****

# 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** | **6/29/2025** | **Steven Gifford** |  |

## 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

Steven Gifford

## Algorithm Cipher

For this project, I implemented a SHA-256 hashing algorithm using Java’s MessageDigest class. The checksum controller accepts a static string and returns a base64-encoded SHA-256 hash. This cipher algorithm is widely regarded as a secure and collision-resistant method of ensuring data integrity. It is part of the Java Cryptography Architecture and is commonly used in secure communications and file verification systems.

## Certificate Generation

Insert a screenshot below of the CER file.

A computer screen with white 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.



## Secondary Testing

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

A screen shot of a computer

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

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

AI-generated content may be incorrect.

## Functional Testing

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

A computer screen shot of a computer screen

AI-generated content may be incorrect.

## Summary

This project involved securing a Spring Boot RESTful web application by implementing cryptographic hashing and SSL communication. I refactored the provided application by adding a controller that performs SHA-256 checksum generation and ensured the application was served over HTTPS. Additionally, I generated a certificate keystore, configured it in the project, and confirmed secure access via a browser. Final testing confirmed the application runs without errors, and the checksum endpoint returns the expected encrypted string.

## Industry Standard Best Practices

To secure the application, I implemented several best practices:

* Used a widely trusted hashing algorithm (SHA-256)
* Configured HTTPS using a self-signed certificate to encrypt traffic
* Ensured the keystore was stored in src/main/resources, aligning with secure deployment practices
* Ran dependency-check analysis to identify and mitigate any known security vulnerabilities in project dependencies  
  These steps reflect modern DevSecOps expectations, integrating security early in the development lifecycle and ensuring encrypted communication for sensitive operations.