****

# 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** | **December 9th, 2024** | **Tristen Bradney** | **Changes to Algorithm Cipher, Certificate Generation, Deploy Cipher, Secure Communications, Secondary Testing, Functional Testing, Summary, and Industry Standard Best Practices** |

## Client



## 

## Developer

Tristen Bradney

## Algorithm Cipher

I recommend the use of the SHA-256 hash algorithm for Artemis Financial's web application to verify data integrity and make sure of secure communication. SHA-256 is effective against current cryptographic attacks, widely used, and follows industry best practices for secure software development. Secure Hash Algorithm 256-bit is a cryptographic hash function that generates a fixed-length, unique, and irreversible 256-bit hash for any given input. It belongs to the SHA-2 family, designed by the NSA, and is considered one of the most secure hashing algorithms available. SHA-256 is largely used for keeping data integrity in secure communications and applications, like checksums and digital signatures.

A hash function takes input data of any length and produces a fixed-length output, or a “hash”, that represents the data uniquely. SHA-256 makes sure that even the smallest change to the input largely alters the hash output, making it great for tamper detection. SHA-256 produces a 256-bit hash value. This high bit level provides a very large address space that reduces the risk of collisions. It also provides more resistance against brute-force attacks compared to older algorithms like MD5 and SHA-1.

While SHA-256 doesn’t strictly rely on random numbers, it is commonly used with cryptographic systems that take advantage of randomness, like key generation and secure session identifiers, to improve security. SHA-256 is not an encryption algorithm, but it is often used alongside symmetric key algorithms for checksum verification, message authentication, and data integrity. When merged into public key infrastructure, SHA-256 is used for creating and verifying digital signatures, which creates secure exchanges between systems.

Earlier hash functions like MD5 and SHA-1 were regularly used but became vulnerable to collision attacks, making them too risky for modern security needs. SHA-2, developed in 2001, addressed these vulnerabilities. SHA-256, specifically, became the standard for cryptographic security because of its balance of security and processing speed. SHA-256 is part of NIST's recommended cryptographic algorithms and is largely used in applications from SSL/TLS certificates to blockchain. It is still secure against known cryptographic attacks and provides thorough protection for data integrity and verification in secure communication. With Artemis Financials’ requirement for secure data transmission and verification, SHA-256 is a great choice. Its ability to generate unique, tamper-proof hashes strengthens the integrity of transmitted data, which provides enhanced security in Artemis Financials’ web application and satisfies their user needs.

## Certificate Generation

A screen shot of a computer

Description automatically generated

## Deploy Cipher

A screenshot of a computer

Description automatically generated

## Secure Communications

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 TestingA screenshot of a computer program Description automatically generated

## Summary

## I implemented SHA-256 hashing in Artemis Financials’ application for data integrity verification. A new /checksum endpoint generates and returns a cryptographic hash of a unique string, "Tristen Bradney’s checksum for Artemis Financial". I chose the SHA-256 algorithm for its strong security effects, that include resistance to collision and tampering.

## I made sure of secure communication by enabling HTTPS through SSL/TLS encryption. I used a keystore to configure the Tomcat server to run securely on port 8443. This prevents unauthorized access and makes sure that data being transmitted between the client and server is encrypted.

## My code follows secure coding practices and clean design principles. I added a dedicated controller class, ChecksumController, to encapsulate the checksum logic. I added a generateChecksum method that follows a nice, modular structure with appropriate exception handling. I also added helper methods, like bytesToHex, for code reusability and to increase readability.

## The new /checksum endpoint responds to HTTP GET requests, generates and returns the SHA-256 checksum of the static string, and makes sure of consistent and predictable behavior without exposing sensitive implementation details.

## My process for adding layer of security included enabling secure communication, implementing the SHA-256 hashing, refactoring the code to be modular, and exception handling. I configured HTTPS using a self-signed certificate stored it in keystore.jks. I configured the server to listen on port 8443, making sure of encrypted client-server communication. My generateChecksum method generates a SHA-256 hash of the unique string "Tristen Bradney’s checksum for Artemis Financial." I converted the hash into a readable hexadecimal format using a helper method, bytesToHex, making sure of data integrity verification. I added a nested controller class, ChecksumController, that follows the MVC design pattern. I segmented methods to improve readability, maintainability, and security. The application now includes thorough exception handling to catch errors during hash generation, preventing the server from exposing unintended information.

## The refactored application follows the Vulnerability Assessment Process Flow by addressing important areas like cryptography, client/server security, APIs, and code quality. The addition of the /checksum endpoint with SHA-256 hashing provides a secure method for verifying data integrity. By enabling HTTPS and using clean coding practices, multiple layers of security have been added to make sure the application follows secure software development protocols.

## These enhancements strengthen the application's resistance to vulnerabilities and show industry-standard security practices. Future improvements could include more in-depth validation mechanisms, secure error logging, and integration of automated static analysis tools for continuous security monitoring.

## Industry Standard Best Practices

While refactoring the Spring Boot application, I applied industry-standard best practices for secure coding to mitigate known security vulnerabilities and maintain the software’s existing security. By following these standards, the application now has modern security principles, making sure of the protection of sensitive data, system integrity, and client trust. I enabled HTTPS using SSL/TLS encryption to secure data transmission between the client and server. I configured a self-signed keystore, making sure that sensitive information transmitted over the network is encrypted and inaccessible to unauthorized users. This keeps user information confidential and prevents interception or man-in-the-middle attacks.

The SHA-256 hashing algorithm, a secure and widely used cryptographic standard, was implemented to verify data integrity. A new /checksum endpoint was created to show this functionality. By hashing the static string "Tristen Bradney’s checksum for Artemis Financial.", the system makes sure that the data cannot be altered without detection.

With refactoring, I followed the principle of least privilege and modularization, making sure that code only performs the specific tasks required. The methods were modularized, like the bytesToHex helper method, to prevent code duplication and increase maintainability. Exception handling was added to avoid exposing stack traces or sensitive error information, decreasing the risk of information leakage. The newly introduced /checksum endpoint follows RESTful design guidelines, making clear, secure, and predictable API behavior. The response format avoids unnecessary details that could be exploited by malicious users.

Using HTTPS makes sure that data exchanged between the client and server is encrypted, protecting it from unauthorized access and interception. SHA-256 hashing guarantees that data integrity can be verified, reducing risks of tampering and increasing reliability. By using secure coding practices, Artemis Financial shows its dedication to protecting client data, creating trust and confidence with its users. Secure systems are important for companies in the financial field, where data breaches could have severe legal and reputational consequences. Following security best practices help protect the application against common vulnerabilities, like MITM attacks (prevented by HTTPS), data tampering (prevented by SHA-256 hashing), and information leakage (prevented with structured exception handling). Modular code and clean API design increases the application’s maintainability, making it easier to find and fix security vulnerabilities in the future. Using secure coding practices makes sure the application can be scaled without compromising its security. Following industry standards like OWASP and NIST guidelines makes sure that the application meets current security expectations. Compliance reduces the risk of legal penalties and shows Artemis Financials’ commitment to thorough security measures.

By using HTTPS, SHA-256 hashing, and secure coding practices, industry-standard best practices were applied to improve the existing security while mitigating vulnerabilities. These practices not only help to protect sensitive user data but also improve long-term maintainability and scalability. Following these standards strengthens the company’s reputation, improves client trust, and protects Artemis Financial from possible security threats and associated risks.