****

# CS 305 Project Two

**Practices for Secure Software Report**

**Table of Contents**

[Document Revision History 3](#_1fob9te)

[Client 3](#_3znysh7)

[Instructions 3](#_2et92p0)

[Developer 4](#_tyjcwt)

[1. Algorithm Cipher 4](#_3dy6vkm)

[2. Certificate Generation 4](#_1t3h5sf)

[3. Deploy Cipher 4](#_4d34og8)

[4. Secure Communications 4](#_2s8eyo1)

[5. Secondary Testing 4](#_17dp8vu)

[6. Functional Testing 5](#_3rdcrjn)

[7. Summary 5](#_26in1rg)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **02/17/2022** | **Sam Maclean** | **Null** |

## Client



## Instructions

Deliver this completed Practices for Secure Software Report documenting your process for writing secure communications and refactoring code that complies with software security testing protocols.

Respond to the steps outlined below and replace the bracketed text with your findings in your own words. If you choose to include images or supporting materials, be sure to insert them throughout.

## Developer

Sam Maclean

## 1. Algorithm Cipher

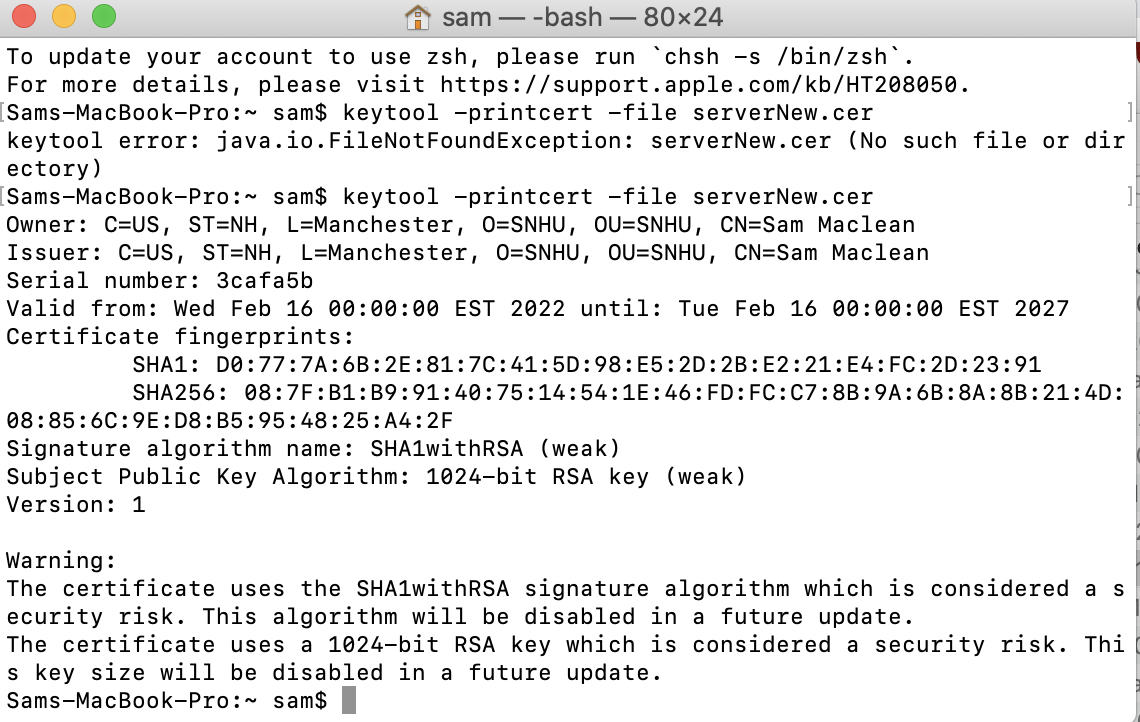
Determine an appropriate encryption algorithm cipher to deploy given the security vulnerabilities, justifying your reasoning. Be sure to address the following:

