****

# 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** | **08/19/2021** | **Sung Kim** |  |

## 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

Sung Kim

## 1. Algorithm Cipher

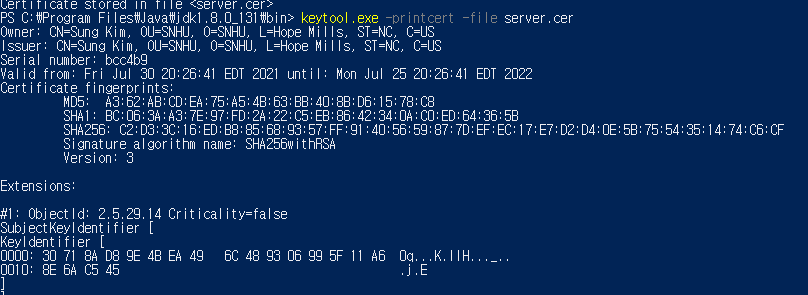
Artemis Financial has requested the use of an algorithmic cipher that complies with modern security standards. In as much, I highly recommend the AES-256 algorithmic cipher. This cipher uses 6 basic operations in 14 rounds to produce a unique hash code. AES has several features that make it a very powerful tool in hindering malicious actors.

AES-256 utilizes a symmetric key, meaning both the encryption and decryption uses the same key. It utilizes 6 basic operations in 14 rounds to obfuscate the original message. To decode it, the same rounds and operations occur in reverse. Each of these rounds produce a round key, which is subsequently used in the next generation of rounds. The code is first converted into bit code, which then is arranged in a 4 x 4 grid. Basic operations occur on the bit grid, such as shifts, additions, xor, substitutions, and rearrangements, and mixing. It uses a substitution hash table, which adds to the complexity of the output code.

AES-256 is a powerful tool in that it can significantly lengthen the time it takes to brute force a key. Since it is a 256-bit key, there are 2256 different combinations of the key, which will take a computer approximately 2.29 x 1032 years to brute force. This is significantly longer than anyone has time to wait for. The NIST has also stated that it is their recommendation even in a post-quantum computing society.

## 2. Certificate Generation

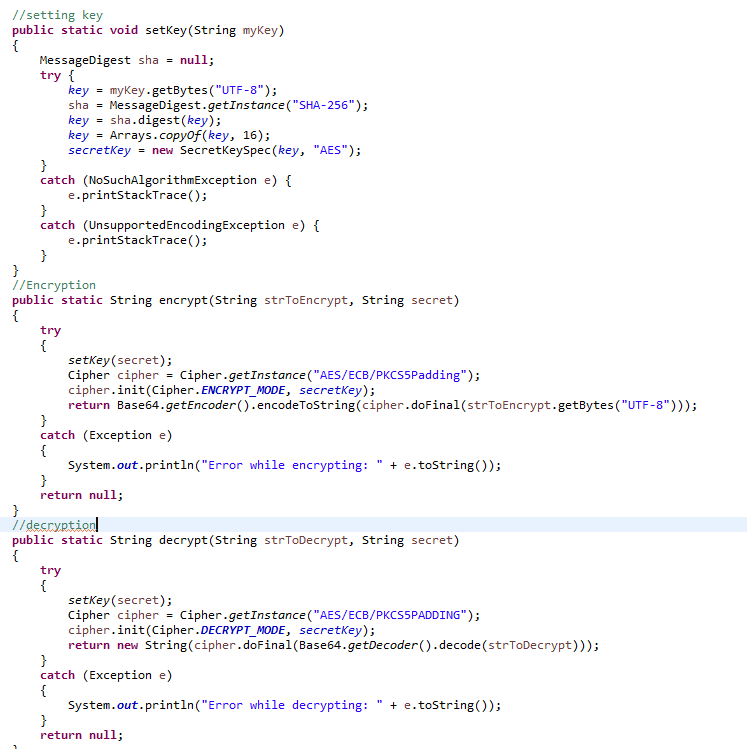
Generate appropriate self-signed certificates using the Java Keytool, which is used through the command line.



