WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export
Create a book
Download as PDF
Printable version

Languages
Deutsch
Español
فارسی
Français
Hrvatski
עברית
日本語
Polski
Português
Русский
Türkçe
Українська
Edit links

SHA-2

From Wikipedia, the free encyclopedia

SHA-2 is a set of cryptographic hash functions designed by the NSA (U.S. National Security Agency).[3] SHA stands for Secure Hash Algorithm. Cryptographic hash functions are mathematical operations run on digital data; by comparing the computed "hash" (the output from execution of the algorithm) to a known and expected hash value, a person can determine the data's integrity. For example, computing the hash of a downloaded file and comparing the result to a previously published hash result can show whether the download has been modified or tampered with.[4] A key aspect of cryptographic hash functions is their collision resistance: nobody should be able to find two different input values that result in the same hash output.

SHA-2 includes significant changes from its predecessor, SHA-1. The SHA-2 family consists of six hash functions with digests (hash values) that are 224, 256, 384 or 512 bits: **SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256**.

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 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, but the initial values are generated using the method described in FIPS PUB 180-4. SHA-2 was published in 2001 by the NIST as a U.S. federal standard (FIPS). The SHA-2 family of algorithms are patented in US 6829355. The United States has released the patent under a royalty-free license.[5]

In 2005, an algorithm emerged for finding SHA-1 collisions in about 2000-times fewer steps than was previously thought possible.[6] Although (as of 2015) no example of a SHA-1 collision has been published yet, the security margin left by SHA-1 is weaker than intended, and its use is therefore no longer recommended for applications that depend on collision resistance, such as digital signatures. 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, as shown in the *Cryptanalysis and validation* section below.[1][2]

SHA-2

General	
Designers	National Security Agency
First published	2001
Series	(SHA-0), SHA-1, SHA-2, SHA-3
Certification	FIPS PUB 180-4, CRYPTREC, NESSIE
Detail	
Digest sizes	224, 256, 384, or 512 bits
Structure	Merkle–Damgård construction
Rounds	64 or 80
Best public cryptanalysis	

A 2011 attack breaks preimage resistance for 57 out of 80 rounds of SHA-512, and 52 out of 64 rounds for SHA-256.[1]

Pseudo-collision attack against up to 46 rounds of SHA-256.[2]

Contents [hide]

Hash standard [edit]

With the publication of FIPS PUB 180-2, 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 algorithms were first published in 2001 in the draft FIPS PUB 180-2, at which time public review and comments were accepted. In August 2002, FIPS PUB 180-2 became the new Secure Hash Standard, replacing FIPS PUB 180-1, which was released in April 1995. The updated standard included the original SHA-1 algorithm, with updated technical notation consistent with that describing the inner workings of the SHA-2 family.[7]

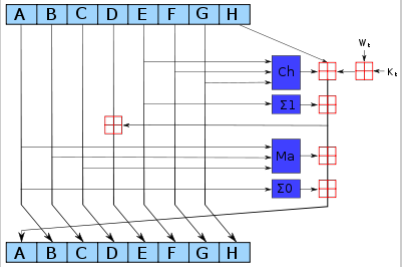
In February 2004, a change notice was published for FIPS PUB 180-2, specifying an additional variant, SHA-224, defined to match the key length of two-key Triple DES.[8] In October 2008, the standard was updated in FIPS PUB 180-3, including SHA-224 from the change notice, but otherwise making no fundamental 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.[9][10][11] Detailed test data and example message digests were also removed from the standard, and provided as separate documents.[12]

In January 2011, NIST published SP800-131A, which specified a move from the current minimum security of 80-bits (provided by SHA-1) allowable for federal government use until the end of 2013, with 112-bit security (provided by SHA-2) being the minimum requirement current thereafter, and the recommended security level from the publication date.[13]

In March 2012, 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.[14]

In July 2012, NIST revised SP800-57, which provides guidance for cryptographic key management. 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.[11] In August 2012, NIST revised SP800-107 in the same manner.[10]

The NIST hash function competition selected a new hash function, SHA-3, in 2012.[15] The SHA-3 algorithm is not derived from SHA-2.