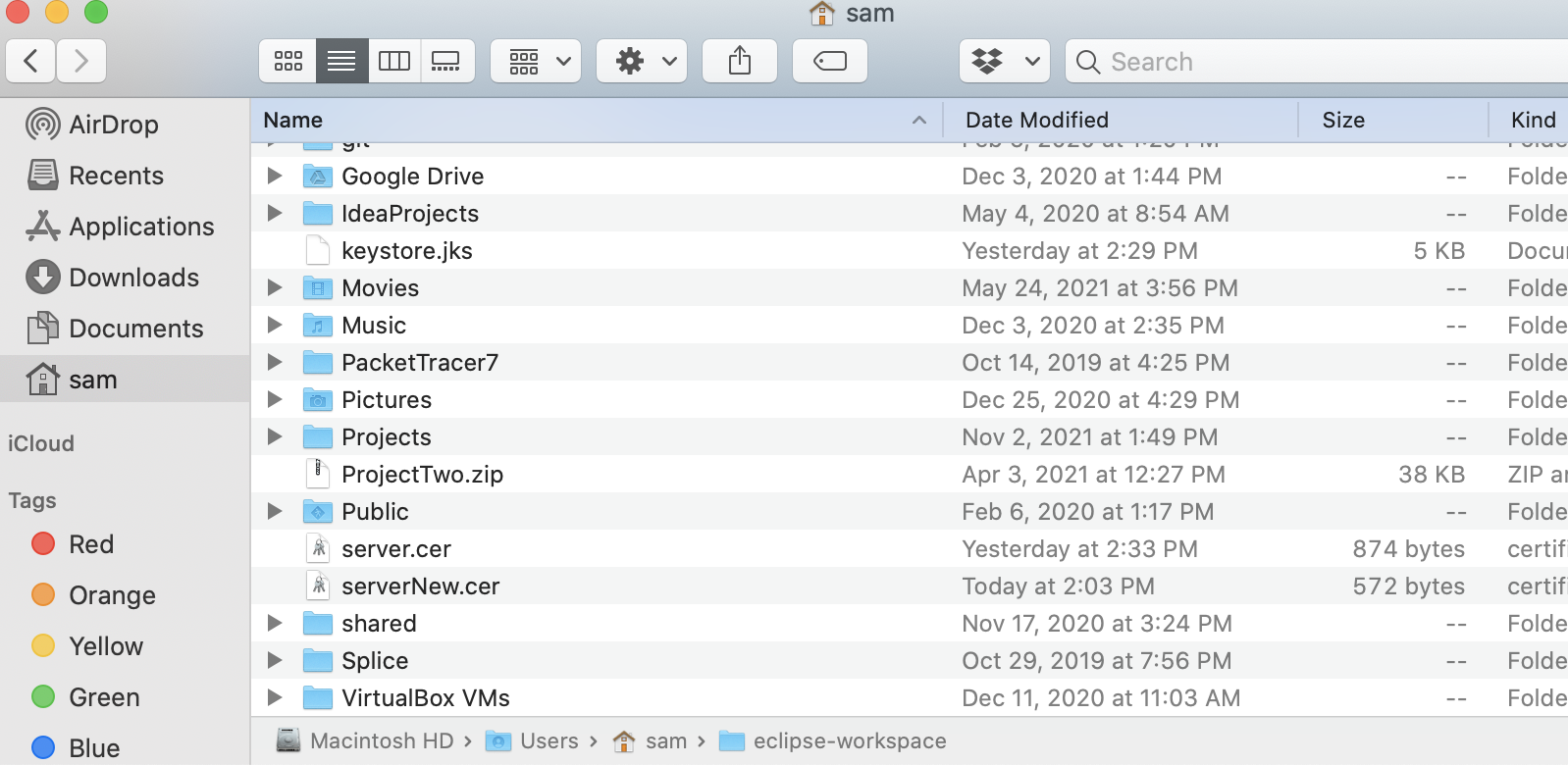
* Provide a brief, high-level overview of the encryption algorithm cipher.
* Discuss the hash functions and bit levels of the cipher.
* Explain the use of random numbers, symmetric vs non-symmetric keys, and so on.
* Describe the history and current state of encryption algorithms.

