****

# CS 305 Project Two

**Practices for Secure Software Report**

Table of Contents

[Document Revision History 3](#_Toc33111302)

[Client 3](#_Toc33111303)

[Instructions 3](#_Toc33111304)

[Developer 4](#_Toc33111305)

[1. Algorithm Cipher 4](#_Toc33111306)

[2. Certificate Generation 4](#_Toc33111307)

[3. Deploy Cipher 4](#_Toc33111308)

[4. Secure Communications 4](#_Toc33111309)

[5. Secondary Testing 4](#_Toc33111310)

[6. Functional Testing 5](#_Toc33111311)

[7. Summary 5](#_Toc33111312)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **2/27/2022** | **Garrett Dunn** |  |

## 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

Garrett Dunn

## 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.

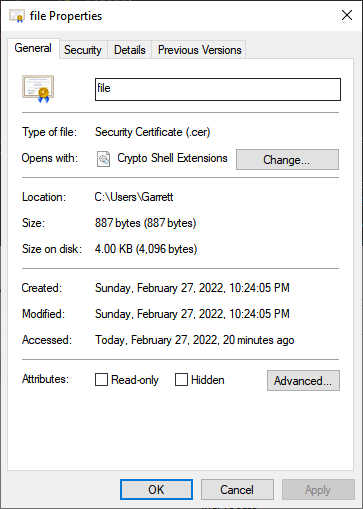
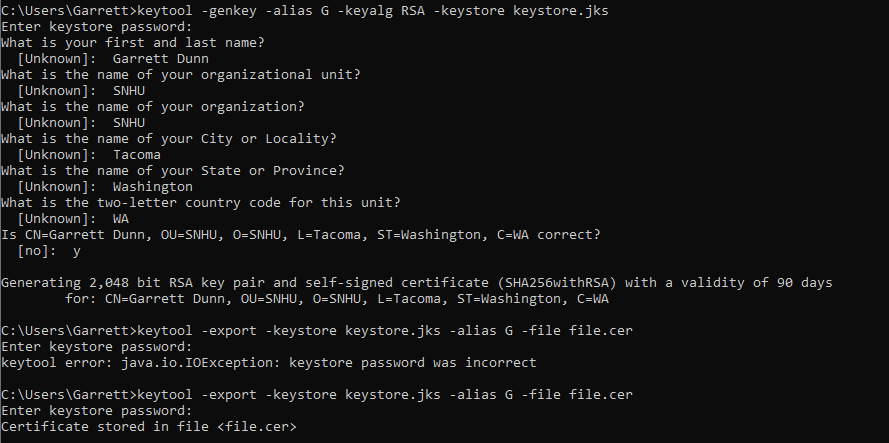
Artemis Financial would benefit from AES with SHA-256 as a hash function as their algorithm cipher. AES has proven to provide solid fortification against all types of cyber-attacks. One exception would be brute-force attacks, as it has been proven vulnerable to, however, the likelihood of this decreases as the byte count (128, 192, 256) becomes more complicated with this algorithm cipher. This algorithm cipher is utilized by many institutions and platforms, including governmental entities and banking systems, which is a testament to how effective it is. AES uses the same key (symmetric) to encrypt and decrypt data whereas non-symmetric keys create a unique key for both the encryption and decryption of data. These keys generate random numbers in lengths of 128, 192, and 256 bits, which makes unauthorized access nearly impossible.

Encryption algorithms date all the way back to almost 600 B.C., where the Spartans developed an encryption device called the scytale which allowed soldiers to send secret messages in battle using a wooden rod and a leather strap that was engraved with the message. The only way to read the message was to have the correctly sized rod. Moving forward in time, one of the most well-known encryption tools was developed in 1918 by a man named Arthur Scherbius. This machine was called the Enigma, and it utilized rotating disks in which keys were embedded, along with an encoded substitution table where each time a new character was typed, the table would change. This was extensively used by the Nazis until the Enigma was cracked, which influenced the allies' victory. Today, encryption is utilized in everyday life. Whether it be withdrawing cash at an ATM machine, or talking to your Amazon Alexa, encryption methods continue to get more robust as each year passes.

