Basic Principles in Networking
Assignment 4 - Integrity
Pair 29:
Nguyen Xuan Binh 887799
Nhut Cao 906939

Section 1: Goals of the experiment

The goal of the experiments below will be to carry out the hashing functions MD5 and SHA1 on Arduino MKR WIFI 1010 with the help of open source libraries. Since the internet gained global presence, information needs to be verified all the time and hash functions serve as a one way function to scramble information into unrecognizable text. Hashing helps protect data integrity over the internet information exchanging.

Section 2: Experimental Setup (Details of the experimental setup step by step)

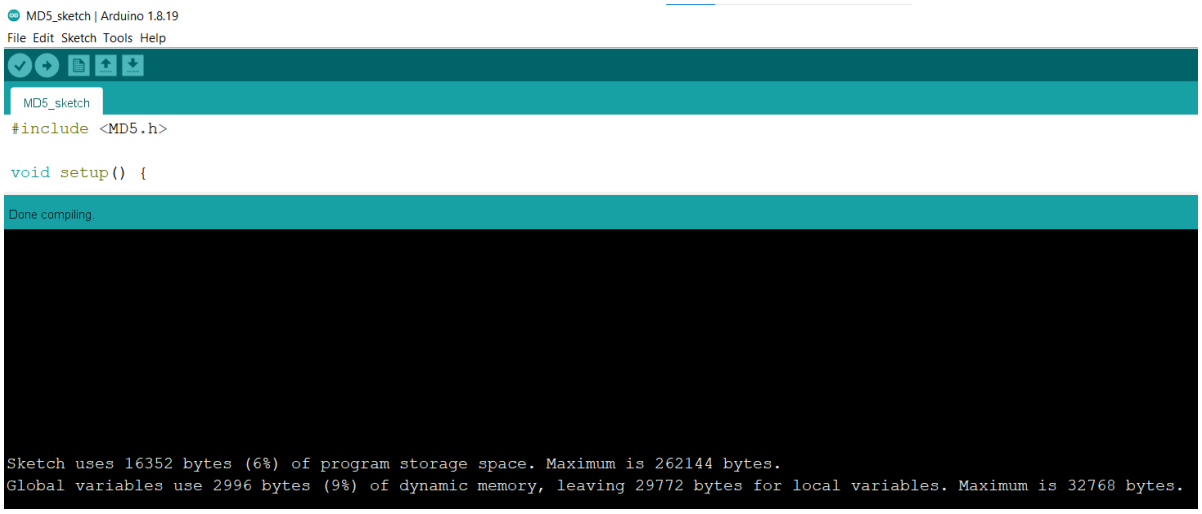
Experiment setup has four parts:

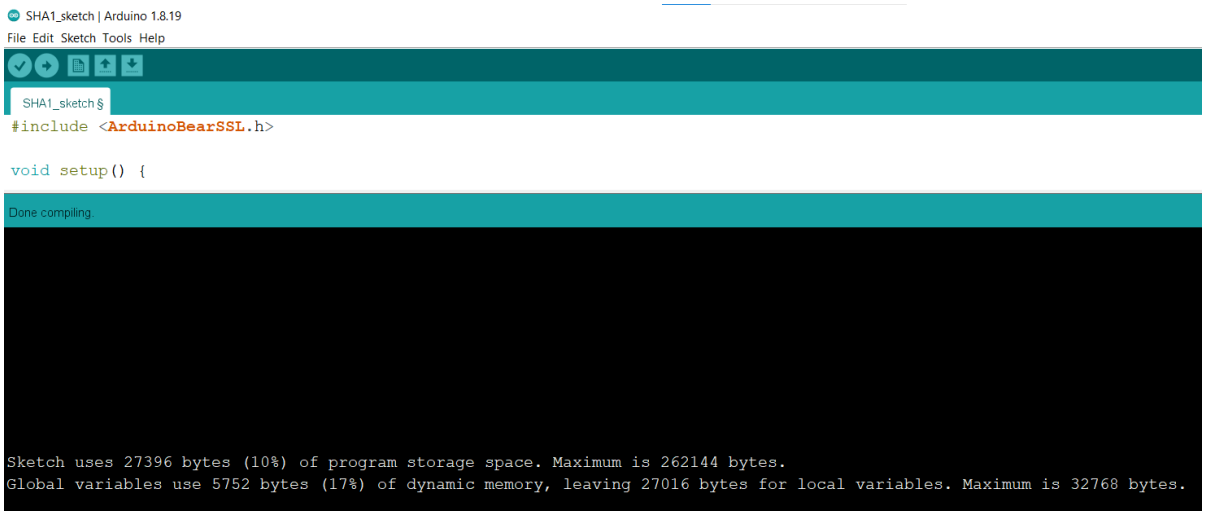
- Import the hashing library by using #include <library-hash-name>
- setup(), which runs only once when the program starts. This set up the Serial Monitor on the Arduino circuit.
- loop(), which runs indefinitely as it waits for input messages to be hashed with MD5 and SHA1. When the message is sent, the Serial Monitor returns the hashed message.
- establishContact(): when the program successfully starts, this will be run once and a message is sent to the Serial Monitor for program usage instructions.

