****

# 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** | **12/9/2021** | **Anthony See** | **Security Checks**  **And refactoring** |

## 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

Anthony See

## 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.

AES, or Advanced Encryption Standard, is the recommended algorithm cipher to use for encryption.

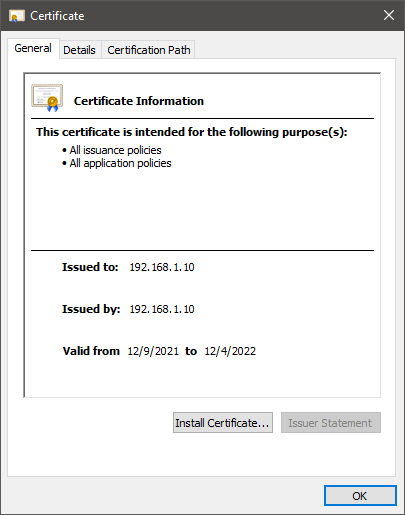
This method is common among the industry and is used in most places, including in the government. AES has multiple different ways to encrypt data, 128-bit, 196-bit, and 256-bit keys. For reference, if using a 256-bit key, it would take leagues longer than the universe has been alive to brute force the key. Each bit level uses multiple rounds to encrypt the data farther, so even if an attacker got a key, they would have to decrypt the right data at the right round. 256-bit AES uses 14 different rounds to encrypt the data, so the attacker would technically need 14 different keys to get the data.

AES was not always the standard for encryption. DES, or Data Encryption Standard, was used. DES only had a 56-bit key, which could be easily brute forced. In some cases over its time of being used, it was. AES was created by two Belgian cryptographers and was then adopted by many different organizations, companies, and governments. DES was quickly phased out in favor of the new, more secure encryption. Because of the power and security behind it, AES-256 would be ideal for Artemis.

## 2. Certificate Generation

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

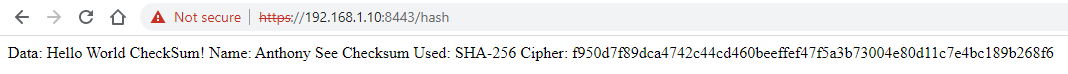
* To demonstrate that the keys were effectively generated, export your certificates (CER file) and submit a screenshot of the CER file below.



## 3. Deploy Cipher

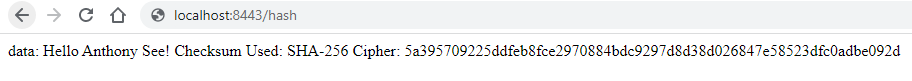
Refactor the code and use security libraries to deploy and implement the encryption algorithm cipher to the software application. Verify this additional functionality with a checksum.

* 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.

## 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, text, application, email

Description automatically generated

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

Text

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.

The main security feature that was added was a self-assigned certificate. This allowed HTTPS to be used with the server securely. It did have troubles with certain browsers because of the naming and it being self-assigned but it still allows HTTPS. A dependency check was, also, done to check for any vulnerabilities in the system. Springwork was updated as well, as it was several versions behind in the given system.

The first step in making sure all the vulnerabilities are patched would be to update the all the libraries that have issues. After this would be creating the certificate so our application can use HTTPS functions. The certificate was created using a keytool that java provides. Then the code was made to allow SHA-256 keys to be created within the application. After that, another dependency check would be done to make sure all vulnerabilities were patched.

A good practice for maintaining security of an application is to make sure everything is up to date. If the owners of a said library found a vulnerability within the library, then it is recommended to update as well. If said update creates an issue, then it would be best to downgrade the version, as a lot of libraries introduce and fix issues in the same updates. Another practice to consider would be to change key algorithms or refractor the code if the program is leaked. This can be mitigated if the company, Artemis, keeps track of who has access to their program.