****

# CS 305 Project Two

**Practices for Secure Software Report**

Table of Contents

[\_Toc33111301](#_Toc33111301)

[Document Revision History 3](#_Toc33111302)

[Client 3](#_Toc33111303)

[Instructions 3](#_Toc33111304)

[Developer 4](#_Toc33111305)

[1. Algorithm Cipher 4](#_Toc33111306)

[2. Certificate Generation 5](#_Toc33111307)

[3. Deploy Cipher 5](#_Toc33111308)

[4. Secure Communications 6](#_Toc33111309)

[5. Secondary Testing 7](#_Toc33111310)

[6. Functional Testing 8](#_Toc33111311)

[7. Summary 9](#_Toc33111312)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **2-21-21** | **Kris Daigle** |  |

## 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

Kristian Daigle

## 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 decided to use an AES-256 cipher for this project. This is a highly secure filter. I chose this for Artemis Financial as they are guarding highly sensitive data. I chose AES because it supports a variety of key sizes. Either the 128 or the 256-bit should work in this case, for this project I used 256 as it’s extremely difficult to crack. It also uses symmetric key creation. This allows the application to encrypt data as needed and deliver keys to users.

The use of symmetric vs non-symmetric keys depends upon the application and what it needs to do. Symmetric keys are shared between the server and the client. Non-symmetric keys include public and private keys with private only being available/known to the client. These are then used to encrypt data. This data stays encrypted until accessed again using the correct key.

The major downside to this is when the data ends up being lost because without the key, there’s no way to recover it. Random number generators can be used to make a unique ID for transactions. This is used to identify things like data transferring or any communications that take place.

Although the use of computer encryption is newer than most sciences, encryption algorithms have been used for some time. By definition, an encryption algorithm is a mathematical formula that changes plaintext into ciphertext. This can be done in reverse also to reveal what was encrypted. Throughout history, this has been used several times to hide sensitive information. This was used with hieroglyphs and other various codes. Codes were used in wartime to protect information as well. For these algorithms to be used, those sending and receiving the messages had the algorithm or key to decode the messages being sent.

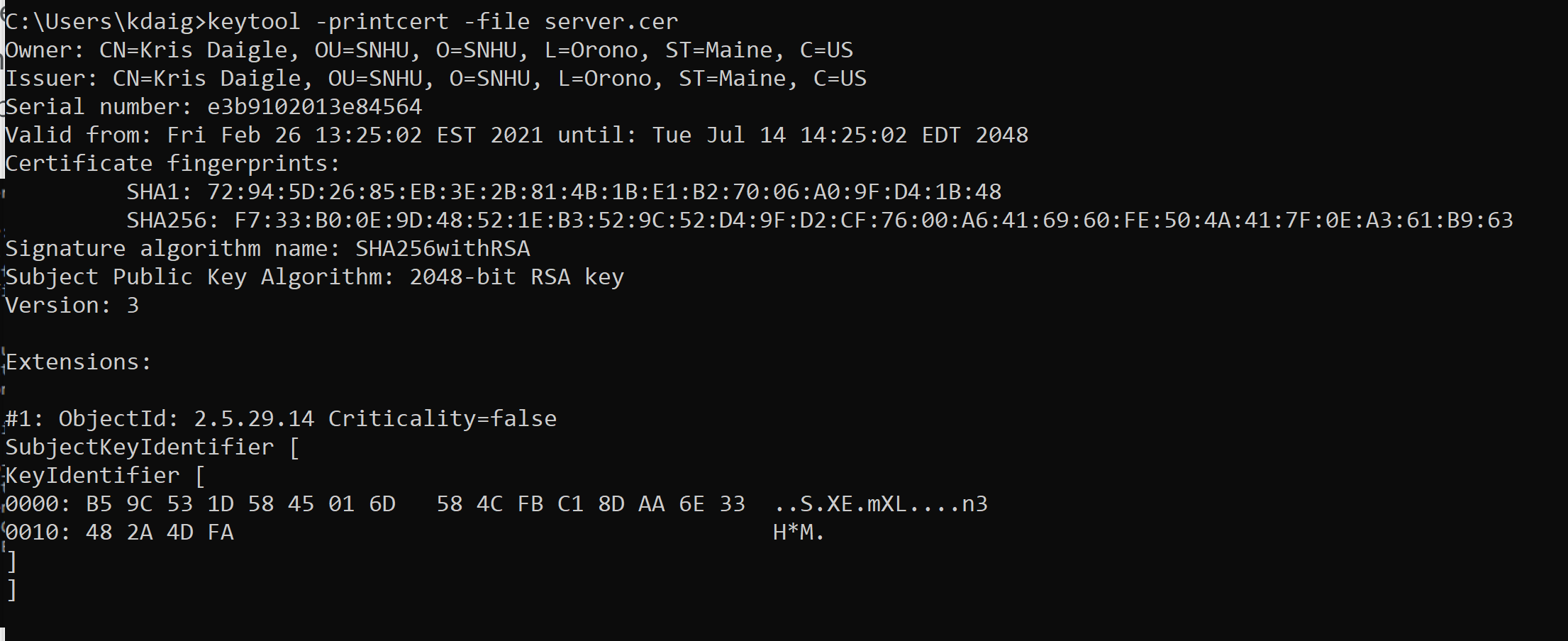
As technology developed with the introduction of electricity, it became possible to use similar encryption to protect information being stored electronically. Enigma Machines were even used during World War I to encrypt messages being sent.