## 2. Certificate Generation

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

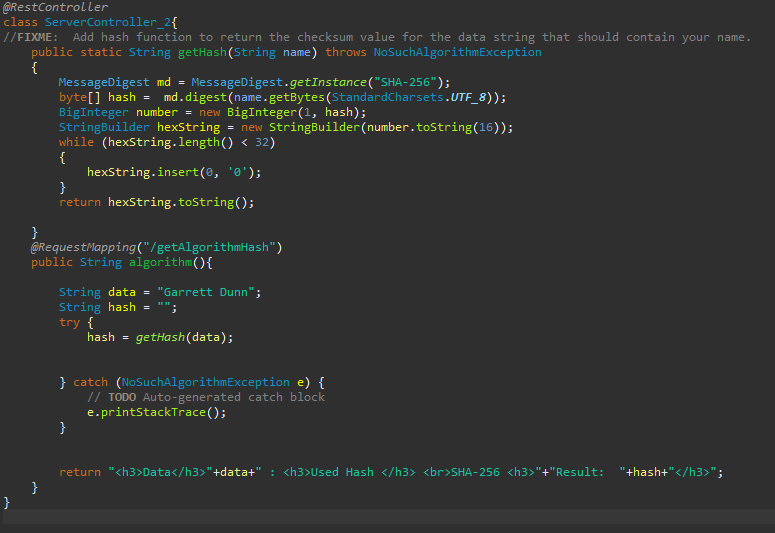
* To demonstrate that the keys were effectively generated, export your certificates (CER file) and submit a screenshot of the CER file below.



## 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.

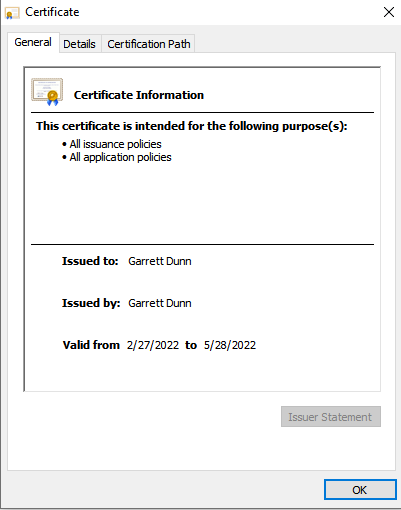
* Insert a screenshot below of the checksum verification. The screenshot must show your name and a unique data string that has been created.

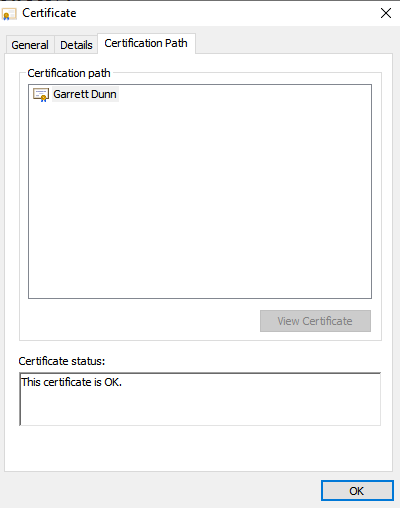


## 4. Secure Communications

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.



