****

# 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** | **6/23/24** | **Justin Phillips** | **Initialized document** |

## 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

Justin Phillips

## Algorithm Cipher

Artemis Financial’s application is a mobile banking application hosted on a webserver which requires a secure, robust encryption algorithm to safely carry sensitive information. My recommendation is to employ SHA-256 as the algorithm cipher for data transmission and communication between the client and server. As per its name, it produces a fixed-length 256-bit hash value (32 bytes) from an input of any size. Due to the immense number of combinations, 2^256, it is immensely difficult and time consuming to crack, and is able to produce a nearly unique hash for any input (Thakkar, 2022). SHA-256’s hash is sent to the recipient and the packet’s SHA-256 hash value is recalculated upon transmission of the packet, meaning there is also data verification through the use of SHA-256, further ensuring file integrity and security (Bhardwaj, 2023). SHA-256 does not use keys but acts as a fantastic encryption method for data transmission. This means it may not be ideal for certain forms or logging into an account on the webapp, but it will be fast for real-time communication.

For these, I also recommend implementing the RSA algorithm, an asymmetrical cipher that uses a 2048 bit key, meaning that it would be nearly impossible to guess or brute force (2^2048 permutations in 256 bytes). RSA employs a public key from the server and a private key from the client, meaning that even if one key is compromised, an attacker likely would not be able to take advantage of it before the key gets refreshed (Kelly, 2009). Similarly to SHA-256, the hash value gets recalculated on packet reception, allowing verification of data integrity. How RSA actually works is that the packet is hashed and encrypted using a private key, creating a unique hash. RSA uses two random large prime numbers to form the basis of its keys (Simmons, 2022).

RSA and SHA-256 are both today considered tried-and-true reliable encryption algorithms. RSA was described in 1977 publicly by Ron Rivest, Adi Shamir, and Leonard Adleman in 1977, and serves as one of the oldest public-key cryptosystems. The original concept for the asymmetrical public-private key system is attributed to Martin Hellman and Whitfield Diffie in just 1976, accompanied by the concept of digital signatures (Rivest). SHA-256 is younger, published in 2001 by the National Institute of Standards and Technology (NIST) for government application. It was part of the SHA-2 family, successor to SHA-1, and continues today to be widely popular and accepted as a robust cipher (Jena, 2023).

## Certificate Generation

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

A screenshot of a computer screen

Description automatically generated

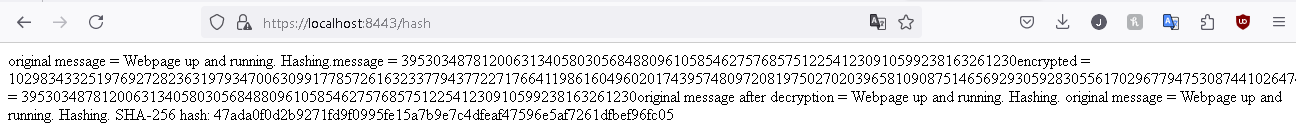
## Deploy Cipher

A close-up of a message

Description automatically generated

## Secure Communications

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



## Secondary Testing

Insert screenshots below of the refactored code executed without errors and the dependency-check report.

Below: Dependency check report

A screenshot of a computer

Description automatically generated

Below: Main application file, SslServerApplication.java

A screenshot of a computer program

Description automatically generated

Below: RSA.java

A screenshot of a computer

Description automatically generated

Below: SHA256.java

A screenshot of a computer program

Description automatically generated

Below: application.properties

A screenshot of a computer

Description automatically generated

Below: pom.xml

A computer screen shot of a program

Description automatically generated

## Functional Testing

Below: Main class executed without errors

A screenshot of a computer

Description automatically generated

Below: SHA ran without errors.

A screenshot of a computer

Description automatically generated

Below: RSA ran without errors.

A screen shot of a computer

Description automatically generated

## Summary

Upon execution of maven-install, I did find that there were several vulnerabilities listed by the OWASP dependency-checker. These vulnerabilities were identical both before and after the code refactoring. My refactoring added a secured RestController, which controls the site’s RESTful API and the /hash route. I also implemented both RSA and SHA-256, which have very long hash outputs that are extremely unlikely to collide and are reliable cryptographic functions. By using RSA for logins and other sensitive forms, and SHA for real-time communication, we can ensure a good balance between performance and security, and guarantee safe data transfer to our clients.

Continued necessary maintenance will mean bimonthly analyses of the static dependency checker OWASP, which can be performed by running the pom.xml under the maven-install configuration, to ensure our dependencies are up to date with no new vulnerabilities discovered.

## Industry Standard Best Practices

I opted to update our dependencies to eliminate vulnerabilities. While the current version of Spring Boot Framework is 3.3.0, using this version meant using SnakeYAML, one of its dependencies for which Spring Boot Framework relies on the vulnerable version 1.x. I instead opted for Spring Boot Framework version 3.1.3. This removed the vulnerabilities created by the outdated Spring Boot Framework and allowed the use of the safer SnakeYAML 2.2, the latest version of that dependency.

Below: Cleaned up all vulnerabilities.

A screenshot of a computer

Description automatically generated

The industry standard best practices for ensuring our application’s security revolve around input validation, principle of least privilege, having secure defaults configured, handling errors, performing code reviews, performing regular tests, patching our software as a live service, and updating our work frequently. What we’ve done to fulfill these goals is create temporary keys and certificates to verify our authenticity and keep things secure moving forward and implement two hash functions for packet transfer and communication that we can use going forward. Of course, I also used some degree of input validation in the implementation of these functions through the use of try and catch blocks, and ran extensive testing on my functions to ensure they work. I also ran the static dependency checker and verified that all vulnerabilities are patched. These actions will save us time and costs down the road and reduce our risks, and we’ll continue to do so by continuing SecDevOps.

**References**

Bhardwaj, R. (2023, November 23). A Definitive Guide to SHA-256 (Secure Hash Algorithms) » Network Interview. *Network Interview*. <https://networkinterview.com/sha-256-secure-hash-algorithms/>

Jena, B. K. (2023, August 29). *A definitive guide to learn the SHA-256 (Secure Hash algorithms)*. Simplilearn.com. https://www.simplilearn.com/tutorials/cyber-security-tutorial/sha-256-algorithm

Kelly, M. (2009) *The RSA Algorithm:  A Mathematical History of the Ubiquitous Cryptological Algorithm*, *swarthmore.edu*. Available at: <https://www.sccs.swarthmore.edu/users/10/mkelly1/rsa.pdf>.

Rivest, R. L. (n.d.). The Early Days of RSA -- History and lessons. In *ACM Turing Award Lecture* [Journal-article]. https://people.csail.mit.edu/rivest/pubs/ARS03.rivest-slides.pdf

Simmons, Gustavus J.. "RSA encryption". *Encyclopedia Britannica*, 29 Dec. 2022, https://www.britannica.com/topic/RSA-encryption.

Thakkar, M. (2022, April 18). *SHA 256 algorithm explained by a cyber security consultant - InfoSec Insights*. InfoSec Insights. https://sectigostore.com/blog/sha-256-algorithm-explained-by-a-cyber-security-consultant/