****

# Practices for Secure Software Report

Table of Contents

[Document Revision History 3](#_Toc102040754)

[Client 3](#_Toc102040755)

[Instructions 3](#_Toc102040756)

[Developer 4](#_Toc102040757)

[1. Algorithm Cipher 4](#_Toc102040758)

[2. Certificate Generation 4](#_Toc102040759)

[3. Deploy Cipher 4](#_Toc102040760)

[4. Secure Communications 4](#_Toc102040761)

[5. Secondary Testing 4](#_Toc102040762)

[6. Functional Testing 4](#_Toc102040763)

[7. Summary 4](#_Toc102040764)

[8. Industry Standard Best Practices 4](#_Toc102040765)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **20241212** | **Brandon Murphy** |  |

## Client



## Instructions

Submit this completed practices for secure software report. Replace the bracketed text with the relevant information. You must document your process for writing secure communications and refactoring code that complies with software security testing protocols.

* Respond to the steps outlined below and include your findings.
* Respond using your own words. You may also choose to include images or supporting materials. If you include them, make certain to insert them in all the relevant locations in the document.
* Refer to the Project Two Guidelines and Rubric for more detailed instructions about each section of the template.

## Developer

Brandon Murphy

## Algorithm Cipher

The encryption algorithm that best suits Artemis Financial is SHA3-512. This hash function is the latest version spinoff of its predecessor before being the SHA1 series of the that specific hash function. The SHA-3 hash function was complete and published by NIST August 2015. The SHA 256 hash function would be a similar option regarding its useability it is faster than the SHA3-512 due to its simplicity. However, it is not as secure as the SHA3-512. When it comes to Artemis financial having an obligation legally and morally to deciding on an appropriate and secure encryption for something as big as customer finances, I think it is important to consider utilizing the newest and most secure hash function. The SHA3-512 uses the KECCAK sponge function to transform an input of any length into a fixed size message digest. The SHA3-512 has a block size of 576 bits and a capacity of 1024 bits. The SHA3-512 produces a 512-bit hash output. You will expect to essentially have longer with no output string which makes for a more difficult algorithm to cipher the encrypted data. The algorithm is implemented on the Leon3 processor. This helps with efficiency regarding the hashing process. Leon3 has a C compiler as well which allows for easier development as well. The SHA is neither symmetric nor asymmetric it is not a cipher it is a cryptographic hash function in this case it generates 512- bit hash value. SHA does not use encryption keys. The hash function to verify files will use the SHA3-512 cipher to create a checksum signature of the provided message. SHA3 is a much more complex algorithm, the state consists of 5×5 lanes, each lane containing 64 bits. The state consists of 64 slices, and each slice contains 5\*5=25 bits. For the SHA3-512, w is equal to 64, and b is equal to 1600. During the initialization and padding phases,

the hash function transforms an input message with arbitrary length to a message with a fixed size.

## Certificate Generation

Insert a screenshot below of the CER file.

