****

# 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** | **{Date}** | **Arison O’Hara-Hulett** |  |

## 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

Arison O’Hara-Hulett

## 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.

The encryption algorithm selected was SHA. SHA is a one way encryption algorithm this is purposeful when comparing String outputs, something like having to see if one photo is really another photo, or if that photo has been digitally altered in someone. This would be useful in NFT’s which are a new way to store a digital collectible. There may be a billion photos or digital renderings of an image. But there are a few that hold the SHA hash that would match the limited collectors edition hash. Symmetric keys vs asymmetric keys is like making copies of a key and giving it out. Symmetric keys are all the same, for in and out things like AES which usually measured in time for its de-obfuscation implement Symmetric keys. Messaging platforms or emails may use asymmetric keys so that senders can encrypt and receivers unencrypted using their secret key things like RSA may use asymmetric keys.

Encryption algorithms date all the way back to ancient Egypt when crude keys were made to unlock secret secret passages that lead further in to the pharaoh’s temples, or quarters. They have then be refined and rediscovered through the ages. Passing a note with a call-sign or by recognizable figure heads, faces symbols, or locations has also updated through the ages. Espionage or spies, royal messengers, guards, have all played their part in developing and discovering the intricacies of the key and their complex algorithms to safeguard our personal or professional data. There are numerous ways to encrypt data, and a number of those ways are algorithms that have been developed to keep curious eyes from peeking at this data. The algorithms that have been written have created a layer of security that can now be measured in time. Some algorithms have been cracked and some have yet to be broken.

## 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.

[Insert screenshot(s) here.]

## 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.

[Insert screenshot(s) here.] Graphical user interface, text, application

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.

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

[Insert screenshot(s) here.]

Graphical user interface, text, application

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

Text

Description automatically generated

[Insert screenshots here.] Graphical user interface, text, application, Teams

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.

[Insert screenshot(s) here.] Table

Description automatically generated with low confidence

## 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 was made to create a new class with two annotations @RestController, and @RequestMapping the RequestMapping was not utilized and the source can be found at

https://localhost.8443/profile. I also implemented a static checksum method that converted key to a hex string for printing. I annotated where I found my byteToHex function as this was the first time having to deal with a bitwise operator on my own. Which I now know a lot more about. I was able to get a running program with no added vulnerabilities to the dependency check as well. In adding to the code the Client / Server connection has been enhanced, utilizing quality code was implemented by using a reputable source from springboot and the @REST annotations ensure quality connection and modular quality code is in use, this also marks the Encapsulation box off as well. Cryptography can be said to be in use by implementing the SHA casing function. This project brings a secure connection and foundation of a great server for the company. By utilizing springboot as the platform java in conjunction of its robust and well documented libraries allow easy to use secure and reliable code. This ensures their data is protected and ready to handle numerous checks & will make it enjoyable and easy to use while staying secure... for now.