Assignment 2 - Hash Functions

Section 1: Goals of the Experiment

The goals of this experiment are to learn how hash functions operate, specifically MD5 and SHA-1, and implement the algorithm utilizing an Arduino board.

Section 2: Experimental Setup

The physical setup consisted of an Arduino Board MKR WIFI 1010 plugged into a laptop via USB type A to USB type B.

To implement the hash function, we first did extensive research to understand exactly how it works. Generally, hash algorithms can be broken down into the following steps (Metamorphosite, 2007):

1. Initialize variables h0-h4.
    a. These variables play an important role for the functionality of the block ciphers and are used later.
2. Take in input as a string
3. Break the string into separate characters
4. Convert the separate characters to their ASCII values and then convert it again to binary.
    a. If the binary values are less than 8 bits long, pad the spaces with zeros. For example, A = 65 = 1000001 is padded with an extra zero at the beginning (01000001) so that it is 8 bits long.
5. Join these characters as binary back together and add a ‘1’ to the end.
6. Append zeros to the end until the message length is equal to 448 mod 512.
7. Take the length of the original string and convert it into binary. The length of the message now should be a multiple of 512.
    a. If the length in binary is less than 64 bits, append zeros to the front of the number until it is 64 bits long.
8. Break down the message into 512 bit chunks.
9. Break down the chunks from step 7 further into 32 bit blocks called ‘words’.
10. Make more “words” from the original chunks we got in step 9 using bitwise operations (XORs).
11. Left shift each ‘word’ by one byte.
12. Make variables A-E and assign them to h0-h4 respectively.
13. Depending on the “word” number, each will go through a separate function. Words 0-19 go to function 1, 20-39 to function 2, 40-59 to function 3 and 60-79 to function 4.
    a. These functions each include an ‘F’ variable that compares some of the A-E variables to output another string of bits that is added to the “word”. Some of these variables are also added to the ‘word’ to get a completely different output that was put in.
    b. The process of these functions are relatively the same but are different in the variables that they use. In the end, the new ‘word’ is set as the new ‘A’ variable, which means each iteration uses the result of the function before it.

14. After step 13, the variables A-E are completely different than the values that they were initially set as in step 12. H0-h4 are assigned to h0-h4 added to A-E respectively.
15. A-E, which are in bit form, are converted into Hexadecimal characters and concatenated together to form the Hash.

Since this process is extensive, we programmed the hash function in several steps. In the code, there are several print statements for debugging. We used the strings “Hello World” and “A Test” as a basis for when we were looking for errors in the code.

MD5 vs. SHA-1

Though the process for programming both hash algorithms were similar, the main difference was in the number of the “words” and what main programmers call the “main loop”. MD5 creates 64 “words” whereas SHA-1 will produce 80 “words”. MD5’s main loop uses different variables and functions compared to SHA-1 as seen in the pseudocode below.

```
for i from 0 to 63
    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]
    A := D
    D := C
    C := B
    B := B + leftrotate(F, s[i])
end for
```

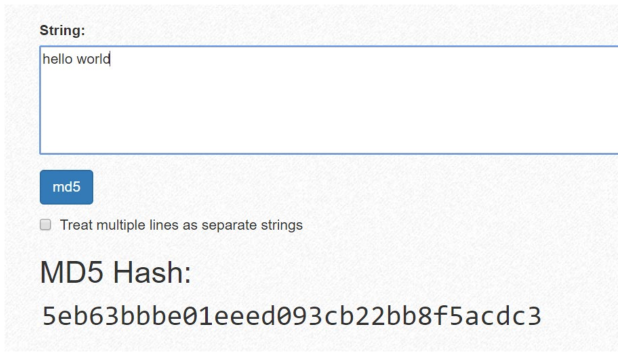
```
for i from 0 to 79
    if 0 ≤ i ≤ 19 then
        f = (b and c) or ((not b) and d)
        k = 0x5A827999
    else if 20 ≤ i ≤ 39
        f = b xor c xor d
        k = 0x6ED9EBA1
    else if 40 ≤ i ≤ 59
        f = (b and c) or (b and d) or (c and d)
        k = 0x8F1BBCDC
    else if 60 ≤ i ≤ 79
        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
```

***Images of psuedocode from Wikipedia (Wikipedia Contributors, 2019a&b)

Section 3: Results and Conclusion

Below are screen captures of the hashes our programs calculated compared to the online hash generators. The experiment is deemed successful our hashed outputs are the same as the generators’.

