****

# 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** | **12/15/2024** | **Joshua Shults** |  |

## 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

Joshua Shults

## Algorithm Cipher

For the algorithm cipher, considering Artemis Financials’ needs, I would recommend the Advanced Encryption Standard that utilizes a 256-bit key format (AES-256). AES is a NIST standardized algorithm that is widespread in its use due to its compliance with common regional mandates and capabilities of producing lengthy cipher keys. AES-256 also works great with hash functions, it is commonly paired with SHA-256. Hash functions further increase the security of sensitive data by providing a digital fingerprint, meaning that if even one character of a key is altered it will drastically change the entire hash which makes malicious tampering obvious.

AES-256 is symmetric in its generation of keys, meaning that a file is encrypted and decrypted using the same key. While this means that if a key is obtained by an outside source it can be used to unlock its associated file, this still is preferrable as it reduces the performance overhead for Artemis, and with secure protocols for storing and passing keys this risk can be mitigated greatly.

AES also uses random number generation when producing cipher keys. This means that there will be no way to reverse engineer key production as its generation is completely random.

As to the history of Algorithm Ciphers, AES was made the industry standard in 2001. The organization that established AES as the standard was the National Institute of Standards and Technology (NIST). AES became the standard due to its bit-key lengths. A larger bit-key means that there are more possible keys for the random generation, and a larger key means that it is incredibly resistant to brute force attempts at decrypting the key.

## Certificate Generation

Insert a screenshot below of the CER file.

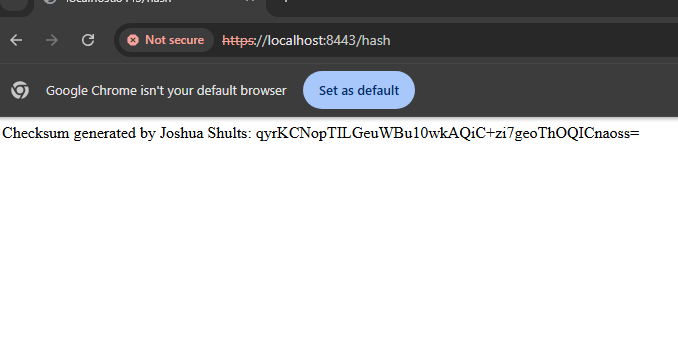
A screenshot of a computer

Description automatically generated

(The CER file is open in the bottom right of the image, the rest was included for the instructor to see its location in the refactored ZIP file)

## Deploy Cipher

Insert a screenshot below of the checksum verification.



## Secure Communications

Insert a screenshot below of the web browser that shows a secure webpage.

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 computer screen shot of a program

Description automatically generated

## Summary

In summary, I introduced AES-256 with a hash function, SHA-256, into the refactored code in section 7 where it is seen running without errors. Also, from the HTTPS we can see that the certificate is valid and that the placeholder data was encrypted via AES-256. HTTPS was enabled using a self-signed certificate, this ensured that the communication was secure and encrypted. By using this method in Artemis Financial they can be assured that all communications between their server and their clients will remain confidential, as well as in the case of a malicious user retrieving data in transit it will only reveal a encrypted message that is not decipherable with AES-256’s large bit-key length. The application also was refactored to include SSL which enforces the HTTPS protocol on the designated port, which for this example was on port 8443.

Though the browser itself did not show “trust” due to the file including a self-signed certificate, which browsers do not trust, this is sufficient to show just for this development. Included in the screenshot was the dev commands for chrome showing that while the browser itself will not trust a self-signed cert, it was still utilizing HTTPS and SSL for secure communications.

## Industry Standard Best Practices

As for industry standard practices I incorporated a checksum verification to show that the code was doing what was intended, producing an encrypted message. I integrated the OWASP Dependency-Check tool to find any new vulnerabilities that could have been introduced by my additions to the code base. I also utilized input validation, by utilizing a @RequestParam to handle any incoming user inputs at an endpoint.