**MD5 ALGORITHM IN CRYPTOGRAPHY ALGORITHM IMPLEMENT IN C**

|  |
| --- |
| /\* |
|  | \* Simple MD5 implementation |
|  | \* |
|  | \* Compile with: gcc -o md5 -O3 -lm md5.c |
|  | \*/ |
|  | #include <stdio.h> |
|  | #include <stdlib.h> |
|  | #include <string.h> |
|  | #include <stdint.h> |
|  |  |
|  | // leftrotate function definition |
|  | #define LEFTROTATE(x, c) (((x) << (c)) | ((x) >> (32 - (c)))) |
|  |  |
|  | // These vars will contain the hash |
|  | uint32\_t h0, h1, h2, h3; |
|  |  |
|  | void md5(uint8\_t \*initial\_msg, size\_t initial\_len) { |
|  |  |
|  | // Message (to prepare) |
|  | uint8\_t \*msg = NULL; |
|  |  |
|  | // Note: All variables are unsigned 32 bit and wrap modulo 2^32 when calculating |
|  |  |
|  | // r specifies the per-round shift amounts |
|  |  |
|  | uint32\_t r[] = {7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, |
|  | 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20, |
|  | 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, |
|  | 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 (in radians) as constants// Initialize variables: |
|  | uint32\_t k[] = { |
|  | 0xd76aa478, 0xe8c7b756, 0x242070db, 0xc1bdceee, |
|  | 0xf57c0faf, 0x4787c62a, 0xa8304613, 0xfd469501, |
|  | 0x698098d8, 0x8b44f7af, 0xffff5bb1, 0x895cd7be, |
|  | 0x6b901122, 0xfd987193, 0xa679438e, 0x49b40821, |
|  | 0xf61e2562, 0xc040b340, 0x265e5a51, 0xe9b6c7aa, |
|  | 0xd62f105d, 0x02441453, 0xd8a1e681, 0xe7d3fbc8, |
|  | 0x21e1cde6, 0xc33707d6, 0xf4d50d87, 0x455a14ed, |
|  | 0xa9e3e905, 0xfcefa3f8, 0x676f02d9, 0x8d2a4c8a, |
|  | 0xfffa3942, 0x8771f681, 0x6d9d6122, 0xfde5380c, |
|  | 0xa4beea44, 0x4bdecfa9, 0xf6bb4b60, 0xbebfbc70, |
|  | 0x289b7ec6, 0xeaa127fa, 0xd4ef3085, 0x04881d05, |
|  | 0xd9d4d039, 0xe6db99e5, 0x1fa27cf8, 0xc4ac5665, |
|  | 0xf4292244, 0x432aff97, 0xab9423a7, 0xfc93a039, |
|  | 0x655b59c3, 0x8f0ccc92, 0xffeff47d, 0x85845dd1, |
|  | 0x6fa87e4f, 0xfe2ce6e0, 0xa3014314, 0x4e0811a1, |
|  | 0xf7537e82, 0xbd3af235, 0x2ad7d2bb, 0xeb86d391}; |
|  |  |
|  | h0 = 0x67452301; |
|  | h1 = 0xefcdab89; |
|  | h2 = 0x98badcfe; |
|  | h3 = 0x10325476; |
|  |  |
|  | // 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.[37] \*/ |
|  |  |
|  | // Pre-processing: padding with zeros |
|  | //append "0" bit until message length in bit ≡ 448 (mod 512) |
|  | //append length mod (2 pow 64) to message |
|  |  |
|  | int new\_len = ((((initial\_len + 8) / 64) + 1) \* 64) - 8; |
|  |  |
|  | msg = calloc(new\_len + 64, 1); // also appends "0" bits |
|  | // (we alloc also 64 extra bytes...) |
|  | memcpy(msg, initial\_msg, initial\_len); |
|  | msg[initial\_len] = 128; // write the "1" bit |
|  |  |
|  | uint32\_t bits\_len = 8\*initial\_len; // note, we append the len |
|  | memcpy(msg + new\_len, &bits\_len, 4); // in bits at the end of the buffer |
|  |  |
|  | // Process the message in successive 512-bit chunks: |
|  | //for each 512-bit chunk of message: |
|  | int offset; |
