MD5

The **MD5 message-digest algorithm** is a widely used hash function producing a 128-bit hash value. MD5 was designed by Ronald Rivest in 1991 to replace an earlier hash function MD4,[3] and was specified in 1992 as RFC 1321.

MD5 can be used as a checksum to verify data integrity against unintentional corruption. Historically it was widely used as a cryptographic hash function; however it has been found to suffer from extensive vulnerabilities. It remains suitable for other non-cryptographic purposes, for example for determining the partition for a particular key in a partitioned database, and may be preferred due to lower computational requirements than more recent Secure Hash Algorithms.[4]

MD5

General	
Designers	Ronald Rivest
First published	April 1992
Series	MD2, MD4, MD5, MD6
Cipher detail	
Digest sizes	128 bit
Block sizes	512 bit
Structure	Merkle–Damgård construction
Rounds	4[1]
Best public cryptanalysis	
A 2013 attack by Xie Tao, Fanbao Liu, and Dengguo Feng breaks MD5 collision resistance in 2^{18} time. This attack runs in less than a second on a regular computer.[2] MD5 is prone to length extension attacks.	

History and cryptanalysis

MD5 is one in a series of message digest algorithms designed by Professor Ronald Rivest of MIT (Rivest, 1992). When analytic work indicated that MD5's predecessor MD4 was likely to be insecure, Rivest designed MD5 in 1991 as a secure replacement. (Hans Dobbertin did indeed later find weaknesses in MD4.)

In 1993, Den Boer and Bosselaers gave an early, although limited, result of finding a "pseudo-collision" of the MD5 compression function; that is, two different initialization vectors that produce an identical digest.

In 1996, Dobbertin announced a collision of the compression function of MD5 (Dobbertin, 1996). While this was not an attack on the full MD5 hash function, it was close enough for cryptographers to recommend switching to a replacement, such as SHA-1 (also compromised since) or RIPEMD-160.

The size of the hash value (128 bits) is small enough to contemplate a birthday attack. MD5CRK was a distributed project started in March 2004 to demonstrate that MD5 is practically insecure by finding a collision using a birthday attack.

MD5CRK ended shortly after 17 August 2004, when collisions for the full MD5 were announced by Xiaoyun Wang, Dengguo Feng, Xuejia Lai, and Hongbo Yu.[5][6] Their analytical attack was reported to take only one hour on an IBM p690 cluster.[7]

On 1 March 2005, Arjen Lenstra, Xiaoyun Wang, and Benne de Weger demonstrated construction of two X.509 certificates with different public keys and the same MD5 hash value, a demonstrably practical collision.[8] The construction included private keys for both public keys. A few days later, Vlastimil Klima described an improved algorithm, able to construct MD5 collisions in a few hours on a single notebook computer.[9] On 18 March 2006, Klima published an algorithm that could find a collision within one minute on a single notebook computer, using a method he calls tunneling.[10]

Various MD5-related RFC errata have been published. In 2009, the United States Cyber Command used an MD5 hash value of their mission statement as a part of their official emblem.[11]

On 24 December 2010, Tao Xie and Dengguo Feng announced the first published single-block (512-bit) MD5 collision.[12] (Previous collision discoveries had relied on multi-block attacks.) For "security reasons", Xie and Feng did not disclose the new attack method. They issued a challenge to the cryptographic community, offering a US$10,000 reward to the first finder of a different 64-byte collision before 1 January 2013. Marc Stevens responded to the challenge and published colliding single-block messages as well as the construction algorithm and sources.[13]

In 2011 an informational RFC 6151[14] was approved to update the security considerations in MD5[15] and HMAC-MD5.[16]

Security

One basic requirement of any cryptographic hash function is that it should be computationally infeasible to find two distinct messages that hash to the same value. MD5 fails this requirement catastrophically. On 31 December 2008, the CMU Software Engineering Institute concluded that MD5 was essentially "cryptographically broken and unsuitable for further use".[17] The weaknesses of MD5 have been exploited in the field, most infamously by the Flame malware in 2012. As of 2019, MD5 continues to be widely used, despite its well-documented weaknesses and deprecation by security experts.[18]

A collision attack exists that can find collisions within seconds on a computer with a 2.6 GHz Pentium 4 processor (complexity of $2^{24.1}$).[19] Further, there is also a chosen-prefix collision attack that can produce a collision for two inputs with specified prefixes within seconds, using off-the-shelf computing hardware (complexity 2^{39}).[20] The ability to find collisions has been greatly aided by the use of off-the-shelf GPUs. On an NVIDIA GeForce 8400GS graphics processor, 16–18 million hashes per second can be computed. An NVIDIA GeForce 8800 Ultra can calculate more than 200 million hashes per second.[21]

These hash and collision attacks have been demonstrated in the public in various situations, including colliding document files[22][23] and digital certificates.[24] As of 2015, MD5 was demonstrated to be still quite widely used, most notably by security research and antivirus companies.[25]

As of 2019, one quarter of widely used content management systems were reported to still use MD5 for password hashing.[18]

Overview of security issues

In 1996, a flaw was found in the design of MD5. While it was not deemed a fatal weakness at the time, cryptographers began recommending the use of other algorithms, such as SHA-1, which has since been found to be vulnerable as well.[26] In 2004 it was shown that MD5 is not collision-resistant.[27] As such, MD5 is not suitable for applications like SSL certificates or digital signatures that rely on this property for digital security. Researchers additionally discovered more serious flaws in MD5, and described a feasible collision attack—a method to create a pair of inputs for which MD5 produces identical checksums.[5][28] Further advances were made in breaking MD5 in 2005, 2006, and 2007.[29] In December 2008, a group of researchers used this technique to fake SSL certificate validity.[24][30]

