 :
   Extend the sixteen 32-bit words into eighty 32-bit words:
   for i from 16 to 79
      w[i] = (w[i-3] xor w[i-8] xor w[i-14] xor w[i-16]) leftrotate 1
Step 5:
   Initialize hash value for this chunk:
   Main loop:

   for i from 0 to 79
      if 0 ≤ i ≤ 19 then
         f = (b and c) or ((not b) and d)
         k = 0x5A827999
      else if 20 ≤ i ≤ 39
         f = b xor c xor d
         k = 0x6ED9EBA1
      else if 40 ≤ i ≤ 59
         f = (b and c) or (b and d) or (c and d)
         k = 0x8F1BBCDC
      else if 60 ≤ i ≤ 79
         f = b xor c xor d
         k = 0xCA62C1D6

      temp = (a leftrotate 5) + f + e + k + w[i]
      e = d
      d = c
      c = b leftrotate 30
      b = a
      a = temp
Step 6:
   Add this chunk's hash to result so far:
   h0 = h0 + a
   h1 = h1 + b
   h2 = h2 + c
   h3 = h3 + d
   h4 = h4 + e
Step 7:
Produce the final hash value (big-endian) as a 160 bit number
```

1.4 SHA- 2 and its Variants [8, 10]

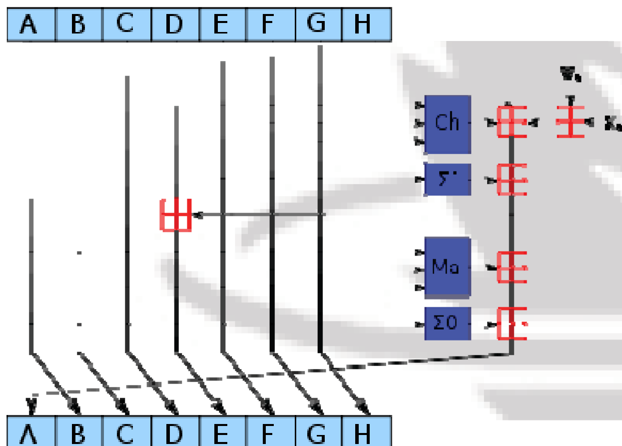
SHA-2 is a set of cryptographic hash functions the variants of SHA-2 areSHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256. SHA-2 includes a significant number of changes from its predecessor, SHA-1. SHA-2 currently consists of a set of six hash functions with digests that are 224, 256, 384 or 512 bits.

SHA-256 and SHA-512 are novel hash functions computed with 32-bit and 64-bit words, respectively. They use different shift amounts and additive constants, but their structures are otherwise virtually identical, differing only in the number of rounds. SHA-224 and SHA-384 are simply

Licensed Under Creative Commons Attribution CC BY

truncated versions of the first two, computed with different initial values. SHA-512/224 and SHA-512/256 are also truncated versions of SHA-512[3.6], but the initial values are generated using the method described in FIPS PUB 180-4 security flaws were identified in SHA-1, namely that a mathematical weakness might exist, indicating that a stronger hash function would be desirable. Although SHA-2 bears some similarity to the SHA-1 algorithm, these attacks have not been successfully extended to SHA-2.

Currently, the best public attacks break preimage resistance 52 rounds of SHA-256 or 57 rounds of SHA-512, and collision resistance for 46 rounds of SHA-256.



NIST added three additional hash functions in the SHA family. The algorithms are collectively known as SHA-2, named after their digest lengths (in bits): SHA-256, SHA-384, and SHA-512.

The updated standard included the original SHA-1[1,4] algorithm, with updated technical notation consistent with that describing the inner workings of the SHA-2 family. Specifying an additional variant, SHA-224, defined to match the key length of two-key Triple DES. The standard was updated in FIPS PUB 180-3, including SHA-224 from the change notice, but otherwise making no fundamental changes to the standard. The primary motivation for updating the standard was relocating security information about the hash algorithms and recommendations for their use to Special Publications 800-107 and 800-57.Detailed test data and example message digests were also removed from the standard, and provided as separate documents. The standard was updated in FIPS PUB 180-4, adding the hash functions SHA-512/224 and SHA-512/256, and describing a method for generating initial values for truncated versions of SHA-512. Additionally, a restriction on padding the input data prior to hash calculation was removed, allowing hash data to be calculated simultaneously with content generation, such as a real-time video or audio feed. Padding the final data block must still occur prior to hash output.

The publication disallows creation of digital signatures with a hash security lower than 112-bits after 2013. The previous revision from 2007 specified the cutoff to be the end of 2010.In August 2012, NIST revised SP800-107 in the same manner. The NIST hash function competition selected a new hash function, SHA-3, in 2012.The SHA-3 algorithm is not derived from SHA-2.