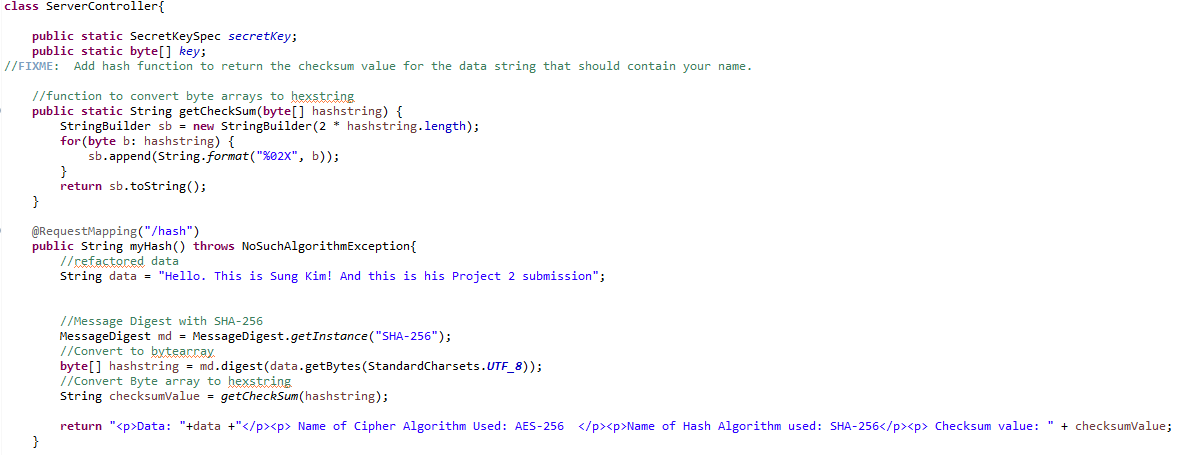
## 3. Deploy Cipher

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.

**CODE:**

**AES-256**

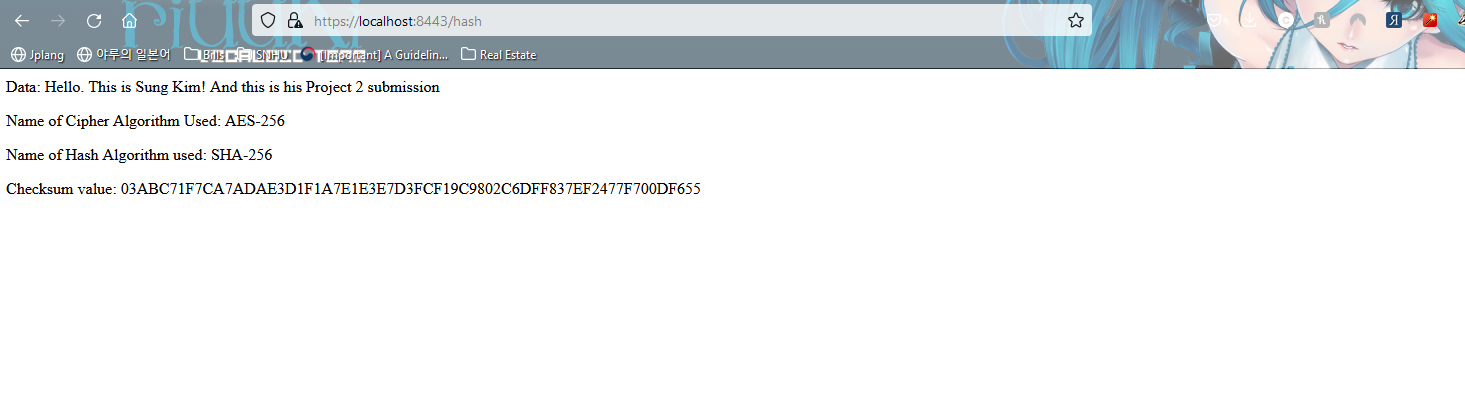
**HASH Function**:



**Evidence of encryption and verification using hash function:**

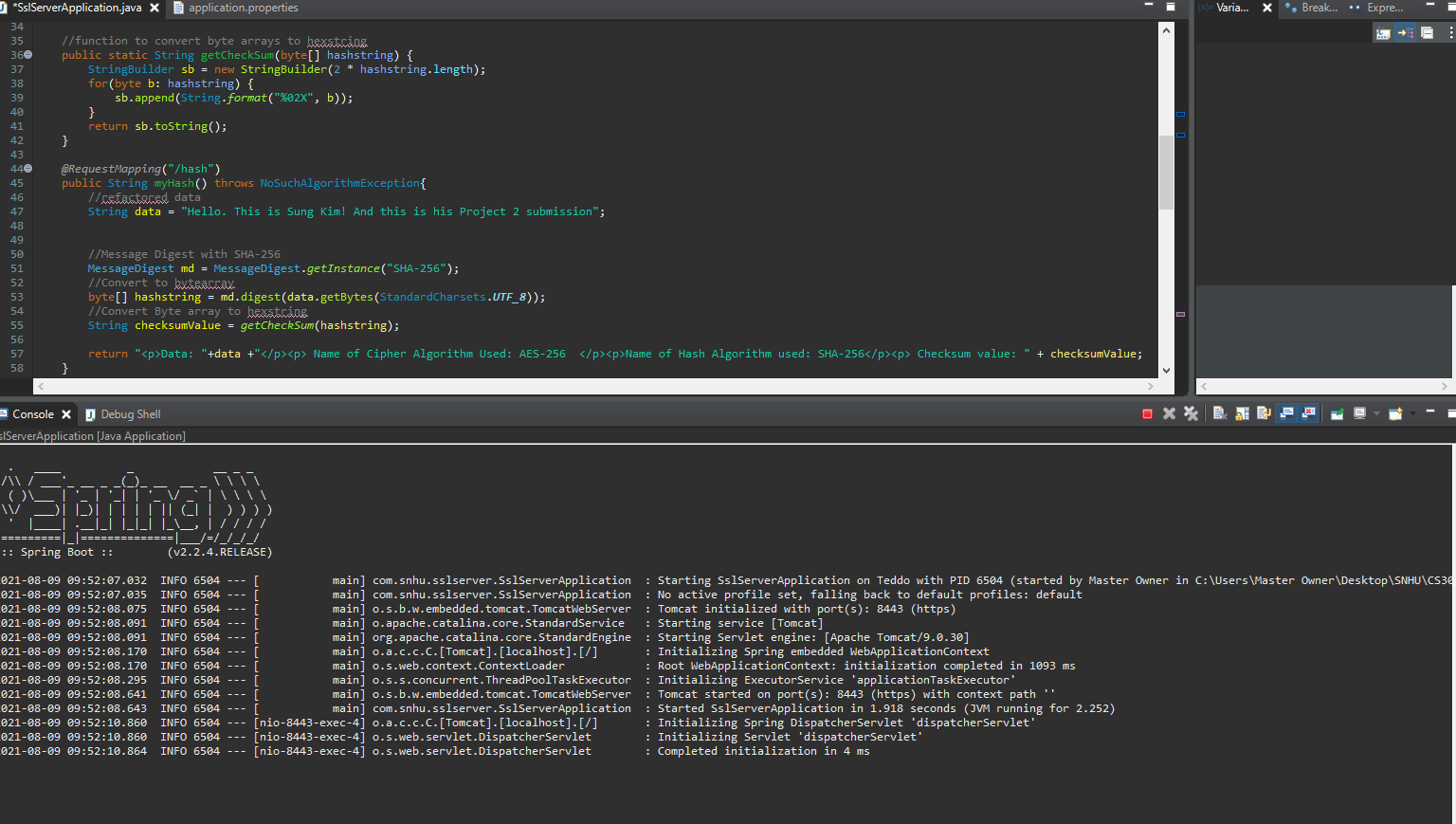
## 4. Secure Communications

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.