One iteration in a SHA-2 family compression function. The blue components perform the following operations:

$$\mathrm{Ch}(E,F,G) = (E \wedge F) \oplus (\neg E \wedge G)$$
$$\mathrm{Ma}(A,B,C) = (A \wedge B) \oplus (A \wedge C) \oplus (B \wedge C)$$
$$\Sigma_0(A) = (A \ggg 2) \oplus (A \ggg 13) \oplus (A \ggg 22)$$
$$\Sigma_1(E) = (E \ggg 6) \oplus (E \ggg 11) \oplus (E \ggg 25)$$

The bitwise rotation uses different constants for SHA-512. The given numbers are for SHA-256. The red ⊞ is addition modulo 2^{32}.

Applications [edit]

For more details on this topic, see Cryptographic hash function § Applications.

The SHA-2 hash function is implemented in some widely used security applications and protocols, including TLS and SSL, PGP, SSH, S/MIME, and IPsec.

SHA-256 is used as part of the process of authenticating Debian GNU/Linux software packages[16] and in the DKIM message signing standard; SHA-512 is part of a system to authenticate archival video from the International Criminal Tribunal of the Rwandan genocide.[17] SHA-256 and SHA-512 are proposed for use in DNSSEC.[18] Unix and Linux vendors are moving to using 256- and 512-bit SHA-2 for secure password hashing.[19]

Several cryptocurrencies like Bitcoin use SHA-256 for verifying transactions and calculating proof-of-work or proof-of-stake. The rise of ASIC SHA-2 accelerator chips has led to the use of scrypt-based proof-of-work schemes.

SHA-1 and SHA-2 are the secure hash algorithms required by law for use in certain U.S. Government applications, including use within other cryptographic algorithms and protocols, for the protection of sensitive unclassified information. FIPS PUB 180-1 also encouraged adoption and use of SHA-1 by private and commercial organizations. SHA-1 is being retired for most government uses; the U.S. National Institute of Standards and Technology says, "Federal agencies ***should*** stop using SHA-1 for...applications that require collision resistance as soon as practical, and must use the SHA-2 family of hash functions for these applications after 2010" (emphasis in original).[20] NIST's directive that U.S. government agencies must stop uses of SHA-1 after 2010[21] was hoped to accelerate migration away from SHA-1.

The SHA-2 functions were not quickly adopted, despite better security than SHA-1. Reasons might include lack of support for SHA-2 on systems running Windows XP SP2 or older[22] and a lack of perceived urgency since SHA-1 collisions have not yet been found. The Google Chrome team announced a plan to make browser gradually stop honoring SHA-1-dependent TLS certificates over a period from late 2014 and early 2015.[23][24][25]

Cryptanalysis and validation [edit]

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 2^L evaluations. This is called a preimage attack and may or may not be practical depending on L and the particular computing environment. The second criterion, finding two different messages that produce the same message digest, known as a collision, requires on average only $2^{L/2}$ evaluations using a birthday attack.

Some of the applications that use cryptographic hashes, such as password storage, are only minimally affected by a collision attack. Constructing a password that works for a given account requires a preimage attack, as well as access to the hash of the original password (typically in the shadow file) which may or may not be trivial. Reversing password encryption (e.g., to obtain a password to

try against a user's account elsewhere) is not made possible by the attacks. (However, even a secure password hash cannot prevent brute-force attacks on weak passwords.)

In the case of document signing, an attacker could not simply fake a signature from an existing document—the attacker would have to produce a pair of documents, one innocuous and one damaging, and get the private key holder to sign the innocuous document. There are practical circumstances in which this is possible; until the end of 2008, it was possible to create forged SSL certificates using an MD5 collision.[26]

Increased interest in cryptographic hash analysis during the SHA-3 competition produced several new attacks on the SHA-2 family, the best of which are given in the table below. Only the collision attacks are of practical complexity; none of the attacks extend to the full round hash function.

