****

# 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** | **2/25/2024** | **Rogan Page** |  |

## 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

Rogan Page

## 1. Algorithm Cipher

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

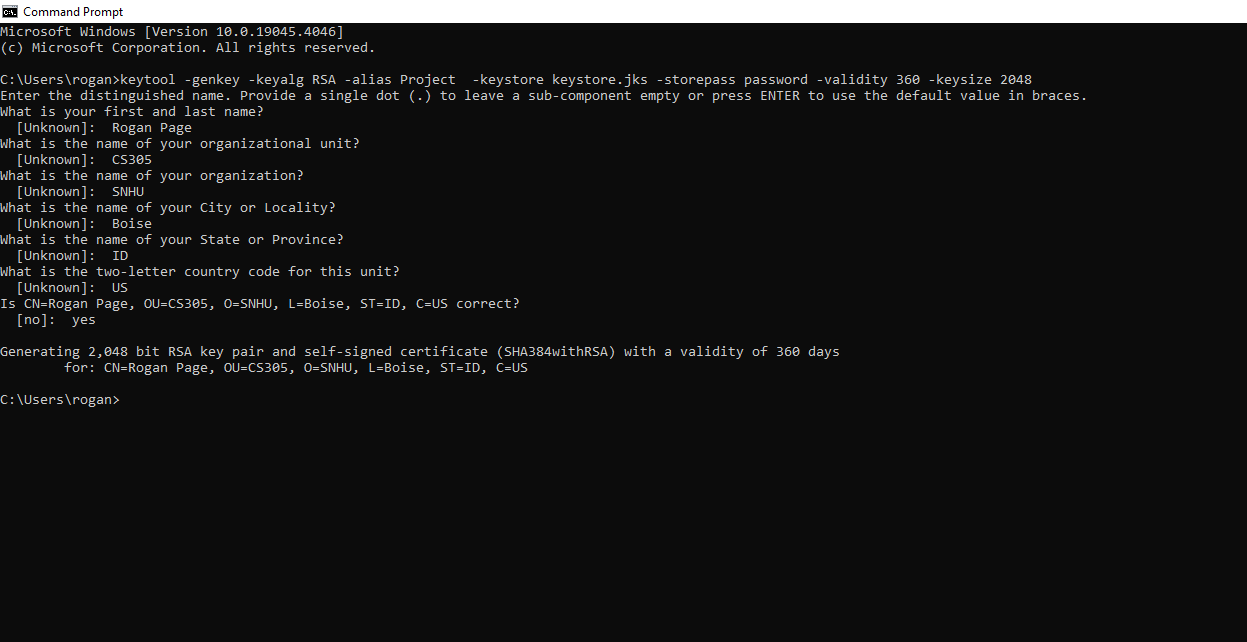
* Provide a brief, high-level overview of the encryption algorithm cipher.
* Discuss the hash functions and bit levels of the cipher.
* Explain the use of random numbers, symmetric vs non-symmetric keys, and so on.
* Describe the history and current state of encryption algorithms.

Artemis Financial seeks to fortify its web application's security by implementing an additional layer of protection to safeguard communications. Given the paramount concern of potential bad actors attempting to illicitly access sensitive financial information, encryption emerges as the foremost solution. Specifically, employing asymmetric encryption would ensure heightened security, wherein the encryption key is public while the decryption key remains private. To achieve the utmost security, especially for external transmissions, the recommendation is to utilize the SHA-256 cipher algorithm with 256-bit keys. SHA-256 encryption offers a robust defense with an extensive array of possible key combinations due to its 256-bit length. Furthermore, leveraging Java's random number generator enhances security by generating a non-reversible checksum using the SHA-256 algorithm, thereby verifying the integrity of transmitted files.

## 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.

A white background with a black border

Description automatically generated

## 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.

* Showing the HTTPS is working but that my Cert isn’t official because it’s self signed.

A white background with a black border

Description automatically generated

A screenshot of a certificate

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

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

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

A screenshot of a computer code

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 my code refactoring, I've introduced a secured RestController to serve as the secure endpoint for hashing operations in my application, aligning with the requirements outlined in the vulnerability assessment diagram. Opting for the SHA-256 hashing algorithm adds an extra layer of security due to its robustness and minimal likelihood of collisions. To uphold the current security standards of the application, I recommend conducting dependency checks on a bi-monthly basis. This proactive approach helps to promptly address any potential vulnerabilities, thereby safeguarding the company's sensitive data. Moreover, retaining the plugins within the pom.xml file ensures that the latest versions of these plugins are utilized, further enhancing the overall security posture of the application.