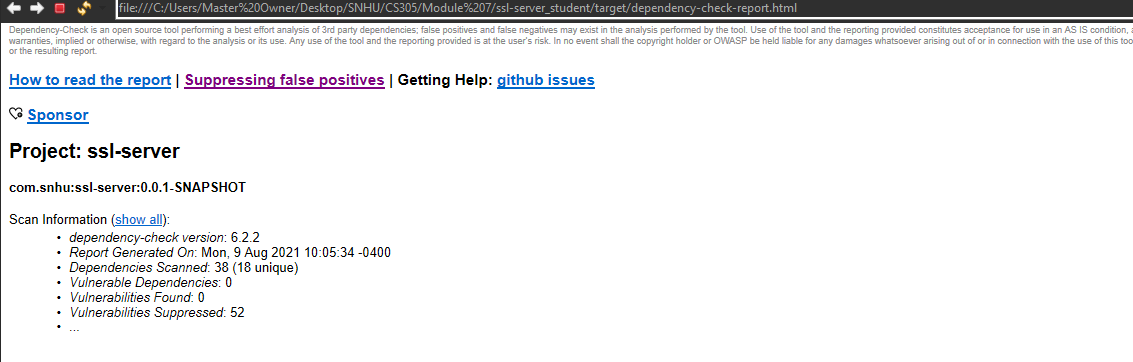


## 5. Secondary Testing

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.

