****

# 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 5](#_Toc102040760)

[4. Secure Communications 5](#_Toc102040761)

[5. Secondary Testing 5](#_Toc102040762)

[6. Functional Testing 7](#_Toc102040763)

[7. Summary 7](#_Toc102040764)

[8. Industry Standard Best Practices 7](#_Toc102040765)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **February 19, 2024** | **Malcolm Williams** |  |

## Client



## Developer

Malcolm Williams

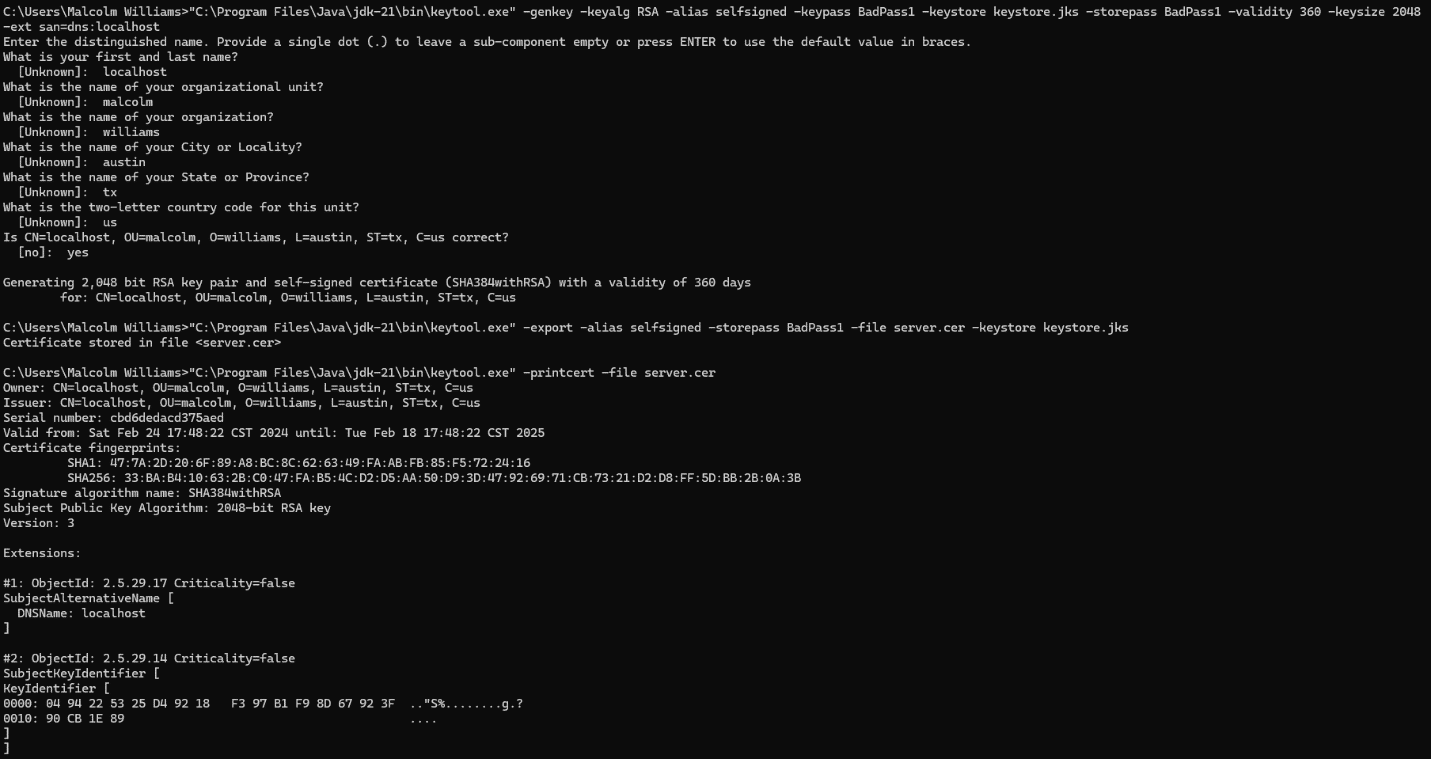
## Algorithm Cipher

SHA-256

SHA-256 is a cryptographic hash function that takes any length of input and produces 256 bits of data that is unique to the input. It is a one-way function, so each input has a computable output, but computing the input from the output is very difficult. The algorithm is collision resistant and has zero known collisions which makes it an ideal choice for digital signatures, and password verification. SHA-256 is part of the SHA-2 hash algorithm group, which also contains 224, 384, and 512 bits. They vary in the size of their outputs and the size of the words used to calculate the output. For example, SHA-256 uses 32-bit words while SHA-512 uses 64-bit words. SHA-256 splits its input into 512-bit chunks and performs 64 logical operations to produce its 256-bit output. This algorithm is deterministic and does not utilize random numbers. It is also not to be confused as an encryption algorithm so there is no key required to produce the hash and the hash cannot be reversed to the original input.

In 2001, the National Institute of Standards and Technology alongside the National Security Agency published the SHA-256 hashing algorithm to succeed SHA-1 because SHA-1 was found to be vulnerable to brute force attacks that could determine the input from the output. SHA-256 is still considered secure in early 2024 and is used in common protocols such as SSL, TLS, and SSH. Even though SHA-2 is currently secure, a new algorithm called SHA-3 was developed in 2015 in case SHA-2 has vulnerabilities that are discovered in the future.

## Certificate Generation

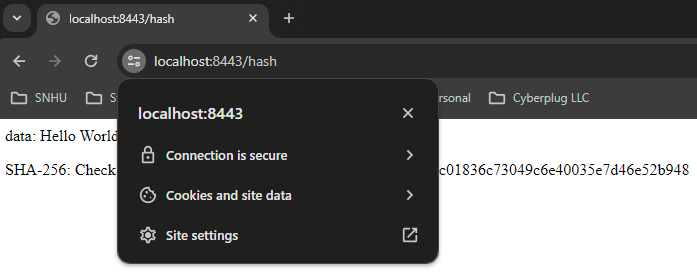
Here, I generated a self-signed certificate with the Java keytool where I used “-ext san=dns:localhost” so the browser would accept the certificate for the localhost address.

## Deploy Cipher

A screenshot of a computer

Description automatically generated

## Secure Communications



## Secondary Testing

The refactored code includes the ServerController class. The added code did not introduce any new security vulnerabilities and was executed without error.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

## Functional Testing

The code is seen running without error:

A screenshot of a computer

Description automatically generated

## Summary

During this process I ran a static test of the code and a dependency check. The dependency checks only noted false alarms or vulnerabilities without a current fix. This static test addresses the “Secure API Interaction” section of the Vulnerability Assessment Process Flow Diagram since this is a Spring application that supports the integration of numerous APIs. The Server Controller which was integrated into the SSL application demonstrates the ability to use a secure hash function, the same technology behind the certificate of authenticity presented to the browser that shows the user that the domain is trustworthy and using secure communications. This ability to use a digital signature address the “Cryptography” block of the Vulnerability Assessment Process Flow Diagram.

## Industry Standard Best Practices

In this software application I ensured that the server used SSL/TLS for secure communications over the internet. Utilizing a digital signature to confirm the website’s identity and encrypting the data that is transmitted combine to form a more secure application for the company. The data used in the hash is not user-generated so there was no need to verify the input data before using it. The dependency check that cross-examined the supply chain of the application with known vulnerability databases was another key step in ensuring that the application remains as secure as possible.