****

# 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** | **4/19/2025** | **Hunter Prince** |  |

## 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

Hunter Prince

## Algorithm Cipher: Recommend an appropriate encryption algorithm cipher to deploy, given the security vulnerabilities, and justify your reasoning. Review the scenario and the supporting materials to support your recommendation. In your practices for secure software report, be certain to address the following actions:

* 1. Provide a brief, high-level overview of the encryption algorithm cipher.
  2. Discuss the hash functions and bit levels of the cipher.
  3. Explain the use of random numbers, symmetric versus non-symmetric keys, and so on.
  4. Describe the history and current state of encryption algorithms.

Artemis Financials’ core mission is to deliver secure financial solutions to clients across the globe. To align with this goal, I recommend implementing SHA-256 as the primary encryption algorithm. This cipher is known for its high level of security, making it virtually impossible to breach through brute-force attacks—it would take many years to crack.

SHA-256 is widely recognized and frequently used in secure communication between financial institutions due to its reliability and resilience. It generates a hash value by compressing input data into a fixed-length output, known as the hash. The randomness incorporated in its bit structure enhances security by making outcomes unpredictable, which significantly reduces the risk of unauthorized access.

Encryption strength is directly tied to bit length—in this case, 256 bits—offering a vast range of potential combinations. This immense number of possibilities makes it extremely difficult for malicious actors to guess the correct encryption key.

There are two main types of encryptions: symmetric and asymmetric. Symmetric encryption, such as AES-256, uses a single key for both encryption and decryption. It is efficient and fast, making it suitable for many real-time applications. On the other hand, asymmetric encryption involves a pair of keys—one public and one private—offering a higher level of security, which is why it is often used for secure internet communications.

Encryption technology has a long and fascinating history, dating back to 600 BC (A brief history of encryption, Thales Group, 2016). Over time, encryption has evolved into a critical tool for protecting sensitive information, reflecting humanity’s enduring commitment to privacy and security.

## Certificate Generation

Insert a screenshot below of the CER file.

A computer screen with text

AI-generated content may be incorrect.

## Deploy Cipher

Insert a screenshot below of the checksum verification.

A screenshot of a computer

AI-generated content may be incorrect.

## Secure Communications

Insert a screenshot below of the web browser that shows a secure webpage.

A screenshot of a computer

AI-generated content may be incorrect.  
  
(This is after enabling HTTPS in my code in application settings, could not visually make it say secure but server log shows otherwise)

## Secondary Testing

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

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.(There are no new vulnerabilities when compared to the first project, there are in fact less with the new refactored code.)

## Functional Testing

Insert a screenshot below of the refactored code executed without errors.

A screenshot of a computer code

AI-generated content may be incorrect.

## Summary

During this assignment, I made a few key changes to strengthen the security of the application while refactoring the code. One of the main updates was adding SHA-256 hashing using Java’s Message Digest, which helps verify data integrity by generating a secure, readable checksum. I also reconfigured the application to run over HTTPS using a self-signed SSL certificate, so all communication is now encrypted and secure—even during local development. These updates were important for protecting against common threats like data tampering and man-in-the-middle attacks.

I also spent time cleaning up the pom.xml file and added the OWASP Dependency-Check plugin to scan for any vulnerabilities in the project’s dependencies. As a result, not only did the code stay secure, but the number of flagged vulnerabilities went down after the refactor. These updates weren’t just about checking boxes, they were about making sure the application is solid, secure, and ready for future development.

## Industry Standard Best Practices

Throughout the project, I followed the industry’s best practices to make sure the software stayed secure while I made changes. I implemented HTTPS for secure communication, kept sensitive info like passwords properly managed, and ran both static and manual testing to catch any vulnerabilities early. I also avoided adding unnecessary dependencies to reduce potential risks. By sticking to these secure coding standards, I helped protect the app from common threats, kept it compliant with industry regulations, and made it easier to maintain in the future. These steps not only improve the reliability of the software but also build trust with users and strengthen the company’s overall security posture.