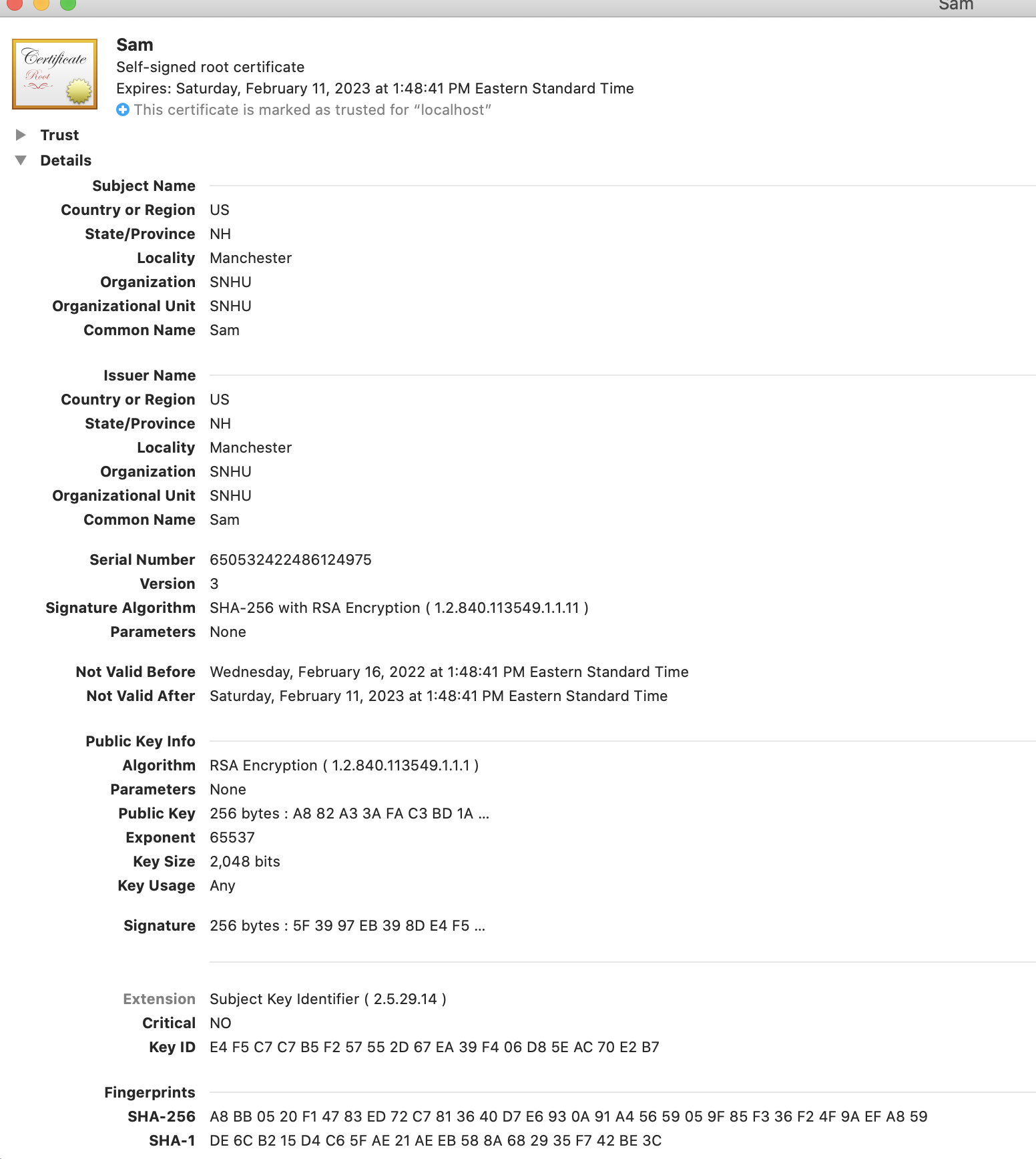
I chose the RSA encryption algorithm cipher that makes use of the Secure Hash Algorithm (SHA)-256 hashing algorithm because it is widely used among developers due to its lack of vulnerabilities which makes it very difficult to exploit. SHA-256 is a cryptographic hash function that computes a hash that is 256 bits long for a given piece of data. This makes it incredibly difficult to reverse the hashing algorithm due to its complexity and a large number of bits. Data is converted to random numbers and letters that have an equal probability of being selected. The resulting hash then uses RSA to encrypt the hash with a secure private key which makes it nearly impossible to decrypt the data.