With the development of more advanced technology, we require more advanced encryption methods and better encryption algorithms. Much like any other science, cryptology is still evolving with the times.

## 2. Certificate Generation

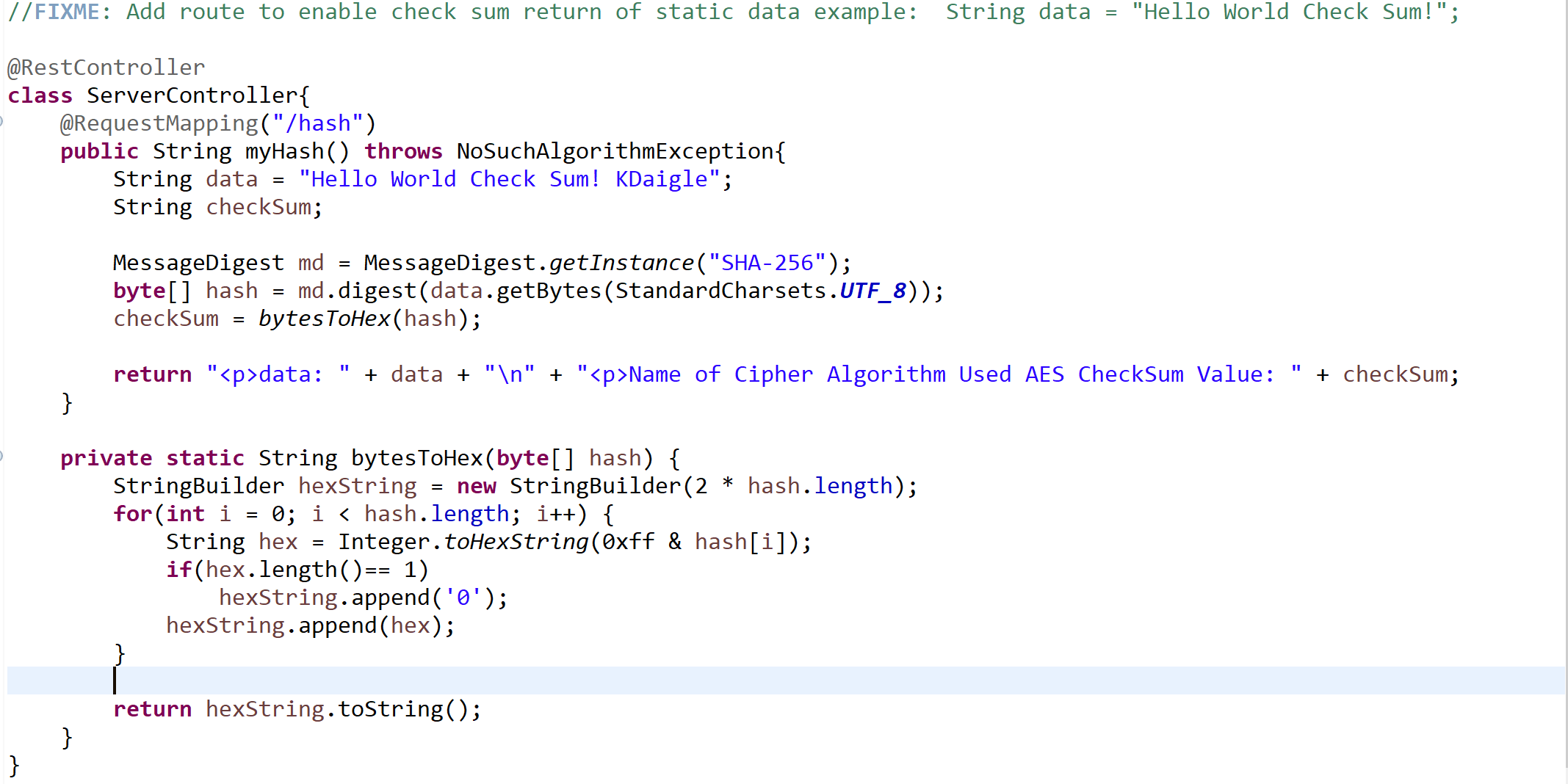
Generate appropriate self-signed certificates using the Java Keytool, which is used through the command line.