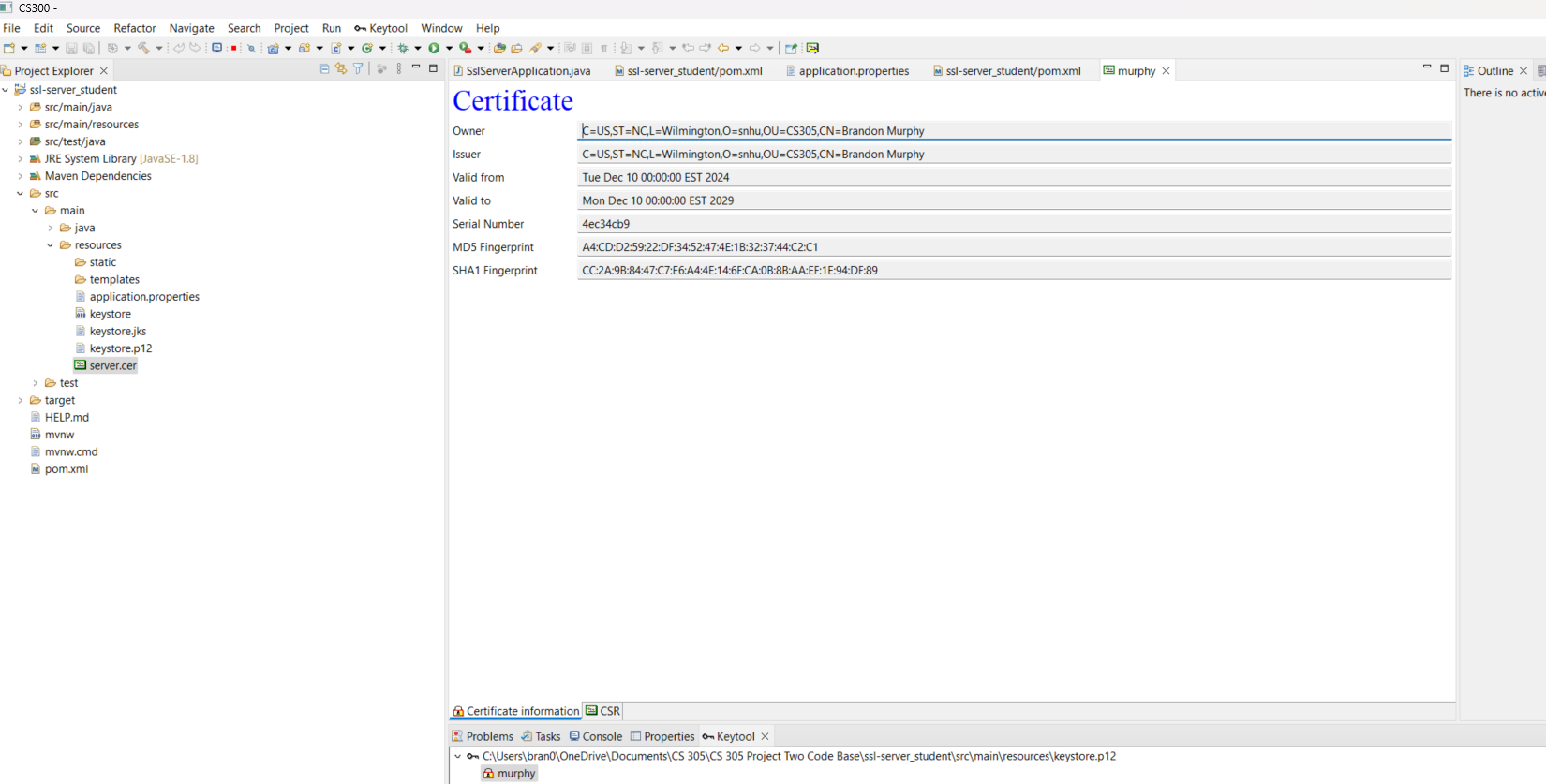
A screenshot of a computer program

Description automatically generated

This is the command prompt printout of the .cer file

A screenshot of a computer

Description automatically generated



This is the .cer file and where it is stored in the project.

