****

# 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** | **8/18/24** | **Tiffany Thai** |  |

## Client



## Developer

Tiffany Thai

## Algorithm Cipher

For Artemis Financial, the selected cipher algorithm must balance security, compliance, and practicality. The selected cipher must serve its purpose. For encrypting highly sensitive financial data, the recommended cipher algorithm: Advanced Encryption Standard (AES), also known as the Rijndael algorithm. AES is widely recognized as one of the most secure encryption algorithms today. AES supports key sizes of 128, 192, and 256 bits, with AES-256 being the most secure and currently recommended for highly sensitive data, like the financial data of Artemis Financial. AES is resistant to various types of cryptographic attacks, including brute force, differential, and linear cryptanalysis.

For the verification step ensuring secure communications, Artemis Financial’s web application is recommended to use SHA-256 algorithm. Secure Hash Algorithm 256-bit (SHA-256) is a cryptographic hash function that produces a 256-bit hash value, which is nearly impossible to reverse-engineer. It is part of the SHA-2 family of algorithms designed by the National Security Agency (NSA) and is widely trusted and used in various security protocols and applications. One of the key features of SHA-256 is its resistance to collision attacks, making it computationally impossible to generate the same hashed output from two different inputs.

Secure random number generators are used to create keys. The quality of the randomness directly affects the security of the encryption. AES is a symmetric key algorithm, meaning the same key is used for both encryption and decryption. This is efficient but requires secure key management. While SHA-256 itself does not use random numbers, when combined with encryption algorithms in secure communication protocols, random numbers are used to ensure the uniqueness of keys or to prevent replay attacks. SHA-256 is a hash function and does not use keys in the way encryption algorithms do. For key exchange, algorithms like RSA or ECC are used. They are asymmetric and use a pair of keys (public and private), but AES is more suitable for the actual encryption of data.

AES was adopted as the encryption standard by NIST in 2001, following the Rijndael algorithm selected through a public competition. It has since become the de facto standard for symmetric encryption. AES is widely used across various industries and remains highly secure. Its continued use and endorsement by organizations like NIST ensure that it remains a trusted choice for encryption.

SHA-256 was developed to provide secure methods of data integrity verification and was introduced as part of the SHA-2 family in response to vulnerabilities discovered in SHA-1. SHA-256 remains one of the most trusted and secure hash functions in use today. Despite being over two decades old, it is still considered secure against known cryptographic attacks, making it suitable for Artemis Financial's data verification needs.

## Certificate Generation

## Generated self-signed certificates using the Java Keytool.

## A screenshot of a computer program Description automatically generated

## A screenshot of a computer Description automatically generated

## Deploy Cipher

## Implemented and deployed a cryptographic hash algorithm, SHA-256, to demonstrate a checksum verification.

A black text on a white background

Description automatically generated

## 

## Secure Communications

## Refactored code to convert HTTP to HTTPS protocol to establish secure communication.

A close up of numbers

Description automatically generated

## Secondary Testing

## Refactored code executed without errors

A screenshot of a computer screen

Description automatically generated

Report of the output from the dependency-check static tester

A screenshot of a server

Description automatically generated

## Functional Testing

## Refactored code for checksum verification executing without errors

A computer screen shot of a program code

Description automatically generated

A computer screen shot of a program

Description automatically generated

The code follows Java and Spring Boot conventions. Logically, the data string “Hello, World! Check Sum!” is hardcoded. While this is suitable for demonstration, in a real-world application, it is better to allow dynamic input so that the checksum can be generated for various data. The catch block throws a **RuntimeException**, which is good for identifying issues during development. In a production environment, additional logging might be helpful to track errors without exposing them to the end user.

## Summary

The refactored code uses SHA-256, a cryptographic hash function, to generate a checksum. SHA-256 is part of the SHA-2 family, which is widely considered secure for cryptographic purposes. By using SHA-256, the application mitigates the risk of vulnerabilities that are present in older or weaker algorithms (like MD5 or SHA-1). The code allows for the generation and delivery of a checksum over HTTP/HTTPS. This is part of client-server communication. The refactoring process included converting the application to use HTTPS instead of HTTP for secure communication between the client and server. HTTPS ensures data transmitted over the network is encrypted, protecting it from interception and tampering.

To further secure the web application, before generating the checksum in a real-world scenario, the dynamic input data should be validated (input validation) to guarantee it meets expected formats and does not contain harmful content. For example, adding checks to strip out any harmful characters or validate data length could enhance security. The code demonstrates an API endpoint /hash that returns a checksum, securing API is crucial to prevent unauthorized access or manipulation. The API endpoint should be protected some sort of authentication and authorization mechanisms to ensure only authorized users can access it.

By addressing these areas, the refactored application aligns with industry standards for secure coding, mitigating potential security vulnerabilities, and ensuring the overall safety and integrity of the software.

## Industry Standard Best Practices

Applying industry-standard best practices for secure coding is essential in refactoring the code for Artemis Financial’s web application. One of the key implementations is the use of SHA-256, a secure cryptographic hash function, to generate checksums. This ensures that data integrity is maintained, and the likelihood of checksum forgery or tampering is minimized. Additionally, enforcing HTTPS for secure communication keeps data in transit from being intercepted or altered, protecting sensitive financial information shared between clients and the server.

Securing coding practices mitigate risks associated with data breaches, maintaining compliance with regulatory requirements, and uphold the company’s reputation as a trustworthy financial service provider. By putting security first in the software development process, Artemis Financial can ensure that its operations remain efficient, its data is protected, and its clients can continue to have confidence in the company's services.

**References**

Ashreet, A. (2023, December 30). Securing Spring Boot Application with HTTPS and Self-Signed Certificates. *Medium*. https://medium.com/@ankita.ashreet/securing-your-spring-boot-app-a-guide-employing-https-with-self-signed-certificates-b16f16ca25d6

Baeldung, & Baeldung. (2024, May 11). *SHA-256 hashing in Java | Baeldung*. Baeldung. https://www.baeldung.com/sha-256-hashing-java

*Getting started | Building a RESTful web service*. (n.d.). Getting Started | Building a RESTful Web Service. <https://spring.io/guides/gs/rest-service>

Obregon, A. (2024, June 17). Basics of SHA-256 Hashing in Java | Medium. *Medium*. https://medium.com/@AlexanderObregon/what-is-sha-256-hashing-in-java-0d46dfb83888