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
Български
Deutsch
Español
فارسی
Français
Hrvatski
Latviešu
日本語
Polski
Português
Русский
Svenska
Edit links

SHA-1

From Wikipedia, the free encyclopedia

In cryptography, **SHA-1** is a cryptographic hash function designed by the United States National Security Agency and is a U.S. Federal Information Processing Standard published by the United States NIST.[2]

SHA-1 produces a 160-bit (20-byte) hash value known as a message digest. A SHA-1 hash value is typically rendered as a hexadecimal number, 40 digits long.

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*. SHA-0 is the original version of the 160-bit hash function published in 1993 under the name "SHA": it was not adopted by many applications. Published in 1995, SHA-1 is very similar to SHA-0, but alters the original SHA hash specification to correct alleged weaknesses. SHA-2, published in 2001, is significantly different from the SHA-1 hash function.

SHA-1 is the most widely used of the existing SHA hash functions, and is employed in several widely used applications and protocols.

SHA-1

General	
Designers	National Security Agency
First published	1993 (SHA-0), 1995 (SHA-1)
Series	(SHA-0), SHA-1, SHA-2, SHA-3
Certification	FIPS PUB 180-4, CRYPTREC (Monitored)
Detail	
Digest sizes	160 bits
Structure	Merkle–Damgård construction
Rounds	80
Best public cryptanalysis	
A 2011 attack by Marc Stevens can produce hash collisions with a complexity between $2^{60.3}$ and $2^{65.3}$ operations.[1] No actual collisions have yet been produced.	

In 2005, cryptanalysts found attacks on SHA-1 suggesting that the algorithm might not be secure enough for ongoing use.[3] NIST required many applications in federal agencies to move to SHA-2 after 2010 because of the weakness.[4] Although no successful attacks have yet been reported on SHA-2, it is algorithmically similar to SHA-1. In 2012, following a long-running competition, NIST selected an additional algorithm, Keccak, for standardization under SHA-3.[5][6] In November 2013 Microsoft announced their deprecation policy on SHA-1 according to which Windows will stop accepting SHA-1 certificates in SSL by 2017.[7] In September 2014 Google announced their deprecation policy on SHA-1 according to which Chrome will stop accepting SHA-1 certificates in SSL in a phased way by 2017.[8] Mozilla is also planning to stop accepting SHA-1-based SSL certificates by 2017.[9][10][11]

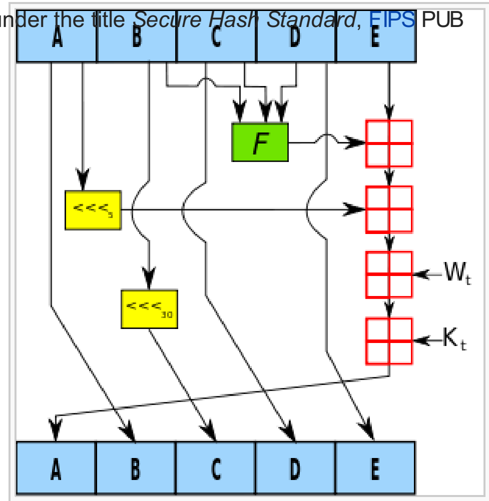
Contents [hide]

The SHA-1 hash function [edit]

This section **does not cite any references or sources**. Please help improve this section by adding citations to reliable sources. Unsourced material may be challenged and removed. *(May 2013)*

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 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*, FIPS PUB 180, by U.S. government standards agency NIST (National Institute of Standards and Technology). 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 in FIPS PUB 180-1 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; this was done, according to the NSA, to correct a flaw in the original algorithm which reduced its cryptographic security. However, the NSA did 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[*citation needed*], supporting the NSA's assertion that the change increased the security.



One iteration within the SHA-1 compression function:
A, B, C, D and E are 32-bit words of the state;
F is a nonlinear function that varies;
$\lll_n$ denotes a left bit rotation by *n* places;
n varies for each operation;
W_t is the expanded message word of round t;
K_t is the round constant of round t;
⊞ denotes addition modulo 2^{32}.

