# CS 305 Module Five Coding Assignment Checksum Verification Template

## Algorithm Cipher

The cipher used is SHA-384.

## Justification

I am using SHA-384 because it is part of the most commonly used cipher suite of TLS 1.3, TLS\_AES\_256\_GCM\_SHA384 (Helme, 2022). SHA-384 has enhanced security compared to SHA-256, making it more resistant to collisions and preimage attacks, and it is also recommended by security professionals to move from the SHA-1 family to the SHA-2 family regardless (Information Technology Laboratory, 2017). SHA-512 could be used in theory, but it is not a part of the standard I have chosen and happens to be slightly slower as well.

To elaborate on collisions, they are a large problem with any deterministic hash function, as they are what drop our time complexity from to , as well as open us up to security problems earlier referred to as preimage attacks, where the hash is reverse engineered since hash functions are not one-to-one, only unto, meaning there is some where the text we are hashing through a mapping function . Note that Hash functions are not groups and only resemble them, their non-symmetric nature (i.e. destructive) means they have no identity element (Gallian, 2016, ch. 2 Groups). However, this is desirable as it means you cannot reverse the message from the checksum, giving an attacker no information on the contents if they obtain the hash / checksum. These facts mean that we have a longer-term viability of our cryptography for Artemis Financial using SHA-384.

## Generate Checksum

[On GitHub at this link](https://github.com/Kubia-Beta/Software-Security-CS305/blob/main/Mod5/ServerApplication.java).

package com.snhu.sslserver;

import java.io.UnsupportedEncodingException;

import java.security.MessageDigest;

import java.security.NoSuchAlgorithmException;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RestController;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

@SpringBootApplication

public class ServerApplication {

public static void main(String[] args) {

SpringApplication.run(ServerApplication.class, args);

}

}

@RestController

class ServerController{

private static final Logger logger = LoggerFactory.getLogger(ServerController.class); // Spin up slf4j logger

@RequestMapping("/hash")

public String myHash(){

final String data = "Connor Sculthorpe"; // My name to hash

String dataHash = null;

try {

dataHash = generateHash(data); // call our hashing function

} catch (NoSuchAlgorithmException | UnsupportedEncodingException e) {

logger.error("Something went wrong with calling the hash function", e); // Log the error if one is thrown

}

// Our hash should be:

// 8d3825ea8fad0009635a6f609e445df83fe85e44f9c848c8f4a876613ad0360c4533dee6aa693de895c800b5c8b3b382

// Thanks to: https://emn178.github.io/online-tools/sha384.html

return "<p>data: " + data + "</p><p>SHA-384 : CheckSum Value: " + dataHash + "</p>";

}

/\*\*

\* Generates a SHA-384 hash from an input string.

\*

\* Time: O(n), θ(1), Ω(1)

\* Space: O(n+k)

\* @param String data

\* @return String dataHashed

\* @throws NoSuchAlgorithmException, UnsupportedEncodingException

\* References: https://stackoverflow.com/questions/5531455/how-to-hash-some-string-with-sha-256-in-java

\*/

private String generateHash(String data) throws NoSuchAlgorithmException, UnsupportedEncodingException {

final MessageDigest md = MessageDigest.getInstance("SHA-384"); // Create message digest obj

final byte[] digest = md.digest(data.getBytes("UTF-8")); // Compute the message digest (byte array)

StringBuilder hexString = new StringBuilder(); // Buffer to move our data into, use StringBuffer for thread safety if needed

for (int i = 0; i < digest.length; i++) {

final String hex = Integer.toHexString(0xff & digest[i]);

if (hex.length() == 1) {

hexString.append('0'); // Leading zeroes

}

hexString.append(hex);

}

return hexString.toString();

}

}

## Verification

A screen shot of a computer

Description automatically generated

References

Gallian, J. A. (2016). Contemporary abstract algebra (9th ed.). Cengage Learning.

Helme, S. (2022, June 30). *Top 1 million analysis - June 2022*. Scott Helme. <https://scotthelme.co.uk/top-1-million-analysis-june-2022/>

Information Technology Laboratory (2017, January 04). *Hash Functions*. U.S. Department of Commerce, National Institute of Standards and Technology. <https://csrc.nist.gov/projects/hash-functions>