# CS 305 Module Five Coding Assignment Checksum Verification Template

## Algorithm Cipher

As a recommendation, I would say SHA-256 is a good choice. It is considered incredibly safe with extremely low odds of collision. As of right now, there is not any known collision to exist. The collision rate is 1 in every 2256.

## Justification

SHA-256 is a hashing algorithm that uses 256 bits. The algorithm is used as a mathematical function that can help to scramble up text into an unreadable mess. It will end up needing a key to unscramble it to make it readable again. Collisions while using a hash function could become an issue. Some issues are minimal, but others could be extremely important and unique data. A collision is when 2 different data statements go through the hash function and result in the same hash value. This could become an issue for many reasons. One is when using the key you would receive unwanted data back, and the other is someone unauthorized could end up seeing private data.

## Generate Checksum

package com.snhu.sslserver;

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 java.security.MessageDigest;

import java.security.NoSuchAlgorithmException;

import java.nio.charset.StandardCharsets;

import java.math.BigInteger;

*@SpringBootApplication*

public class ServerApplication {

public static void main(String[] args) {

SpringApplication.*run*(ServerApplication.class, args);

}

}

*@RestController*

class ServerController{

//**FIXME**: Add hash function to return the checksum value for the data string that should contain your name.

//Used NoSuchAlgorithmExeption to run as Java application

public static String hashFunc(String data) throws NoSuchAlgorithmException{

MessageDigest md = MessageDigest.*getInstance*("SHA-256");

byte[] biteDigest = md.digest(data.getBytes(StandardCharsets.***UTF\_8***));

BigInteger num = new BigInteger(1, biteDigest);

StringBuilder stringAsNum = new StringBuilder(num.toString(16));

while(stringAsNum.length() < 32) {

stringAsNum.insert(0,'0');

}

return stringAsNum.toString();

}

*@RequestMapping*("/hash")

public String myHash() throws Exception{

//String data = "Hello Joe Smith!";

String data = "Hello Tiffany McDonnell!";

String checkSum = *hashFunc*(data);

return "<p>data:"+ data + " : SHA-256 " + checkSum;

}

}

## Verification

A screenshot of a computer

Description automatically generated

**Citations**

Callaghan, P. (n.d.). *Why you should use SHA-256 in evidence authentication*. Pagefreezer Blog. https://blog.pagefreezer.com/sha-256-benefits-evidence-authentication

*Java cryptography - message digest*. Online Courses and eBooks Library. (n.d.). https://www.tutorialspoint.com/java\_cryptography/java\_cryptography\_message\_digest.htm

Knudsen, J. (n.d.). *Java cryptography*. O’Reilly Online Learning. https://learning.oreilly.com/library/view/java-cryptography/1565924029/ch06s01.html#ch06-5-fm2xml

*Secure hash algorithms*. Secure Hash Algorithms - Practical Cryptography for Developers. (n.d.). https://cryptobook.nakov.com/cryptographic-hash-functions/secure-hash-algorithms