At FSE 2012, researchers at Sony gave a presentation suggesting pseudo-collision attacks could be extended to 52 rounds on SHA-256 and 57 rounds on SHA-512 by building upon the biclique pseudo-preimage attack.[27]

Published in	Year	Attack method	Attack	Variant	Rounds	Complexity
New Collision Attacks Against Up To 24-step SHA-2[28]	2008	Deterministic	Collision	SHA-256	24/64	$2^{28.5}$
				SHA-512	24/80	$2^{32.5}$
Preimages for step-reduced SHA-2[29]	2009	Meet-in-the-middle	Preimage	SHA-256	42/64	$2^{251.7}$
					43/64	$2^{254.9}$
				SHA-512	42/80	$2^{502.3}$
					46/80	$2^{511.5}$
Advanced meet-in-the-middle preimage attacks[30]	2010	Meet-in-the-middle	Preimage	SHA-256	42/64	$2^{248.4}$
				SHA-512	42/80	$2^{494.6}$
Higher-Order Differential Attack on Reduced SHA-256[2]	2011	Differential	Pseudo-collision	SHA-256	46/64	2^{178}
					46/64	2^{46}
Bicliques for Preimages: Attacks on Skein-512 and the SHA-2 family[1]	2011	Biclique	Preimage	SHA-256	45/64	$2^{255.5}$
				SHA-512	50/80	$2^{511.5}$
			Pseudo-preimage	SHA-256	52/64	2^{255}
				SHA-512	57/80	2^{511}
Improving Local Collisions: New Attacks on Reduced SHA-256[31]	2013	Differential	Collision	SHA-256	31/64	$2^{65.5}$
			Pseudo-collision	SHA-256	38/64	2^{37}
Branching Heuristics in Differential Collision Search with Applications to SHA-512[32]	2014	Heuristic differential	Pseudo-collision	SHA-512	38/80	$2^{40.5}$

Official validation [edit]

Main article: Cryptographic Module Validation Program

Implementations of all FIPS-approved security functions can be officially validated through the CMVP program, jointly run by the National Institute of Standards and Technology (NIST) and the Communications Security Establishment (CSE). For informal verification, a package to generate a high number of test vectors is made available for download on the NIST site; the resulting verification, however, does not replace the formal CMVP validation, which is required by law for certain applications.

As of December 2013, there are over 1300 validated implementations of SHA-256 and over 900 of SHA-512, with only 5 of them being capable of handling messages with a length in bits not a multiple of eight while supporting both variants (see SHS Validation List).

Examples of SHA-2 variants [edit]

Hash values of empty string.

```
SHA224("")
0x d14a028c2a3a2bc9476102bb288234c415a2b01f828ea62ac5b3e42f
SHA256("")
0x e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
SHA384("")
0x 38b060a751ac96384cd9327eb1b1e36a21fdb71114be07434c0cc7bf63f6e1da274edebfe76f65fbd51ad2f14898b95b
SHA512("")
0x cf83e1357eefb8bdf1542850d66d8007d620e4050b5715dc83f4a921d36ce9ce47d0d13c5d85f2b0ff8318d2877eec2f63b931bd47417a81a538327af927da3e
SHA512/224("")
0x 6ed0dd02806fa89e25de060c19d3ac86cabb87d6a0ddd05c333b84f4
SHA512/256("")
0x c672b8d1ef56ed28ab87c3622c5114069bdd3ad7b8f9737498d0c01ecef0967a
```

Even a small change in the message will (with overwhelming probability) result in a mostly different hash, due to the avalanche effect. For example, adding a period to the end of the sentence:

```
SHA224("The quick brown fox jumps over the lazy dog")
0x 730e109bd7a8a32b1cb9d9a09aa2325d2430587ddbc0c38bad911525
SHA224("The quick brown fox jumps over the lazy dog.")
0x 619cba8e8e05826e9b8c519c0a5c68f4fb653e8a3d8aa04bb2c8cd4c
```

Pseudocode [edit]