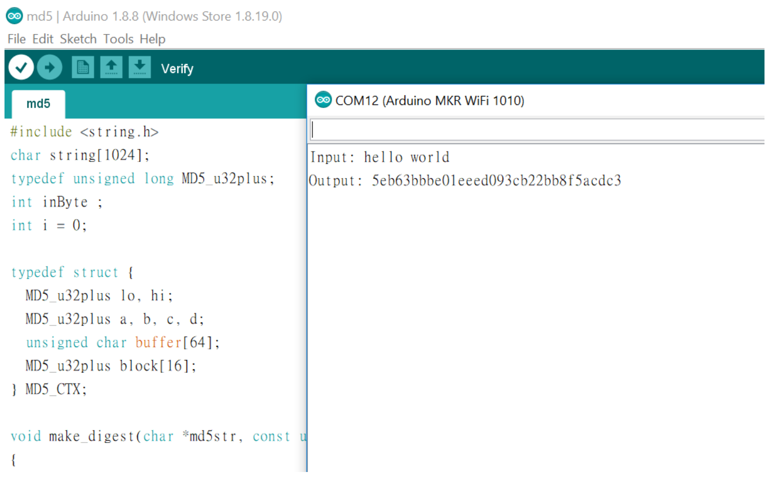
MD5:



```
hello world
```

MD5 Hash:

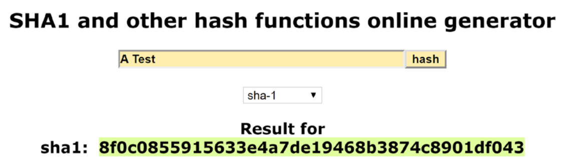
```
5eb63bbbe01eeed093cb22bb8f5acdc3
```


md5 | Arduino 1.8.8 (Windows Store 1.8.19.0)
File Edit Sketch Tools Help
Verify
md5
COM12 (Arduino MKR WiFi 1010)

```
#include <string.h>
char string[1024];
typedef unsigned long MD5_u32plus;
int inByte ;
int i = 0;

typedef struct {
  MD5_u32plus lo, hi;
  MD5_u32plus a, b, c, d;
  unsigned char buffer[64];
  MD5_u32plus block[16];
} MD5_CTX;

void make_digest(char *md5str, const u
{
```

```
Input: hello world
Output: 5eb63bbbe01eeed093cb22bb8f5acdc3
```

SHA-1:

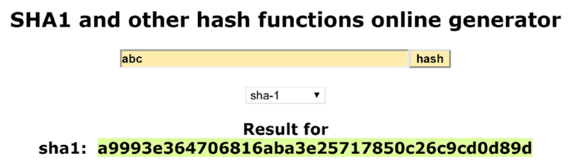


SHA1 and other hash functions online generator

A Test | hash

sha-1

Result for
sha1: 8f0c0855915633e4a7de19468b3874c8901df043

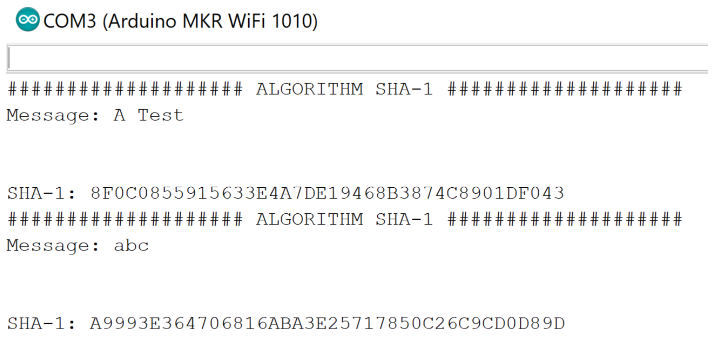


SHA1 and other hash functions online generator

abc | hash

sha-1

Result for
sha1: a9993e364706816aba3e25717850c26c9cd0d89d



COM3 (Arduino MKR WiFi 1010)

```
#################### ALGORITHM SHA-1 ####################
Message: A Test


SHA-1: 8F0C0855915633E4A7DE19468B3874C8901DF043
#################### ALGORITHM SHA-1 ####################
Message: abc


SHA-1: A9993E364706816ABA3E25717850C26C9CD0D89D
```

Section 4:Given Questions

1.Of the two mentioned hash function, which one should be used for Security Application? Provide explanation.