* 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

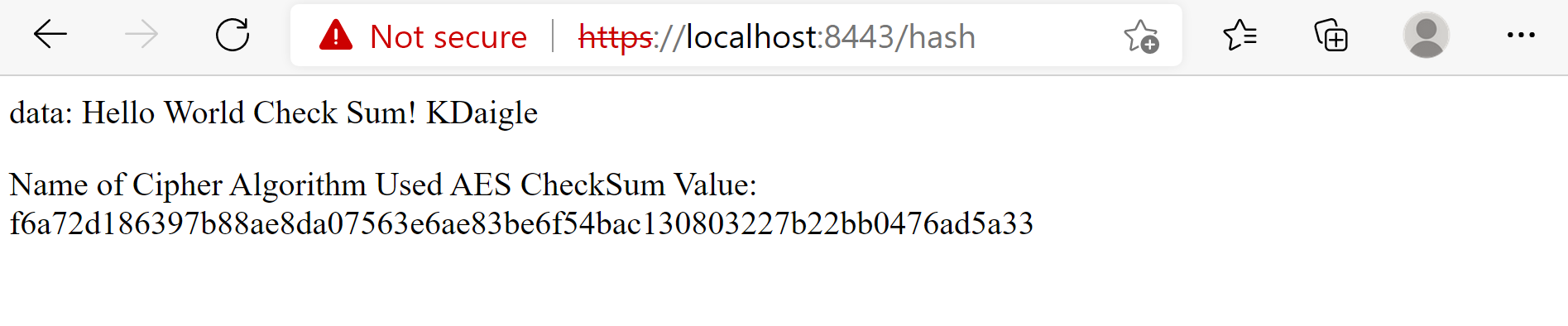
Refactor the code and use security libraries to deploy and implement the encryption algorithm cipher to the software application. Verify this additional functionality with a checksum.

