****

# 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** | **8/12/22** | **Jaelyn Sloan** | **Initial release; code files attached** |

## 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

Jaelyn Sloan

## 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 best algorithm cipher for this project would be AES encryption and SHA-256 for message digest and hash values. These algorithm ciphers are very commonly used to ensure secure communication practices between servers. In December 2015, the US Federal Government Department of Defense released a policy to declare SHA-256 as the new standard for hash functions (Staten, 2016). In May 2002, AES was declared the new US Federal standard encryption algorithm (Lake, 2020). SHA-256 is able to create secure communications in digital signature verification, integrity checks, SSL handshake, and password hashing. Collision chances while using SHA-256 are almost impossible due to the hash function of this algorithm cipher.

SHA-256 hash function uses a 256-bit digest, meaning that there are possible combinations, which is why the collision chances are so low. This also means that it would be extremely difficult for a brute force attack to succeed. AES has three different bit lengths: 128-bit, 192-bit, and 256-bit. Using the 256-bit encryption key is much more secure against brute force attacks, however it uses much more processing power and may take longer to execute than 128-bit encryption (Bernstein & Cobb, 2021). In this case, it would be beneficial to use 256-bit encryption because the system is complex and has a large amount of data.

AES uses symmetric keys, also known as a secret key, to encrypt and decrypt data. Symmetric ciphers use the same key for encryption and decryption, so the sender and receiver must both have the same key to access the information (Bernstein & Cobb, 2021). SHA-256 uses cryptographic hashing to scramble data and generate random hash keys, preventing an intruder from easily accessing secure information. It is deterministic, meaning that any change in the input will result in a change of the random hash value, making it much simpler to authenticate files. (Callaghan, 2020).

Before AES became the standard encryption algorithm in the US, the Data Encryption Standard (DES) was the federal standard. From 1974 to 2002, DES was used to secure private data across the US. Eventually we began seeing that there were many flaws to DES which allowed intruders to crack the encryption code and access private information (Thakkar, 2021). Since then, we have upgraded to using AES as the standard for encryption algorithms and SHA-256 as the standard for hash functions.

## 2. Certificate Generation

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

Text

Description automatically generated

Graphical user interface, text, application, email

Description automatically generatedGraphical user interface, text, application

Description automatically generated

## 3. Deploy Cipher

Graphical user interface, application, Word

Description automatically generatedRefactor 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.

## 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, application, Word

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.

Dependency check before:

Graphical user interface, text, application, email

Description automatically generated

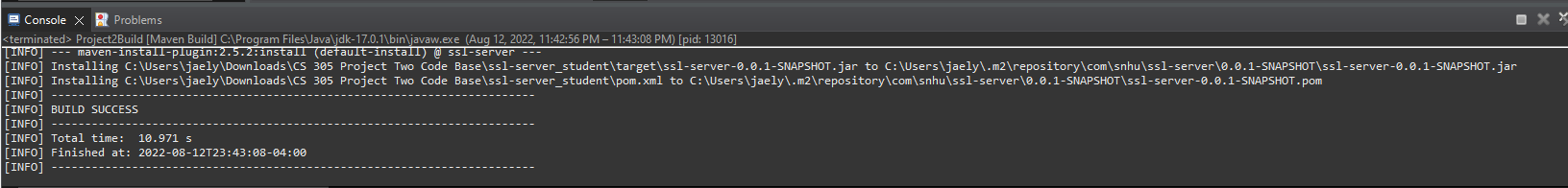
Updated pom.xml file

Text

Description automatically generated

Text

Description automatically generated



Refactored code:

Text

Description automatically generated

Graphical user interface

Description automatically generated

Dependency check after:

Graphical user interface, text, application, email

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.

After manually reviewing my code, I found that there were a few places that needed in-line comments to make my code more understandable and readable by other developers. I added comments on line 20, 24, and 46 as shown below:







I also added more detailed notes in the comments that were already present to make it easier to understand. To mitigate this vulnerability in the future, I suggest practicing writing in-line comments while writing the code to ensure that I am explaining what exactly the code is for instead of going back after the project is completed and adding comments (Sloan, 2022).

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 first area of security that I addressed when refactoring the code was cryptography. I was able to implement AES encryption and SHA-256 hash functions to securely encrypt our program data. The API was made secure by developing and using a RESTful application. The client/server aspect is made secure by creating and using an SSL certificate for our application.

I initially developed the code to create a RESTful application using SHA-256 to generate hash keys which keep the data secure. Then, it was necessary to create an SSL certificate to authenticate our RESTful application for the client/server. Using the SSL layer is a very efficient and secure way to encrypt and protect data in our system. This increases the customer satisfaction with our program, as well as the reputation of the company.

The best way to maintain the security of our software application is to regularly perform a dependency check to ensure that new vulnerabilities do not emerge and are corrected as soon as possible. It would also be beneficial to regularly update all plugins and packages to reduce the number of vulnerabilities that could occur.

References

Bernstein, C., & Cobb, M. (2021, September 24). *Advanced Encryption Standard (AES)*. SearchSecurity. <https://www.techtarget.com/searchsecurity/definition/Advanced-Encryption-Standard>

Callaghan, P. (2020, August 19). *Why You Should Use SHA-256 in Evidence Authentication*. Pagefreezer. <https://blog.pagefreezer.com/sha-256-benefits-evidence-authentication>

Lake, J. (2020, February 17). *What is AES encryption and how does it work?* Comparitech. <https://www.comparitech.com/blog/information-security/what-is-aes-encryption/>

Staten, R. (n.d.). *DoD SHA-256.* CHIPS. <https://www.doncio.navy.mil/chips/ArticleDetails.aspx?ID=7375>

Thakkar, M. (2021, December 15). *What Is DES Encryption? A Look at the DES Algorithm*. InfoSec Insights. <https://sectigostore.com/blog/what-is-des-encryption-a-look-at-the-des-algorithm/>