****

# 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** | **6/16/2023** | **Ty Simpson** |  |

## Client



## Developer

Ty Simpson

## Algorithm Cipher

We will implement the SHA-256 Algorithm Cipher. SHA-256 is an industry standard and has no known vulnerabilities. The chances of collisions, where two messages share the same hash values, is unlikely due to there being a total of 2256 possible hash values. Avoiding collisions is important – for example, attackers wouldn’t need to know exact passwords if they were able to use a password that generated the same hash.

## Certificate Generation

A screenshot of a computer

Description automatically generated with medium confidence

## Deploy Cipher

A screenshot of a computer

Description automatically generated with medium confidence

## Secure Communications

A screenshot of a computer

Description automatically generated with medium confidence

## Secondary Testing

A screenshot of a computer program

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with low confidence

## Functional Testing

A screenshot of a computer program

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

## Summary

Through refactoring the code, I employed a cryptographic algorithm through generating a checksum. I employed a Rest Controller – creating the “/hash” that we would navigate to test the hash. I also generated a keystore and certificate that validates and secures the site. However, since the certificate was self-signed the browser would not implement the https protocol. I ran a dependency check, and while the code I implemented did not generate any new vulnerabilities – vulnerabilities did exist. It seems that many of them can be fixed by upgrading these dependencies to their most current and most stable versions.

# References

N-able. (2019, September 12). *SHA-256 Algorithm Overview*. Retrieved from N-Able: https://www.n-able.com/blog/sha-256-encryption