As of 2010, the CMU Software Engineering Institute considers MD5 "cryptographically broken and unsuitable for further use",[17] and most U.S. government applications now require the SHA-2 family of hash functions.[31] In 2012, the Flame malware exploited the weaknesses in MD5 to fake a Microsoft digital signature.[32]

Collision vulnerabilities

In 1996, collisions were found in the compression function of MD5, and Hans Dobbertin wrote in the RSA Laboratories technical newsletter, "The presented attack does not yet threaten practical applications of MD5, but it comes rather close ... in the future MD5 should no longer be implemented ... where a collision-resistant hash function is required."[33]

In 2005, researchers were able to create pairs of PostScript documents[34] and X.509 certificates[35] with the same hash. Later that year, MD5's designer Ron Rivest wrote that "md5 and sha1 are both clearly broken (in terms of collision-resistance)".[36]

On 30 December 2008, a group of researchers announced at the 25th Chaos Communication Congress how they had used MD5 collisions to create an intermediate certificate authority certificate that appeared to be legitimate when checked by its MD5 hash.[24] The researchers used a PS3 cluster at the EPFL in Lausanne, Switzerland[37] to change a normal SSL certificate issued by RapidSSL into a working CA certificate for that issuer, which could then be used to create other certificates that would appear to be legitimate and issued by RapidSSL. Verisign, the issuers of RapidSSL certificates, said they stopped issuing new certificates using MD5 as their checksum algorithm for RapidSSL once the vulnerability was announced.[38] Although Verisign declined to revoke existing certificates signed using MD5, their response was considered adequate by the authors of the exploit (Alexander Sotirov, Marc Stevens, Jacob Appelbaum, Arjen Lenstra, David Molnar, Dag Arne Osvik, and Benne de Weger).[24] Bruce Schneier wrote of the attack that "we already knew that MD5 is a broken hash function" and that "no one should be using MD5 anymore".[39] The SSL researchers wrote, "Our desired impact is that Certification Authorities will stop using MD5 in issuing new certificates. We also hope that use of MD5 in other applications will be reconsidered as well."[24]

In 2012, according to Microsoft, the authors of the Flame malware used an MD5 collision to forge a Windows code-signing certificate.[32]

MD5 uses the Merkle–Damgård construction, so if two prefixes with the same hash can be constructed, a common suffix can be added to both to make the collision more likely to be accepted as valid data by the application using it. Furthermore, current collision-finding techniques allow specifying an arbitrary *prefix*: an attacker can create two colliding files that both begin with the same content. All the attacker needs to generate two colliding files is a template file with a 128-byte block of data, aligned on a 64-byte boundary, that can be changed freely by the collision-finding algorithm. An example MD5 collision, with the two messages differing in 6 bytes, is:

```
d131dd02c5e6eec4 693d9a0698aff95c 2fcab58712467eab 4004583eb8fb7f89
55ad340609f4b302 83e488832571415a 085125e8f7cdc99f d91dbdf280373c5b
d8823e3156348f5b ae6dacd436c919c6 dd53e2b487da03fd 02396306d248cda0
e99f33420f577ee8 ce54b67080a80d1e c69821bcb6a88393 96f9652b6ff72a70
```

```
d131dd02c5e6eec4 693d9a0698aff95c 2fcab50712467eab 4004583eb8fb7f89
55ad340609f4b302 83e4888325f1415a 085125e8f7cdc99f d91dbd7280373c5b
d8823e3156348f5b ae6dacd436c919c6 dd53e23487da03fd 02396306d248cda0
e99f33420f577ee8 ce54b67080280d1e c69821bcb6a88393 96f965ab6ff72a70
```

Both produce the MD5 hash `79054025255fb1a26e4bc422aef54eb4`.[40] The difference between the two samples is that the leading bit in each nibble has been flipped. For example, the 20th byte (offset 0x13) in the top sample, 0x87, is 10000111 in binary. The leading bit in the byte (also the leading bit in the first nibble) is flipped to make 00000111, which is 0x07, as shown in the lower sample.

Later it was also found to be possible to construct collisions between two files with separately chosen prefixes. This technique was used in the creation of the rogue CA certificate in 2008. A new variant of parallelized collision searching using MPI was proposed by Anton Kuznetsov in 2014, which allowed finding a collision in 11 hours on a computing cluster.[41]

Preimage vulnerability

In April 2009, an attack against MD5 was published that breaks MD5's preimage resistance. This attack is only theoretical, with a computational complexity of $2^{123.4}$ for full preimage.[42][43]

Applications

MD5 digests have been widely used in the software world to provide some assurance that a transferred file has arrived intact. For example, file servers often provide a pre-computed MD5 (known as md5sum) checksum for the files, so that a user can compare the checksum of the downloaded file to it. Most unix-based operating systems include MD5 sum utilities in their distribution packages; Windows users may use the included PowerShell function "Get-FileHash", the included command line function "certutil -hashfile <filename> md5",[44][45] install a Microsoft utility,[46][47] or use third-party applications. Android ROMs also use this type of checksum.