Pseudocode for the SHA-256 algorithm follows. Note the great increase in mixing between bits of the `w[16..63]` words compared to SHA-1.

```
Note 1: All variables are 32 bit unsigned integers and addition is calculated modulo 2^32
Note 2: For each round, there is one round constant k[i] and one entry in the message schedule array w[i], 0 ≤ i ≤ 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

Initialize hash values:
(first 32 bits of the fractional parts of the square roots of the first 8 primes 2..19):
h0 := 0x6a09e667
h1 := 0xbb67ae85
h2 := 0x3c6ef372
h3 := 0xa54ff53a
h4 := 0x510e527f
h5 := 0x9b05688c
h6 := 0x1f83d9ab
h7 := 0x5be0cd19

Initialize array of round constants:
(first 32 bits of the fractional parts of the cube roots of the first 64 primes 2..311):
k[0..63] :=
   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

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)

Process the message in successive 512-bit chunks:
break message into 512-bit chunks
for each chunk
    create a 64-entry message schedule array w[0..63] of 32-bit words
    (The initial values in w[0..63] don't matter, so many implementations zero them here)
    copy chunk into first 16 words w[0..15] of the message schedule array

    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

    Initialize working variables to current hash value:
    a := h0
    b := h1
    c := h2
    d := h3
    e := h4
    f := h5
    g := h6
    h := h7

    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

    Add the compressed chunk to the current hash value:
    h0 := h0 + a
    h1 := h1 + b
    h2 := h2 + c
    h3 := h3 + d
    h4 := h4 + e
    h5 := h5 + f
    h6 := h6 + g
    h7 := h7 + h

Produce the final hash value (big-endian):
digest := hash := h0 append h1 append h2 append h3 append h4 append h5 append h6 append h7
```

The computation of the `ch` and `maj` values can be optimized the same way as described for SHA-1.

SHA-224 is identical to SHA-256, except that:

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

```
SHA-224 initial hash values (in big endian):
(The second 32 bits of the fractional parts of the square roots of the 9th through 16th primes 23..53)
h[0..7] :=
    0xc1059ed8, 0x367cd507, 0x3070dd17, 0xf70e5939, 0xffc00b31, 0x68581511, 0x64f98fa7, 0xbefa4fa4
```

SHA-512 is identical in structure to SHA-256, but:

