ELEC-C7420 Basic Principles in Networking
Part II: Security

Assignment II: Implementing Hash functions for Digital Signatures

Members: Beatriz Glaser
Thuy Määttä

1. Goals of the experiment

The goal of this experiment is to explore two types of hash functions and gain a basic understanding of their algorithm, implementation and use. With this exercise we can learn how to secure digital information better, as well as begin to understand the prevention of data breaches and cyber attacks. Furthermore, the experiment tackles and incentivizes curiosity about integrity and cryptography, which is a topic of growing importance in today's world.

2. Experimental Setup

a. For both experiments we used:
   i. Arduino MKR WiFi 1010 board
   ii. Micro USB cable
   iii. Arduino IDE

b. MD5 implementation
   i. Library: ArduinoMD5 (https://github.com/tzikis/ArduinoMD5)
   ii. Step by step setup:
      1. Have a working Arduino IDE with SAMD installed
      2. Connect the Arduino MKR WiFi 1010 board and port to the IDE
      3. Download the above mentioned library
      4. Create a new sketch
         a. Import the downloaded library (MD5.h)
         b. Write a "setup" function to establish serial communication between Arduino and the computer via a USB cable
         c. Implement code in a "loop" function
            i. The program prompts a plain text message from user
            ii. Then prints out the MD5 coded version of the message
      5. Compile and upload code to the board
      6. Test with Serial Monitor

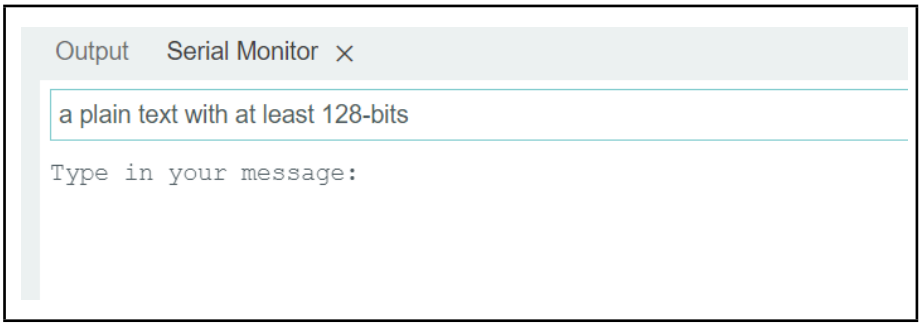
c. SHA-3 implementation
   i. Library: Crypto by Rhys Weatherly
   ii. Step by step setup:
      1. Have a working Arduino IDE with SAMD installed
      2. Connect the Arduino MKR WiFi 1010 board and port to the IDE

3. Download the above mentioned library
4. Create a new sketch
    a. Import the downloaded library (Crypto.h, SHA3.h)
    b. Write a “setup” function to establish serial communication between Arduino and the computer via a USB cable
    c. Implement code in a “loop” function
        i. The program prompts a plain text message from user
        ii. Then prints out the SHA3 coded version of the message
5. Compile and upload code to the board
6. Test with Serial Monitor

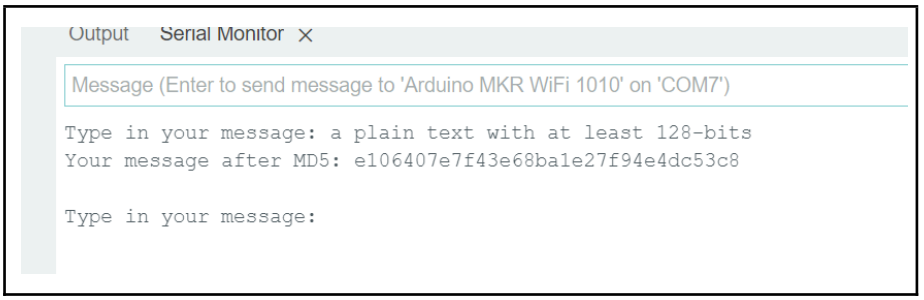
3. Results

a. MD5 implementation

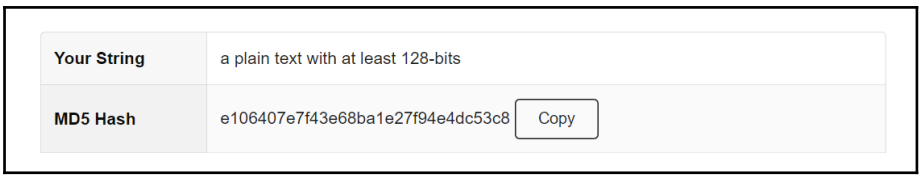
i. After uploading the code, run the Serial Monitor and write a message (e.g.: a 128-bit plain text)



ii. The program then prints out the encoded message after MD5 hashing, and prompts for a new input.



iii. We can double check the hashing through an online MD5 hashing generator (https://www.md5hashgenerator.com/), proving the program works!