As it is easy to generate MD5 collisions, it is possible for the person who created the file to create a second file with the same checksum, so this technique cannot protect against some forms of malicious tampering. In some cases, the checksum cannot be trusted (for example, if it was obtained over the same channel as the downloaded file), in which case MD5 can only provide error-checking functionality: it will recognize a corrupt or incomplete download, which becomes more likely when downloading larger files.

Historically, MD5 has been used to store a one-way hash of a password, often with key stretching.[48][49] NIST does not include MD5 in their list of recommended hashes for password storage.[50]

MD5 is also used in the field of electronic discovery, to provide a unique identifier for each document that is exchanged during the legal discovery process. This method can be used to replace the Bates stamp numbering system that has been used for decades during the exchange of paper documents. As above, this usage should be discouraged due to the ease of collision attacks.

Algorithm

MD5 processes a variable-length message into a fixed-length output of 128 bits. The input message is broken up into chunks of 512-bit blocks (sixteen 32-bit words); the message is always padded even if its original length is divisible by 512 (see RFC 1321, section 3.1). The padding works as follows: first, a single bit, 1, is appended to the end of the message. This is followed by as many zeros as are required to bring the length of the message up to 64 bits fewer than a multiple of 512. The remaining bits are filled up with 64 bits representing the length of the original message, modulo 2^{64}.

The main MD5 algorithm operates on a 128-bit state, divided into four 32-bit words, denoted A, B, C, and D. These are initialized to certain fixed constants. The main algorithm then uses each 512-bit message block in turn to modify the state. The processing of a message block consists of four similar stages, termed *rounds*; each round is composed of 16 similar operations based on a non-linear function F, modular addition, and left rotation. Figure 1 illustrates one operation within a round. There are four possible functions; a different one is used in each round:

$$\begin{aligned} F(B,C,D) &= (B \wedge C) \vee (\neg B \wedge D) \\ G(B,C,D) &= (B \wedge D) \vee (C \wedge \neg D) \\ H(B,C,D) &= B \oplus C \oplus D \\ I(B,C,D) &= C \oplus (B \vee \neg D) \end{aligned}$$

$\oplus, \wedge, \vee, \neg$ denote the XOR, AND, OR and NOT operations respectively.

Figure 1. One MD5 operation. MD5 consists of 64 of these operations, grouped in four rounds of 16 operations. F is a nonlinear function; one function is used in each round. M_i denotes a 32-bit block of the message input, and K_i denotes a 32-bit constant, different for each operation. $<<<_s$ denotes a left bit rotation by s places; s varies for each operation. $\boxplus$ denotes addition modulo 2^{32}.

Pseudocode

The MD5 hash is calculated according to this algorithm.[51] All values are in little-endian.

```
// : All variables are unsigned 32 bit and wrap modulo 2^32 when calculating
var int s[64], K[64]
var int i

// s specifies the per-round shift amounts
s[ 0..15] := { 7, 12, 17, 22,  7, 12, 17, 22,  7, 12, 17, 22,  7, 12, 17, 22 }
s[16..31] := { 5,  9, 14, 20,  5,  9, 14, 20,  5,  9, 14, 20,  5,  9, 14, 20 }
s[32..47] := { 4, 11, 16, 23,  4, 11, 16, 23,  4, 11, 16, 23,  4, 11, 16, 23 }
s[48..63] := { 6, 10, 15, 21,  6, 10, 15, 21,  6, 10, 15, 21,  6, 10, 15, 21 }

// Use binary integer part of the sines of integers (Radians) as constants:
for i from 0 to 63 do
    K[i] := floor(2^32 × abs(sin(i + 1)))
end for
// (Or just use the following precomputed table):
K[ 0.. 3] := { 0xd76aa478, 0xe8c7b756, 0x242070db, 0xc1bdceee }
K[ 4.. 7] := { 0xf57c0faf, 0x4787c62a, 0xa8304613, 0xfd469501 }
K[ 8..11] := { 0x698098d8, 0x8b44f7af, 0xffff5bb1, 0x895cd7be }
K[12..15] := { 0x6b901122, 0xfd987193, 0xa679438e, 0x49b40821 }
K[16..19] := { 0xf61e2562, 0xc040b340, 0x265e5a51, 0xe9b6c7aa }
K[20..23] := { 0xd62f105d, 0x02441453, 0xd8a1e681, 0xe7d3fbc8 }
K[24..27] := { 0x21e1cde6, 0xc33707d6, 0xf4d50d87, 0x455a14ed }
```

```
K[28..31] := { 0xa9e3e905, 0xfcefa3f8, 0x676f02d9, 0x8d2a4c8a }
K[32..35] := { 0xfffa3942, 0x8771f681, 0x6d9d6122, 0xfde5380c }
K[36..39] := { 0xa4beea44, 0x4bdecfa9, 0xf6bb4b60, 0xbebfbc70 }
K[40..43] := { 0x289b7ec6, 0xeaa127fa, 0xd4ef3085, 0x04881d05 }
K[44..47] := { 0xd9d4d039, 0xe6db99e5, 0x1fa27cf8, 0xc4ac5665 }
K[48..51] := { 0xf4292244, 0x432aff97, 0xab9423a7, 0xfc93a039 }
K[52..55] := { 0x655b59c3, 0x8f0ccc92, 0xffeff47d, 0x85845dd1 }
K[56..59] := { 0x6fa87e4f, 0xfe2ce6e0, 0xa3014314, 0x4e0811a1 }
K[60..63] := { 0xf7537e82, 0xbd3af235, 0x2ad7d2bb, 0xeb86d391 }

// Initialize variables:
var int a0 := 0x67452301   // A
var int b0 := 0xefcdab89   // B
var int c0 := 0x98badcfe   // C
var int d0 := 0x10325476   // D

// Pre-processing: adding a single 1 bit
append "1" bit to message<
 // Notice: the input bytes are considered as bit strings,
 //  where the first bit is the most significant bit of the byte.[52]

// Pre-processing: padding with zeros
append "0" bit until message length in bits ≡ 448 (mod 512)

// Notice: the two padding steps above are implemented in a simpler way
  //  in implementations that only work with complete bytes: append 0x80
  //  and pad with 0x00 bytes so that the message length in bytes ≡ 56 (mod 64).

append original length in bits mod 2^64 to message

// Process the message in successive 512-bit chunks:
for each 512-bit chunk of padded message do
    break chunk into sixteen 32-bit words M[j], 0 ≤ j ≤ 15
    // Initialize hash value for this chunk:
    var int A := a0
    var int B := b0
    var int C := c0
    var int D := d0
    // Main loop:
    for i from 0 to 63 do
        var int F, g
        if 0 ≤ i ≤ 15 then
            F := (B and C) or ((not B) and D)
            g := i
        else if 16 ≤ i ≤ 31 then
            F := (D and B) or ((not D) and C)
            g := (5×i + 1) mod 16
        else if 32 ≤ i ≤ 47 then
            F := B xor C xor D
            g := (3×i + 5) mod 16
        else if 48 ≤ i ≤ 63 then
            F := C xor (B or (not D))
            g := (7×i) mod 16
        // Be wary of the below definitions of a,b,c,d
        F := F + A + K[i] + M[g]  // M[g] must be a 32-bit block
        A := D
        D := C
        C := B
        B := B + leftrotate(F, s[i])
    end for
    // Add this chunk's hash to result so far:
    a0 := a0 + A
    b0 := b0 + B
    c0 := c0 + C
    d0 := d0 + D
end for

var char digest[16] := a0 append b0 append c0 append d0 // (Output is in little-endian)
```