Examples of SHA-2[6, 9] variants

1. SHA224("")
   0x
   d14a028c2a3a2bc9476102bb288234c415a2b01f828ea62
   ac5b3e42f
2. SHA256("")
   0x
   e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934c
   a495991b7852b855
3. SHA384("")
   0x
   38b060a751ac96384cd9327eb1b1e36a21fdb71114be074
   34c0cc7bf63f6e1da274edebfe76f65fbd51ad2f14898b95b
4. SHA512("")
   cf83e1357eefb8bdf1542850d66d8007d620e4050b5715d
   c83f4a921d36ce9ce47d0d13c5d85f2b0ff8318d2877eec2f
   63b931bd47417a81a538327af927da3e
5. SHA512/224("")
   0x
   6ed0dd02806fa89e25de060c19d3ac86cabb87d6a0ddd05
   c333b84f4
6. SHA512/256("")
   0x
   c672b8d1ef56ed28ab87c3622c5114069bdd3ad7b8f9737
   498d0c01ecef0967a

Pseudocode for the SHA-256 algorithm follows:
Note 1: All variables are 32 bit unsigned integers and addition is calculated modulo 232
Note 2: For each round, there is one round constant k[i] and one entry in the message schedule array w[i], $0 \le i \le 63$
Note 3: The compression function uses 8 working variables, a through h
Note 4: Big-endian convention is used when expressing the constants in this pseudocode, and when parsing message block data from bytes to words, for example, the first word of the input message "abc" after padding is 0x61626380

step 1: Initialize hash values:

first 32 bits of the fractional parts of the square roots of the first 8 primes 2..19
Step 2:
Initialize array of round constants:
first 32 bits of the fractional parts of the cube roots of the first 64 primes 2..311
step 3:
Pre-processing:
append the bit '1' to the message
append k bits '0', where k is the minimum number >= 0 such that the resulting message
   length (modulo 512 in bits) is 448.
append length of message (without the '1' bit or padding), in bits, as 64-bit big-endian integer
   (this will make the entire post-processed length a multiple of 512 bits)
Step 4:
Process the message in successive 512-bit chunks:
break message into 512-bit chunks
for each chunk

Licensed Under Creative Commons Attribution CC BY

create a 64-entry message schedule array w[0..63] of 32-bit words

Step 5:

Extend the first 16 words into the remaining 48 words w[16..63] of the message schedule array:

for i from 16 to 63

s0 := (w[i-15] rightrotate 7) xor (w[i-15] rightrotate 18) xor (w[i-15] rightshift 3)

s1 := (w[i-2] rightrotate 17) xor (w[i-2] rightrotate 19) xor (w[i-2] rightshift 10)

w[i] := w[i-16] + s0 + w[i-7] + s1

Step 6:

Initialize working variables to current hash value

Compression function main loop:

for i from 0 to 63

S1 := (e rightrotate 6) xor (e rightrotate 11) xor (e rightrotate 25)

ch := (e and f) xor ((not e) and g)

temp1 := h + S1 + ch + k[i] + w[i]

S0 := (a rightrotate 2) xor (a rightrotate 13) xor (a rightrotate 22)

maj := (a and b) xor (a and c) xor (b and c)

temp2 := S0 + maj

h := g

g := f

f := e

e := d + temp1

d := c

c := b

b := a

a := temp1 + temp2

Step 7:

Add the compressed chunk to the current hash value

Step 8:

Produce the final hash value (big-endian):

digest := hash := h0 append h1 append h2 append h3 append h4 append h5 append h6 append h7. SHA-224 is identical to SHA-256[11], except that:

- the initial hash values h0 through h7 are different, and
- the output is constructed by omitting h7.

1.5 Comparison of Different SHA Functions

Algorithm and variant		Output size (bits)	Internal state size (bits)	Block size (bits)	Max message size (bits)	Word size (bits)	Rounds	Bitwise operations	Collisions found	Example Performance (MiB/s)
MD5 (as reference)		128	128	512	$2^{64}-1$	32	64	and, or, xor, rot	Yes	335
SHA-0		160	160	512	$2^{64}-1$	32	80	and, or, xor, rot	Yes	-
SHA-1		160	160	512	$2^{64}-1$	32	80	and, or, xor, rot	Theoretical attack	192
SHA-2	*SHA-224* *SHA-256*	224 256	256	512	$2^{64}-1$	32	64	and, or, xor, shr, rot	None	139
	SHA-384 *SHA-512* *SHA-512/224* *SHA-512/256*	384 512 224 256	512	1024	$2^{128}-1$	64	80	and, or, xor, shr, rot	None	154

1.6 Proposed Work

This research paper consists of comparisons between different secure hashing algorithms. Each algorithm takes the time for the computation of hash value. By computing the time required from each of these algorithm and finding the algorithm which will require the less amount of time for computation of the hash value we can combine the best secure hashing algorithm with network security algorithm so as to increase the security of the data being sent.

References

[1] http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf
[2] Schneier on Security: Cryptanalysis of SHA-1
[3] "Crypto++ 5.6.0 Benchmarks". Retrieved 2013-06-13.
[4] "Cryptanalysis of MD5 & SHA-1" (PDF).
[5] "Collisions for 72-step and 73-step SHA-1: Improvements in the Method of Characteristics".
[6] "SHA-1 Collision Search Graz".
[7] SHA-1 hash function under pressure – heise Security
[8] Classification and Generation of Disturbance Vectors for Collision Attacks against SHA-1
[9] Cryptanalysis of MD5 & SHA-1
[10] Henri Gilbert, Helena Handschuh: Security Analysis of SHA-256 and Sisters. Selected Areas in cryptography 2003
[11] http://www.unixwiz.net/techtips/iguide-crypto-hashes.html

Author Profile

Priyanka Vadhera has obtained her B. Tech degree from maharishi Dayanand University in 2012. She is also persuing M.Tech degree from Maharishi Dayanand University, Rohtak, India. Her areas of interest are wireless security, networking and signal processing.

Bhumika Lall has obtained her B. Tech and M. tech degree. she is working as assistant professor in B. S. Anangpuria institute of technology and management affiliated to Maharishi Dayanand University, Rohtak. Her areas of interest are wireless security, networking and signal processing.

Licensed Under Creative Commons Attribution CC BY