Applications [edit]

Cryptography [edit]

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

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 flaw in the first implementations of the firmware allowed for an attacker to bypass the system's security scheme.[12]

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 from most government uses; the U.S. National Institute of Standards and Technology said, "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),[13] though that was later relaxed.[14]

A prime motivation for the publication of the Secure Hash Algorithm was the Digital Signature Standard, in which it is incorporated.

The SHA hash functions have been used for the basis of the SHACAL block ciphers.

Data integrity [edit]

Revision control systems such as Git and Mercurial use SHA-1 not for security but for ensuring that the data has not changed due to accidental corruption. Linus Torvalds has said about Git: "If you have disk corruption, if you have DRAM corruption, if you have any kind of problems at all, Git will notice them. It's not a question of if, it's a guarantee. You can have people who try to be malicious. They won't succeed. [...] 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 that, OK, it's a huge security feature. It has nothing at all to do with security, it's just the best hash you can get. [...] I guarantee you, if you put your data in Git, you can trust the fact that five years later, after it was converted from your hard disk to DVD to whatever new technology and you copied it along, five years later you can verify that the data you get back out is the exact same data you put in. [...] One of the reasons I care is for the kernel, we had a break in on one of the BitKeeper sites where people tried to corrupt the kernel source code repositories."[15] Nonetheless, without the second preimage resistance of SHA-1, signed commits and tags would no longer secure the state of the repository as they only sign the root of a Merkle tree.[*citation needed*]

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 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. The second criterion, finding two different messages that produce the same message digest, namely a *collision,* requires on average only about 1.2 × $2^{L/2}$ evaluations using a birthday attack. For the latter reason the strength of a hash function is usually compared to a symmetric cipher of half the message digest length. Thus SHA-1 was originally thought to have 80-bit strength.

Cryptographers have 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 terms of practical security, a major concern about these new attacks is that they might pave the way to more efficient ones. Whether this is the case is yet to be seen, but a migration to stronger hashes is believed to be prudent. Some of the applications that use cryptographic hashes, like 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, 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 can't 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.[16]

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.[17] 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. A simple improvement to prevent these attacks is to hash twice: $\mathrm{SHA}_d(\textit{message}) = \mathrm{SHA}(\mathrm{SHA}(0^b \,\|\, \textit{message}))$ (the length of 0^b, zero block, is equal to the block size of the hash function).

Attacks [edit]

In early 2005, Rijmen and Oswald published an attack on a reduced version of SHA-1—53 out of 80 rounds—which finds collisions with a computational effort of fewer than 2^{80} operations.[18]

In February 2005, an attack by Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu was announced.[19] The attacks can find collisions in the full version of SHA-1, requiring fewer than 2^{69} operations. (A brute-force search would require 2^{80} operations.)

The authors write: "In particular, our analysis is built upon the original differential attack on SHA-0 [*sic*], the near collision attack on SHA-0, the multiblock collision techniques, as well as the message modification techniques used in the collision search attack on MD5. Breaking SHA-1 would not be possible without these powerful analytical techniques."[20] The authors have presented a collision for 58-round SHA-1, found with 2^{33} hash operations. The paper with the full attack description was published in August 2005 at the CRYPTO conference.

In an interview, Yin states that, "Roughly, we exploit the following two weaknesses: One is that the file preprocessing step is not complicated enough; another is that certain math operations in the first 20 rounds have unexpected security problems."[21]

On 17 August 2005, an improvement on the SHA-1 attack was announced on behalf of Xiaoyun Wang, Andrew Yao and Frances Yao at the CRYPTO 2005 rump session, lowering the complexity required for finding a collision in SHA-1 to 2^{63}.[22] On 18 December 2007 the details of this result were explained and verified by Martin Cochran.[23]

Christophe De Cannière and Christian Rechberger further improved the attack on SHA-1 in "Finding SHA-1 Characteristics: General Results and Applications,"[24] receiving the Best Paper Award at ASIACRYPT 2006. A two-block collision for 64-round SHA-1 was presented, found using unoptimized methods with 2^{35} compression function evaluations. Since this attack requires the equivalent of about 2^{35} evaluations, it is considered to be a significant theoretical break.[25] Their attack was extended further to 73 rounds (of 80) in 2010 by Grechnikov.[26] In order to find an actual collision in the full 80 rounds of the hash function, however, massive amounts of computer time are required. To that end, a collision search for SHA-1 using the distributed computing platform BOINC began August 8, 2007, organized by the Graz University of Technology. The effort was abandoned May 12, 2009 due to lack of progress.[27]