Instead of the formulation from the original RFC 1321 shown, the following may be used for improved efficiency (useful if assembly language is being used – otherwise, the compiler will generally optimize the above code. Since each computation is dependent on another in these formulations, this is often slower than the above method where the nand/and can be parallelised):

```
( 0 ≤ i ≤ 15): F := D xor (B and (C xor D))
(16 ≤ i ≤ 31): F := C xor (D and (B xor C))
```

MD5 hashes

The 128-bit (16-byte) MD5 hashes (also termed *message digests*) are typically represented as a sequence of 32 hexadecimal digits. The following demonstrates a 43-byte ASCII input and the corresponding MD5 hash:

```
MD5("The quick brown fox jumps over the lazy dog") =
9e107d9d372bb6826bd81d3542a419d6
```

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:

```
MD5("The quick brown fox jumps over the lazy dog.") =
e4d909c290d0fb1ca068ffaddf22cbd0
```

The hash of the zero-length string is:

```
MD5("") =
d41d8cd98f00b204e9800998ecf8427e
```

The MD5 algorithm is specified for messages consisting of any number of bits; it is not limited to multiples of eight bits (octets, bytes). Some MD5 implementations such as md5sum might be limited to octets, or they might not support *streaming* for messages of an initially undetermined length.

Implementations

Below is a list of cryptography libraries that support MD5:

- Botan
- Bouncy Castle
- cryptlib
- Crypto++
- Libgcrypt
- Nettle
- OpenSSL
- wolfSSL

See also

- Comparison of cryptographic hash functions
- Hash function security summary
- HashClash
- MD5Crypt
- md5deep