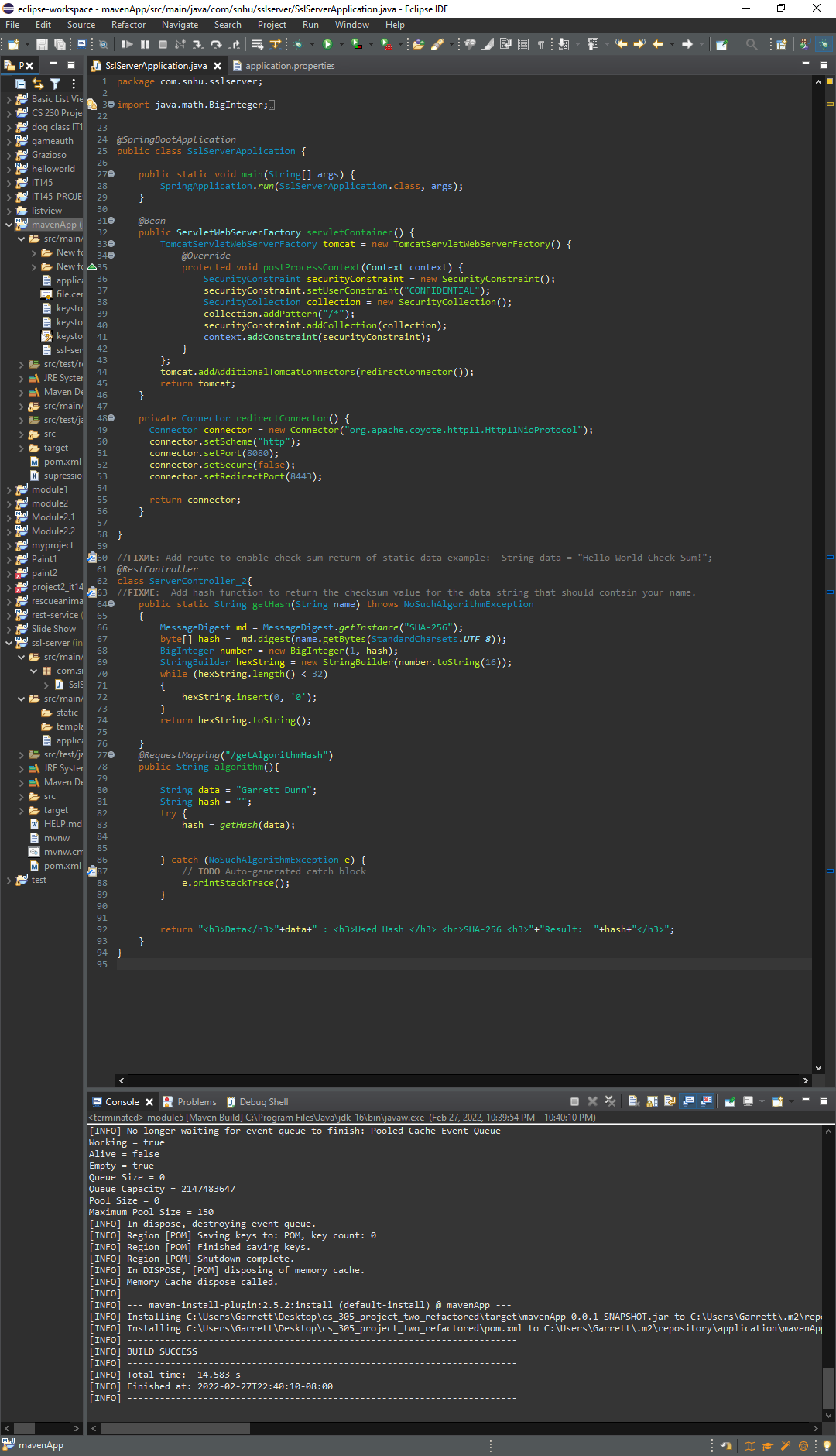


No matter what I did I couldn’t get it secure, I even added it to the trusted root certification authorities store in mmc. I don’t know what I am doing wrong.