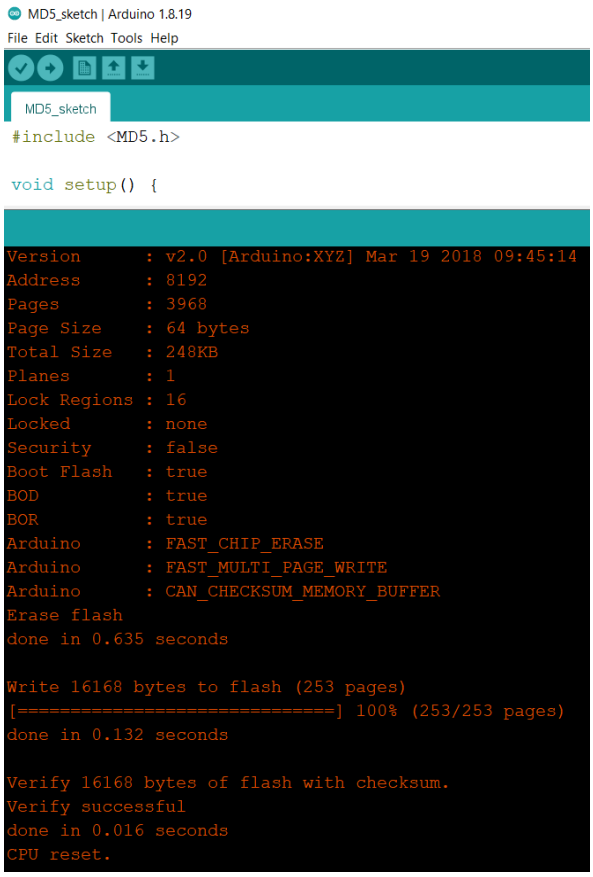
I used two open-source libraries to carry out the hashes:

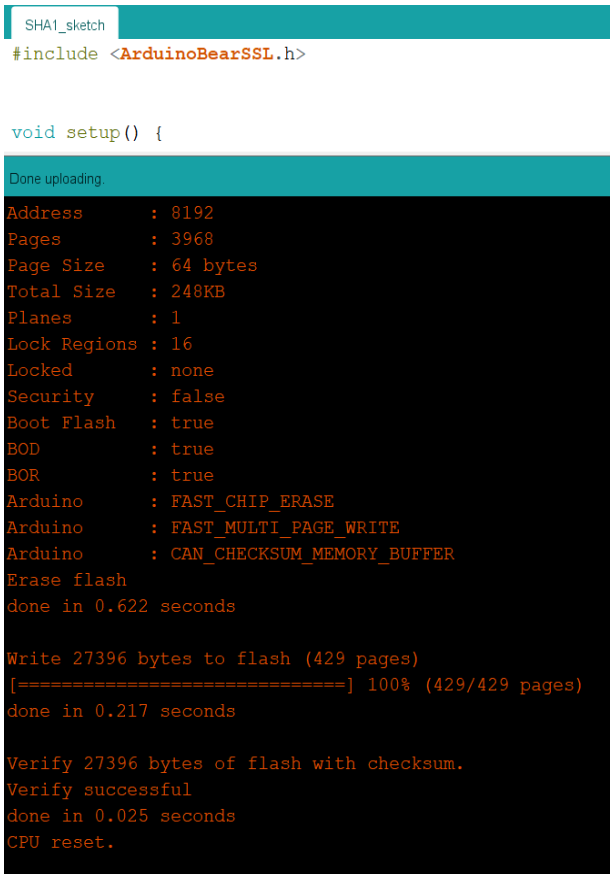
- For MD5, the library is from https://github.com/tzikis/ArduinoMD5
  After that, Sketch > Include Library > Add .Zip Library and choose the downloaded zip.
  Refresh the application, then Sketch > Include Library > (Contributed Libraries) MD5
- For SHA1, the library is downloaded directly from Arduino library manager:
  Tools > Manage Libraries. The library name is ArduinoBearSSL. Source code at https://github.com/arduino-libraries/ArduinoBearSSL