- md5sum
- MD6
- SHA-1
- SHA-2

References

1. Rivest, R. (April 1992). "Step 4. Process Message in 16-Word Blocks" (https://datatracker.ietf.org/doc/html/rfc1321#section-3.4). *The MD5 Message-Digest Algorithm* (https://www.rfc-editor.org/rfc/rfc1321). IETF. p. 5. sec. 3.4. doi:10.17487/RFC1321 (https://doi.org/10.17487%2FRFC1321). RFC 1321 (https://datatracker.ietf.org/doc/html/rfc1321). Retrieved 10 October 2018.
2. Xie Tao; Fanbao Liu; Dengguo Feng (2013). "Fast Collision Attack on MD5" (https://eprint.iacr.org/2013/170.pdf) (PDF). *Cryptology ePrint Archive*. Archived (https://web.archive.org/web/20210202021701/https://eprint.iacr.org/2013/170.pdf) (PDF) from the original on 2 February 2021. Retrieved 3 December 2013.
3. Ciampa, Mark (2009). *CompTIA Security+ 2008 in depth* (https://archive.org/details/comptiasecurity20000ciam). Australia; United States: Course Technology/Cengage Learning. p. 290 (https://archive.org/details/comptiasecurity20000ciam/page/290). ISBN 978-1-59863-913-1.
4. Kleppmann, Martin (2 April 2017). *Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems* (1 ed.). O'Reilly Media. p. 203. ISBN 978-1449373320.
5. J. Black, M. Cochran, T. Highland: A Study of the MD5 Attacks: Insights and Improvements (http://www.cs.colorado.edu/~jrblack/papers/md5e-full.pdf) Archived (https://web.archive.org/web/20150101093005/http://www.cs.colorado.edu/%7Ejrblack/papers/md5e-full.pdf) 1 January 2015 at the Wayback Machine, 3 March 2006. Retrieved 27 July 2008.
6. Hawkes, Philip; Paddon, Michael; Rose, Gregory G. (13 October 2004). "Musings on the Wang et al. MD5 Collision" (https://web.archive.org/web/20181105220829/https://eprint.iacr.org/2004/264). *Cryptology ePrint Archive*. Archived from the original (https://eprint.iacr.org/2004/264) on 5 November 2018. Retrieved 10 October 2018.
7. Bishop Fox (26 September 2013). "Fast MD5 and MD4 Collision Generators" (https://web.archive.org/web/20170426171733/http://www.bishopfox.com/resources/tools/other-free-tools/md4md5-collision-code/). *BishopFox*. Archived from the original (http://www.bishopfox.com/resources/tools/other-free-tools/md4md5-collision-code/) on 26 April 2017. Retrieved 10 February 2014.
8. Lenstra, Arjen; Wang, Xiaoyun; Weger, Benne de (1 March 2005). "Colliding X.509 Certificates" (http://eprint.iacr.org/2005/067). *Cryptology ePrint Archive*. Archived (https://web.archive.org/web/20170523000238/http://eprint.iacr.org/2005/067) from the original on 23 May 2017. Retrieved 10 October 2018.
9. Klíma, Vlastimil (5 March 2005). "Finding MD5 Collisions – a Toy For a Notebook" (http://eprint.iacr.org/2005/075). *Cryptology ePrint Archive*. Archived (https://web.archive.org/web/20170517175404/http://eprint.iacr.org/2005/075) from the original on 17 May 2017. Retrieved 10 October 2018.
10. Vlastimil Klima: Tunnels in Hash Functions: MD5 Collisions Within a Minute (http://eprint.iacr.org/2006/105) Archived (https://web.archive.org/web/20110806052832/http://eprint.iacr.org/2006/105) 6 August 2011 at the Wayback Machine, Cryptology ePrint Archive Report 2006/105, 18 March 2006, revised 17 April 2006. Retrieved 27 July 2008.
11. "Code Cracked! Cyber Command Logo Mystery Solved" (https://www.wired.com/dangerroom/2010/07/code-cracked-cyber-command-logos-mystery-solved/). *USCYBERCOM*. Wired News. 8 July 2010. Archived (https://web.archive.org/web/20140217212938/http://www.wired.com/dangerroom/2010/07/code-cracked-cyber-command-logos-mystery-solved/) from the original on 17 February 2014. Retrieved 29 July 2011.
12. Tao Xie; Dengguo Feng (2010). "Construct MD5 Collisions Using Just A Single Block Of Message" (http://eprint.iacr.org/2010/643) (PDF). Archived (https://web.archive.org/web/20170514185806/http://eprint.iacr.org/2010/643) from the original on 14 May 2017. Retrieved 28 July 2011.
13. "Marc Stevens – Research – Single-block collision attack on MD5" (http://marc-stevens.nl/research/md5-1block-collision/). Marc-stevens.nl. 2012. Archived (https://web.archive.org/web/20170515070731/http://marc-stevens.nl/research/md5-1block-collision/) from the original on 15 May 2017. Retrieved 10 April 2014.

