****

# 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** | **15 October 2024** | **Asa Brown** |  |

## Client



## Developer

Asa Brown

## Algorithm Cipher

I recommend using the Advanced Encryption Standard (AES) with a 256-bit key to encrypt archive files at Artemis Financial. AES-256 is a symmetric cipher known for its strong security. It utilizes hash functions to convert plaintext into unreadable ciphertext. While it offers high-level security, its computational demands can impact performance. Best practices for information protection include multi-factor authentication, regular software updates, least privilege access, and vulnerability assessments.

AES-256 faces some risks such as poor key management, insecure implementations, and potential vulnerabilities from advances in quantum computing. Additionally, Artemis Financial must comply with regulations like the Gramm-Leach-Bliley Act and PCI DSS to protect consumer data.

Alternatives to AES-256 may be considered if compatibility issues with older systems arise or simpler ciphers like AES-128 can adequately secure the data.

## Certificate Generation

A screenshot of a certificate

Description automatically generated

## Deploy Cipher

A screenshot of a computer

Description automatically generated

## Secure Communications

A screenshot of a computer

Description automatically generated

## Secondary Testing

A computer screen shot of a program

Description automatically generated

A screenshot of a computer

Description automatically generated

***Vulnerabilities Prior to False Positive Suppression***

## Functional Testing

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

***Dependency-check & suppression.xml included***

A screenshot of a check

Description automatically generated

***Vulnerabilities After False Positive Suppression***

## Summary & Industry Standard Best Practices

For this project, I was provided the code base for the SslServerApplication. Sections of code I refactored were in the SslServerApplication.java, application.properties, pom.xml, and suppression.xml files, in addition to adding the .jks and .cer files within the root. Within the java file, I implemented the checksum controller class to make sure that the data’s integrity was maintained and remained unchanged, and this also required the creation of a certificate.

By altering properties within the application.properties file, I could configure the port and relay information about the keystore. The port was configured to use port:8443 for HTTPS communications. The dependency-check plugins were also implemented within the pom.xml file, and after suppressing the vulnerabilities via the inclusion of a suppression.xml file, false positives were filtered out.

Following the vulnerability assessment process, REST API was used to handle GET requests for the /hash endpoint, and the hashing algorithm SHA-256 was utilized to generate checksums, a very solid way to ensure data integrity, and a try/catch statement was used to handle potential errors that could be introduced to the program. By refactoring this code, multiple sections in the vulnerability Assessment Process were addressed including APIs, Cryptography, Code Error, and overall Code Quality.