****

# 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** | **8/14/2022** | **Noah Coleman** |  |

## 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

Noah Coleman

## 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.

For this project I recommend SHA-256 cipher algorithm. This algorithm creates keys using 256 bits, making it incredibly difficult to infiltrate. The algorithm uses a checksum to check the validity of the file using which is non-reversible.

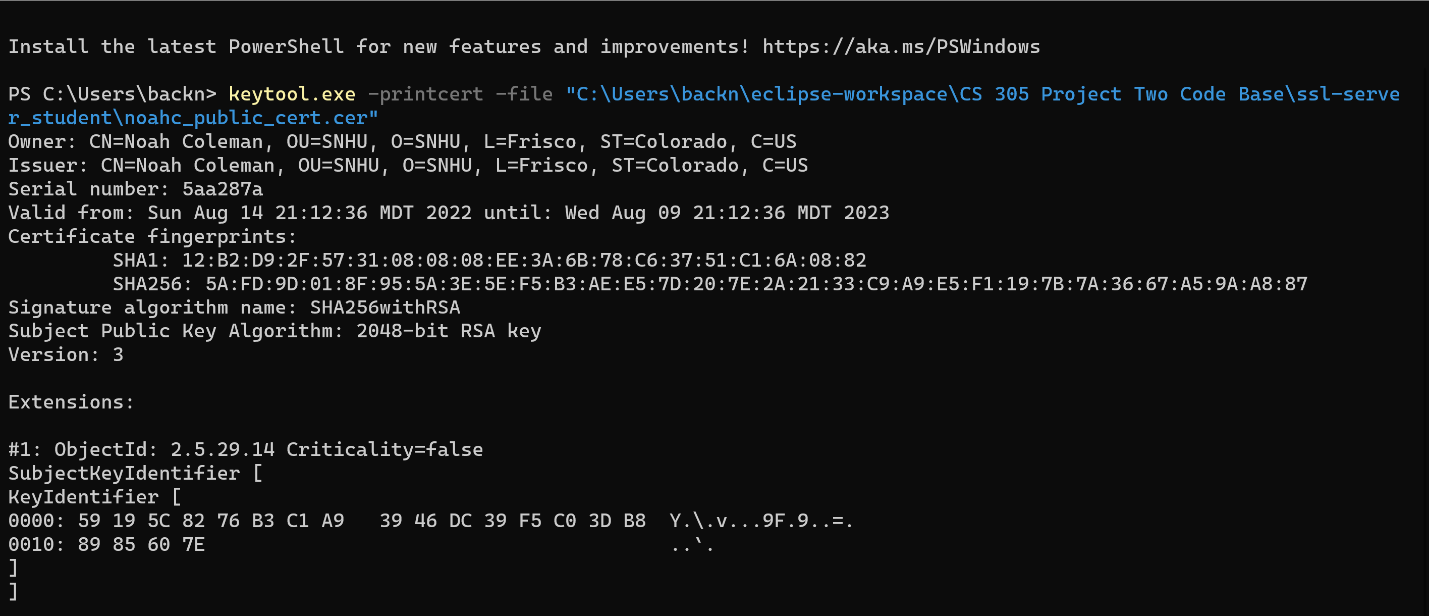
To keep information secure algorithms use keys to encrypt and decrypt data. These keys are protected using hash functions and bit levels. Hash functions are important as they return data when given a key, but this key is accessed using some equation. If the mathematics level is high enough this equation can be difficult or impossible to guess, meaning that the data saved in the hash table is only available if the key is known. Bit levels are important as the larger the key, the harder it is to guess. A 256 bit key will take more time to parse but be more secure than a 128 bit key.

Two kinds of keys are symmetric and asymmetric. Symmetric keys are when “the same key is used for both encrypting and decrypting the data. That key must remain secret for the system to remain secure” (Manico 6). These keys are best for internal systems where trusted users are accessing information within an organization. Asymmetric keys contain a public and a private key. “Data encrypted with the public key can only be decrypted by the private key. These algorithms can be used for both encryption and signing” (Manico 6). This is more useful for a financial system where the public needs to take financial request and the company needs to verify they are coming from the correct user.

## 2. Certificate Generation

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

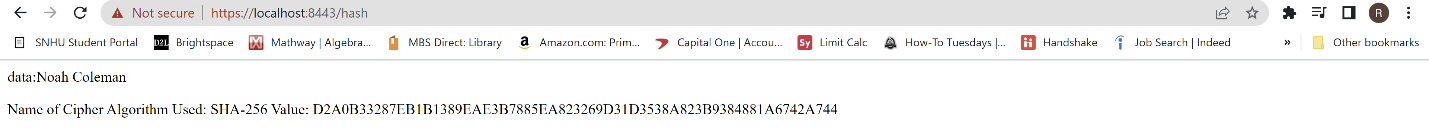
* To demonstrate that the keys were effectively generated, export your certificates (CER file) and submit a screenshot of the CER file below.