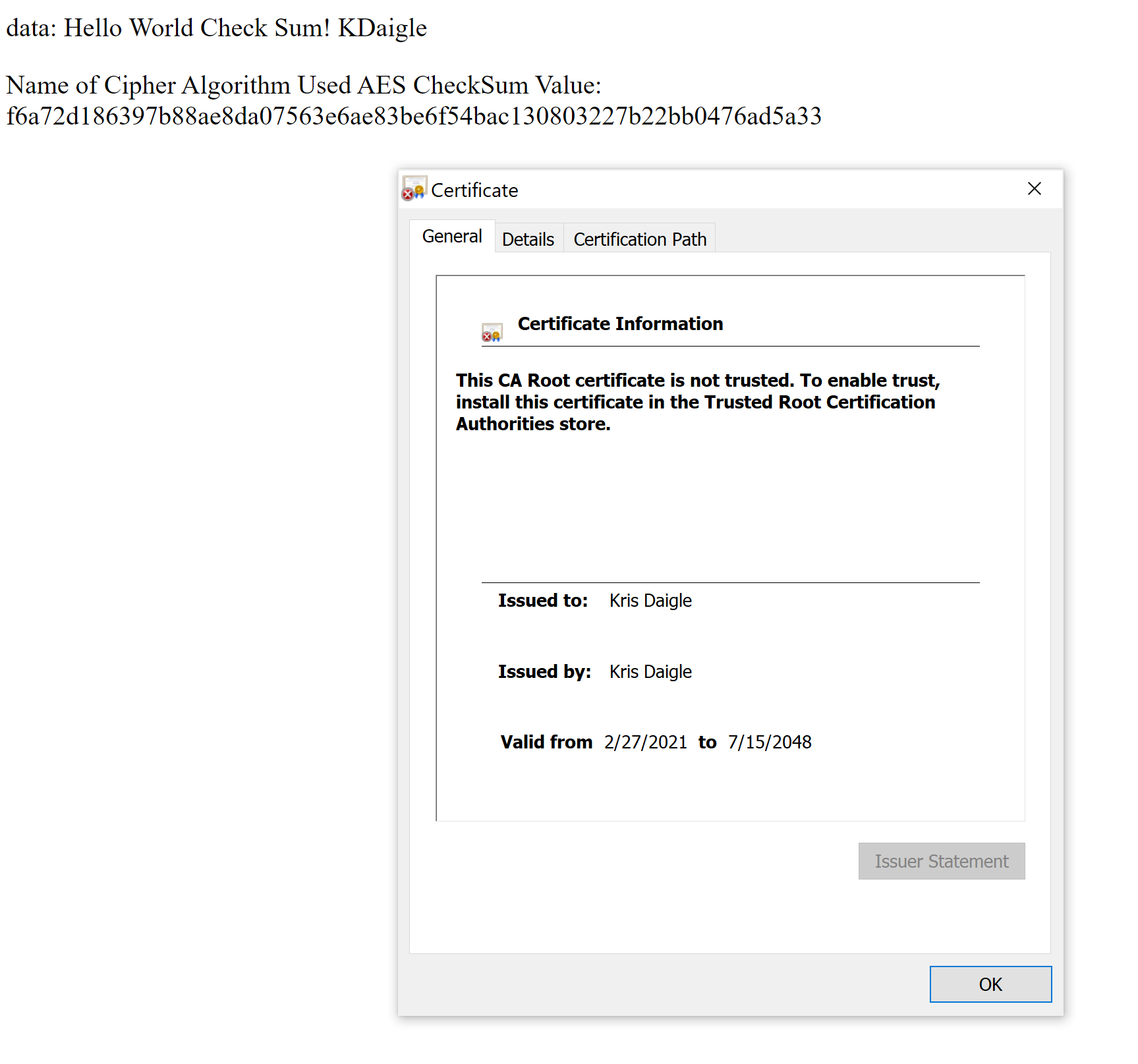


## 4. Secure Communications

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.

