****

# 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 5](#_Toc102040760)

[4. Secure Communications 5](#_Toc102040761)

[5. Secondary Testing 6](#_Toc102040762)

[6. Functional Testing 9](#_Toc102040763)

[7. Summary 10](#_Toc102040764)

[8. Industry Standard Best Practices 14](#_Toc102040765)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **12/15/2024** | **Paulina Weaver** |  |

## 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

Paulina Weaver

## Algorithm Cipher

The encryption algorithm cipher that I recommend for Artemis Financial is SHA-256. SHA-256 is a secure hashing algorithm that takes an input of any size and converts it into a fixed 256-bit output, also known as a hash or checksum (Griškėnas, 2023). This hashing process is one-way, meaning once the data is transformed into a hash, it cannot be reversed to reveal the original input. SHA-256 is part of the SHA-2 family of cryptographic hash functions, which was introduced in 2001 by the National Security Agency (NSA) and the National Institute of Standards and Technology (NIST) (Jena, 2024). It was developed to address vulnerabilities in older hashing algorithms like SHA-1, which became susceptible to collision and brute-force attacks as technology advanced.

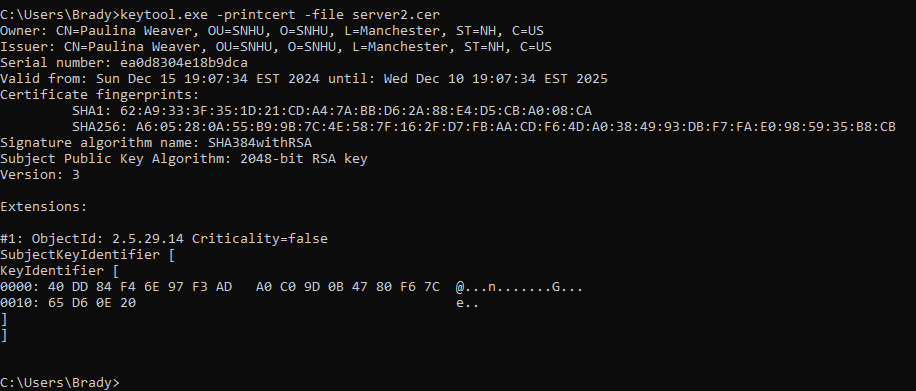
A key feature of SHA-256 is its ability to provide high-level security and protect the integrity of data. The algorithm divides input data into 512-bit blocks and processes them through 64 rounds of mathematical operations, including logical functions, bitwise shifts, and modular addition. The result is a unique 256-bit hash value that serves as a secure checksum. A checksum acts as a digital fingerprint for the input data, allowing systems to verify its integrity. If even the smallest change is made to the data, the resulting hash will be completely different, which makes SHA-256 highly reliable for detecting tampering or corruption.

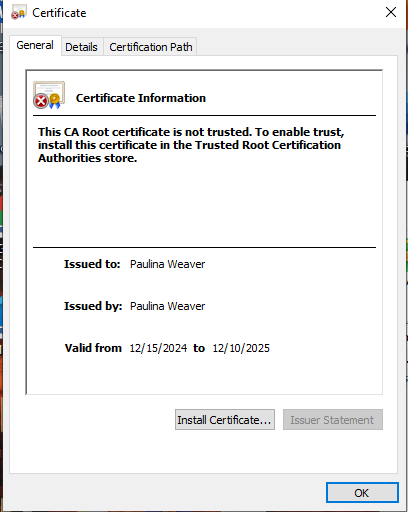
Unlike encryption algorithms that use keys to encode and decode information, hashing algorithms like SHA-256 are one-way processes. This means they do not rely on keys but instead generate a fixed output that cannot be reversed. While encryption can use symmetric keys (where the same key is used to encrypt and decrypt data) or asymmetric keys (a public-private key pair), hashing focuses solely on producing a collision-resistant checksum. To be secure, a hashing algorithm must meet two important criteria: it should not allow the original input to be determined from the hash, and it should avoid collisions, where two different inputs result in the same hash. By meeting these criteria, SHA-256 ensures that data integrity can be verified with confidence.

