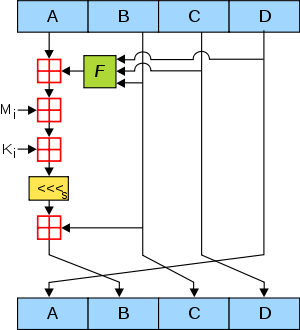
The **MD5 message-digest algorithm** is a widely used [cryptographic hash function](https://en.wikipedia.org/wiki/Cryptographic_hash_function) producing a 128-[bit](https://en.wikipedia.org/wiki/Bit) (16-byte) [hash value](https://en.wikipedia.org/wiki/Hash_value), typically expressed in text format as a 32 digit [hexadecimal](https://en.wikipedia.org/wiki/Hexadecimal) number. MD5 has been utilized in a wide variety of cryptographic applications, and is also commonly used to verify [data integrity](https://en.wikipedia.org/wiki/Data_integrity).

Algorithm [[edit](https://en.wikipedia.org/w/index.php?title=MD5&action=edit&section=7)]

[](https://en.wikipedia.org/wiki/File:MD5.svg)

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. *Mi* denotes a 32-bit block of the message input, and *Ki* denotes a 32-bit constant, different for each operation. [left shift](https://en.wikipedia.org/wiki/File:Lll.png)*S* denotes a left bit rotation by *s* places; *s* varies for each operation. [Addition](https://en.wikipedia.org/wiki/File:Boxplus.png)Denotes addition modulo 232.

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 [padded](https://en.wikipedia.org/wiki/Padding_(cryptography)) so that its length is divisible by 512. 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 less than a multiple of 512. The remaining bits are filled up with 64 bits representing the length of the original message, modulo 264.

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](https://en.wikipedia.org/wiki/Modular_addition), and left rotation. Figure 1 illustrates one operation within a round. There are four possible functions *F*; a different one is used in each round:

\begin{align}
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{align}


\oplus, \wedge, \vee, \neg Denote the [XOR](https://en.wikipedia.org/wiki/XOR), [AND](https://en.wikipedia.org/wiki/Logical_conjunction), [OR](https://en.wikipedia.org/wiki/Logical_disjunction) and [NOT](https://en.wikipedia.org/wiki/Negation) operations respectively.

### Pseudo code[[edit](https://en.wikipedia.org/w/index.php?title=MD5&action=edit&section=8)]

The MD5 hash is calculated according to this algorithm. All values are in [little-endian](https://en.wikipedia.org/wiki/Endianness).

//*Note: All variables are unsigned 32 bit and wrap modulo 2^32 when calculating*

**Var** *int* [64] s, K

//*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

K[i]:= floor (232 × abs (sin (i + 1)))

**End for**

// *(Or just use the following precompiled 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 bits strings,

where the first bit is the most significant bit of the byte.[[48]](https://en.wikipedia.org/wiki/MD5#cite_note-48)

//*Pre-processing: padding with zeros*

**append** "0" bit **until** message length in bits ≡ 448 (mod 512)

**append** original length in bits **mod** (2 **pow** 64) **to** message

//*Process the message in successive 512-bit chunks:*

**for each** *512-bit* chunk **of** message

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

**if** 0 ≤ i ≤ 15 **then**

F := (B **and** C) **or** ((**not** B) **and** D)

g := i

**else if** 16 ≤ i ≤ 31

F := (D **and** B) **or** ((**not** D) **and** C)

g := (5×i + 1) **mod** 16

**else if** 32 ≤ i ≤ 47

F := B **xor** C **xor** D

g := (3×i + 5) **mod** 16

**else if** 48 ≤ i ≤ 63

F := C **xor** (B **or** (**not** D))

g := (7×i) **mod** 16

dTemp := D

D := C

C := B

B := B + **leftrotate**((A + F + K[i] + M[g]), s[i])

A := dTemp

**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)*

//*leftrotate function definition*

**leftrotate** (x, c)

**return** (x << c) **binary or** (x >> (32-c));

*Note: Instead of the formulation from the original*[*RFC 1321*](https://tools.ietf.org/html/rfc1321)*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 parallelized):*

(0 ≤ i ≤ 15): F: = D **xor** (B **and** (C **xor** D))

(16 ≤ i ≤ 31): F: = C **xor** (D **and** (B **xor** C))

MD5 hashes [[edit](https://en.wikipedia.org/w/index.php?title=MD5&action=edit&section=9)]

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

MD5 ("[The quick brown fox jumps over the lazy dog](https://en.wikipedia.org/wiki/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](https://en.wikipedia.org/wiki/Avalanche_effect). For example, adding a period to the end of the sentence:

MD5 ("[The quick brown fox jumps over the lazy dog](https://en.wikipedia.org/wiki/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 bit ([octets](https://en.wikipedia.org/wiki/Octet_(computing)), [bytes](https://en.wikipedia.org/wiki/Byte)) as shown in the examples above. Some MD5 implementations such as [md5sum](https://en.wikipedia.org/wiki/Md5sum) might be limited to octets, or they might not support *streaming* for messages of an initially undetermined length