At the Rump Session of CRYPTO 2006, Christian Rechberger and Christophe De Cannière claimed to have discovered a collision attack on SHA-1 that would allow an attacker to select at least parts of the message.[28][29]

In 2008, an attack methodology by Stéphane Manuel reported hash collisions with an estimated theoretical complexity of 2^{51} to 2^{57} operations.[30] However he later retracted that claim after finding that local collision paths were not actually independent, and finally quoting for the most efficient a collision vector that was already known before this work.[31]

Cameron McDonald, Philip Hawkes and Josef Pieprzyk presented a hash collision attack with claimed complexity 2^{52}

at the Rump session of Eurocrypt 2009.[32] However, the accompanying paper, "Differential Path for SHA-1 with complexity $O(2^{52})$" has been withdrawn due to the authors' discovery that their estimate was incorrect.[33]

As of 2012, the most efficient attack against SHA-1 is considered to be the one by Marc Stevens[34] with an estimated cost of $2.77M to break a single hash value by renting CPU power from cloud servers.[35] Stevens developed this attack in a project called HashClash,[36] implementing a differential path attack. On 8 November 2010, he claimed he had a fully working near-collision attack against full SHA-1 working with an estimated complexity equivalent to $2^{57.5}$ SHA-1 compressions. He estimates this attack can be extended to a full collision with a complexity around 2^{61}.

SHA-0 [edit]

At CRYPTO 98, two French researchers, Florent Chabaud and Antoine Joux, presented an attack on SHA-0 (Chabaud and Joux, 1998): collisions can be found with complexity 2^{61}, fewer than the 2^{80} for an ideal hash function of the same size.

In 2004, Biham and Chen found near-collisions for SHA-0—two messages that hash to nearly the same value; in this case, 142 out of the 160 bits are equal. They also found full collisions of SHA-0 reduced to 62 out of its 80 rounds.

Subsequently, on 12 August 2004, a collision for the full SHA-0 algorithm was announced by Joux, Carribault, Lemuet, and Jalby. This was done by using a generalization of the Chabaud and Joux attack. Finding the collision had complexity 2^{51} and took about 80,000 CPU hours on a supercomputer with 256 Itanium 2 processors. (Equivalent to 13 days of full-time use of the computer.)

On 17 August 2004, at the Rump Session of CRYPTO 2004, preliminary results were announced by Wang, Feng, Lai, and Yu, about an attack on MD5, SHA-0 and other hash functions. The complexity of their attack on SHA-0 is 2^{40}, significantly better than the attack by Joux *et al.*[37][38]

In February 2005, an attack by Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu was announced which could find collisions in SHA-0 in 2^{39} operations.[19][39]

Another attack in 2008 applying the boomerang attack brought the complexity of finding collisions down to $2^{33.6}$, which is estimated to take 1 hour on an average PC.[40]

In light of the results for SHA-0, some experts suggested that plans for the use of SHA-1 in new cryptosystems should be reconsidered. After the CRYPTO 2004 results were published, NIST announced that they planned to phase out the use of SHA-1 by 2010 in favor of the SHA-2 variants.[41]

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, in any way, the formal CMVP validation, which is required by law for certain applications.

As of December 2013, there are over 2000 validated implementations of SHA-1, with 14 of them capable of handling messages with a length in bits not a multiple of eight (see SHS Validation List).

Examples and pseudocode [edit]

Example hashes [edit]

These are examples of SHA-1 message digests in hexadecimal and in Base64 binary to ASCII text encoding.

```
SHA1("The quick brown fox jumps over the lazy dog")
gives hexadecimal: 2fd4e1c67a2d28fced849ee1bb76e7391b93eb12
gives Base64 binary to ASCII text encoding: L9ThxnotKPzthJ7hu3bnORuT6xI=
```

Even a small change in the message will, with overwhelming probability, result in a completely different hash due to the avalanche effect. For example, changing `dog` to `cog` produces a hash with different values for 81 of the 160 bits:

```
SHA1("The quick brown fox jumps over the lazy cog")
gives hexadecimal: de9f2c7fd25e1b3afad3e85a0bd17d9b100db4b3
gives Base64 binary to ASCII text encoding: 3p8sf9JeGzr60+haC9F9mxANtLM=
```