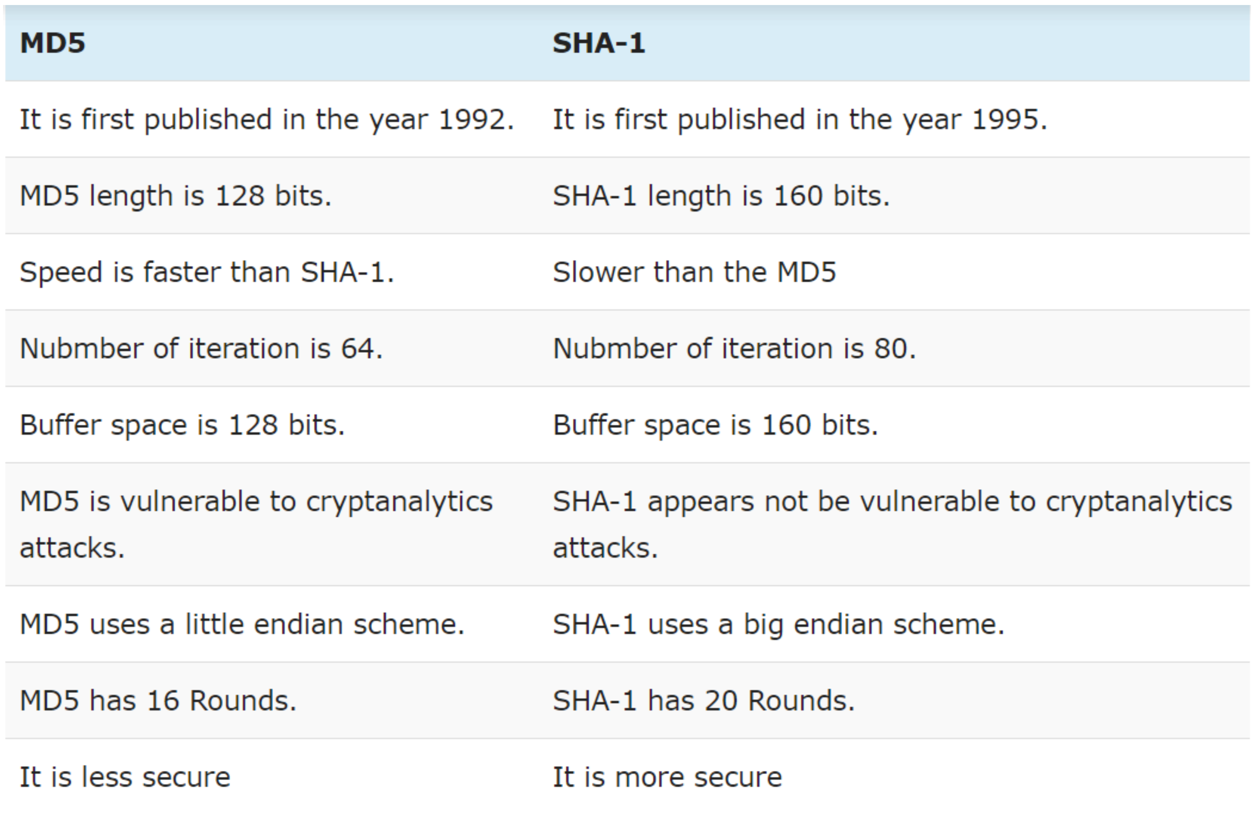
The buffer space of SHA-1 is 32 bits more than MD5's. If we want to find a message for a digest value, MD5 needs to execute 2^{128} operations,while SHA-1 needs to execute 2^{160} operation. It's easier to reverse the encryption of MD5 by running a brute force attack.In addition, MD5 has been found to be insecure in structure. It seems that SHA-1 is much more secure than MD5 and it should be used for Security Application.

2.Please explain in brief what makes hash functions resistant to attacks. Provide an exemplary brief case study.

Simply define a hash function like:
Input 123456 -----> Output 1+2+3+4+5+6 = 21
Input 993 -----> Output 9+9+3 = 21
The function is irreversible because you can't get the original text by just knowing the definition of the hush function and the output (unless running a brute force attack). The design of a hash function is to make an irreversible function and be collision resistant to protect the text.

Since every hash function with more inputs than outputs, generally it's hard to find two inputs that hash to the same output especially with a complex calculation. If a website wants to record the user's account numbers and passwords, it's dangerous to save the text directly because hackers can access all the passwords by entering the database. Also, if we put the password to a hash function first, the hacker can't get the original password.The principle is to just hash the password provided by the user and authenticate it by comparing it with what's stored in the database.

