****

# Practices for Secure Software Report

Table of Contents

[Document Revision History 3](#_Toc102040754)

[Client 3](#_Toc102040755)

[Instructions 3](#_Toc102040756)

[Developer 4](#_Toc102040757)

[1. Algorithm Cipher 4](#_Toc102040758)

[2. Certificate Generation 4](#_Toc102040759)

[3. Deploy Cipher 4](#_Toc102040760)

[4. Secure Communications 4](#_Toc102040761)

[5. Secondary Testing 4](#_Toc102040762)

[6. Functional Testing 4](#_Toc102040763)

[7. Summary 4](#_Toc102040764)

[8. Industry Standard Best Practices 4](#_Toc102040765)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **08/10/2023** | **Carlos Stockl** | **Project two** |

## Client



## Instructions

Submit this completed practices for secure software report. Replace the bracketed text with the relevant information. You must document your process for writing secure communications and refactoring code that complies with software security testing protocols.

* Respond to the steps outlined below and include your findings.
* Respond using your own words. You may also choose to include images or supporting materials. If you include them, make certain to insert them in all the relevant locations in the document.
* Refer to the Project Two Guidelines and Rubric for more detailed instructions about each section of the template.

## Developer

[Carlos Stockl]

## Algorithm Cipher

To encrypt long-term archive files, Artemis Financial is requesting an encryption algorithm recommendation. The biggest issue that may come up for these files would be bad actors somehow acquiring access to these files.

The solution for this problem is to encrypt the files so much that they would be useless even if they were somehow stolen. Since the files will not be moved anywhere, there will be no need for Asymmetric\* keys. Also, since these files will be archived for the long term, there is no need for these files to be encrypted quickly.

Because of the reasons listed above, I would recommend using the SHA-256 cipher algorithm with 256-bit keys for the encryption of these files. SHA-256 encryption is the most secure default Java option available because SHA-256 provides the highest level of bitwise encryption. SHA-256 also uses the Symmetrical\* encryption keys. Since Artemis Financial will be the only party accessing these encrypted files, Symmetrical\* keys will work perfectly. The SHA-256 algorithm also uses Java’s random number generation to ensure that every encrypted file is very secure. The use of random numbers allows for the cipher to securely create an Irreversible checksum that verifies the legitimacy of each file.

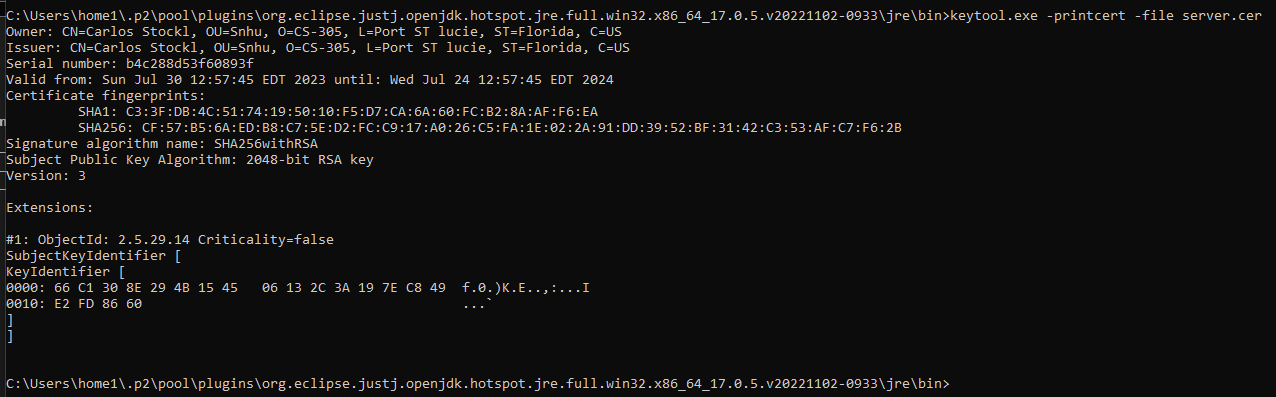
To verify files, the hash function will utilize the SHA-256 cipher to generate a checksum signature for the given message.

Symmetrical encryption keys utilize a single key for encoding and decoding data, while asymmetrical keys involve a public key for encoding and a private key for decoding.

## Certificate Generation

By using the Java Key tool, I generated sufficient personally signed certificates into the command line.

Below is a screenshot demonstrating how my key is generated using the command prompt:



## Deploy Cipher

After modifying the Refactor code in the SslServerApplication.java I added security libraries to deploy and implement the encryption algorithm cipher. I was able to verify the results in checksum, it shows a unique data string created.