|  | for(offset=0; offset<new\_len; offset += (512/8)) { |
|  |  |
|  | // break chunk into sixteen 32-bit words w[j], 0 ≤ j ≤ 15 |
|  | uint32\_t \*w = (uint32\_t \*) (msg + offset); |
|  |  |
|  | #ifdef DEBUG |
|  | printf("offset: %d %x\n", offset, offset); |
|  |  |
|  | int j; |
|  | for(j =0; j < 64; j++) printf("%x ", ((uint8\_t \*) w)[j]); |
|  | puts(""); |
|  | #endif |
|  |  |
|  | // Initialize hash value for this chunk: |
|  | uint32\_t a = h0; |
|  | uint32\_t b = h1; |
|  | uint32\_t c = h2; |
|  | uint32\_t d = h3; |
|  |  |
|  | // Main loop: |
|  | uint32\_t i; |
|  | for(i = 0; i<64; i++) { |
|  |  |
|  | #ifdef ROUNDS |
|  | uint8\_t \*p; |
|  | printf("%i: ", i); |
|  | p=(uint8\_t \*)&a; |
|  | printf("%2.2x%2.2x%2.2x%2.2x ", p[0], p[1], p[2], p[3], a); |
|  |  |
|  | p=(uint8\_t \*)&b; |
|  | printf("%2.2x%2.2x%2.2x%2.2x ", p[0], p[1], p[2], p[3], b); |
|  |  |
|  | p=(uint8\_t \*)&c; |
|  | printf("%2.2x%2.2x%2.2x%2.2x ", p[0], p[1], p[2], p[3], c); |
|  |  |
|  | p=(uint8\_t \*)&d; |
|  | printf("%2.2x%2.2x%2.2x%2.2x", p[0], p[1], p[2], p[3], d); |
|  | puts(""); |
|  | #endif |
|  |  |
|  |  |
|  | uint32\_t f, g; |
|  |  |
|  | if (i < 16) { |
|  | f = (b & c) | ((~b) & d); |
|  | g = i; |
|  | } else if (i < 32) { |
|  | f = (d & b) | ((~d) & c); |
|  | g = (5\*i + 1) % 16; |
|  | } else if (i < 48) { |
|  | f = b ^ c ^ d; |
|  | g = (3\*i + 5) % 16; |
|  | } else { |
|  | f = c ^ (b | (~d)); |
|  | g = (7\*i) % 16; |
|  | } |
|  |  |
|  | #ifdef ROUNDS |
|  | printf("f=%x g=%d w[g]=%x\n", f, g, w[g]); |
|  | #endif |
|  | uint32\_t temp = d; |
|  | d = c; |
|  | c = b; |
|  | printf("rotateLeft(%x + %x + %x + %x, %d)\n", a, f, k[i], w[g], r[i]); |
|  | b = b + LEFTROTATE((a + f + k[i] + w[g]), r[i]); |
|  | a = temp; |
|  |  |
|  |  |
|  |  |
|  | } |
|  |  |
|  | // Add this chunk's hash to result so far: |
|  |  |
|  | h0 += a; |
|  | h1 += b; |
|  | h2 += c; |
|  | h3 += d; |
|  |  |
|  | } |
|  |  |
|  | // cleanup |
|  | free(msg); |
|  |  |
|  | } |
|  |  |
|  | int main(int argc, char \*\*argv) { |
|  |  |
|  | if (argc < 2) { |
|  | printf("usage: %s 'string'\n", argv[0]); |
|  | return 1; |
|  | } |
|  |  |
|  | char \*msg = argv[1]; |
|  | size\_t len = strlen(msg); |
|  |  |
|  | // benchmark |
|  | // int i; |
|  | // for (i = 0; i < 1000000; i++) { |
|  | md5(msg, len); |
|  | // } |
|  |  |
|  | //var char digest[16] := h0 append h1 append h2 append h3 //(Output is in little-endian) |
|  | uint8\_t \*p; |
|  |  |
|  | // display result |
|  |  |
|  | p=(uint8\_t \*)&h0; |
|  | printf("%2.2x%2.2x%2.2x%2.2x", p[0], p[1], p[2], p[3], h0); |
|  |  |
|  | p=(uint8\_t \*)&h1; |
|  | printf("%2.2x%2.2x%2.2x%2.2x", p[0], p[1], p[2], p[3], h1); |
|  |  |
|  | p=(uint8\_t \*)&h2; |
|  | printf("%2.2x%2.2x%2.2x%2.2x", p[0], p[1], p[2], p[3], h2); |
|  |  |
|  | p=(uint8\_t \*)&h3; |
|  | printf("%2.2x%2.2x%2.2x%2.2x", p[0], p[1], p[2], p[3], h3); |
|  | puts(""); |
|  |  |
|  | return 0; |
|  | } |

