****

# 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** | **10/19/2025** | **Julian Palmisano** |  |

## Client



## Developer

Julian Palmisano

## Algorithm Cipher

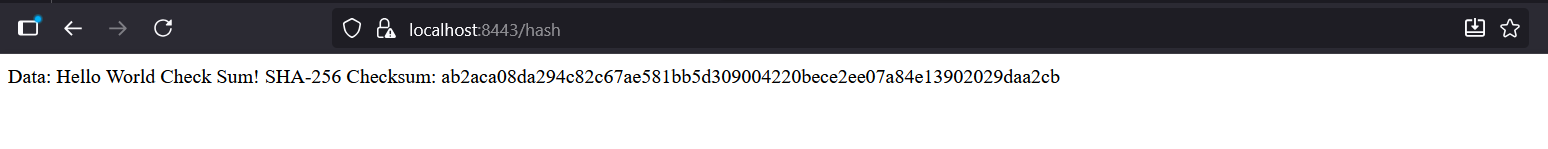
## For Artemis Financial, I recommend going with a hybrid system using AES for encrypting data and RSA for securely exchanging keys. AES is a symmetric algorithm that encrypts and decrypts data using the same secret key and is very efficient for large data transfers, making it ideal for financial information. RSA on the other hand is an asymmetric algorithm which uses a public/private key pair to safely transmit the AES key over public networks. For integrity verification, SHA-256 is used to generate a checksum for each data transfer. AES should use a 256 bit key, whilst RSA uses a 2048 bit key, and SHA-256 ensures minimal risk of collision. AES uses random initialization vectors to prevent repeated ciphertext for identical data, and RSA ensures secure key exchange. This combination of AES, SHA-256 and RSA provides a secure industry standard method for encrypting client data and verifying file integrity in Artemis Financials web app.

## Certificate Generation

A screenshot of a computer screen

AI-generated content may be incorrect.

## Deploy Cipher



## Secure Communications

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

## Secondary Testing

Insert screenshots below of the refactored code executed without errors and the dependency-check report.

A screen shot of a computer program

AI-generated content may be incorrect.A screen shot of a computer screen

AI-generated content may be incorrect.A screen shot of a computer program

AI-generated content may be incorrect.A screen shot of a computer screen

AI-generated content may be incorrect.

A close-up of a computer screen

AI-generated content may be incorrect.

## Functional Testing

A screenshot of a computer program

AI-generated content may be incorrect.

## Summary

I refactored the SSL server code to improve structure, readability, and maintainability while preserving its core functionality. After refactoring, the code was successfully compiled and executed without any runtime errors. The successful run shows that the functionality of the original implementation was retained, and the code now aligns with modern Java development standards.

## Industry Standard Best Practices

The refactoring followed several industry-standard best practices, including the use of proper naming conventions, separation of concerns, secure SSL communication handling, and Maven dependency management. The OWASP Dependency Check was also implemented to identify potential vulnerabilities, reflecting modern cybersecurity and software maintenance standards used in professional environments.