SHA-256 has become the standard for secure hashing algorithms due to its strong collision resistance and proven reliability. Its development was a response to the limitations of earlier algorithms like SHA-1, which are now considered insecure due to advances in computing power. Today, SHA-256 is widely trusted and used by government agencies, financial institutions, and technology companies to protect data integrity and ensure the authenticity of sensitive information (Jena, 2024).

## Certificate Generation

Insert a screenshot below of the CER file.





## Deploy Cipher

Insert a screenshot below of the checksum verification.

A screenshot of a computer

Description automatically generated

## Secure Communications

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

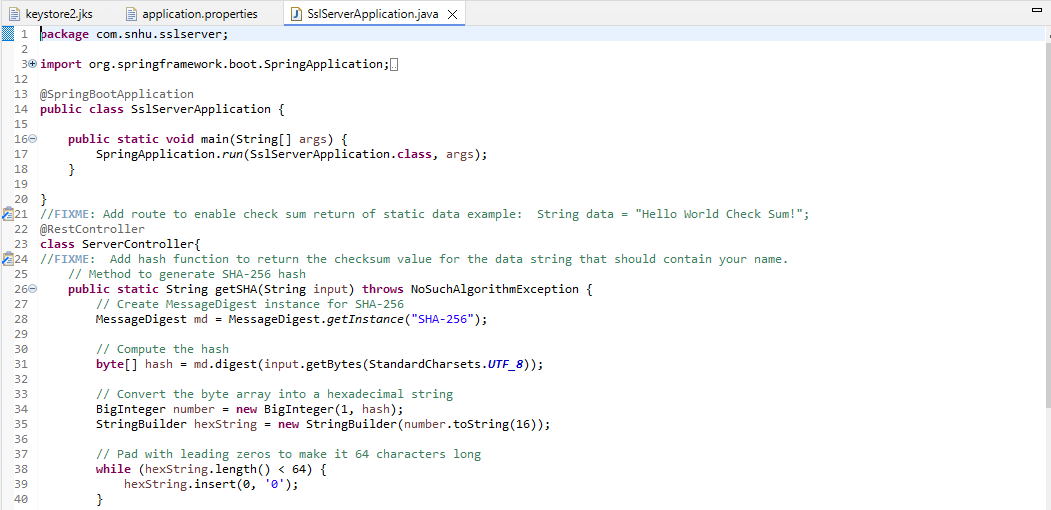
A screenshot of a computer

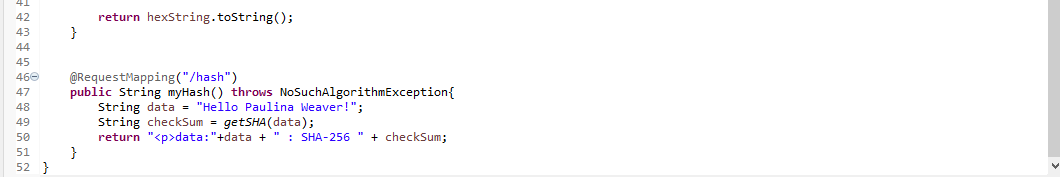
Description automatically generated

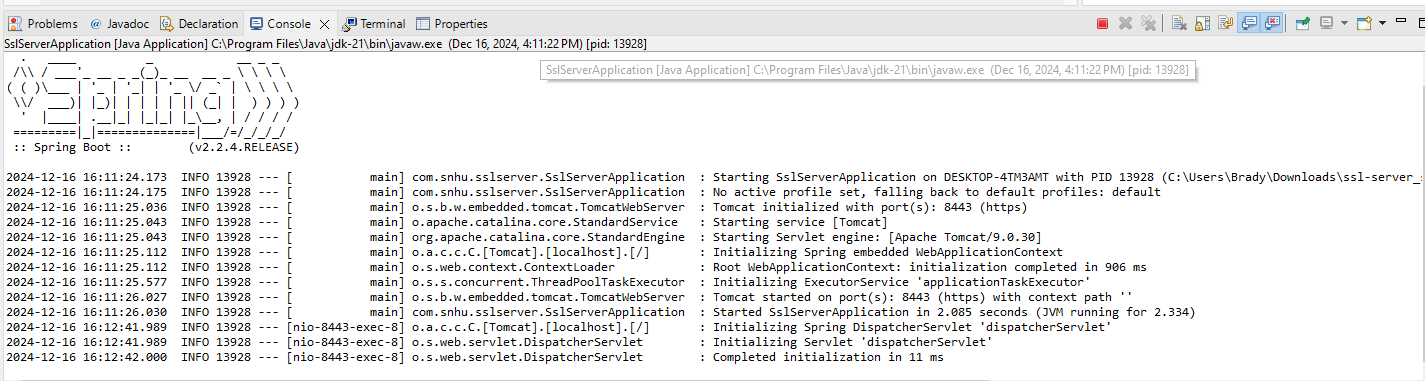
## Secondary Testing

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

SslServerApplication.java







Application.properties

A screenshot of a computer

Description automatically generated

Dependency-check prior to refactoring code:

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Dependency-check after refactoring:

A screenshot of a computer

Description automatically generated

A screenshot of a computer code

Description automatically generated

Pom.xml

A screenshot of a computer

Description automatically generated

## Functional Testing

Insert a screenshot below of the refactored code executed without errors.

sslServerApplication.java

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

Application.properties

A screenshot of a computer

Description automatically generated

Pom.xml

A screenshot of a computer