3.Provide a comparison between MD5 and SHA-1. Overall, which one do you think performs better than the other one?



MD5	SHA-1
It is first published in the year 1992.	It is first published in the year 1995.
MD5 length is 128 bits.	SHA-1 length is 160 bits.
Speed is faster than SHA-1.	Slower than the MD5
Nubmber of iteration is 64.	Nubmber of iteration is 80.
Buffer space is 128 bits.	Buffer space is 160 bits.
MD5 is vulnerable to cryptanalytics attacks.	SHA-1 appears not be vulnerable to cryptanalytics attacks.
MD5 uses a little endian scheme.	SHA-1 uses a big endian scheme.
MD5 has 16 Rounds.	SHA-1 has 20 Rounds.
It is less secure	It is more secure

MD5 and SHA-1 hashing algorithms provide one way encryption. They ideally store passwords as generally we won’t need to see the unencrypted version. MD5 uses a hash length of 16 bytes and SHA1 uses 20 bytes. This means that SHA1 is slower but more secure.

Section 5: References

Metamorphosite. (2007, Nov. 12). *How Hash Algorithms Work* [Blog post]. Retrieved from http://www.metamorphosite.com/one-way-hash-encryption-sha1-data-software

The Ex. (2017, Jan 9). *Algoritma SHA-1 dan Implementasi pada Mikrokontroller Arduino* [YouTube video]. Retrieved from https://www.youtube.com/watch?v=37BqArsJWGE

Wikipedia contributors. (2019, March 7). SHA-1. In Wikipedia, The Free Encyclopedia. Retrieved from https://en.wikipedia.org/w/index.php?title=SHA-1&oldid=886669102

Wikipedia contributors. (2019, February 22). MD5. In Wikipedia, The Free Encyclopedia. Retrieved from https://en.wikipedia.org/w/index.php?title=MD5&oldid=884548944