After setting up everything, we verify and upload the sketches of MD5 and SHA1







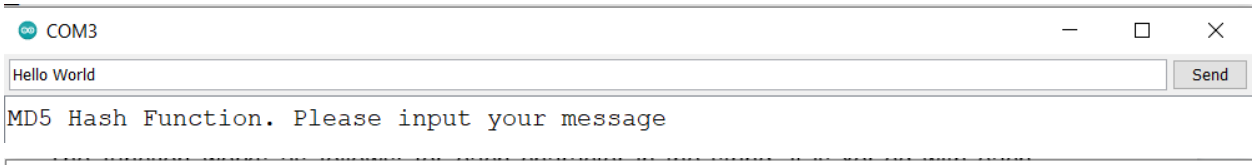


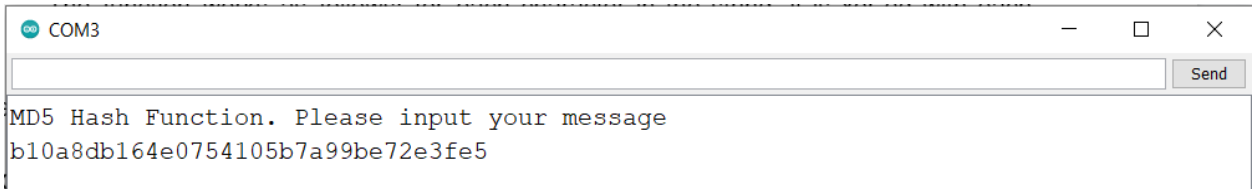
=> Sketch verification and upload are successful for both MD5 and SHA sketches

Section 3: Results & Conclusion

MD5 Hash Results

- Hash the message “Hello World”



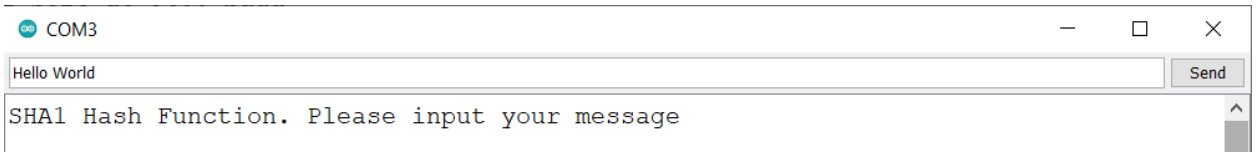


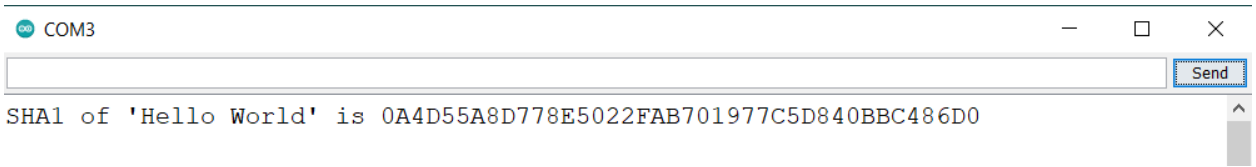
- Hash the message “Computer Networks is fun”

- Hash the message “The MD5 hashing algorithm is already cryptographically broken”