- the message is broken into 1024-bit chunks,
- the initial hash values and round constants are extended to 64 bits,
- there are 80 rounds instead of 64,
- the message schedule array w has 80 64-bit words instead of 64 32-bit words,
- to extend the message schedule array w, the loop is from 16 to 79 instead of from 16 to 63,
- the round constants are based on the first 80 primes 2..409,
- the word size used for calculations is 64 bits long,
- the appended length of the message (before pre-processing), in *bits*, is a 128-bit big-endian integer, and
- the shift and rotate amounts used are different.

```
SHA-512 initial hash values (in big-endian):

h[0..7] := 0x6a09e667f3bcc908, 0xbb67ae8584caa73b, 0x3c6ef372fe94f82b, 0xa54ff53a5f1d36f1,
           0x510e527fade682d1, 0x9b05688c2b3e6c1f, 0x1f83d9abfb41bd6b, 0x5be0cd19137e2179

SHA-512 round constants:

k[0..79] := [ 0x428a2f98d728ae22, 0x7137449123ef65cd, 0xb5c0fbcfec4d3b2f, 0xe9b5dba58189dbbc, 0x3956c25bf348b538,
              0x59f111f1b605d019, 0x923f82a4af194f9b, 0xab1c5ed5da6d8118, 0xd807aa98a3030242, 0x12835b0145706fbe,
              0x243185be4ee4b28c, 0x550c7dc3d5ffb4e2, 0x72be5d74f27b896f, 0x80deb1fe3b1696b1, 0x9bdc06a725c71235,
              0xc19bf174cf692694, 0xe49b69c19ef14ad2, 0xefbe4786384f25e3, 0x0fc19dc68b8cd5b5, 0x240ca1cc77ac9c65,
              0x2de92c6f592b0275, 0x4a7484aa6ea6e483, 0x5cb0a9dcbd41fbd4, 0x76f988da831153b5, 0x983e5152ee66dfab,
              0xa831c66d2db43210, 0xb00327c898fb213f, 0xbf597fc7beef0ee4, 0xc6e00bf33da88fc2, 0xd5a79147930aa725,
              0x06ca6351e003826f, 0x142929670a0e6e70, 0x27b70a8546d22ffc, 0x2e1b21385c26c926, 0x4d2c6dfc5ac42aed,
              0x53380d139d95b3df, 0x650a73548baf63de, 0x766a0abb3c77b2a8, 0x81c2c92e47edaee6, 0x92722c851482353b,
              0xa2bfe8a14cf10364, 0xa81a664bbc423001, 0xc24b8b70d0f89791, 0xc76c51a30654be30, 0xd192e819d6ef5218,
              0xd69906245565a910, 0xf40e35855771202a, 0x106aa07032bbd1b8, 0x19a4c116b8d2d0c8, 0x1e376c085141ab53,
              0x2748774cdf8eeb99, 0x34b0bcb5e19b48a8, 0x391c0cb3c5c95a63, 0x4ed8aa4ae3418acb, 0x5b9cca4f7763e373,
              0x682e6ff3d6b2b8a3, 0x748f82ee5defb2fc, 0x78a5636f43172f60, 0x84c87814a1f0ab72, 0x8cc702081a6439ec,
              0x90befffa23631e28, 0xa4506cebde82bde9, 0xbef9a3f7b2c67915, 0xc67178f2e372532b, 0xca273eceea26619c,
              0xd186b8c721c0c207, 0xeada7dd6cde0eb1e, 0xf57d4f7fee6ed178, 0x06f067aa72176fba, 0x0a637dc5a2c898a6,
              0x113f9804bef90dae, 0x1b710b35131c471b, 0x28db77f523047d84, 0x32caab7b40c72493, 0x3c9ebe0a15c9bebc,
              0x431d67c49c100d4c, 0x4cc5d4becb3e42b6, 0x597f299cfc657e2a, 0x5fcb6fab3ad6faec, 0x6c44198c4a475817]

SHA-512 Sum & Sigma:

S0 := (a rightrotate 28) xor (a rightrotate 34) xor (a rightrotate 39)
S1 := (e rightrotate 14) xor (e rightrotate 18) xor (e rightrotate 41)
```

```
s0 := (w[i-15] rightrotate 1) xor (w[i-15] rightrotate 8) xor (w[i-15] rightshift 7)
s1 := (w[i-2] rightrotate 19) xor (w[i-2] rightrotate 61) xor (w[i-2] rightshift 6)
```

SHA-384 is identical to SHA-512, except that:

- the initial hash values `h0` through `h7` are different (taken from the 9th through 16th primes), and
- the output is constructed by omitting `h6` and `h7`.

```
SHA-384 initial hash values (in big-endian):

h[0..7] := 0xcbbb9d5dc1059ed8, 0x629a292a367cd507, 0x9159015a3070dd17, 0x152fecd8f70e5939,
           0x67332667ffc00b31, 0x8eb44a8768581511, 0xdb0c2e0d64f98fa7, 0x47b5481dbefa4fa4
```

SHA-512/t is identical to SHA-512 except that:

- the initial hash values `h0` through `h7` are given by the *SHA-512/t IV generation function*,
- the output is constructed by truncating the concatenation of `h0` through `h7` at *t* bits,
- *t* equal to 384 is not allowed, instead SHA-384 should be used as specified, and
- *t* values 224 and 256 are especially mentioned as approved.

The *SHA-512/t IV generation function* evaluates a *modified SHA-512* on the ASCII string "SHA-512/*t*", substituted with the decimal representation of *t*. The *modified SHA-512* is the same as SHA-512 except its initial values `h0` through `h7` have each been XORed with the hexadecimal constant `0xa5a5a5a5a5a5a5a5`.

Comparison of SHA functions [edit]

In the table below, *internal state* means the "internal hash sum" after each compression of a data block.

