****

# 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** | **4-16-2022** | **Gavin Scott** |  |

## 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

Gavin Scott

## 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 application, I would recommend the AES algorithm cipher. This cipher is FIPS approved, and it supports keys up to 256 bits. It is one of the most commonly used open-source solutions, making it tried and true. This cipher runs in symmetric encryption, meaning it only has one key that encrypts and decrypts. One of the downsides to this software is that it can be more difficult to implement, and may ultimately impact performance and the production process. While bigger keys mean usually equate to more security, they also equate to a higher computation time.

Hash functions are used to encrypt plaintext. If given a message, it is easy to create the hash, while if given the hash, it is extremely difficult to generate the message. These are often used in conjunction with signatures as the signatures will show if the hash has been tampered with and if the message has been altered.

A symmetric encryption involves the use of only one key, a private key for both the decryption and encryption of data. A non-symmetric encryption uses two keys, the public and private key. Random numbers play a key role in key generation, making it difficult for a hacker to figure out the key.

Encryption goes as far back as the spartan empire through using a leather strap wrapped around a rod. The first computer-based encryption was used in the early 70s by IBM, which was the DES and it remained in use until being cracked in 1997. In 1976, Whitfield Diffie and Martin Hellman came up with the concept of using a pair of keys to encrypt and decrypt. In 2000, AES replaces the system Diffie and Hellman created in favor of single key encryption.

## 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.

Text

Description automatically generated

## 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.

Graphical user interface, application, Word

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.

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

Graphical user interface, text, application

Description automatically generated

While new dependencies were introduced at first, updating the spring boot version removed most of the dependences.

## 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.

A screenshot of a computer

Description automatically generated with medium 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 each section, we added layers of security, such as refactoring the code to include a checksum validation and the use of https protocol using self-signed certificates. The package versions were also updated in order to comply with current security standards.

In ensuring the proper security measures are taken, the software application needs must be identified first. This will be unique to each situation, and the security focuses will shift project to project depending on their needs. The dependency check is also important in understanding possible vulnerabilities and how the project depends on each package included.