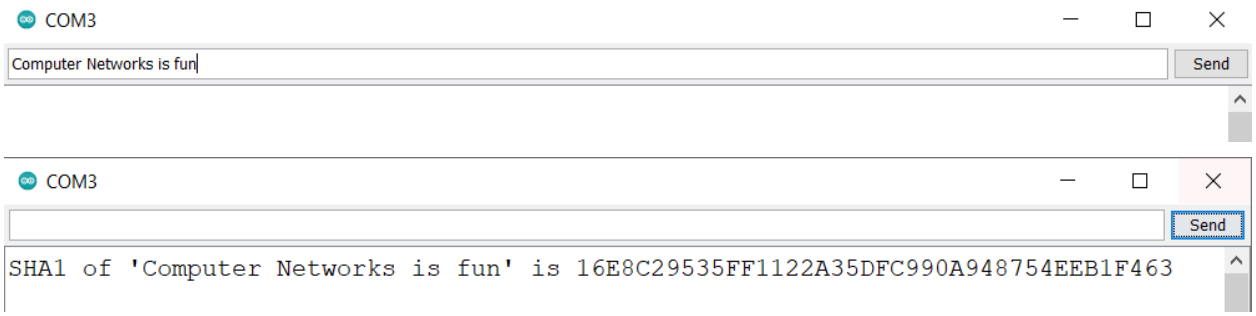
SHA1 Hash Results

- Hash the message “Hello World”

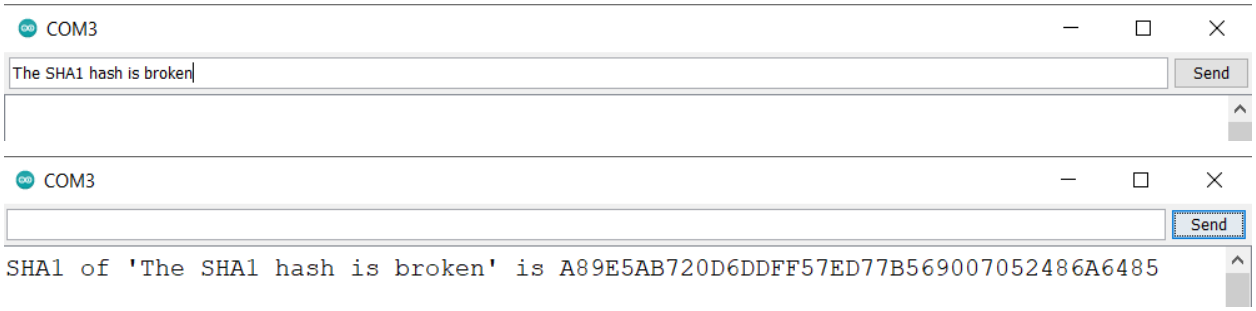




- Hash the message “Computer Networks is fun”



- Hash the message “The SHA1 hash is broken”



=> Conclusion: Hashing functions MD5 and SHA1 can be used to scramble messages beyond recognition.

Section 4: Answer of the given questions

- **Question 1: Of the two mentioned hash function, would you use one for Security Application? Why? If not, provide an alternative.**
  - Currently, both of these hash functions are considered insecure. For Message Digest Algorithm 5 (MD5), it has been cryptographically broken and considered unsafe. Therefore, MD5 should not be used for Security Application.
  - Since 2005, Secure Hash Algorithm 1 (SHA-1) no longer remains secure against well-funded opponents and by 2020, chosen-prefix attacks against SHA-1 have been common. Therefore, SHA1 should not also be used for Security Application.
  => Alternatives: secure SHA family, such as SHA2 and SHA3 are common practice for hashing purposes in Security Application, at least currently. They are much more secure than both SHA1 and MD5.

- **Question 2: Please explain in brief what makes hash functions resistant to attacks. Provide an exemplary brief case study.**
  A secure hash function is resistant to attacks because they are resistant to three types of attacks, mainly:

1. Collision-resistant: it is hard to find any two different inputs input1 and input2 such that hash(input1) = hash(input2)

2. Preimage resistant: Given H, it is hard to find an input such that H = hash(input).
3. Second preimage resistant: Given an input m1, it should be hard to find another input, input2 (not equal to input1) such that hash(input1) = hash(input2).

Brief case study: An application of hashing functions is storing the hash of the passwords in the database. When the user input the password, it is hashed and compared with the stored hash in the database. If the hashes match then the user is authenticated. When the database is exploited, the attacker sees the hashes but they cannot calculate the original password because hashes functions are one-way.
To tackle preimage resistance, the attacker has to run a high-end CPU that calculates trillions of hashes to break any of the passwords in the database. It becomes increasingly more difficult if he tries to target only one password in the database. Second preimage resistant decreases the chance of the attacker finding any other string whose hash matches the hash of the user's password.