The hash of the zero-length string is:

```
SHA1("")
gives hexadecimal: da39a3ee5e6b4b0d3255bfef95601890afd80709
gives Base64 binary to ASCII text encoding: 2jmj7l5rSw0yVb/vlWAYkK/YBwk=
```

SHA-1 pseudocode [edit]

Pseudocode for the SHA-1 algorithm follows:

```
Note 1: All variables are unsigned 32-bit quantities and wrap modulo 2^32 when
calculating, except for
        ml, the message length, which is a 64-bit quantity, and
        hh, the message digest, which is a 160-bit quantity.
Note 2: All constants in this pseudo code are in big endian.
        Within each word, the most significant byte is stored in the leftmost byte
position

Initialize variables:

h0 = 0x67452301
h1 = 0xEFCDAB89
h2 = 0x98BADCFE
h3 = 0x10325476
h4 = 0xC3D2E1F0

ml = message length in bits (always a multiple of the number of bits in a character).

Pre-processing:
append the bit '1' to the message e.g. by adding 0x80 if message length is a multiple
of 8 bits.
append 0 ≤ k < 512 bits '0', such that the resulting message length in bits
   is congruent to 448 (mod 512)
append ml, in a 64-bit big-endian integer. Thus, the total length is a multiple of 512
bits.

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

    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

    Initialize hash value for this chunk:
    a = h0
    b = h1
    c = h2
    d = h3
    e = h4

    Main loop:[42]
    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

    Add this chunk's hash to result so far:
    h0 = h0 + a
    h1 = h1 + b
    h2 = h2 + c
    h3 = h3 + d
    h4 = h4 + e

Produce the final hash value (big-endian) as a 160 bit number:
hh = (h0 leftshift 128) or (h1 leftshift 96) or (h2 leftshift 64) or (h3 leftshift 32)
or h4
```

The number `hh` is the message digest, which can be written in hexadecimal (base 16), but is often written using Base64 binary to ASCII text encoding.

The constant values used are chosen to be nothing up my sleeve numbers: the four round constants `k` are 2^{30} times the square roots of 2, 3, 5 and 10. The first four starting values for `h0` through `h3` are the same with the MD5 algorithm, and the fifth (for `h4`) is similar.

Instead of the formulation from the original FIPS PUB 180-1 shown, the following equivalent expressions may be used to compute `f` in the main loop above:

```
(0  ≤ i ≤ 19): f = d xor (b and (c xor d))                (alternative 1)
(0  ≤ i ≤ 19): f = (b and c) xor ((not b) and d)          (alternative 2)
(0  ≤ i ≤ 19): f = (b and c) + ((not b) and d)            (alternative 3)
(0  ≤ i ≤ 19): f = vec_sel(d, c, b)                       (alternative 4)

(40 ≤ i ≤ 59): f = (b and c) or (d and (b or c))          (alternative 1)
(40 ≤ i ≤ 59): f = (b and c) or (d and (b xor c))         (alternative 2)
(40 ≤ i ≤ 59): f = (b and c) + (d and (b xor c))          (alternative 3)
(40 ≤ i ≤ 59): f = (b and c) xor (b and d) xor (c and d)  (alternative 4)
(40 ≤ i ≤ 59): f = vec_sel(c, b, c xor d)                 (alternative 5)
```

Max Locktyukhin has also shown[43] that for the rounds 32–79 the computation of:

```
w[i] = (w[i-3] xor w[i-8] xor w[i-14] xor w[i-16]) leftrotate 1
```

can be replaced with:

```
w[i] = (w[i-6] xor w[i-16] xor w[i-28] xor w[i-32]) leftrotate 2
```

This transformation keeps all operands 64-bit aligned and, by removing the dependency of `w[i]` on `w[i-3]`, allows efficient SIMD implementation with a vector length of 4 like x86 SSE instructions.

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

Note that performance will vary not only between algorithms, but also with the specific implementation and hardware used. The OpenSSL tool has a built-in "speed" command that benchmarks the various algorithms on the user's system.