## 2. Certificate Generation

Generate appropriate self-signed certificates using the Java Keytool, which is used through the command line.

* To demonstrate that the keys were effectively generated, export your certificates (CER file) and submit a screenshot of the CER file below.

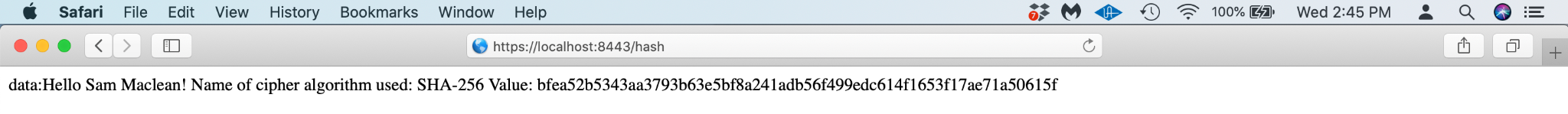




## 3. Deploy Cipher

Refactor the code and use security libraries to deploy and implement the encryption algorithm cipher to the software application. Verify this additional functionality with a checksum.

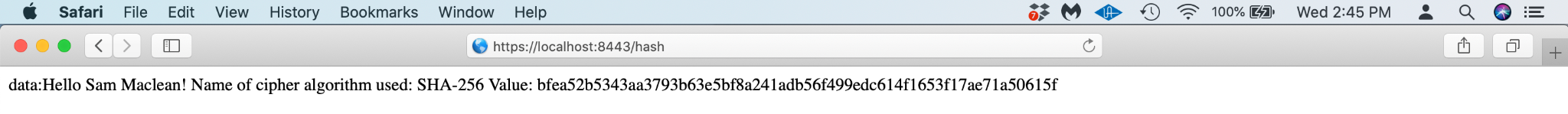
* Insert a screenshot below of the checksum verification. The screenshot must show your name and a unique data string that has been created.



## 4. Secure Communications

Refactor the code to convert HTTP to the HTTPS protocol. Compile and run the refactored code to verify secure communication by typing **https://localhost:8443/hash** in a new browser window to demonstrate that the secure communication works successfully.