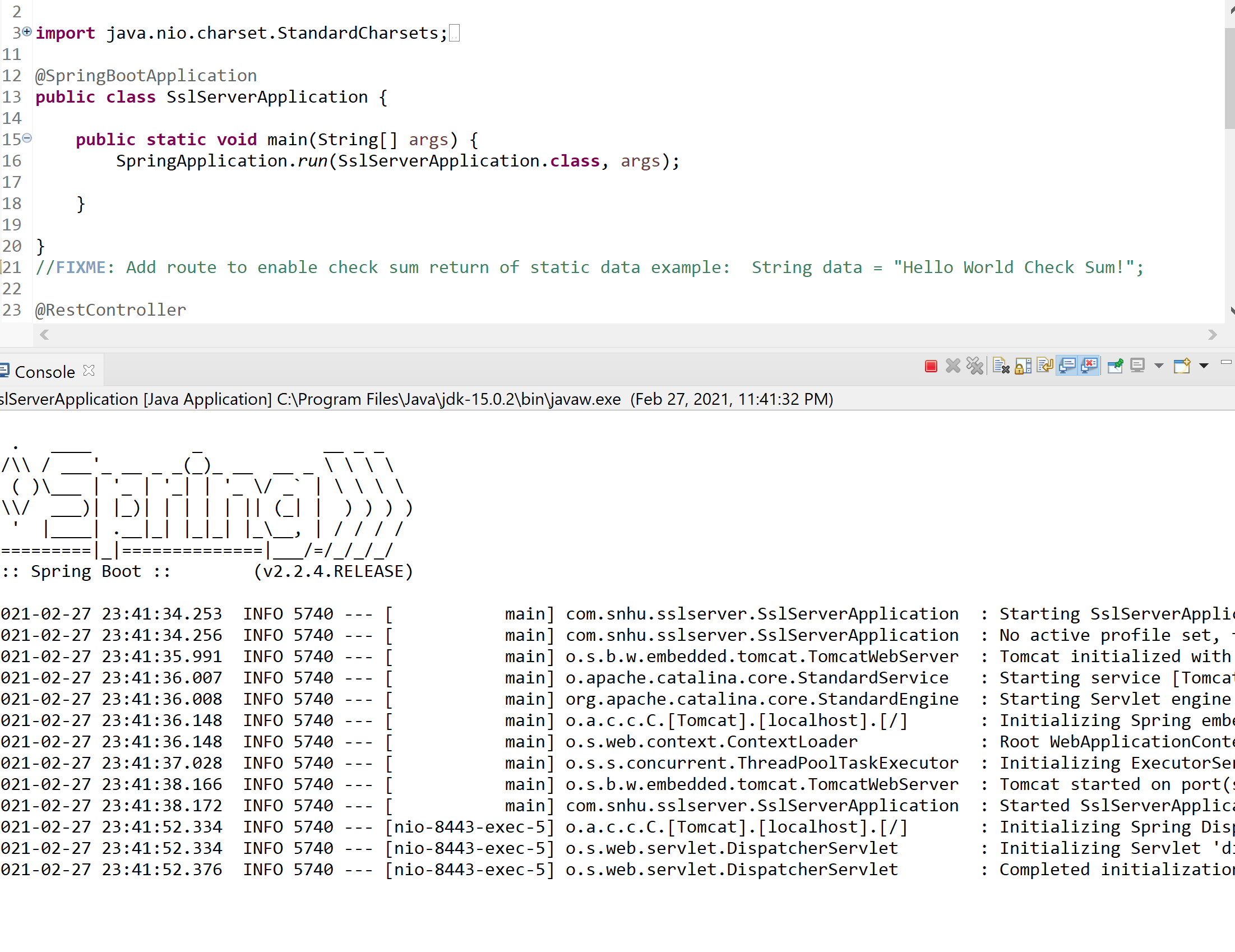




## 5. Secondary Testing

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

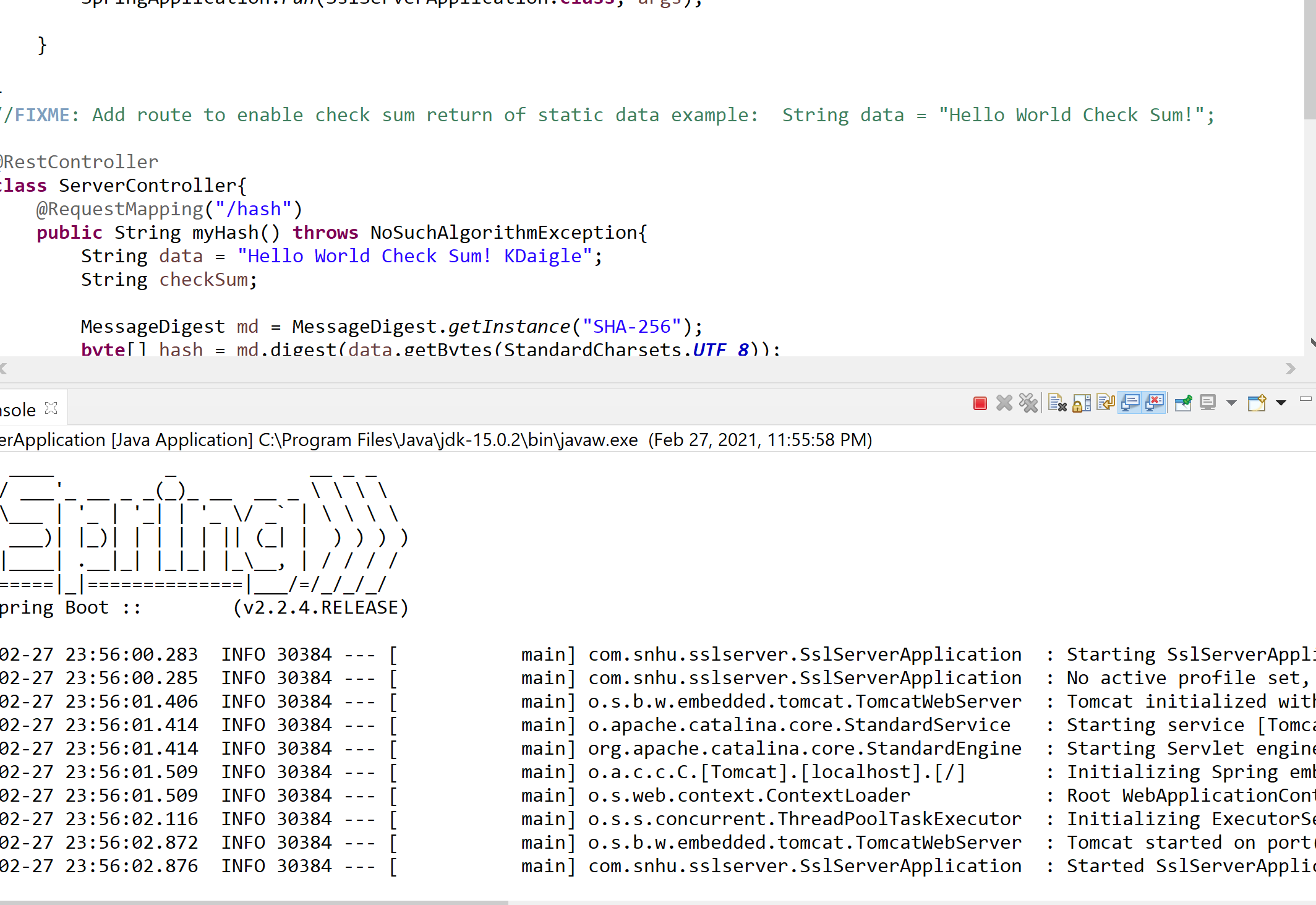




## 6. Functional Testing

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.

Refactoring this code addressed the areas of API, cryptography, client/server, and code quality. We were able to create our own self-signed certificate and created keys used for this application which connect with 256-bit AES encryption. This ensures that all of the sensitive information that the users have stored within the application will be safe.

Adding these security measures is valuable to Artemis Financial, as this will maintain the user’s trust, and keep their business running. Security breaches will break the faith of the users and will lead to losses in business. Maintaining security is an excellent preventative measure to avoid such things from occurring.

Best practices for maintaining the current security are to update the certificate as needed based upon expiration, and checking for vulnerabilities regularly (especially after updates or changes). Moving forward, it’s also good practice to suppress false positives within the dependency report to minimize what is being handled while checking for vulnerabilities. Lastly, it’s important to check for updates in dependent software to ensure that the program is up-to-date and running effectively.