Further information: Merkle–Damgård construction

view · talk · edit

Comparison of SHA functions

Algorithm and variant		Output size (bits)	Internal state size (bits)	Block size (bits)	Max message size (bits)	Rounds	Operations	Security (bits)	Example performance[34] (MiB/s)
MD5 (as reference)		128	128 (4 × 32)	512	$2^{64} - 1$	64	And, Xor, Rot, Add (mod 2^{32}), Or	<64 (collisions found)	335
SHA-0		160	160 (5 × 32)	512	$2^{64} - 1$	80	And, Xor, Rot, Add (mod 2^{32}), Or	<80 (collisions found)	-
SHA-1		160	160 (5 × 32)	512	$2^{64} - 1$	80		<80 (theoretical attack[35] in 2^{61} operations)	192
SHA-2	*SHA-224*	224	256 (8 × 32)	512	$2^{64} - 1$	64	And, Xor, Rot, Add (mod 2^{32}), Or, Shr	112	139
	SHA-256	256						128	
	SHA-384	384	512 (8 × 64)	1024	$2^{128} - 1$	80	And, Xor, Rot, Add (mod 2^{64}), Or, Shr	192	154
	SHA-512	512						256	
	SHA-512/224	224						112	
	SHA-512/256	256						128	
SHA-3	*SHA3-224*	224	1600 (5 × 5 × 64)	1152	Unlimited	24	And, Xor, Rot, Not	112	-
	SHA3-256	256		1088				128	
	SHA3-384	384		832				192	
	SHA3-512	512		576				256	
	SHAKE128	*d* (arbitrary)		1344				min(*d*/2, 128)	-
	SHAKE256	*d* (arbitrary)		1088				min(*d*/2, 256)	

In the bitwise operations column, "rot" stands for rotate no carry, and "shr" stands for right logical shift. All of these algorithms employ modular addition in some fashion except for SHA-3.

The performance numbers above were for a single-threaded implementation on an AMD Opteron 8354 running at 2.2 GHz under Linux in 64-bit mode, and serve only as a rough point for general comparison. More detailed performance measurements on modern processor architectures are given in the table below.

CPU architecture	Frequency	Algorithm	Word size (bits)	Cycles/byte x86	MiB/s x86	Cycles/byte x86-64	MiB/s x86-64
Intel Ivy Bridge	3.5 GHz	SHA-256	32-bit	16.80	199	13.05	256
		SHA-512	64-bit	43.66	76	8.48	394
AMD Piledriver	3.8 GHz	SHA-256	32-bit	22.87	158	18.47	196
		SHA-512	64-bit	88.36	41	12.43	292

The performance numbers labeled 'x86' were running using 32-bit code on 64-bit processors, whereas the 'x86-64' numbers are native 64-bit code. While SHA-256 is designed for 32-bit calculations, it does benefit from code optimized for 64-bit processors. 32-bit implementations of SHA-512 are significantly slower than their 64-bit counterparts. Variants of both algorithms with different output sizes will perform similarly, since the message expansion and compression functions are identical, and only the initial hash values and output sizes are different. The best implementations of MD5 and SHA-1 perform between 4.5 and 6 cycles per byte on modern processors.

Testing was performed by the University of Illinois at Chicago on their hydra8 system running an Intel Xeon E3-1275 V2 at a clock speed of 3.5 GHz, and on their hydra9 system running an AMD A10-5800K at a clock speed of 3.8 GHz.[36] The referenced cycles per byte speeds above are the median performance of an algorithm digesting a 4,096 byte message using the SUPERCOP cryptographic benchmarking software.[37] The MiB/s performance is extrapolated from the CPU clockspeed on a single core, real world performance will vary due to a variety of factors.