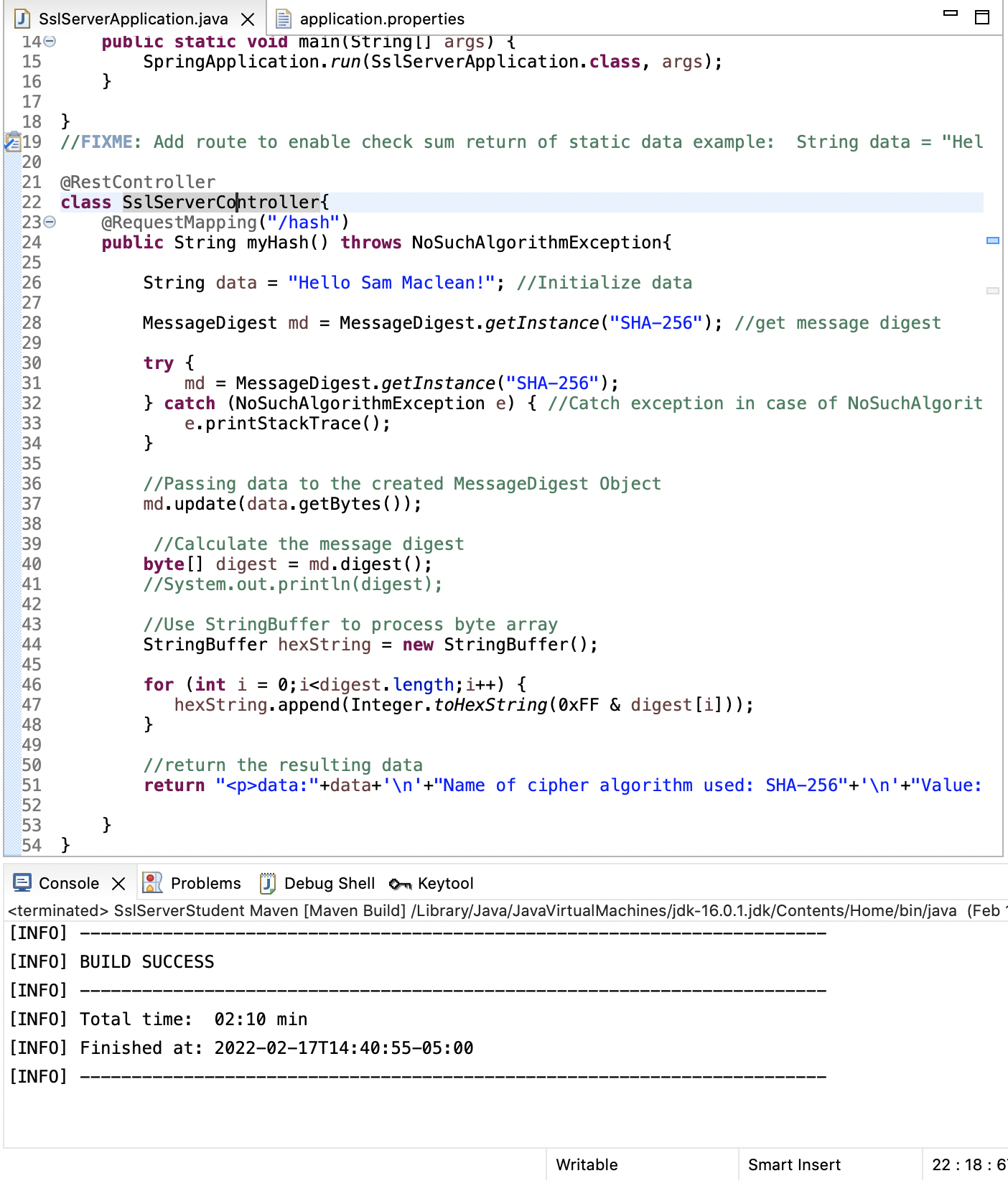
* Insert a screenshot below of the web browser that shows a secure webpage.

