****

# 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** | **February 25, 2024** | **Brandon Quinones** |  |

## 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

Brandon Quinones

## Algorithm Cipher

The best algorithm cipher I would recommend is the RSA algorithm. It is an asymmetric cypher meaning that it uses both a public key and private key. This is more secure than a symmetric cipher, and it is legal to use as an export. With an RSA Algorithm we can provide the client with a public key to encrypt their data using our algorithm without sharing the private key. Our algorithm with encrypt this, once the encrypted data is sent only we can use the private key to decrypt this data.

## Certificate Generation

A screenshot of a computer program

Description automatically generated

## Deploy Cipher

A screenshot of a computer

Description automatically generated

## Secure Communications

Tried everything I could research, from installing certificates to the machine and converting the .jks file to pkcs12 and adding the certificate to trusted certificates on the browser. Nothing seemed to work to remove the red scratch from the https element and have the browser show a secure connection.

A screenshot of a computer

Description automatically generated

## Secondary Testing

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

A screenshot of a computer

Description automatically generated

## Functional Testing

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

A screenshot of a computer

Description automatically generated

## Summary

For this ssl server the hashing function works as designed with minimal vulnerabilities. This RESTful API uses secure connection to the server because it only connects to the local host port if the valid certificate is present and the correct password is given in the application.properties file. This is something that would need to be issued specifically by the employer and cannot be shared outside the company. The signature is encrypted using RSA method, into SHA 256 hash. The cipher of the actual document itself is in SHA 512. The SHA 256 and 512 hashing methods are used because they are small and have a low chance of collision.

## Industry Standard Best Practices

To best secure this RESTful API the vulnerability check would need to be executed every two weeks, and revised for any changes, especially after updates to plugins in maven. The certificate must also be kept up to date, and renewed yearly, or for more security 3-6 months. The best practice is to get a certificate validated by a trusted authority. This way the browser can recognize it as a valid certificate and will give the user confidence of a secure connection.