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[45] (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[46] in 2^{61} operations)	192
SHA-2	*SHA-224* *SHA-256*	224 256	256 (8 × 32)	512	$2^{64} - 1$	64	And, Xor, Rot, Add (mod 2^{32}), Or, Shr	112 128	139
	SHA-384 *SHA-512* *SHA-512/224* *SHA-512/256*	384 512 224 256	512 (8 × 64)	1024	$2^{128} - 1$	80	And, Xor, Rot, Add (mod 2^{64}), Or, Shr	192 256 112 128	154
SHA-3	*SHA3-224* *SHA3-256* *SHA3-384* *SHA3-512*	224 256 384 512	1600 (5 × 5 × 64)	1152 1088 832 576	Unlimited	24	And, Xor, Rot, Not	112 128 192 256	-
	SHAKE128 *SHAKE256*	*d* (arbitrary) *d* (arbitrary)		1344 1088				min(*d*/2, 128) min(*d*/2, 256)	-

See also [edit]

- Comparison of cryptographic hash functions
- cryptlib
- Crypto++
- Digital timestamping
- Hashcash
- Hash collision
- International Association for Cryptologic Research
- Libgcrypt
- md5deep
- OpenSSL
- PolarSSL
- RIPEMD-160
- Secure Hash Standard
- sha1sum
- Tiger (cryptography)
- Whirlpool (cryptography)

Notes [edit]

1. ^ Marc Stevens (19 June 2012). "Attacks on Hash Functions and Applications" (PDF). *PhD thesis*.
2. ^ http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf
3. ^ Schneier, Bruce (February 18, 2005). "Schneier on Security: Cryptanalysis of SHA-1".
4. ^ http://csrc.nist.gov/groups/ST/toolkit/secure_hashing.html
5. ^ Schneier on Security: NIST Hash Workshop Liveblogging (5)
6. ^ Hash cracked – heise Security
7. ^ "SHA1 Deprecation Policy". Microsoft. 2013-11-12. Retrieved 2013-11-14.
8. ^ "Intent to Deprecate: SHA-1 certificates". Google. 2014-09-03. Retrieved 2014-09-04.
9. ^ "Bug 942515 - stop accepting SHA-1-based SSL certificates with notBefore >= 2014-03-01 and notAfter >= 2017-01-01, or any SHA-1-based SSL certificates after 2017-01-01". Mozilla. Retrieved 2014-09-04.
10. ^ "CA:Problematic Practices - MozillaWiki". Mozilla. Retrieved 2014-09-09.
11. ^ "Phasing Out Certificates with SHA-1 based Signature Algorithms | Mozilla Security Blog". Mozilla. 2014-09-23.

Retrieved 2014-09-24.