Section 6: Appendix

MD5 Code

```
#include <string.h>
char string[1024];
typedef unsigned long MD5_u32plus;
int inByte ;
int i = 0;

typedef struct {
  MD5_u32plus lo, hi;
  MD5_u32plus a, b, c, d;
  unsigned char buffer[64];
  MD5_u32plus block[16];
} MD5_CTX;

void make_digest(char *md5str, const unsigned char *digest, int len) /* {{{ */
{
  static const char hexits[17] = "0123456789abcdef";
  int i;

  for (i = 0; i < len; i++) {
    md5str[i * 2]       = hexits[digest[i] >> 4];
    md5str[(i * 2) + 1] = hexits[digest[i] &  0x0F];
  }
  md5str[len * 2] = '\0';
}

/*
 * The basic MD5 functions.
 *
 * F and G are optimized compared to their RFC 1321 definitions for
 * architectures that lack an AND-NOT instruction, just like in Colin Plumb's
 * implementation.
 */
#define F(x, y, z)      ((z) ^ ((x) & ((y) ^ (z))))
#define G(x, y, z)      ((y) ^ ((z) & ((x) ^ (y))))
#define H(x, y, z)      ((x) ^ (y) ^ (z))
#define I(x, y, z)      ((y) ^ ((x) | ~(z)))

/*
 * The MD5 transformation for all four rounds.
 */
#define STEP(f, a, b, c, d, x, t, s) \
  (a) += f((b), (c), (d)) + (x) + (t); \
  (a) = (((a) << (s)) | (((a) & 0xffffffff) >> (32 - (s)))); \
  (a) += (b);

/*
 * SET reads 4 input bytes in little-endian byte order and stores them
 * in a properly aligned word in host byte order.
 *
 * The check for little-endian architectures that tolerate unaligned
 * memory accesses is just an optimization.  Nothing will break if it
 * doesn't work.
 */
#if defined(__i386__) || defined(__x86_64__) || defined(__vax__)
# define SET(n) \
  (*(MD5_u32plus *)&ptr[(n) * 4])
# define GET(n) \
  SET(n)
#else
# define SET(n) \
  (ctx->block[(n)] = \
  (MD5_u32plus)ptr[(n) * 4] | \
  ((MD5_u32plus)ptr[(n) * 4 + 1] << 8) | \
  ((MD5_u32plus)ptr[(n) * 4 + 2] << 16) | \
  ((MD5_u32plus)ptr[(n) * 4 + 3] << 24))
# define GET(n) \
  (ctx->block[(n)])
#endif

/*
 * This processes one or more 64-byte data blocks, but does NOT update
 * the bit counters.  There are no alignment requirements.
 */
static const void *body(void *ctxBuf, const void *data, size_t size)
{
  MD5_CTX *ctx = (MD5_CTX*)ctxBuf;
  const unsigned char *ptr;
  MD5_u32plus a, b, c, d;
  MD5_u32plus saved_a, saved_b, saved_c, saved_d;

  ptr = (unsigned char*)data;

  a = ctx->a;
  b = ctx->b;
  c = ctx->c;
  d = ctx->d;
  do {
    saved_a = a;
    saved_b = b;
    saved_c = c;
    saved_d = d;

/* Round 1 */
    STEP(F, a, b, c, d, SET(0), 0xd76aa478, 7)
    STEP(F, d, a, b, c, SET(1), 0xe8c7b756, 12)
    STEP(F, c, d, a, b, SET(2), 0x242070db, 17)
    STEP(F, b, c, d, a, SET(3), 0xc1bdceee, 22)
    STEP(F, a, b, c, d, SET(4), 0xf57c0faf, 7)
    STEP(F, d, a, b, c, SET(5), 0x4787c62a, 12)
    STEP(F, c, d, a, b, SET(6), 0xa8304613, 17)
    STEP(F, b, c, d, a, SET(7), 0xfd469501, 22)
    STEP(F, a, b, c, d, SET(8), 0x698098d8, 7)