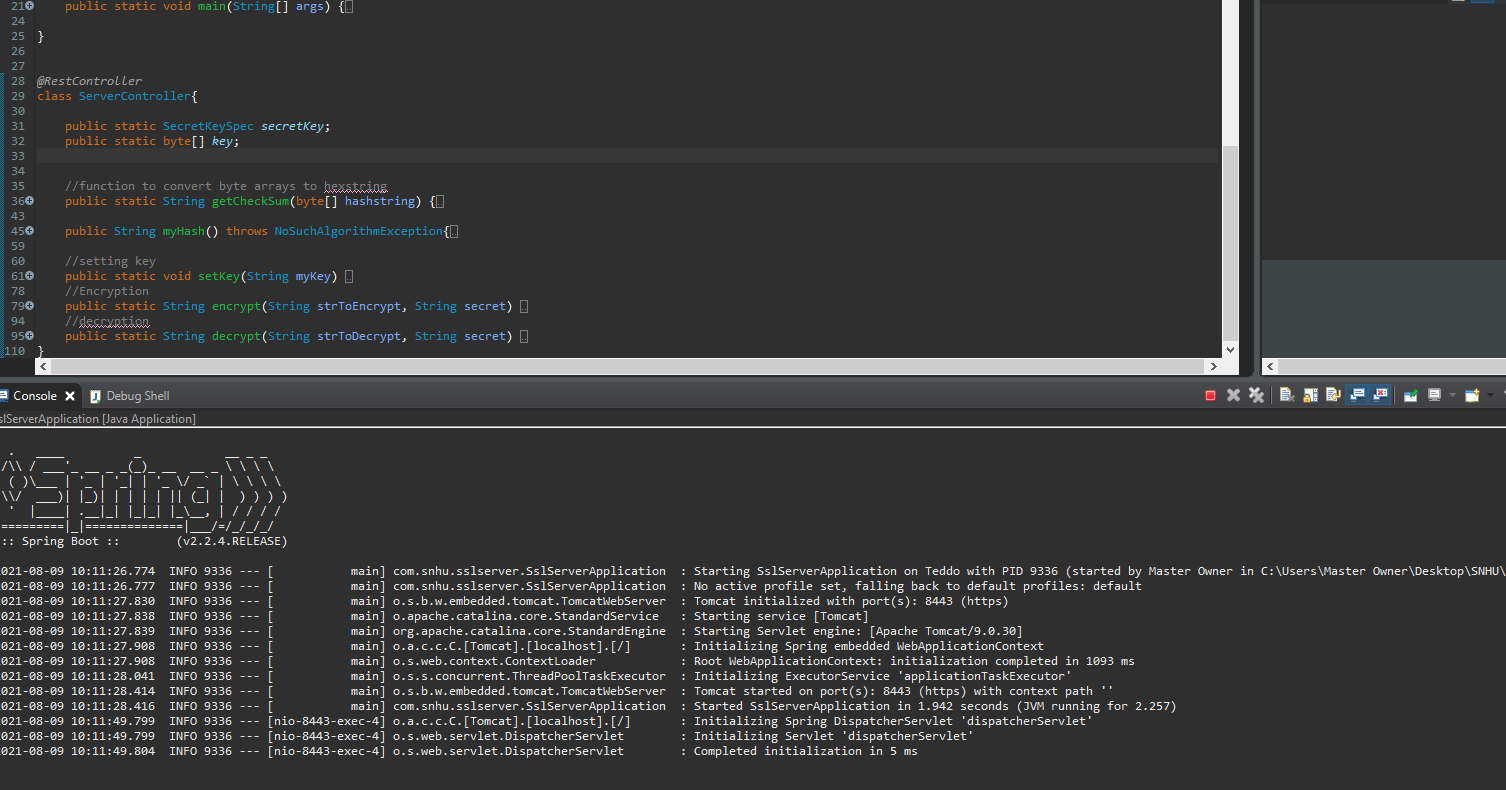


**Dependency Check Report with False Positives and Minor Known issues removed. 52 vulnerabilities suppressed.**



## 6. Functional Testing

Identify syntactical, logical, and security vulnerabilities for the software application by manually reviewing code.



## 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.

Input validation: There was no user generated input required for this project. If user input was required, a simple input limiter with restricted character types can be utilized by using a compare function with regex and string length.

APIs: All APIs are up to date being forcefully updated to a newer version. This is done through updating the version number in the pom.xml file provided in the main folder. We have updated the dependency check from the 5.3 version to the 6.2.2 version.

Cryptography: Utilization of the AES-256 algorithm and SHA hash algorithm was used to validate and verify the correct usage of AES-256. Manual key and RNG key generation are both provided for convenience.

Client/Server: Tomcat utilizes configurations that do generate false positives. The method of access is through SSL, which require certifications located in the keystore P.12 of type PKCS12. It also utilizes port 8443, which is not the main listening port of 8080 (default).

Code Error: Several Try/catch statements were utilized so that if an error is caught the program does not crash. This can be very disastrous if a financial institution goes offline from a simple flaw in the code.

Code Quality: We used comments to add to the readability of the code, which decreases the chances of code being flawed due to human oversight. We also added kill lines for every buffer that is created so that we do not risk open buffers that can be exposed to memory attacks. We also standardized the character set so that the code is consistent among different characters from different language modalities.

Encapsulation: The code is separated into different classes, which restrict access to internal data members. However, for the purpose of this project, there are only a few data members that use several external classes from imported libraries. This restricts broken code to a few classes, while the reliable dependencies maintain integrity. They have also been updated to ensure fewer known data breaches.

Software security is essentially the practice of slowing down a malicious threat from gaining unwanted to access. We have implemented encryption to make it harder for a threat to use the data. We have used hashing to validate that the data that we are sending is correct and has not been corrupted during the encryption process. We have also used certificates to limit access and secure the connection between the server and the client. We have refactored the code to optimize for readability, thus decreasing the chance of human error. We have also statically analyzed the code and eliminated the false positives that are generated. We have also updated the software dependencies to ensure that known vulnerabilities have been eliminated.

We recommend that Artemis Financial frequently update their dependencies and do frequent static analysis to make sure that there are no known vulnerabilities. We also recommend that they use containerization, whenever they are updating to ensure a safe rollback without data corruption if that is necessary. It also implements firewall protection in the case of DDoS attacks, which is standard in most cloud container services, such as GCP and AWS. We recommend the use of the vulnerability Flow Diagram to methodically analyze threat opportunities.

References:

National Institute of Standards and Technology, Chen, L., Jordan, S., Liu, Y., Moody, D., Peralta, R., Perlner, R., & Smith-Tone, D. (2016, February). *Report on Post-Quantum Cryptography* (NIST IR 8105). US Department of Commerce. https://csrc.nist.gov/csrc/media/publications/nistir/8105/final/documents/nistir\_8105\_draft.pdf

Oracle. (2017). *Java Security Standard Algorithm Names*. Java Security Standard Algorithm Names. https://docs.oracle.com/javase/9/docs/specs/security/standard-names.html#cipher-algorithm-names