14. Turner, Sean (March 2011). "RFC 6151 – Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms" (https://tools.ietf.org/html/rfc6151). *Internet Engineering Task Force*. doi:10.17487/RFC6151 (https://doi.org/10.17487%2FRFC6151). Archived (https://web.archive.org/web/20170615213134/https://tools.ietf.org/html///rfc6151) from the original on 15 June 2017. Retrieved 11 November 2013.
15. Rivest, Ronald L. (April 1992). "RFC 1321 – The MD5 Message-Digest Algorithm" (https://tools.ietf.org/html/rfc1321). *Internet Engineering Task Force*. doi:10.17487/RFC1321 (https://doi.org/10.17487%2FRFC1321). hdl:1721.1/149165 (https://hdl.handle.net/1721.1%2F149165). Archived (https://web.archive.org/web/20210409200613/https://tools.ietf.org/html/rfc1321) from the original on 9 April 2021. Retrieved 5 October 2013.
16. Krawczyk, Hugo; Bellare, Mihir; Canetti, Ran (February 1997). "RFC 2104 – HMAC: Keyed-Hashing for Message Authentication" (https://tools.ietf.org/html/rfc2104). *Internet Engineering Task Force*. doi:10.17487/RFC2104 (https://doi.org/10.17487%2FRFC2104). Archived (https://web.archive.org/web/20210415003434/https://tools.ietf.org/html/rfc2104) from the original on 15 April 2021. Retrieved 5 October 2013.
17. Dougherty, Chad R. (31 December 2008). "Vulnerability Note VU#836068 MD5 vulnerable to collision attacks" (https://www.kb.cert.org/vuls/id/836068). *Vulnerability notes database*. CERT Carnegie Mellon University Software Engineering Institute. Archived (https://web.archive.org/web/20110726144513/http://www.kb.cert.org/vuls/id/836068) from the original on 26 July 2011. Retrieved 3 February 2017.
18. Cimpanu, Catalin. "A quarter of major CMSs use outdated MD5 as the default password hashing scheme" (https://www.zdnet.com/article/a-quarter-of-major-cmss-use-outdated-md5-as-the-default-password-hashing-scheme/). *ZDNet*. Archived (https://web.archive.org/web/20210124103125/https://www.zdnet.com/article/a-quarter-of-major-cmss-use-outdated-md5-as-the-default-password-hashing-scheme/) from the original on 24 January 2021. Retrieved 17 June 2019.
19. M.M.J. Stevens (June 2007). *On Collisions for MD5* (http://www.win.tue.nl/hashclash/On%20Collisions%20for%20MD5%20-%20M.M.J.%20Stevens.pdf) (PDF) (Master's thesis). Archived (https://web.archive.org/web/20170517115509/http://www.win.tue.nl/hashclash/On%20Collisions%20for%20MD5%20-%20M.M.J.%20Stevens.pdf) (PDF) from the original on 17 May 2017. Retrieved 31 March 2010.
20. Marc Stevens; Arjen Lenstra; Benne de Weger (16 June 2009). "Chosen-prefix Collisions for MD5 and Applications" (https://web.archive.org/web/20111109092157/https://documents.epfl.ch/users/l/le/lenstra/public/papers/lat.pdf) (PDF). *École Polytechnique Fédérale de Lausanne*. Archived from the original (https://documents.epfl.ch/users/l/le/lenstra/public/papers/lat.pdf) (PDF) on 9 November 2011. Retrieved 31 March 2010.
21. "New GPU MD5 cracker cracks more than 200 million hashes per second" (http://bvernoux.free.fr/md5/index.php). Archived (https://web.archive.org/web/20110511121239/http://bvernoux.free.fr/md5/index.php) from the original on 11 May 2011. Retrieved 25 March 2011.
22. Magnus Daum, Stefan Lucks. "Hash Collisions (The Poisoned Message Attack)" (https://web.archive.org/web/20100327141611/http://th.informatik.uni-mannheim.de/people/lucks/HashCollisions/). *Eurocrypt 2005 rump session*. Archived from the original (http://th.informatik.uni-mannheim.de/People/lucks/HashCollisions/) on 27 March 2010.
23. Max Gebhardt; Georg Illies; Werner Schindler (31 October 2005). "A Note on the Practical Value of Single Hash Collisions for Special File Formats" (https://web.archive.org/web/20080917182949/http://csrc.nist.gov/groups/ST/hash/documents/Illies_NIST_05.pdf) (PDF). *National Institute of Standards and Technology*. Archived from the original (http://csrc.nist.gov/groups/ST/hash/documents/Illies_NIST_05.pdf) (PDF) on 17 September 2008.
24. Sotirov, Alexander; Marc Stevens; Jacob Appelbaum; Arjen Lenstra; David Molnar; Dag Arne Osvik; Benne de Weger (30 December 2008). "MD5 considered harmful today" (http://www.win.tue.nl/hashclash/rogue-ca/). Archived (https://web.archive.org/web/20170325033522/http://www.win.tue.nl/hashclash/rogue-ca/) from the original on 25 March 2017. Retrieved 30 December 2008. Announced (https://events.ccc.de/congress/2008/Fahrplan/events/3023.en.html) Archived (https://web.archive.org/web/20181116081156/https://events.ccc.de/congress/2008/Fahrplan/events/3023.en.html) 16 November 2018 at the Wayback Machine at the 25th Chaos Communication Congress.
25. "Poisonous MD5 – Wolves Among the Sheep | Silent Signal Techblog" (http://blog.silentsignal.eu/2015/06/10/poisonous-md5-wolves-among-the-sheep/). 10 June 2015. Archived (https://web.archive.org/web/20150610115424/http://blog.silentsignal.eu/2015/06/10/poisonous-md5-wolves-among-the-sheep/) from the original on 10 June 2015. Retrieved 10 June 2015.
26. Hans Dobbertin (Summer 1996). "The Status of MD5 After a Recent Attack" (http://ftp.arnes.si/packages/crypto-tools/rsa.com/cryptobytes/crypto2n2.pdf.gz). *CryptoBytes*. Retrieved 22 October 2013.