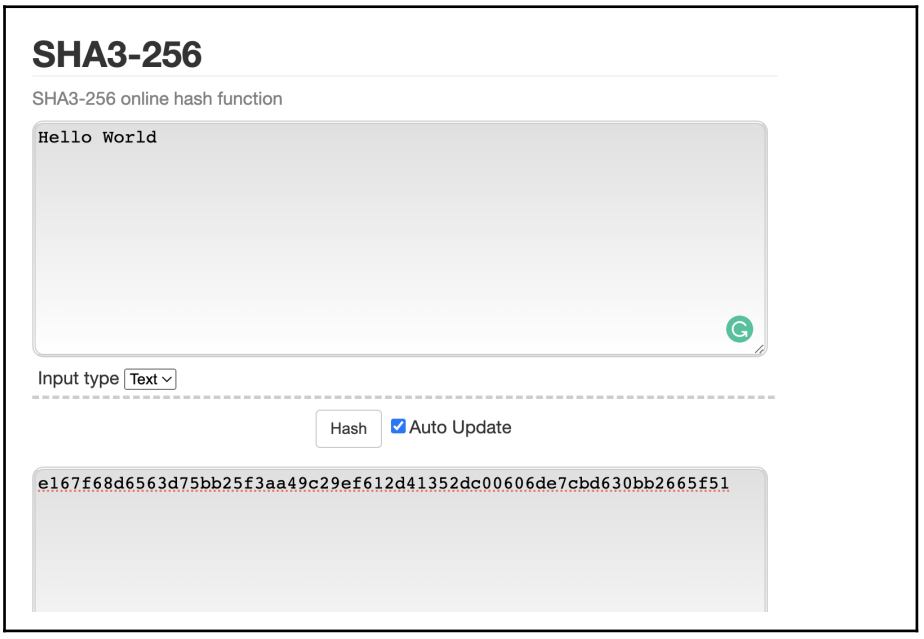


b. SHA-3 implementation

i. After uploading the code, run the Serial Monitor and write a message

ii. The program successfully encrypt the message and print out the result

iii. Double check the hashing through an online hash calculator (https://emn178.github.io/online-tools/sha3_256.html), proving the program works



4. Questions

a. *Of the two mentioned hash functions, would you use one for Security Application? Why? If not, provide an alternative.*

SHA-3 is considered a good choice for Security Application due to its strong resistance to attacks. It uses a sponge construction that provides a higher level of security than older hash functions like MD5 and SHA-1. SHA-3 produces hash values that are longer and more uniformly distributed than previous hash functions, making them harder to guess or reverse-engineer. Additionally, it can reduce the risk of collisions from different applications or inputs. Moreover, it was selected as standard by the National Institute of Standards and Technology (NIST). SHA-3 offers output sizes of 224, 256, 384, and 512 bits, allowing for the accommodation of different security requirements and performance needs.

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

Hash functions are resistant to attacks because they are collision resistant, which means it is difficult to find two messages that produce the same hash value. Secondly, they are preimage resistant, meaning it's challenging to reconstruct the original message from the hash value alone. They are also sensitive to input changes as small changes in the input should result in entirely different output hash values. Also, extending a hash value to include additional data while still producing a valid value is difficult.

As an example, consider storing passwords. In the past, it was common for websites and applications to store passwords in plaintext or use weak encryption algorithms, which are vulnerable and easy to attack. These were not secure and easily hacked. With the development of the internet and the growing IT world, there has been an increase in the security required around passwords. Therefore, nowadays they are often securely stored using hash functions. When a user generates a password, it is hashed and the resulting hash value is stored. When a user logs in, their password is hashed and compared to the previously stored hash value. Using a strong hash function for each password makes recovering the original password from the hash value difficult. This contributes to data integrity, prevents attacks and data breaches.

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

MD5’s algorithm uses a 128 bits message digest, which is a somewhat limited size. This means there is a high probability that you are able to generate two different messages that produce the same hash value, which is not secure. Its compression function is also relatively weak, which makes it easier for attackers to find different inputs with the same intermediate hash value.

Differently, the SHA-3 algorithm uses hash sizes up to 512 bits, which makes it less likely to suffer collision attacks. It also uses a different design approach: sponge construction, where it “absorbs” the input into a state and “squeezes” the state to produce the hash value. SHA-3 is also resistant to length extension attacks.

Overall, MD5 is a lot older and outdated compared to SHA-3. MD5 can still be used for non-cryptographic applications and where the risk of attack is low. However, SHA-3 seems to be a lot more advanced, secure and prepared for today’s world.

d. *What does it mean for a hash algorithm to be broken?*

A hash algorithm is considered “broken” when it is vulnerable to an attack, which means it is not completely capable of providing security. In other words, the algorithm cannot be trusted for cryptographic applications and should not be used.

When broken, it is easy for an attacker to generate, for example, collision attacks. A collision attack means that two different inputs are producing the same hash value, which is unsafe since these values are used to detect data integrity. If two inputs produce the same value, then one could interchange the inputs without being noticed.

5. Conclusion

In conclusion, we learned how different hash algorithms preserve integrity and provide security when handling data. While MD5 has an older design and is more vulnerable to attacks, SHA-3 is a newer algorithm that generates more secure hash values. Overall, it is useful, on professional and personal levels, to explore hash functions and knowing how to verify data integrity.

6. Annex

a. Source code for MD5 implementation

integrity.ino

```
#include "MD5.h"

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

  while (!Serial) {
    ; // Wait for serial port to connect
  }

}

void loop() {

  Serial.print("Type in your message: ");
  while (Serial.available() == 0) {}
  String s = Serial.readString();
  int s_len = s.length();

  Serial.println(s);

  char* char_msg = new char[s_len + 1];
  strcpy(char_msg, s.c_str());


  unsigned char* h = MD5::make_hash(char_msg);

  char *md5_s = MD5::make_digest(h, 16);
  free(h);

  Serial.print("Your message after MD5: ");
  Serial.println(md5_s);
  Serial.println();

  free(md5_s);

}

```

b. Source code for SHA-3 implementation