****

# 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** |
| --- | --- | --- | --- |
| **1.5** | **18 February 2022** | **Oleo Luberis** | **Document security of application** |

## 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

Oleo Luberis

## 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.

As a developer, I would choose AES (Advanced Encryption Standard) encryption. The fact the government made AES the standard in 2002, shows it works. The government left DES (Data Encryption Standard) because the 56 bits were small and allowed for brute force attacks. AES encryption comes in three main flavors, 128, 192 and 256. At AES 128 the key size is 10. At AES 192 key size is 12. Lastly AES 256 has 14 keys. These keys are used to transform the AES cipher making the bits harder to crack. The more the keys the harder the encryption is to crack.

The hash function is used to check data integrity. If the hash is changed in any way by even 1 bit, the whole message will produce a different hash. This is a quick and easy way to see if the file has been tampered with.

We use random numbers to confuse hackers, as they have no way to find out if the information, they are trying to steal is real or random information. We use random numbers to generate information that will throw off any hacks as they are receiving garbage. Symmetric keys are using one key to encrypt and decrypt a message. If both party’s keep this key secure, this works well. If the key is stolen, all messages are a free for all to all interceptors. Non-symmetric keys are used, both parties have a public key to scramble messages. Each party has their own key, used for encryption and decryption.

In the past, the United States and other powers had a problem. How to send information across lands without it being intercepted. This is and was not an easy task. If the messenger got caught, the message was found and intercepted. These countries began trying to outsmart each other by ciphering information to make it hard for the other side to understand the information whether I was stolen or not. Sometimes the letters were offset and used to create a new messaging system. Even when in the enemy’s hands the message is still intact and unreadable. Unless the enemy figures out the code. Today we use AES as this is what the government states is the minimum standard. This encryption can last multiple centuries before the key is decrypted. The only thing that can make this useless is if computer sped up in speed and computational power.

## 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, text, application, email

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.

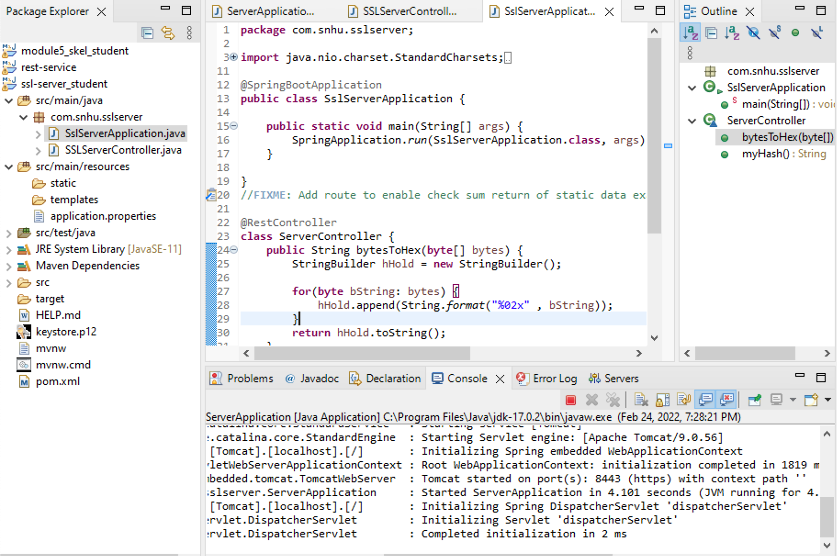
* Insert a screenshot below of the web browser that shows a secure webpage.

Could not get this to work with tomcat. I get the https to show in tomcat, but chrome will not access it.

## 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

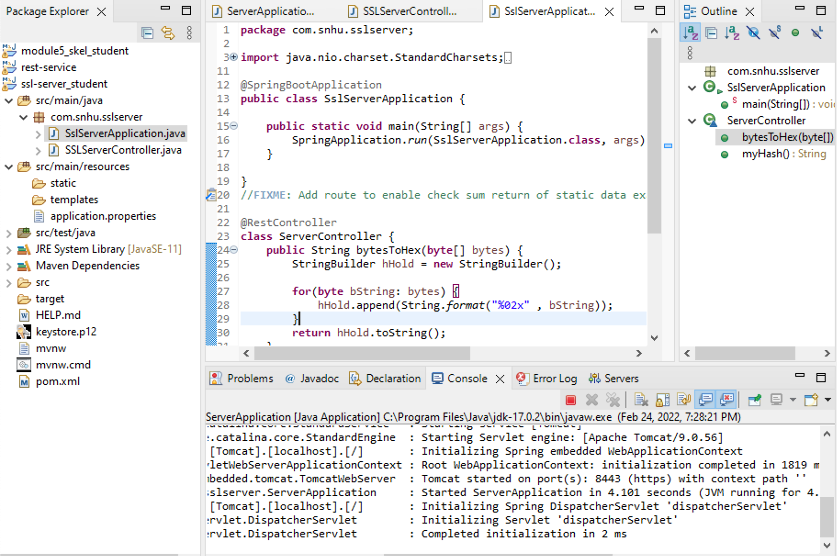




## 6. Functional Testing

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.

`

## 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 this lesson of Refactored code, I learned a lot. Learned to create keystore certificates, setup and use tomcat, setup a server, encrypt my name in 1 and 256 bits. I also learned to use maven for checking dependencies against a dependency site (which checks for new dependencies). One of my hated and became easy task was editing the pom file. Having to check for a start and closed version of the tag your using creates a love to hate problem. One aspect of this training I liked was creating the certificates. I don’t even know how many certificates I created and stored on my computer. I created 128-bit encryption and 256 for https encryptions and used them in this application. If a company does not have a security plan, you would lose business. Your company would also have many lawsuits because you did not provide security for user data and information. Having a good plan and security allows companies to run successful business and continue the flow of information between the user and your company. Companies as of late have been getting hacked left and right. This is because the advancement in computing allows for faster brute force, and employees getting tricked by phishing. Phishing is one of the ways hacker gain access to a company. The employees are either not trained opens emails they should not or send information or send information to the wrong people. Another pain is if you let data from people overseas get hacked and sold. This can create a war scenario.

The dependencies that maven check, will help keep hackers at bay. I believe we should check dependencies regularly to keep our software safe. For businesses that deal in stocks and monies, should use 256 bits encryption as a loss can be catastrophic.