27. Xiaoyun Wang; Hongbo Yu (2005). "How to Break MD5 and Other Hash Functions" (https://web.archive.org/web/20090521024709/http://merlot.usc.edu/csac-f06/papers/Wang05a.pdf) (PDF). *Advances in Cryptology – Lecture Notes in Computer Science*. pp. 19–35. Archived from the original (http://merlot.usc.edu/csac-f06/papers/Wang05a.pdf) (PDF) on 21 May 2009. Retrieved 21 December 2009.
28. Xiaoyun Wang, Dengguo ,k.,m.,m, HAVAL-128 and RIPEMD, Cryptology ePrint Archive Report 2004/199, 16 August 2004, revised 17 August 2004. Retrieved 27 July 2008.
29. Marc Stevens, Arjen Lenstra, Benne de Weger: Vulnerability of software integrity and code signing applications to chosen-prefix collisions for MD5 (http://www.win.tue.nl/hashclash/SoftIntCodeSign/) Archived (https://web.archive.org/web/20071213023720/http://www.win.tue.nl/hashclash/SoftIntCodeSign/) 13 December 2007 at the Wayback Machine, 30 November 2007. Retrieved 27 July 2008.
30. Stray, Jonathan (30 December 2008). "Web browser flaw could put e-commerce security at risk" (https://web.archive.org/web/20130828142658/http://news.cnet.com/8301-1009_3-10129693-83.html). CNET.com. Archived from the original (https://news.cnet.com/8301-1009_3-10129693-83.html) on 28 August 2013. Retrieved 24 February 2009.
31. "NIST.gov — Computer Security Division — Computer Security Resource Center" (https://web.archive.org/web/20110609064344/http://csrc.nist.gov/groups/ST/hash/policy.html). Csrc.nist.gov. Archived from the original (http://csrc.nist.gov/groups/ST/hash/policy.html) on 9 June 2011. Retrieved 9 August 2010.
32. "Flame malware collision attack explained" (https://web.archive.org/web/20120608225029/http://blogs.technet.com/b/srd/archive/2012/06/06/more-information-about-the-digital-certificates-used-to-sign-the-flame-malware.aspx). Archived from the original (http://blogs.technet.com/b/srd/archive/2012/06/06/more-information-about-the-digital-certificates-used-to-sign-the-flame-malware.aspx) on 8 June 2012. Retrieved 7 June 2012.
33. Dobbertin, Hans (Summer 1996). "The Status of MD5 After a Recent Attack" (ftp://ftp.rsasecurity.com/pub/cryptobytes/crypto2n2.pdf) (PDF). *RSA Laboratories CryptoBytes* (FTP). p. 1. Retrieved 10 August 2010. "The presented attack does not yet threaten practical applications of MD5, but it comes rather close. .... [*sic*] in the future MD5 should no longer be implemented... [*sic*] where a collision-resistant hash function is required." (To view documents see Help:FTP)
34. "Schneier on Security: More MD5 Collisions" (http://www.schneier.com/blog/archives/2005/06/more_md5_collis.html). Schneier.com. Archived (https://web.archive.org/web/20210411035935/https://www.schneier.com/blog/archives/2005/06/more_md5_collis.html) from the original on 11 April 2021. Retrieved 9 August 2010.
35. "Colliding X.509 Certificates" (http://www.win.tue.nl/~bdeweger/CollidingCertificates/). Win.tue.nl. Archived (https://web.archive.org/web/20170515022608/http://www.win.tue.nl/~bdeweger/CollidingCertificates/) from the original on 15 May 2017. Retrieved 9 August 2010.
36. "[Python-Dev] hashlib — faster md5/sha, adds sha256/512 support" (http://mail.python.org/pipermail/python-dev/2005-December/058850.html). Mail.python.org. 16 December 2005. Archived (https://web.archive.org/web/20210506232819/https://mail.python.org/pipermail/python-dev/2005-December/058850.html) from the original on 6 May 2021. Retrieved 9 August 2010.
37. "Researchers Use PlayStation Cluster to Forge a Web Skeleton Key" (http://blog.wired.com/27bstroke6/2008/12/berlin.html). *Wired*. 31 December 2008. Archived (https://web.archive.org/web/20090421023048/http://blog.wired.com/27bstroke6/2008/12/berlin.html) from the original on 21 April 2009. Retrieved 31 December 2008.
38. Callan, Tim (31 December 2008). "This morning's MD5 attack — resolved" (https://web.archive.org/web/20090116180944/http://blogs.verisign.com/ssl-blog/2008/12/on_md5_vulnerabilities_and_mit.php). Verisign. Archived from the original (https://blogs.verisign.com/ssl-blog/2008/12/on_md5_vulnerabilities_and_mit.php) on 16 January 2009. Retrieved 31 December 2008.
39. Bruce Schneier (31 December 2008). "Forging SSL Certificates" (http://www.schneier.com/blog/archives/2008/12/forging_ssl_cer.html). Schneier on Security. Archived (https://web.archive.org/web/20201109014745/https://www.schneier.com/blog/archives/2008/12/forging_ssl_cer.html) from the original on 9 November 2020. Retrieved 10 April 2014.
40. Eric Rescorla (17 August 2004). "A real MD5 collision" (https://web.archive.org/web/20140815234704/http://www.rtfm.com/movabletype/archives/2004_08.html#001055). *Educated Guesswork (blog)*. Archived from the original (http://www.rtfm.com/movabletype/archives/2004_08.html#001055) on 15 August 2014. Retrieved 13 April 2015.