See also [edit]

Cryptography portal

- Comparison of cryptographic hash functions
- Digital timestamping
- Hash-based message authentication code (HMAC)
- Hashcash
- International Association for Cryptologic Research (IACR)
- `sha1sum` (`sha224sum`, `sha256sum`, `sha384sum` and `sha512sum`) commands
- scrypt

References [edit]

1. ^ a b c Dmitry Khovratovich, Christian Rechberger and Alexandra Savelieva (2011). "Bicliques for Preimages: Attacks on Skein-512 and the SHA-2 family" (PDF). *IACR Cryptology ePrint Archive*. 2011:286.
2. ^ a b c Mario Lamberger and Florian Mendel (2011). "Higher-Order Differential Attack on Reduced SHA-256" (PDF). *IACR Cryptology ePrint Archive*. 2011:37.
3. ^ "On the Secure Hash Algorithm family" (PDF).
4. ^ "Cryptographic Hash Function". About.com. Retrieved 2014-08-18.
5. ^ "Licensing Declaration for US patent 6829355.". Retrieved 2008-02-17.
6. ^ "Schneier on Security: Cryptanalysis of SHA-1". Schneier.com. Retrieved 2011-11-08.
7. ^ Federal Register Notice 02-21599, Announcing Approval of FIPS Publication 180-2
8. ^ FIPS 180-2 with Change Notice 1
9. ^ Federal Register Notice E8-24743, Announcing Approval of FIPS Publication 180-3
10. ^ a b FIPS SP 800-107 Recommendation for Applications Using Approved Hash Algorithms
11. ^ a b FIPS SP 800-57 Recommendation for Key Management: Part 1: General
12. ^ NIST Algorithm Examples, Secure Hashing
13. ^ FIPS SP 800-131A Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths
14. ^ Federal Register Notice 2012-5400, Announcing Approval of FIPS Publication 180-4

14. ^ Federal Register Notice 2012-5400, Announcing Approval of FIPS Publication 180-4
15. ^ "NIST Selects Winner of Secure Hash Algorithm (SHA-3) Competition". Retrieved 24 February 2015.
16. ^ "Debian codebase in Google Code". Google. Retrieved 2011-11-08.[dead link]
17. ^ John Markoff, A Tool to Verify Digital Records, Even as Technology Shifts, *New York Times*, January 26, 2009
18. ^ RFC 5702,RFC-Editor.org
19. ^ Ulrich Drepper, Unix crypt with SHA-256/512
20. ^ National Institute on Standards and Technology Computer Security Resource Center, NIST's Policy on Hash Functions, accessed March 29, 2009.
21. ^ "Secure Hashing". *NIST*. Retrieved 2010-11-25.
22. ^ Microsoft Corporation,Overview of Windows XP Service Pack 3
23. ^ Chromium Blog, September 5, 2014, Gradually sunsetting SHA-1
24. ^ [1]
25. ^ Filippo Valsorda, The Unofficial Chrome SHA1 Deprecation FAQ
26. ^ Alexander Sotirov, Marc Stevens, Jacob Appelbaum, Arjen Lenstra, David Molnar, Dag Arne Osvik, Benne de Weger, MD5 considered harmful today: Creating a rogue CA certificate, accessed March 29, 2009.
27. ^ Ji Li, Takanori Isobe and Kyoji Shibutani, Sony China Research Laboratory and Sony Corporation, Converting Meet-in-the-Middle Preimage Attack into Pseudo Collision Attack: Application to SHA-2
28. ^ Somitra Kumar Sanadhya and Palash Sarkar (2008). "New Collision Attacks Against Up To 24-step SHA-2" (PDF). *IACR Cryptology ePrint Archive*. 2008:270.
29. ^ Kazumaro Aoki, Jian Guo, Krystian Matusiewicz, Yu Sasaki, and Lei Wang (2009). "Preimages for step-reduced SHA-2". *Advances in Cryptology - ASIACRYPT 2009*. Lecture Notes in Computer Science (Springer Berlin Heidelberg) **5912**: 578–597. doi:10.1007/978-3-642-10366-7_34. ISBN 978-3-642-10366-7. ISSN 0302-9743.
30. ^ Jian Guo, San Ling, Christian Rechberger, and Huaxiong Wang (2010). "Advanced meet-in-the-middle preimage attacks: First results on full Tiger, and improved results on MD4 and SHA-2" (PDF). *Advances in Cryptology - ASIACRYPT 2010*. Lecture Notes in Computer Science (Springer Berlin Heidelberg) **6477**: 56–75. doi:10.1007/978-3-642-17373-8_4. ISBN 978-3-642-17373-8. ISSN 0302-9743.
31. ^ Florian Mendel, Tomislav Nad, Martin Schläffer (2013). "Improving Local Collisions: New Attacks on Reduced SHA-256". *Advances in Cryptology – EUROCRYPT 2013*. Lecture Notes in Computer Science (Springer Berlin Heidelberg) **7881**: 262–278. doi:10.1007/978-3-642-38348-9_16. ISBN 978-3-642-38348-9. ISSN 0302-9743.
32. ^ Maria Eichlseder and Florian Mendel and Martin Schläffer (2014). "Branching Heuristics in Differential Collision Search with Applications to SHA-512" (PDF). *IACR Cryptology ePrint Archive*. 2014:302.
33. ^ "Crypto++ 5.6.0 Benchmarks". Retrieved 2013-06-13.
34. ^ Found on an AMD Opteron 8354 2.2 GHz processor running 64-bit Linux[33]
35. ^ "Cryptanalysis of MD5 & SHA-1" (PDF). Retrieved 2013-04-25.