A screenshot of a certificate

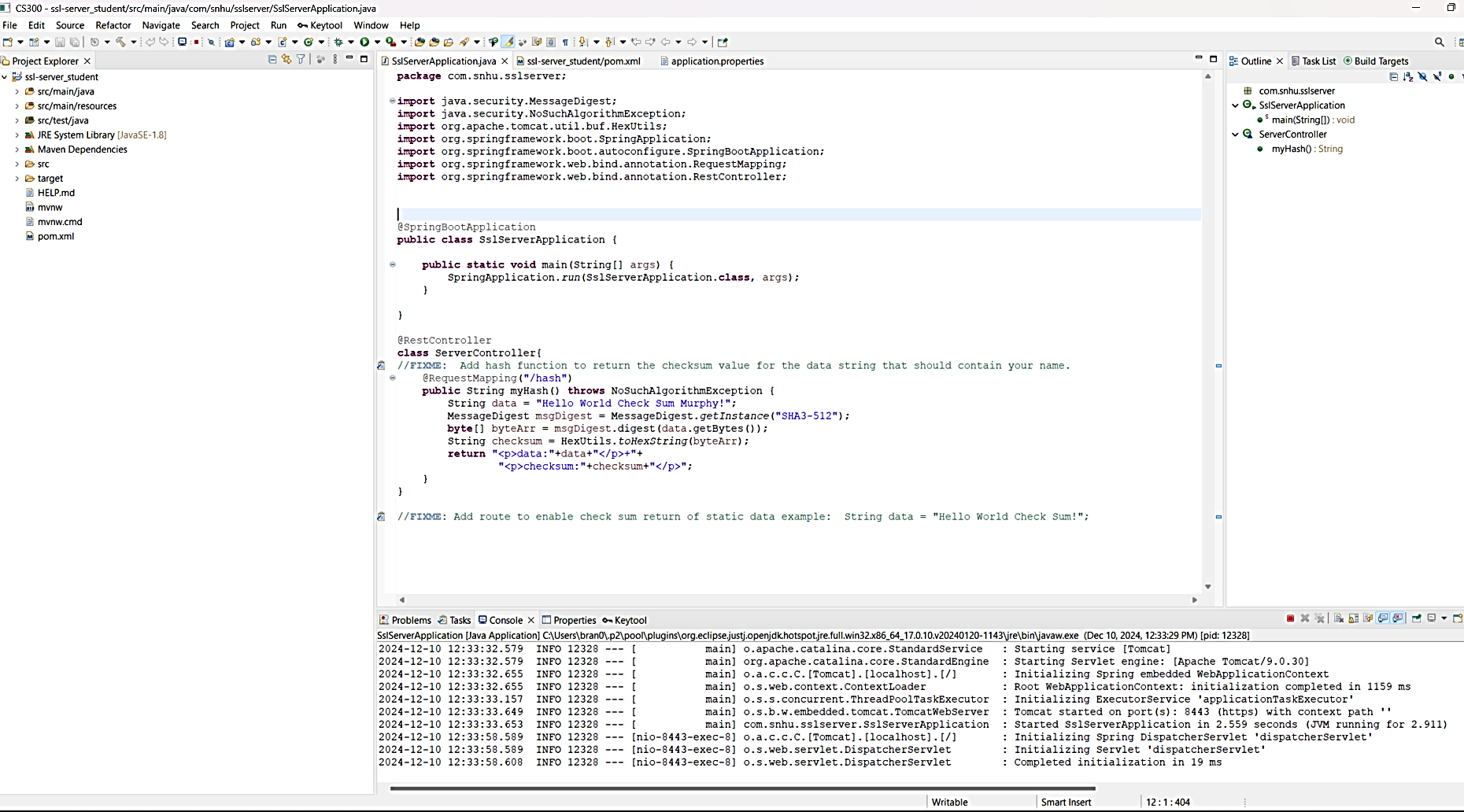
Description automatically generated

This is the certificate information

## Deploy Cipher

Insert a screenshot below of the checksum verification.

This is a screenshot of the running code without errors.



This is the refactored code for the checksum verification.

**package** com.snhu.sslserver;

**import** java.security.MessageDigest;

**import** java.security.NoSuchAlgorithmException;

**import** org.apache.tomcat.util.buf.HexUtils;

**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;

@SpringBootApplication

**public** **class** SslServerApplication {

**public** **static** **void** main(String[] args) {

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

}

}

@RestController

**class** ServerController{

@RequestMapping("/hash")

**public** String myHash() **throws** NoSuchAlgorithmException {

String data = "Hello World Check Sum Murphy!";

MessageDigest msgDigest = MessageDigest.*getInstance*("SHA3-512");

**byte**[] byteArr = msgDigest.digest(data.getBytes());

String checksum = HexUtils.*toHexString*(byteArr);

**return** "<p>data:"+data+"</p>+"+

"<p>checksum:"+checksum+"</p>";

}

}

## Secure Communications

Insert a screenshot below of the web browser that shows a secure webpage.

A screenshot of a computer

Description automatically generated

This is the checksum verification using <https://localhost:8443/hash> webpage.

## Secondary Testing

Insert screenshots below of the refactored code executed without errors and the dependency-check report. This is the dependency check done utilizing the latest version 11.1.1

A screenshot of a computer

Description automatically generated

This is the dependency check showing the build was a success with the refactored code. In the pom.xml file.

A screenshot of a computer

Description automatically generated

The only thing I changed here is really just the version I used is the latest 11.1.1 version.

## Functional Testing

Insert a screenshot below of the refactored code executed without errors.

A screenshot of a computer

Description automatically generated

## Summary

In my code refactoring, I started by updating to the latest Maven dependency check version from 5.3.0 to 11.1.1 in the pom.xml file. Regarding the vulnerability flow diagram to insure I had secure input and representations I referenced the secure coding guidelines from the Oracle site. To incorporate this with the given code and task provided I started by creating a @Restcontroller class to create a RESTful web service to secure information exchanged. I then used a @RequestMapping to apply it to my controller class for the hash. For my main method I have a public string myhash with an exception because I am requesting an algorithm, and it is not available yet in the environment. To help validate input I made sure to add specific parameters and characteristics I want my input/output to produce since the goal here for “checksum verification” is to convert a string of data into an encrypted message utilizing a hash function. I called MessageDigest to use it’s SHA3-512 algorithm, then referenced the MessageDigest site for the method. Using algorithms, we are able to satisfy the use of cryptography as well. Since there is proper input validation and API interactions and utilizing cryptography, this essentially comes together to secure client/server data. When I run my code there is no errors, the code it short to reduce the potential attack surface also makes it easy to read, update, and add too, this is good for the quality of the code.

## Industry Standard Best Practices

For this I already had a file with some already existing code, so I first made sure all the files the dependencies were up to date and used the already existing layout to build upon and make it more secure. I did not really delete anything that was already there, I just added to and made it simple to keep the project agile while enhancing the security of the data. I utilized simple in line comments to guide to keep it organized of what is going on. Then I also used proper coded indentation as well. It is good to use industry standard best practices because it allows other developers in the future to understand your work and update it if necessary. It also makes it easier and more organized for you to go back and make changes. Keeping things organized in a standard way and commenting is good for the software development community as a whole.