    STEP(F, d, a, b, c, SET(9), 0x8b44f7af, 12)
    STEP(F, c, d, a, b, SET(10), 0xffff5bb1, 17)
    STEP(F, b, c, d, a, SET(11), 0x895cd7be, 22)
    STEP(F, a, b, c, d, SET(12), 0x6b901122, 7)
    STEP(F, d, a, b, c, SET(13), 0xfd987193, 12)
    STEP(F, c, d, a, b, SET(14), 0xa679438e, 17)
    STEP(F, b, c, d, a, SET(15), 0x49b40821, 22)

/* Round 2 */
    STEP(G, a, b, c, d, GET(1), 0xf61e2562, 5)
    STEP(G, d, a, b, c, GET(6), 0xc040b340, 9)
    STEP(G, c, d, a, b, GET(11), 0x265e5a51, 14)
    STEP(G, b, c, d, a, GET(0), 0xe9b6c7aa, 20)
    STEP(G, a, b, c, d, GET(5), 0xd62f105d, 5)
    STEP(G, d, a, b, c, GET(10), 0x02441453, 9)
    STEP(G, c, d, a, b, GET(15), 0xd8a1e681, 14)
    STEP(G, b, c, d, a, GET(4), 0xe7d3fbc8, 20)
    STEP(G, a, b, c, d, GET(9), 0x21e1cde6, 5)
    STEP(G, d, a, b, c, GET(14), 0xc33707d6, 9)
    STEP(G, c, d, a, b, GET(3), 0xf4d50d87, 14)
    STEP(G, b, c, d, a, GET(8), 0x455a14ed, 20)
    STEP(G, a, b, c, d, GET(13), 0xa9e3e905, 5)
    STEP(G, d, a, b, c, GET(2), 0xfcefa3f8, 9)
    STEP(G, c, d, a, b, GET(7), 0x676f02d9, 14)
    STEP(G, b, c, d, a, GET(12), 0x8d2a4c8a, 20)

/* Round 3 */
```

```
    STEP(H, a, b, c, d, GET(5), 0xfffa3942, 4)
    STEP(H, d, a, b, c, GET(8), 0x8771f681, 11)
    STEP(H, c, d, a, b, GET(11), 0x6d9d6122, 16)
    STEP(H, b, c, d, a, GET(14), 0xfde5380c, 23)
    STEP(H, a, b, c, d, GET(1), 0xa4beea44, 4)
    STEP(H, d, a, b, c, GET(4), 0x4bdecfa9, 11)
    STEP(H, c, d, a, b, GET(7), 0xf6bb4b60, 16)
    STEP(H, b, c, d, a, GET(10), 0xbebfbc70, 23)
    STEP(H, a, b, c, d, GET(13), 0x289b7ec6, 4)
    STEP(H, d, a, b, c, GET(0), 0xeaa127fa, 11)
    STEP(H, c, d, a, b, GET(3), 0xd4ef3085, 16)
    STEP(H, b, c, d, a, GET(6), 0x04881d05, 23)
    STEP(H, a, b, c, d, GET(9), 0xd9d4d039, 4)
    STEP(H, d, a, b, c, GET(12), 0xe6db99e5, 11)
    STEP(H, c, d, a, b, GET(15), 0x1fa27cf8, 16)
    STEP(H, b, c, d, a, GET(2), 0xc4ac5665, 23)

/* Round 4 */
    STEP(I, a, b, c, d, GET(0), 0xf4292244, 6)
    STEP(I, d, a, b, c, GET(7), 0x432aff97, 10)
    STEP(I, c, d, a, b, GET(14), 0xab9423a7, 15)
    STEP(I, b, c, d, a, GET(5), 0xfc93a039, 21)
    STEP(I, a, b, c, d, GET(12), 0x655b59c3, 6)
    STEP(I, d, a, b, c, GET(3), 0x8f0ccc92, 10)
    STEP(I, c, d, a, b, GET(10), 0xffeff47d, 15)
    STEP(I, b, c, d, a, GET(1), 0x85845dd1, 21)
    STEP(I, a, b, c, d, GET(8), 0x6fa87e4f, 6)
    STEP(I, d, a, b, c, GET(15), 0xfe2ce6e0, 10)
    STEP(I, c, d, a, b, GET(6), 0xa3014314, 15)
    STEP(I, b, c, d, a, GET(13), 0x4e0811a1, 21)
    STEP(I, a, b, c, d, GET(4), 0xf7537e82, 6)
    STEP(I, d, a, b, c, GET(11), 0xbd3af235, 10)
    STEP(I, c, d, a, b, GET(2), 0x2ad7d2bb, 15)
    STEP(I, b, c, d, a, GET(9), 0xeb86d391, 21)

    a += saved_a;
    b += saved_b;
    c += saved_c;
    d += saved_d;

    ptr += 64;
