****

# 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** | **04/21/2024** | **Max R Maysonet-Ramirez** |  |

## 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

Max R Maysonet-Ramirez

## Algorithm Cipher

1. Recommend an appropriate encryption algorithm cipher to deploy, given the security vulnerabilities, and justify your reasoning. Review the scenario and the supporting materials to support your recommendation. In your practices for secure software report, be sure to address the following:
   1. Provide a brief, high-level overview of the encryption algorithm cipher.
   2. Discuss the hash functions and bit levels of the cipher.
   3. Explain the use of random numbers, symmetric versus non-symmetric keys, and so on.
   4. Describe the history and current state of encryption algorithms.

Artemis Financial is a FinTech company that contains highly sensitive data, and this requires that the data stays protected during transportation and at rest. Bad actors are most likely going to want to access that information to pursue financial gain.

With this, my 1st recommendation would be to implement encryption methods. This can allow the data to be encrypted while being transported and at rest. For the intruder to access the data the intruder would need to gain access to the key to decrypt the hash. This can be used with a CA that would deploy a public key that encrypts the data, and the only form of decrypting would be to use the private key from the CA.

The **SHA-256** would work nicely for this implementation. SHA-256 is a cryptographic function that takes an input and produces a fixed-size 256 length hash, which is seemingly random to the intruder. SHA-256 is nearly impossible to reverse engineer and has an extremely high collision resistance.

This method would provide Artemis Financial will the security it needs in it’s communications and will allow to securely lock it’s data away from access to bad actors.

## Certificate Generation

A screenshot of a computer program

Description automatically generated

## Deploy Cipher

A screenshot of a computer

Description automatically generated

## Secure Communications

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

## Secondary Testing

A screen shot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

## Functional Testing

A screen shot of a computer program

Description automatically generated

## Summary

Upon refactoring the code that was provided to me, I thought it would be efficient to apply a cryptographic hash function (SHA-256) for securing data. This adheres to the “Cryptography” step in the flow diagram because I’m encrypting the data to secure it, lowering the chance of vulnerabilities. I also used the “bytesToHex” function securely converts the byte array to a hex string, which uses checksums or hashes. This step regards the encapsulation section of the process flow diagram.   
  
By using the @requestmapping securely, I am able to utilize a HTTPS connection through port 8443. This aligns with the “APIs” stage and all of the data therein is also encrypted. The only part of the string is the “/hash” part.

To add layers of security, I used SSL/TLS. I used a local CA to assign a “selfsigned” certificate that secures the channel between the client and the server. I also used input handling as well. MessageDigest should be validated to prevent injection attacks. This corresponds to “Input Validation”. Another security layer I’ve added is a secure hash algorithm such as SHA-256 that is known for it’s resilience against hash collision attacks. Finally the last layer was the code quality.

## Industry Standard Best Practices

I refactored the SSL server code to make it safer using common secure coding rules that the tech industry recommends. I used a strong method called SHA-256 to check that data hasn't been tampered with, and made sure the server only talks through secure connections (HTTPS). By following these rules, I helped the application stay safe from known threats. These steps are important because they keep the company's data safe, prevent hacking, and show customers that the company is trustworthy, which is really good for business.