Description automatically generated

## Summary

The architecture of the application was reviewed to ensure secure communication over HTTPS. To achieve this, the server was configured to run on port 8443 with SSL encryption enabled using a self-signed certificate. A Java keystore (JKS) was created to store the certificate, and it was imported into Eclipse. I then updated the application.properties file with the required keystore configurations, including the keystore location, password, and alias. This setup ensures that SSL encryption is properly enabled, providing secure data transmission and reducing the risk of interception or tampering.

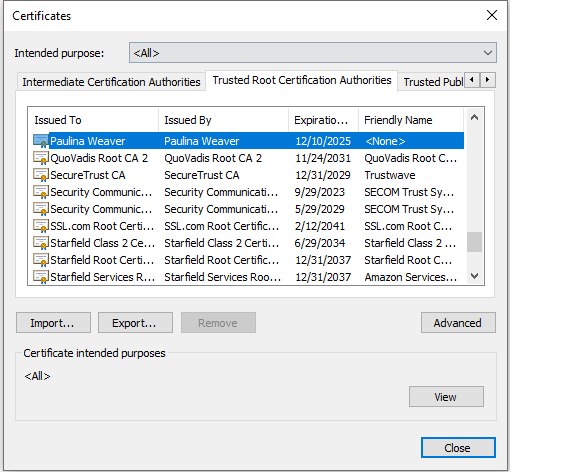
A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

While setting this up, I encountered an issue where my web browser did not trust the self-signed certificate. This is a known issue with self-signed certificates, as they are not recognized by default as trusted authorities. In an attempt to resolve this, I added the certificate to the “Trusted Root Certification Authorities” in Windows and imported it into the trusted certificates in Chrome. Despite this, the browser still did not trust the certificate. However, SSL encryption was still functional, ensuring secure communication between the client and server. Once a trusted certificate is used in a production environment, this issue will no longer arise.



A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

For APIs, the refactored code utilizes the Spring framework to establish secure API interactions. By implementing SSL encryption, the code ensures that API communication is protected, reducing the risk of man-in-the-middle attacks and ensuring that the data exchanged is secure. This improvement addresses potential vulnerabilities related to insecure API interactions.