Additional reading [edit]

- Henri Gilbert, Helena Handschuh: Security Analysis of SHA-256 and Sisters. Selected Areas in Cryptography 2003: pp175–193
- "Proposed Revision of Federal Information Processing Standard (FIPS) 180, Secure Hash Standard". *Federal Register* **59** (131): 35317–35318. 1994-07-11. Retrieved 2007-04-26.

External links [edit]

- Descriptions of SHA-256, SHA-384, and SHA-512 from NIST
- SHA-2 Checker – SHAChecker check your SSL compatibility for SHA-2.
- Specifications for a Secure Hash Standard (SHS) – Draft for proposed SHS (SHA-0)
- Secure Hash Standard (SHS) – Proposed SHS (SHA-0)
- CSRC Cryptographic Toolkit – Official NIST site for the Secure Hash Standard
  - FIPS 180-4: Secure Hash Standard (SHS) (PDF, 1.7 MB) – Current version of the Secure Hash Standard (SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512), March 2012
- Test vectors for SHA-256/384/512 from the NESSIE project
- Test vectors for SHA-1, SHA-2 from NIST site
- NIST Cryptographic Hash Project SHA-3 competition
- RFC 3874: A 224-bit One-way Hash Function: SHA-224.
- RFC 6234: US Secure Hash Algorithms SHA and SHA-based HMAC and HKDF. Contains sample C implementation.

v · t · e	**Hash functions & message authentication codes**
	Security summary
Common functions	MD5 · SHA-1 · **SHA-2** · SHA-3/Keccak
SHA-3 finalists	BLAKE · Grøstl · JH · Skein · Keccak (winner)
Other functions	FSB · ECOH · GOST · HAS-160 · HAVAL · LM hash · MDC-2 · MD2 · MD4 · MD6 · N-Hash · RadioGatún · RIPEMD · SipHash · Snefru · Streebog · SWIFFT · Tiger · VSH · WHIRLPOOL · crypt(3) (DES)
MAC algorithms	DAA · CBC-MAC · HMAC · OMAC/CMAC · PMAC · VMAC · UMAC · Poly1305-AES
Authenticated encryption modes	CCM · CWC · EAX · GCM · IAPM · OCB
Attacks	Collision attack · Preimage attack · Birthday attack · Brute force attack · Rainbow table · Distinguishing attack · Side-channel attack · Length extension attack
Design	Avalanche effect · Hash collision · Merkle–Damgård construction
Standardization	CRYPTREC · NESSIE · NIST hash function competition
Utilization	Salt · Key stretching · Message authentication
v · t · e	**Cryptography**
	History of cryptography · Cryptanalysis · Cryptography portal · Outline of cryptography
	Symmetric-key algorithm · Block cipher · Stream cipher · Public-key cryptography · Cryptographic hash function · Message authentication code · Random numbers · Steganography

Categories: Cryptographic hash functions | Checksum algorithms | National Security Agency cryptography