  } while (size -= 64);

  ctx->a = a;
  ctx->b = b;
  ctx->c = c;
  ctx->d = d;

  return ptr;
}

void MD5Init(void *ctxBuf)
{
  MD5_CTX *ctx = (MD5_CTX*)ctxBuf;
  ctx->a = 0x67452301;
  ctx->b = 0xefcdab89;
  ctx->c = 0x98badcfe;
  ctx->d = 0x10325476;

  ctx->lo = 0;
  ctx->hi = 0;
}

void MD5Update(void *ctxBuf, const void *data, size_t size)
{
  MD5_CTX *ctx = (MD5_CTX*)ctxBuf;
  MD5_u32plus saved_lo;
  MD5_u32plus used, free;

  saved_lo = ctx->lo;
  if ((ctx->lo = (saved_lo + size) & 0x1fffffff) < saved_lo) {
    ctx->hi++;
  }
  ctx->hi += size >> 29;

  used = saved_lo & 0x3f;

  if (used) {
    free = 64 - used;

    if (size < free) {
      memcpy(&ctx->buffer[used], data, size);
      return;
    }

    memcpy(&ctx->buffer[used], data, free);
    data = (unsigned char *)data + free;
    size -= free;
    body(ctx, ctx->buffer, 64);
  }

  if (size >= 64) {
    data = body(ctx, data, size & ~(size_t)0x3f);
    size &= 0x3f;
  }

  memcpy(ctx->buffer, data, size);
}

void MD5Final(unsigned char *result, void *ctxBuf)
{
  MD5_CTX *ctx = (MD5_CTX*)ctxBuf;
  MD5_u32plus used, free;

  used = ctx->lo & 0x3f;

  ctx->buffer[used++] = 0x80;

  free = 64 - used;

  if (free < 8) {
    memset(&ctx->buffer[used], 0, free);
    body(ctx, ctx->buffer, 64);
    used = 0;
    free = 64;
  }

  memset(&ctx->buffer[used], 0, free - 8);

  ctx->lo <<= 3;
  ctx->buffer[56] = ctx->lo;
  ctx->buffer[57] = ctx->lo >> 8;
  ctx->buffer[58] = ctx->lo >> 16;
  ctx->buffer[59] = ctx->lo >> 24;
  ctx->buffer[60] = ctx->hi;
  ctx->buffer[61] = ctx->hi >> 8;
  ctx->buffer[62] = ctx->hi >> 16;
  ctx->buffer[63] = ctx->hi >> 24;

  body(ctx, ctx->buffer, 64);

  result[0] = ctx->a;
  result[1] = ctx->a >> 8;
  result[2] = ctx->a >> 16;
  result[3] = ctx->a >> 24;
  result[4] = ctx->b;
  result[5] = ctx->b >> 8;
  result[6] = ctx->b >> 16;
  result[7] = ctx->b >> 24;
  result[8] = ctx->c;
  result[9] = ctx->c >> 8;
  result[10] = ctx->c >> 16;
  result[11] = ctx->c >> 24;
  result[12] = ctx->d;
  result[13] = ctx->d >> 8;
  result[14] = ctx->d >> 16;
  result[15] = ctx->d >> 24;

  memset(ctx, 0, sizeof(*ctx));
}

void do_md5(char *arg)
{
  char md5str[33];
  MD5_CTX context;
  unsigned char digest[16];

  md5str[0] = '\0';
  MD5Init(&context);
  MD5Update(&context, arg, strlen(arg));
  MD5Final(digest, &context);

  make_digest(md5str, digest, 16);
  Serial.println(md5str);
}

void setup() {
  Serial.begin(9600);
}

void loop() {
  i=0;
  inByte = 0;

  Serial.print("Input: ");
  while(Serial.available()>0)
  {
    inByte = Serial.read();
    string[i] = inByte;
    //Serial.print(string[i]);
    i++  ;
  }
   Serial.println(string);
   Serial.print("Output: ");
   do_md5(string);
   while(Serial.available()<=0)
   {

   }
   memset(string, 0, sizeof(string));
   //delay(10000);
}
```