Below is a screenshot demonstrating encryption was created successfully, I used internet explorer to browse. The correct path is entered:

A screenshot of a computer

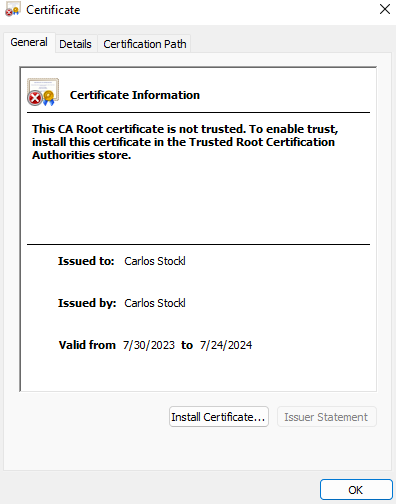
Description automatically generated

## Secure Communications

I was able to convert HTTP to the HTTPS protocol using the new modify Refactor code, In compiling and running the refactored code, I was able to verify secure communication by typing [**https://localhost:8443/hash**](https://localhost:8443/hash) **on the internet explorer browser ,** Below is a screenshot demonstrating the code worked successfully. A screenshot of a computer

Description automatically generated

Below is a screenshot demonstrating a secure connection, it is simply because I am using a self-signed certificate. If you look at the details, I can prove the connection was encrypted via sha256RSA.

 A screenshot of a computer

Description automatically generated

## Secondary Testing

## Below is a screenshot demonstrating that the refactored code executed without errors and the dependency-check report run smoothly.

## A screen shot of a computer program Description automatically generated

## Below is a screenshot using dependency check to refactoring code and works.A screenshot of a computer Description automatically generated

## Functional Testing

Below is a screenshot of my refactored code without errors running smoothly.

A screenshot of a computer program

Description automatically generated

Below is a screenshot of my application properties code without any errors.

A screen shot of a computer

Description automatically generated

Below is a screenshot of my pom.xml file code without any errors. I changed the version from 5.3.0 to 6.3.0 it shows no errors and that it is running smoothly.

A screen shot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

## Summary

For my project, my focus was on cryptography, server/client relationships, and the quality of codes. For code quality, I addressed it by accounting for coding practices, patterns, and security. To secure and protect sensitive data, I implemented a 256-bit algorithm. The script or data for this example was my name; however, for Artemis Financial, the script would be used for financial planning, estimates, important documents, or vital client information. The information would be secured through encryption prior to being stored.

The client/server relationship was addressed in this project through enforcing a TLS connection to the application on the web which secures both the connection and the communication between client and server. Encryption was utilized in both the data and the TLS connection providing a more secure communication. In the end, all codes were checked for vulnerabilities and errors.

An SSL certificate was the first level of protection I inserted for the customer. This certificate guarantees the website's security and lets users know that every interaction on the site is encrypted. This enhances Artemis Financials’ value by fostering user confidence and enhancing the security of data and communications. The SHA-256 encryption technique I introduced served as the second tier of protection. Utilizing TLS to encrypt a communication session is insufficient. The data itself must be encrypted as well. This increases security since if someone were to get access to the server hosting the data, they would have access to extremely private records. You ensure that the data is secure when you If someone were to steal the data from the server, they would be unable to figure out what they had taken. HTTPS enforcement was the final security measure I added to this application. This guarantees that the session cannot be carried out via standard HTTP. To secure both the server and the client, each interaction between the user and the internet site must occur via an encrypted connection. This has added benefit since it prevents older browsers from having the option of a connection that is not adequately protected, which can improve our capacity to thwart things like man-in-the-middle attacks.

## Industry Standard Best Practices

 Regular dependency checks to keep an eye out for the new vulnerabilities that are being identified daily are among the best practices for preserving the security of the software program as it is now configured, secure coding for adding all the other modules required, input validation for any inputs taken from any users, building a strong API that allows specific things to be done but also clarifies for the client the mechanics of it all, and standard code reviews to guarantee that any modifications made are checked for any errors or unexpected interactions with already existing code.

These can be accomplished in a variety of methods and to an extent appropriate to Artemis Financials' needs. For instance, they can update it to RSA-2048 or another standard if necessary if a code review reveals that SHA-256 is no longer a workable security choice in the future. Rather than attempting to blacklist each new danger that surfaces, input validation may be carried out by simply permitting categories of input and white-listing accessible inputs. Any new dangers identified by the NIST or other organizations that scan code for vulnerabilities can be brought to light by regular dependency checks. If new risks are discovered, they can be reduced by following the advice on the reporting sites or by moving to a module that does not require reliance. Secure coding should always be used, and it will need to be adopted moving forward. Care must be taken while developing the API to prevent users from carrying out any tasks that are not explicitly permitted for their level of access. By using these best practices, the application should continue to work flawlessly for many years and give Artemis Financial all the capabilities they require.