41. Anton A. Kuznetsov. "An algorithm for MD5 single-block collision attack using high performance computing cluster" (http://eprint.iacr.org/2014/871.pdf) (PDF). IACR. Archived (https://web.archive.org/web/20160604093753/https://eprint.iacr.org/2014/871.pdf) (PDF) from the original on 4 June 2016. Retrieved 3 November 2014.
42. Yu Sasaki; Kazumaro Aoki (16 April 2009). "Finding Preimages in Full MD5 Faster Than Exhaustive Search". *Advances in Cryptology - EUROCRYPT 2009*. Lecture Notes in Computer Science. Vol. 5479. Springer Berlin Heidelberg. pp. 134–152. doi:10.1007/978-3-642-01001-9_8 (https://doi.org/10.1007%2F978-3-642-01001-9_8). ISBN 978-3-642-01000-2.
43. Ming Mao and Shaohui Chen and Jin Xu (2009). "Construction of the Initial Structure for Preimage Attack of MD5". *2009 International Conference on Computational Intelligence and Security*. Vol. 1. IEEE Computer Society. pp. 442–445. doi:10.1109/CIS.2009.214 (https://doi.org/10.1109%2FCIS.2009.214). ISBN 978-0-7695-3931-7. S2CID 16512325 (https://api.semanticscholar.org/CorpusID:16512325).
44. "Finding Checksum Values in Windows 10" (https://answers.microsoft.com/en-us/windows/forum/all/finding-checksum-values-in-windows-10/dbc3c569-4b5a-4967-8810-c25255cdc1fd). Microsoft Community. Archived (https://web.archive.org/web/20240111185213/https://answers.microsoft.com/en-us/windows/forum/all/finding-checksum-values-in-windows-10/dbc3c569-4b5a-4967-8810-c25255cdc1fd) from the original on 11 January 2024. Retrieved 23 November 2023.
45. "certutil" (https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/certutil). *certutil*. Microsoft Learn. Archived (https://web.archive.org/web/20231123190431/https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/certutil) from the original on 23 November 2023. Retrieved 23 November 2023.
46. "Availability and description of the File Checksum Integrity Verifier utility" (https://support.microsoft.com/kb/841290/en-us). Microsoft Support. 17 June 2013. Archived (https://web.archive.org/web/20150215153720/http://support.microsoft.com/kb/841290/en-us) from the original on 15 February 2015. Retrieved 10 April 2014.
47. "How to compute the MD5 or SHA-1 cryptographic hash values for a file" (https://support.microsoft.com/kb/889768/en-us). Microsoft Support. 23 January 2007. Archived (https://web.archive.org/web/20150309230754/http://support.microsoft.com/kb/889768/en-us) from the original on 9 March 2015. Retrieved 10 April 2014.
48. "FreeBSD Handbook, Security – DES, Blowfish, MD5, and Crypt" (https://www.freebsd.org/cgi/man.cgi?crypt(3)). Archived (https://web.archive.org/web/20170218081024/https://www.freebsd.org/cgi/man.cgi?crypt(3)) from the original on 18 February 2017. Retrieved 19 October 2014.
49. "Synopsis – man pages section 4: File Formats" (http://docs.oracle.com/cd/E26505_01/html/816-5174/policy.conf-4.html). Docs.oracle.com. 1 January 2013. Archived (https://web.archive.org/web/20160304204917/http://docs.oracle.com/cd/E26505_01/html/816-5174/policy.conf-4.html) from the original on 4 March 2016. Retrieved 10 April 2014.
50. NIST SP 800-132 (http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-132.pdf) Archived (https://web.archive.org/web/20161201044854/http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-132.pdf) 1 December 2016 at the Wayback Machine Section 5.1
51. "Reference Source" (https://referencesource.microsoft.com/#System.Workflow.Runtime/MD5HashHelper.cs,5a97802b6014fccc,references). Archived (https://web.archive.org/web/20210621192842/https://referencesource.microsoft.com/#System.Workflow.Runtime/MD5HashHelper.cs,5a97802b6014fccc,references) from the original on 21 June 2021. Retrieved 23 December 2020.
52. RFC 1321, section 2, "Terminology and Notation", Page 2.

Further reading

- Berson, Thomas A. (1992). "Differential Cryptanalysis Mod 2^{32} with Applications to MD5". *EUROCRYPT*. pp. 71–80. ISBN 3-540-56413-6.
- Bert den Boer; Antoon Bosselaers (1993). "Collisions for the Compression Function of MD5". *Advances in Cryptology – EUROCRYPT '93*. EUROCRYPT. Berlin; London: Springer. pp. 293–304. ISBN 978-3-540-57600-6.
- Hans Dobbertin, Cryptanalysis of MD5 compress. Announcement on Internet, May 1996. "CiteSeerX" (http://citeseer.ist.psu.edu/dobbertin96cryptanalysis.html). Citeseer.ist.psu.edu. Archived (https://web.archive.org/w

eb/20080624043846/http://citeseer.ist.psu.edu/dobbertin96cryptanalysis.html) from the original on 24 June 2008. Retrieved 9 August 2010.

- Dobbertin, Hans (1996). "The Status of MD5 After a Recent Attack" (http://ftp.arnes.si/packages/crypto-tools/rsa.com/cryptobytes/crypto2n2.pdf.gz). *CryptoBytes*. **2** (2).
- Xiaoyun Wang; Hongbo Yu (2005). "How to Break MD5 and Other Hash Functions" (https://web.archive.org/web/20090521001714/http://www.infosec.sdu.edu.cn/uploadfile/papers/How%20to%20Break%20MD5%20and%20Other%20Hash%20Functions.pdf) (PDF). *EUROCRYPT*. ISBN 3-540-25910-4. Archived from the original (http://www.infosec.sdu.edu.cn/uploadfile/papers/How%20to%20Break%20MD5%20and%20Other%20Hash%20Functions.pdf) (PDF) on 21 May 2009. Retrieved 6 March 2008.

External links

- W3C recommendation on MD5 (http://www.w3.org/TR/1998/REC-DSig-label/MD5-1_0) Archived (https://web.archive.org/web/20141228122551/http://www.w3.org/TR/1998/REC-DSig-label/MD5-1_0) 28 December 2014 at the Wayback Machine
- MD5 Calculator (https://fe-tool.com/en-us/hash/md5) Archived (https://web.archive.org/web/20221116025344/https://fe-tool.com/en-us/hash/md5) 16 November 2022 at the Wayback Machine