## 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.

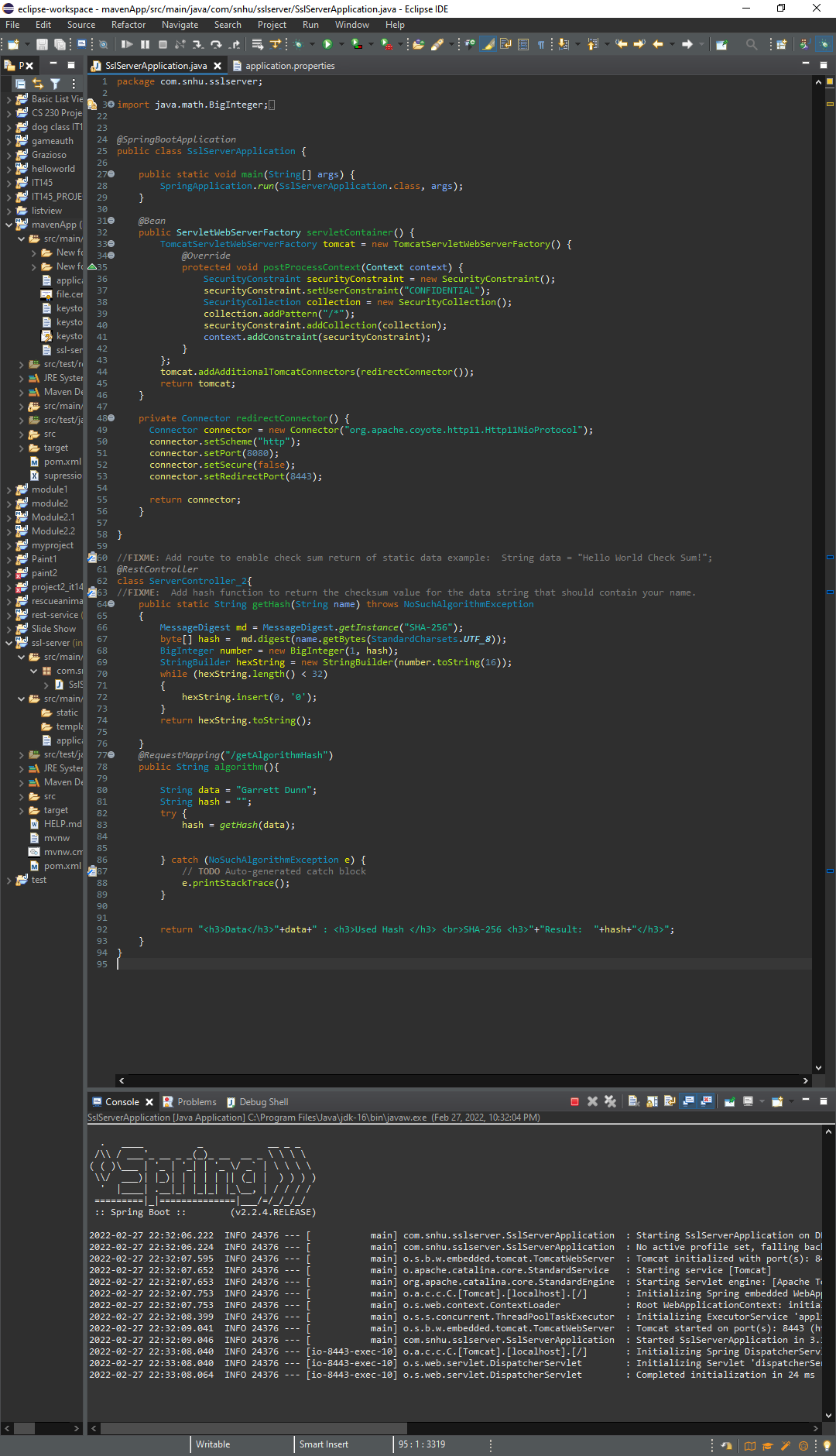
* Include the following below:
  + A screenshot of the refactored code executed without errors
  + A screenshot of the dependency check report



## 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.

I encrypted the data after code refactorization by utilizing a hash function. This bolstered the security of the API, especially since this is a RESTful application. To ensure that data can be exchanged more securely, I utilized a certificate. This improves the security of the client and server. The utilization of a try and catch clause reaffirmed that the integrity of the code would be secured. Making sure that the system was up to date on the Spring Boot and Tomcat versions improved the security of the application immensely and provided a safeguard against any known vulnerabilities. All these steps ensure that Artemis Financial’s system is as secure as possible for its clients.

Ensuring the security of a system is vital in today’s technological climate. With so many potential threats daily, making sure that your consumer's data is the most important thing web developers face. By constantly monitoring encryption methods used, maintaining dependencies, and reviewing the code base, you can ensure that the system is as secure as possible and that the data is safe from unauthorized access.