## 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.

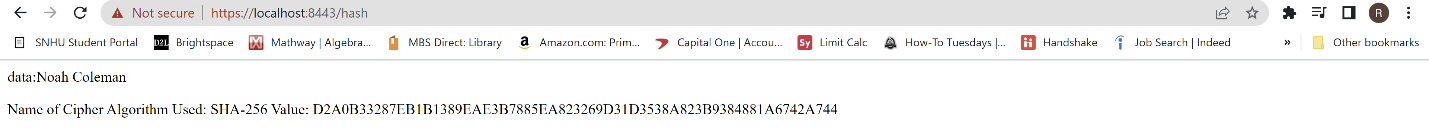
* Insert a screenshot below of the checksum verification. The screenshot must show your name and a unique data string that has been created.



## 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.



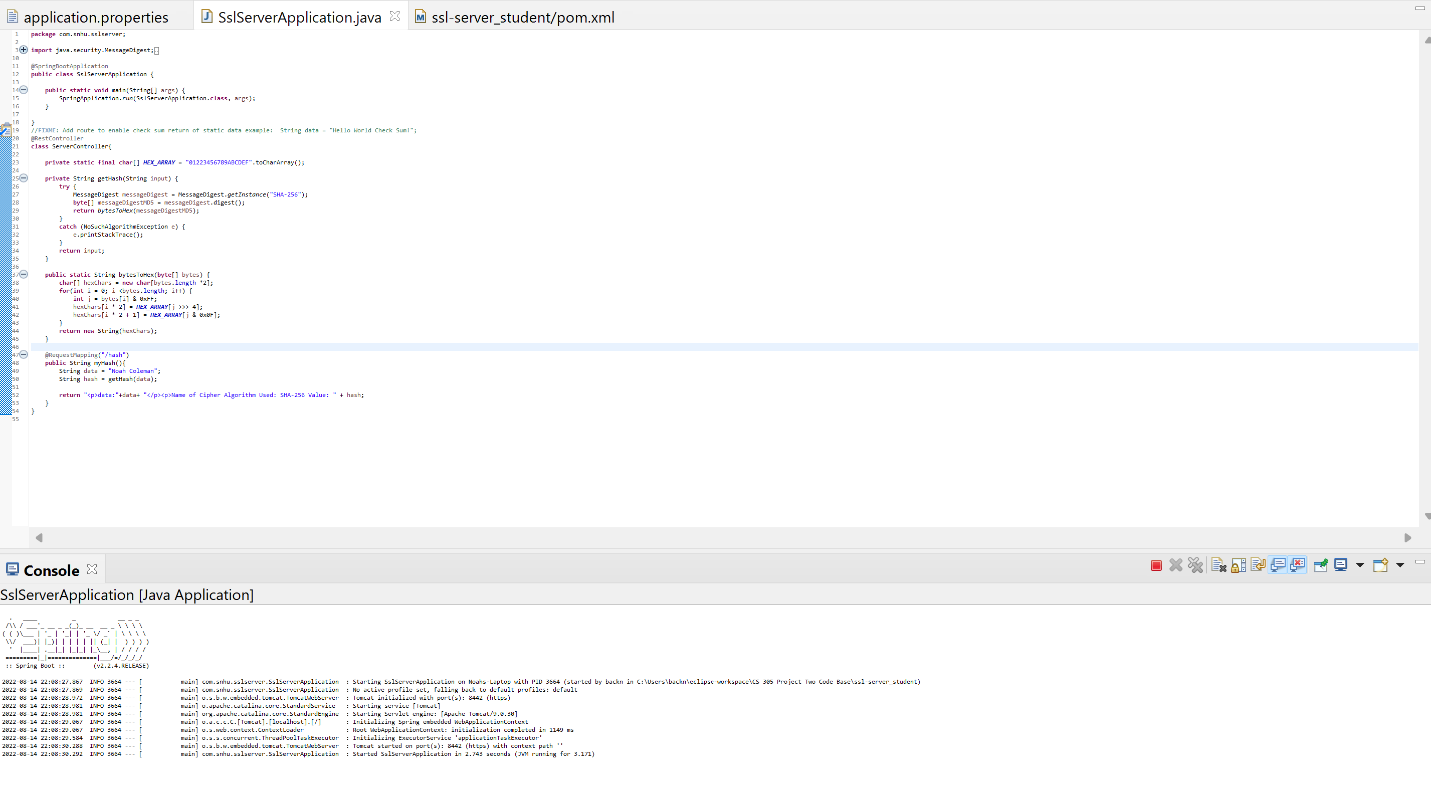
Graphical user interface, text, application, email

Description automatically generated

## 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



Graphical user interface, application

Description automatically generated

## 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.

Graphical user interface, text, application

Description automatically generated

## 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.

In refactoring the code, I added and refactored the RestController class as well as the ServerController class. These classes create a checksum using the SHA-256 algorithm. This addresses the areas of cryptography, code error, and code quality. By making sure the code is running well without errors, we know the security can be trusted. Adding security to the software application adds to the company’s wellbeing by creating trust. If the customers know the company is trustworthy, they will bank with them and give them more of their money to invest. Without security people would bank elsewhere. For maintaining the current security, we should run the maven dependency checks often, and make sure new vulnerabilities are removed or altered in the code base.

References

Manico, J., & Detlefsen, A. (2015). *Iron-clad java building secure web applications*. McGraw-Hill Education.