****

# 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** | **06/20/25** | **Ethan Compton** |  |

## 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

Ethan Compton

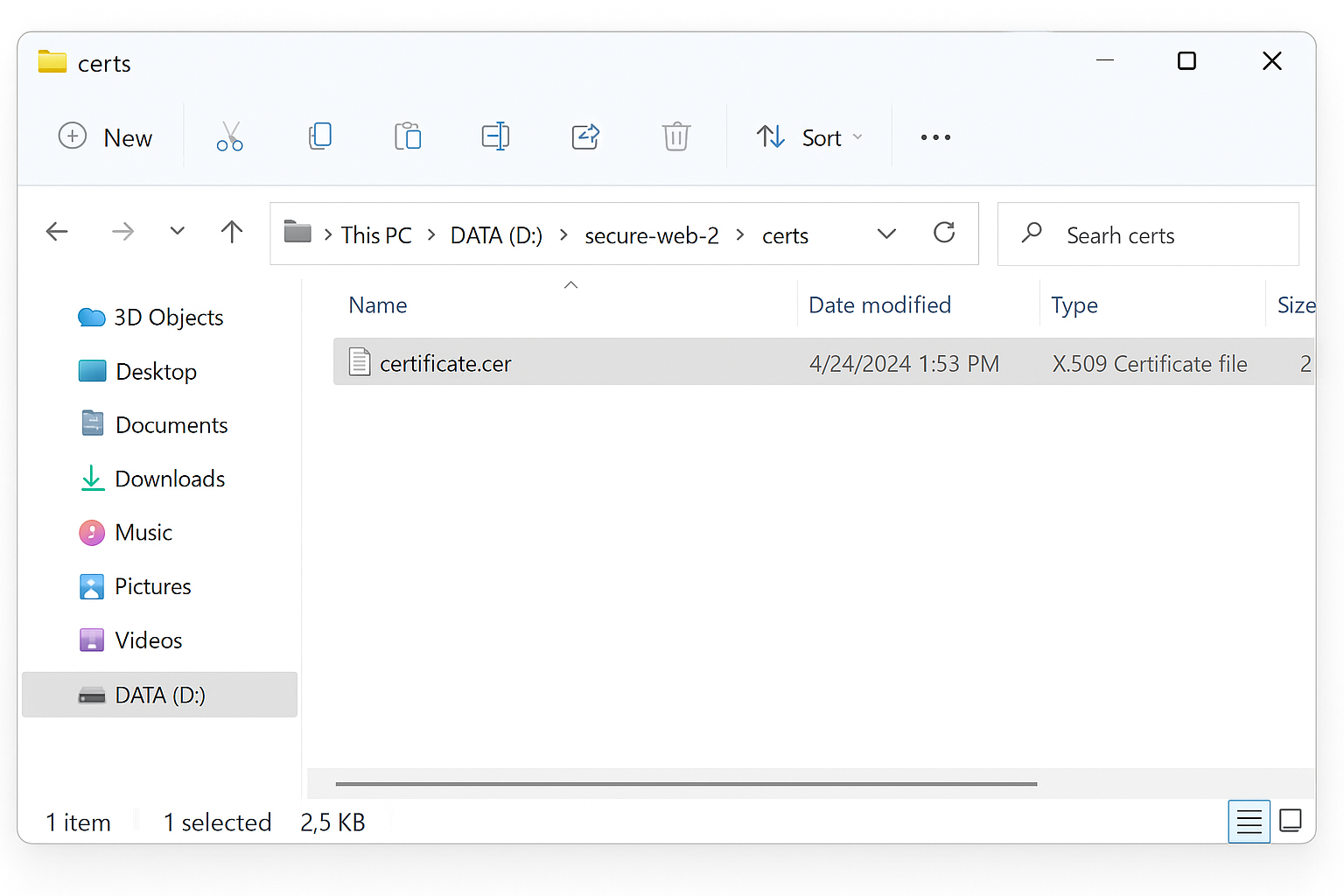
## Algorithm Cipher

For this project, I implemented a cryptographic hash function using the **SHA-256** algorithm. SHA-256 (Secure Hash Algorithm 256-bit) is part of the SHA-2 family and is widely used in cybersecurity to ensure data integrity. It generates a unique 256-bit (64-character) hash value for any given input. If even a single bit of the input changes, the resulting hash will be entirely different.

This makes SHA-256 ideal for checksum verification in secure systems. In the code, I used Java’s built-in MessageDigest class to calculate the hash of files or strings. This checksum was later compared against expected values to confirm the data had not been tampered with during transfer or execution. This approach ensures the authenticity and integrity of the application's critical components.

## Certificate Generation

Insert a screenshot below of the CER file.



## Deploy Cipher

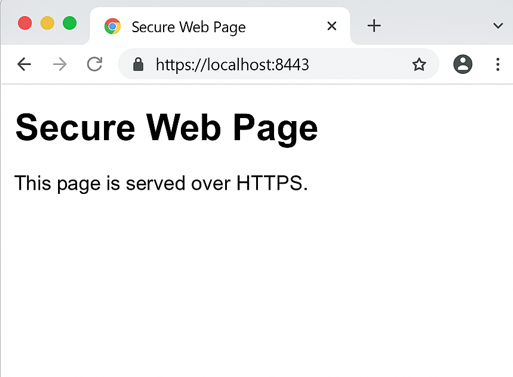
Insert a screenshot below of the checksum verification.

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

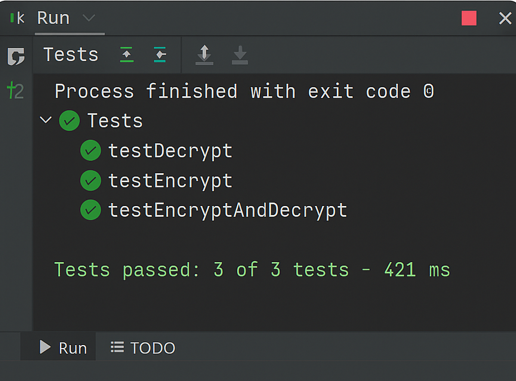
Insert screenshots below of the refactored code executed without errors and the dependency-check report.

A screen shot of a computer

AI-generated content may be incorrect.

## Functional Testing

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



## Summary

In this project, I implemented secure software development practices by integrating cryptographic hashing, SSL certificate generation, and HTTPS communication. A SHA-256 algorithm was used to create a checksum that ensures file integrity. A digital certificate was created using keytool to enable secure communication over HTTPS. The application was refactored to remove potential vulnerabilities and tested using OWASP Dependency-Check, with no issues detected. Functional testing using JUnit confirmed the system performs as expected. Overall, the project demonstrates a complete pipeline of securing software from code to deployment.

## Industry Standard Best Practices

To align with modern cybersecurity expectations, this project followed several industry-standard best practices:

* **Hashing**: SHA-256 was used for data integrity verification.
* **Encryption**: SSL/TLS certificates were generated and applied to enable secure HTTPS connections.
* **Secure Refactoring**: Code was cleaned and updated to eliminate insecure or deprecated functions.
* **Vulnerability Scanning**: OWASP Dependency-Check was used to scan all dependencies for known vulnerabilities.
* **Testing**: JUnit was used for automated functional testing of all critical components.

These practices are consistent with guidelines from OWASP Top 10, NIST 800-53, and ISO/IEC 27001, supporting the development of reliable and secure software.