12. ^ Felix "tmbinc" Domke (2008-04-24). "Thank you, Datel.". Retrieved 2014-10-05. "For verifiying the hash (which is the only thing they verify in the signature), they have chosen to use a function (strncmp) which stops on the first nullbyte – with a positive result. Out of the 160 bits of the SHA1-hash, up to 152 bits are thrown away."
13. ^ National Institute on Standards and Technology Computer Security Resource Center, NIST's March 2006 Policy on Hash Functions, accessed September 28, 2012.
14. ^ National Institute on Standards and Technology Computer Security Resource Center, NIST's Policy on Hash Functions, accessed September 28, 2012.
15. ^ "Tech Talk: Linus Torvalds on git". Retrieved November 13, 2013.
16. ^ 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
17. ^ Niels Ferguson, Bruce Schneier, and Tadayoshi Kohno, Cryptography Engineering, John Wiley & Sons, 2010. ISBN 978-0-470-47424-2
18. ^ Cryptology ePrint Archive
19. ^ *a* *b* Schneier on Security: SHA-1 Broken
20. ^ MIT.edu, Massachusetts Institute of Technology
21. ^ Fixing a hole in security | Tech News on ZDNet
22. ^ Schneier on Security: New Cryptanalytic Results Against SHA-1
23. ^ Notes on the Wang et al. 2^{63} SHA-1 Differential Path
24. ^ Christophe De Cannière, Christian Rechberger (2006-11-15). "Finding SHA-1 Characteristics: General Results and Applications".
25. ^ "IAIK Krypto Group – Description of SHA-1 Collision Search Project". Retrieved 2009-06-30.
26. ^ "Collisions for 72-step and 73-step SHA-1: Improvements in the Method of Characteristics". Retrieved 2010-07-24.
27. ^ "SHA-1 Collision Search Graz". Retrieved 2009-06-30.
28. ^ SHA-1 hash function under pressure – heise Security
29. ^ Crypto 2006 Rump Schedule
30. ^ Stéphane Manuel. "Classification and Generation of Disturbance Vectors for Collision Attacks against SHA-1" (PDF). Retrieved 2011-05-19.
31. ^ Stéphane Manuel. "Classification and Generation of Disturbance Vectors for Collision Attacks against SHA-1". Retrieved 2012-10-04. *the most efficient disturbance vector is Codeword2 first reported by Jutla and Patthak*
32. ^ SHA-1 collisions now 2^52
33. ^ International Association for Cryptologic Research
34. ^ Cryptanalysis of MD5 & SHA-1
35. ^ When Will We See Collisions for SHA-1?
36. ^ HashClash - Framework for MD5 & SHA-1 Differential Path Construction and Chosen-Prefix Collisions for MD5
37. ^ Freedom to Tinker " Blog Archive " Report from Crypto 2004
38. ^ Francois Grieu (Wed, 18 Aug 2004 05:06:02 +0200). "Re: Any advance news from the crypto rump session?". Newsgroup: sci.crypt. Usenet: fgrieu-05A994.05060218082004@individual.net. Check date values in: `|date=` (help)
39. ^ **(Chinese)** Sdu.edu.cn, Shandong University
40. ^ Stéphane Manuel, Thomas Peyrin (2008-02-11). "Collisions on SHA-0 in One Hour".
41. ^ National Institute of Standards and Technology
42. ^ http://www.faqs.org/rfcs/rfc3174.html
43. ^ Locktyukhin, Max; Farrel, Kathy (2010-03-31), "Improving the Performance of the Secure Hash Algorithm (SHA-1)", *Intel Software Knowledge Base* (Intel), retrieved 2010-04-02
44. ^ "Crypto++ 5.6.0 Benchmarks". Retrieved 2013-06-13.
45. ^ Found on an AMD Opteron 8354 2.2 GHz processor running 64-bit Linux[44]
46. ^ "Cryptanalysis of MD5 & SHA-1" (PDF). Retrieved 2013-04-25.

References [edit]

- Florent Chabaud, Antoine Joux: Differential Collisions in SHA-0. CRYPTO 1998. pp56–71
- Eli Biham, Rafi Chen, Near-Collisions of SHA-0, Cryptology ePrint Archive, Report 2004/146, 2004 (appeared on CRYPTO 2004), IACR.org
- Xiaoyun Wang, Hongbo Yu and Yiqun Lisa Yin, Efficient Collision Search Attacks on SHA-0, CRYPTO 2005, CMU.edu
- Xiaoyun Wang, Yiqun Lisa Yin and Hongbo Yu, Finding Collisions in the Full SHA-1, Crypto 2005 MIT.edu
- Henri Gilbert, Helena Handschuh: Security Analysis of SHA-256 and Sisters. Selected Areas in Cryptography 2003: pp175–193
- http://www.unixwiz.net/techtips/iguide-crypto-hashes.html
- "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.

- A. Cilardo, L. Esposito, A. Veniero, A. Mazzeo, V. Beltran, E. Ayugadé, A CellBE-based HPC application for the analysis of vulnerabilities in cryptographic hash functions, High Performance Computing and Communication international conference, August 2010

External links [edit]

- 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
- RFC 3174 (with sample C implementation)
- Interview with Yiqun Lisa Yin concerning the attack on SHA-1
- Explanation of the successful attacks on SHA-1 (3 pages, 2006)
- Cryptography Research – Hash Collision Q&A
- Online SHA1 hash crack using Rainbow tables
- Hash Project Web Site: software- and hardware-based cryptanalysis of SHA-1
- SHA-1 at DMOZ
- Lecture on SHA-1 on YouTube by Christof Paar

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 | Broken hash functions | Checksum algorithms | National Security Agency cryptography