****

# 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** | **6/25/25** | **Yusuf Bajwa** |  |

## 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

Yusuf Bajwa

## Algorithm Cipher

For this project, I recommend using AES (Advanced Encryption Standard) as the encryption algorithm. AES is one of the most trusted ciphers today and is widely used to secure data.

Overview:

AES is a symmetric-key algorithm, meaning the same key is used to encrypt and decrypt data. It’s fast, secure, and reliable, which makes it ideal for protecting sensitive information like communications and stored data.

Hash Functions and Bit Levels:  
While AES itself is used for encryption, SHA-256 is used here for hashing, providing a 256-bit hash. AES supports key sizes of 128, 192, and 256 bits, with 256-bit keys offering the highest level of security.

Random Numbers and Keys:  
AES depends on strong random number generation to create secure keys. Using symmetric keys simplifies key management, though the key must be kept private to maintain security.

History and Current State:  
AES was developed as a replacement for older, weaker algorithms like DES. It’s been the standard since 2001 and is still considered secure today, used by governments and major companies worldwide.

Overall, AES combined with SHA-256 hashing and SSL/TLS provides layered protection that meets modern security standards.

## Certificate Generation

Insert a screenshot below of the CER file.

A screenshot of a computer

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.

## Secondary Testing

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

Refactored Code Running in Eclipse

A screenshot of a computer

AI-generated content may be incorrect.

Dependency Check Report (0 Errors and Vulnerabilities were discovered)

A screenshot of a computer

AI-generated content may be incorrect.

## Functional Testing

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

Refactored code running, without any errors.

A screenshot of a computer

AI-generated content may be incorrect.

## Summary

I updated the code to add SHA-256 hashing, which helps make sure the data isn’t messed with. The checksum part was refactored to meet security standards and keep things secure. I also set up SSL/TLS with a certificate, so the site runs over HTTPS for safer communication.

This lines up with the vulnerability assessment process, mainly:

* Risk Identification — Adding hashing makes sure data stays the same.
* Secure Development — Refactoring with SHA-256 adds stronger protection.
* Verification/Testing — I tested everything and ran a dependency check to confirm no known issues.

Additionally, I layered in HTTPS with the SSL cert and checked for vulnerable libraries to tighten things up further.

## Industry Standard Best Practices

For this project, I followed standard secure coding practices to keep the app safe. I used SHA-256 hashing to create secure checksums, which helps make sure the data hasn’t been messed with. I also set up SSL/TLS so communication between the client and server is encrypted.

On top of that, I ran an OWASP Dependency-Check to scan for known security issues in the project’s dependencies. That helps catch vulnerabilities before they become a real problem, luckily there were no problems detected.

Overall, sticking to these security best practices protects the app, keeps user data safe, and helps the company avoid issues down the line like data breaches or trust problems.