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

MD5	SHA1
Stands for Message Digest	Stands for Secure Hash Algorithm
Can have 128 bits length of message digest	Can have 160 bits length of message digest
Speed is faster than SHA1	Speed is slower than MD5
Simpler than SHA1	More complex than MD5
To make out the initial message the aggressor would want 2^128 operations whereas exploitation the MD5 algorithmic program.	In SHA1 it will be 2^160 that makes it quite troublesome to seek out.
MD5 provides indigent/poor security.	SHA1 provides balanced/tolerable security.
If the assailant needs to seek out the 2 messages having identical message digest then assailant would need to perform 2^64 operations	SHA1 needs to perform 2^80 operations, bigger when compared to MD5.

MD5 performs faster than SHA1 but SHA1 is more secure, and both are already fast so overall if one strictly needs security then SHA1 is a better option.

- **Question 4: What does it mean for a hash algorithm to be broken?**

- The hash algorithm is broken if it is possible to uncover the original message or any other message whose hash matches the known hash without the knowledge of the original message.
- There are three fundamentally different types of attacks on a secure hash:

1. Collision attack: Two different inputs: input1 and input2 are freely chosen, under the constraint such that hash(input1) = hash(input2). This is generally the easiest type of attack. It's the type of attack that's been known for several years against MD5.
2. Preimage attack: is generally much more difficult - Given only the hash value H, try to recover any possible input such that H = hash(input)
3. Second-preimage attack: given an input1, find a different input2 that produces the same hash from a particular hash of input1, or hash(input1) = hash(input2)

Section 5: Annex of the hashing sketch

MD5 Hashing Function sketch

```
#include <MD5.h>

void setup() {

  // put your setup code here, to run once:

  Serial.begin(9600);   // initialize the serial port at 9600 baud

  while (!Serial) {

        ;  // wait for serial port to connect

        } // wait for serial port to connect

  establishContact();  // wait for incoming data

  } /* setup */


void loop() {

  // put your main code here, to run repeatedly:

  String input = "";             // serial input character
  if (Serial.available() > 0){        // if you have data input
        input = Serial.readString();        // read the whole input
        char charInput[input.length() + 1];
        input.toCharArray(charInput, input.length()); // adding the string input as char * type
```

```
        unsigned char* hash = MD5::make_hash(charInput);
        //generate the digest (hex encoding) of our hash
        char *md5str = MD5::make_digest(hash, 16);
         //print it on our serial monitor
       Serial.println(md5str);
       //Give the Memory back to the System if you run the md5 Hash generation in a loop
       free(md5str);
       //free dynamically allocated 16 byte hash from make_hash()
         free(hash);

  } // if Serial.available() > 0

}  /* loop */

void establishContact(){
  if (Serial.available() <= 0) {
     Serial.print("MD5 Hash Function. Please input your message");
  }
  Serial.println();
} // establishContact()
```

SHA1 Hashing Function sketch

```
#include <ArduinoBearSSL.h>

void setup() {

  // put your setup code here, to run once:

  Serial.begin(9600);   // initialize the serial port at 9600 baud

  while (!Serial) {

        ;  // wait for serial port to connect

        } // wait for serial port to connect

  establishContact();  // wait for incoming data

  } /* setup */

void loop() {

  // put your main code here, to run repeatedly:
```

```
  if (Serial.available() > 0){        // if you have data input
        String input = Serial.readString();        // read the whole input
        char charInput[input.length() + 1];
        input.toCharArray(charInput, input.length()); // adding the string input as char * type
         printSHA1(charInput);
        //std::string result = sha1(inputString);
        // Serial.print(result.toString());
   } // if Serial.available() > 0
}  /* loop */

void printSHA1(char* str) {
  Serial.print("SHA1 of '");
  Serial.print(str);
  Serial.print("' is ");

  SHA1.beginHash();
  SHA1.print(str);
  SHA1.endHash();

  printResult();
}

void printResult()
{
  while (SHA1.available()) {
    byte b = SHA1.read();

    if (b < 16) {
      Serial.print("0");
    }

    Serial.print(b, HEX);
  }
  Serial.println();
}

void establishContact(){
   if (Serial.available() <= 0) {
       Serial.print("SHA1 Hash Function. Please input your message");
   }
   Serial.println();
} // establishContact()
```