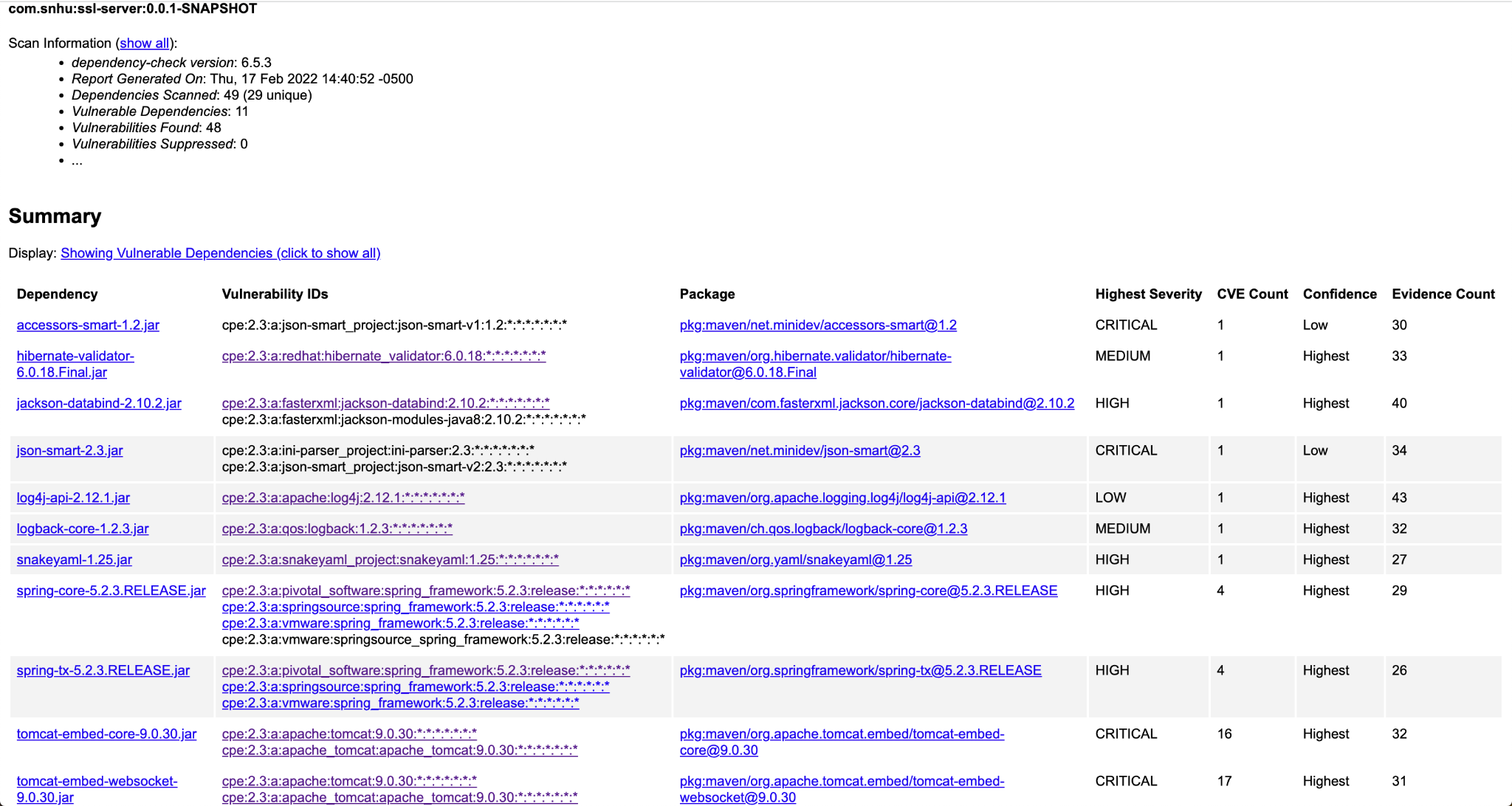


## 5. Secondary Testing

Complete a secondary static testing of the refactored code using the dependency check tool to ensure code complies with software security enhancements. You only need to focus on the code you have added as part of the refactoring. Complete the dependency check and review the output to ensure you did not introduce additional security vulnerabilities.

* Include the following below:
  + A screenshot of the refactored code executed without errors
  + A screenshot of the dependency check report