References:
http://www.miraclesalad.com/webtools/md5.php
https://github.com/scottmac/arduino/blob/master/md5/md5.pde

SHA-1 Code

```
#include <string.h>

String msg;
uint32_t wrds[80], h[5], A, B, C, D, E;
String bin = "";
int inByte = 0;
char string[1024];
int i;

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
}//setup

//STEP 0
void resetSHA1() {
  h[0] = 0x67452301;
  h[1] = 0xEFCDAB89;
  h[2] = 0x98BADCFE;
  h[3] = 0x10325476;
  h[4] = 0xC3D2E1F0;
  bin = "";
  memset(string, 0, sizeof(string));
}//resetSHA1

String toBin(int data) {
  String bin = String(data, BIN);

  while(bin.length() < 8) {
    bin = '0' + bin;
  }
  return bin;
}//toBin

uint32_t rank (int base, int rank) {
  uint32_t result = 1;
  for(int i = 0; i < rank; ++i) {
    result *= 2;
  }
  return result;
}//rank

uint32_t bit2Int(String data) {
  uint32_t result = 0, temp = 0;
  int pos;
  int dtSize = data.length();
  for(int i = 0; i < dtSize; i++) {
    pos = dtSize - i;
    if(data[pos] == '1'){
      temp = rank(2, i-1);
      result += temp;
    }
  }
  return result;
}//bit2Int

static uint32_t rol(const uint32_t value, const size_t bits) {
  return (value << bits) | (value >> (32 - bits));
}//rol

uint32_t func1(uint32_t words) {
  uint32_t F, result;
  F = (~B & D) | (B & C);
  result = rol(A, 5) + 0x5A827999 + F + E + words;
  E = D;
  D = C;
  C = rol(B, 30);
  B = A;
  A = result;
  return result;
}//func1

uint32_t func2(uint32_t words) {
  uint32_t F, result;
  F = (B ^ C) ^ D;
  result = rol(A, 5) + F + E + 0x6ED9EBA1 + words;
  E = D;
  D = C;
  C = rol(B, 30);
  B = A;
  A = result;
  return result;
}//func2

uint32_t func3(uint32_t words) {
  uint32_t F, result;
  F = ((B & C) | (B & D)) | (C & D);
  result = rol(A, 5) + F + E + 0x8F1BBCDC + words;
  E = D;
  D = C;
  C = rol(B, 30);
  B = A;
  A = result;
  return result;
}//func3

uint32_t func4(uint32_t words) {
  uint32_t F, result;
  F = (B ^ C) ^ D;
  result = rol(A, 5) + F + E + 0xCA62C1D6 + words;
  E = D;
  D = C;
  C = rol(B, 30);
  B = A;
  A = result;
  return result;
}//func4

void loop() {
```

```
// put your main code here, to run repeatedly:
resetSHA1();
Serial.println("#################### ALGORITHM SHA-1 ####################");
Serial.print("Message: ");
while(Serial.available() > 0)
{
  inByte = Serial.read();
  if((inByte >= 'A' && inByte <= 125) | inByte == ' ')
  {
    msg += char(inByte);
  }
}
//while(!Serial.available());
//msg = Serial.readString();
Serial.println(msg);
//Serial.println(msg.length());


//Change ASCII of character to binary, then concat to 1 string
for (int var = 0; var < (int)msg.length(); ++var) {
  //Serial.println(msg[var], BIN);
  bin.concat(toBin((int)msg[var]));
}//for loop

//add 1 bit at last
bin = bin + "1";


//add zeros until 448 bit
int dtlen = bin.length();
for (int var = 0; var <448 - dtlen; ++var) {
  bin.concat('0');
}//for loop

//add 64 bit for size
int msglen = msg.length() * 8;
//Serial.println(msglen);
String binSize = toBin(msglen);
int binSizeln = binSize.length();
for (int var = 0; var < 64 - binSizeln; ++ var) {
  binSize = "0" + binSize;
}//for loop

//bind 448bit and 64bit be 1 chunk
```

```
bin.concat(binSize);

//print string of "bin" variable, prints the entire 512 bit string, used mostly for debugging
/*for (int i = 0; i < bin.length(); i++){
  if (i % 75 == 0 && i > 1) {
    Serial.println();
  }//if
  Serial.print(bin[i]);
}//for
Serial.println("\n");*/

//show block 0-15, and convert binary string to int 32
//breaks chunks into 16 32 bit words
int idx = 0;
String kata = "";
for(int var = 0; var < 512; ++var) {
  if(var % 32 == 0) {
    idx = var/32;
    if(kata != "") {
      wrds[idx-1] = bit2Int(kata);
    }//if
    //printing for debugging - prints words 1-15
    /*Serial.println();
    Serial.print("Words ");
    Serial.print(idx);
    Serial.print(": ");*/
    kata = "";
  }//if
  kata += bin[var];
  //Serial.print(bin[var]);
}//for
Serial.println();

wrds[15] = bit2Int(kata);

//STEP 9 create all words until 80 words
for(int i = 16; i < 80; i++){
  wrds[i] = ((wrds[i - 3] ^ wrds[i - 8]) ^ wrds[i - 14]) ^ wrds[i - 16];
  wrds[i] = rol(wrds[i], 1);

  //print for debugging - prints the word segments after XORs
  /*Serial.print("Words ");
  Serial.print(i);
  Serial.print(": ");
  Serial.println(wrds[i], BIN);*/
}//for
//Serial.println();

//STEP 10
A = h[0];
B = h[1];
C = h[2];
D = h[3];
E = h[4];

for (int i = 0; i < 80; i++) {
  if(i < 20)
    wrds[i] = func1(wrds[i]);
  else if (i < 40)
    wrds[i] = func2(wrds[i]);
  else if (i < 60)
    wrds[i] = func3(wrds[i]);
  else
    wrds[i] = func4(wrds[i]);

  /*Serial.print("Step ");
  Serial.println(i);
  Serial.print("A: ");
  Serial.println(A, BIN);
  Serial.print("B: ");
  Serial.println(B, BIN);
  Serial.print("C: ");
  Serial.println(C, BIN);
  Serial.print("D: ");
  Serial.println(D, BIN);
  Serial.print("E: ");
  Serial.println(E, BIN);*/
}//for
//Serial.println();
h[0] = h[0] + A;
h[1] = h[1] + B;
h[2] = h[2] + C;
h[3] = h[3] + D;
h[4] = h[4] + E;


  //printing for debugging - prints final segments in Binary and Hex
 /*for(int i = 0; i < 5; i++) {
    Serial.print("h");
    Serial.print(i);
    Serial.print("=");
    Serial.println(h[i], BIN);
    Serial.print("h");
    Serial.print(i);
    Serial.print("=");
    Serial.println(h[i], HEX);
  }*/

  Serial.println();
  Serial.print("SHA-1: ");
  Serial.print(h[0], HEX);
  Serial.print(h[1], HEX);
  Serial.print(h[2], HEX);
  Serial.print(h[3], HEX);
  Serial.print(h[4], HEX);
  Serial.println();

  while(Serial.available() <= 0)
  {

  }
  msg = "";
}//loop
```