****

# 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 5](#_Toc102040761)

[5. Secondary Testing 6](#_Toc102040762)

[6. Functional Testing 7](#_Toc102040763)

[7. Summary 8](#_Toc102040764)

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

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **10/19/24** | **Joseph Caron** |  |

## 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

Joseph Caron

## Algorithm Cipher

For Artemis Financial’s needs, I recommend using SHA-256. It is the most widely used encryption algorithm ciphers when it comes to protecting personal and financial data. It is considered to be practically impossible to break with current technological capabilities, so it will absolutely keep all of the information safe from unauthorized access, even in transit over the internet. This encryption algorithm cipher is so secure because there are 2256 possible hash values. To put that in perspective, it would take longer than the estimated age of the universe to brute force one’s way through information hashed with SHA-256.

The bit level of SHA-256 is, as the name implies, 256 bits. The algorithm cipher takes any input, regardless of size, and turns it into a 256 bit hash. Hashing is a bit different from encryption, because the output is completely randomized, has no comprehensible relation to the input, and cannot be reverse engineered to discover the input from the hash. With encryption algorithms, you generally have two types of keys depending on the cipher chosen: symmetric and asymmetric. Symmetric keys are simpler to use, as one key both encrypts and decrypts the information. Asymmetric keys, on the other hand, are more complex because they require two keys, a public key and a private key. The information is encrypted using the public key, and then the private key is used to decrypt the information. SHA-256, however, does not use a key to encrypt data, so it cannot be said to be either symmetric or asymmetric. Whatever data is converted to a hash will be converted into the same hash, no matter where it is done. A key is then used to convert the hash back into plain text. Encryption methods have been around for centuries, though the use of computer systems in encryption has obviously been more recent than that. As technology improves and computer capabilities increase, security continues to be very important and so encryption must evolve just as quickly.

## Certificate Generation

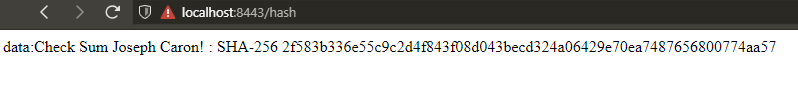
A computer screen shot of a black screen

Description automatically generated

## Deploy Cipher

A screen shot of a computer program

Description automatically generated



## Secure Communications

A screen shot of a computer

Description automatically generated



A screenshot of a computer

Description automatically generated

## Secondary Testing

A computer screen with colorful text

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

## Functional Testing

A computer screen shot of a program code

Description automatically generated

## Summary

To enhance the security of the program, I refactored the code in the SslServerApplication.java file. I wrote in a RestController which includes a hash cipher to generate a check sum to help protect transmitted data. Further to this, I also created a self-signed safety certificate and connected it to the program. HTTP was changed to HTTPS protocol and I also ran a MAVEN dependency check to make sure I didn’t introduce any new vulnerabilities by adding in the security measures that I did.

## Industry Standard Best Practices

While upgrading the security of the program in question, I followed industry standard best practices by ensuring that I did not introduce more vulnerabilities into my code as I wrote it. By implementing a checksum I ensured that the validity of the data wouldn’t be compromised. I put in a self-signed certificate to add site identity verification and data encryption, though I would recommend obtaining a new certificate from a trusted CA, as self-signed certificates could cause security issues in the future. Keeping to industry standard best practices is always the best idea in general, and specifically when it comes to software security. Keeping private information, such as personal identifiable information and financial information, secure and private is essential for a company that deals with that type of information on a daily basis. Any lapse could lead to data leaks or breaches, which besides being unethical to simply allow would also bring down the reputation of the company and could severely harm business.