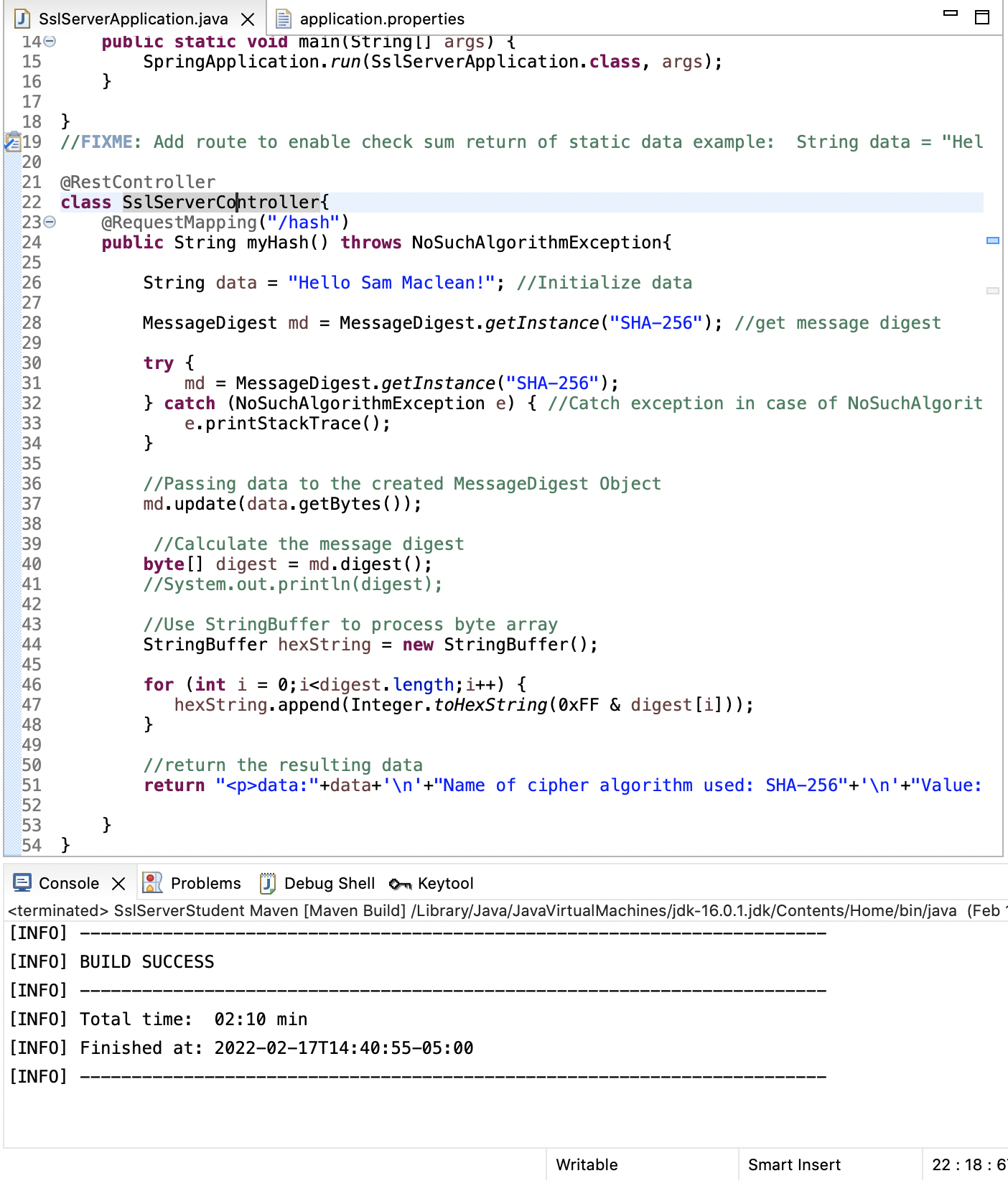




## 6. Functional Testing

Identify syntactical, logical, and security vulnerabilities for the software application by manually reviewing code.

* Complete this functional testing and include a screenshot below of the refactored code executed without errors.



## 7. Summary

Discuss how the code has been refactored and how it complies with security testing protocols. Be sure to address the following:

* Refer to the Vulnerability Assessment Process Flow Diagram and highlight the areas of security that you addressed by refactoring the code.
* Discuss your process for adding layers of security to the software application and the value that security adds to the company’s overall wellbeing.
* Point out best practices for maintaining the current security of the software application to your customer.

This program uses input validation RESTful APIs to securely communicate data between the client and server by encrypting data with RSA encryption and the SHA-256 hashing algorithm. The code was refactored to take in a string of data that gets processed through the SHA-256 algorithm to turn the original string of data into a series of numbers and letters that are incredibly difficult to reverse. I also used Cryptography in the Vulnerability Assessment Process Flow Diagram by encrypting the string of data with SHA-256. I handled code errors like exceptions by catching a NoSuchAlgorithm exception which is common when requesting input for the message digest part of the code.

I started by generating a certificate that allows the user to securely communicate with the website since the certificate establishes trust. Then I proceeded to add the SHA-256 hashing algorithm to encrypt the financial data that may be transferred to and from client and server. I also caught an exception to make sure the program does not break and expose additional vulnerabilities when utilizing the Artemis Financial website. These layers of security help protect against attacks due to security vulnerabilities within a code for a website. To maintain security for customers that use a software application, it is important to keep all plugins and other programs up to date by generating a dependency check report to see if any vulnerabilities are present and making sure the code aligns with best practices when referencing the Vulnerability Assessment Process Flow Diagram.