The Cryptography layer was enhanced by integrating SHA-256 encryption for securely hashing sensitive input data. This ensures that sensitive information is never transmitted or stored in plain text, ensuring data confidentiality. The getSHA method was added to securely hash input data. The method uses the MessageDigest class from Java's java.security package to generate a SHA-256 hash of the input string. The input string is converted into a byte array using StandardCharsets.UTF\_8. Then, the MessageDigest.getInstance("SHA-256") method is called to create an instance of the SHA-256 algorithm. The digest method computes the hash of the byte array, returning the result as a byte array. This byte array is then converted into a BigInteger to represent the hash value as a numerical value, which is subsequently transformed into a hexadecimal string using a StringBuilder. Finally, the string is padded with leading zeros to ensure it is always 64 characters long, as required for a SHA-256 hash.

A screenshot of a computer code

Description automatically generated

Code error handling was added to ensure that any exceptions that may occur during the hashing process are properly managed. In particular, the getSHA method uses the NoSuchAlgorithmException to handle cases where the SHA-256 algorithm might not be available in the Java environment. This exception is thrown if there’s an issue with the cryptographic algorithm, which ensures the program does not crash unexpectedly and provides a mechanism to catch and handle the error.

A screenshot of a computer program

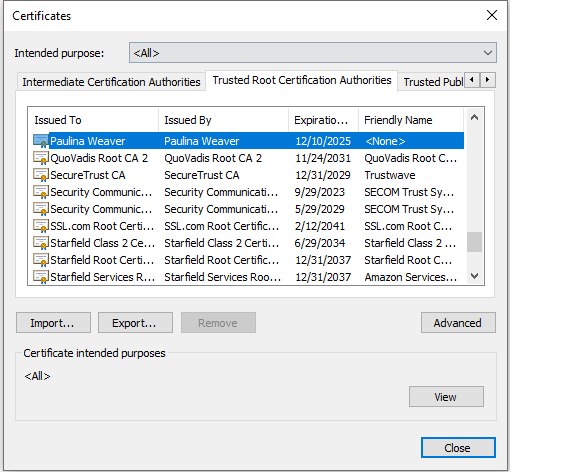
Description automatically generated

For Client/Server communication, the use of SSL encryption ensures that data exchanged between the client and server is encrypted, preventing eavesdropping and maintaining secure communication. This directly addresses potential security risks related to insecure data transmission.

Code Quality was improved by ensuring secure data transmission and the use of SHA-256 cryptography, both of which contribute to best practices for secure coding. The refactored code also incorporated Encapsulation by isolating the cryptographic function within the getSHA method, which securely handles the hashing process. This encapsulation ensures that sensitive data, such as user input, is processed securely without exposing it to unnecessary vulnerabilities.

A screenshot of a computer code

Description automatically generated



A screenshot of a computer

Description automatically generated

## Industry Standard Best Practices

Following industry-standard best practices for secure coding is crucial to protect the software application from known security threats. One important practice I implemented was the use of SSL encryption for secure communication (OWASP, n.d.). By setting up the application to run over HTTPS using a self-signed certificate and specifying the Keystore in the application.properties file, I ensured that all data transmitted between the client and server was encrypted. This mitigates the risk of man-in-the-middle attacks, where an attacker could intercept and manipulate sensitive data. Encrypting communication is a fundamental best practice to protect user information and maintain the confidentiality of data in transit.

Additionally, I implemented SHA-256 hashing for secure data storage and integrity verification. Using the SHA-256 algorithm, I ensured that sensitive data, like user inputs, could be securely hashed and verified. This is particularly important for verifying data integrity, such as ensuring that files or inputs have not been tampered with. While I used a static string for hashing in this example, in actual scenarios, this method is especially useful for securely storing passwords by hashing them before saving them in a database. This way, even if the database is hacked, the passwords remain unreadable.  Using well-known algorithms like SHA-256 helps avoid weak hash functions, which can lead to problems like hash collisions or brute force attacks.

Following these secure coding practices helps protect sensitive customer data, which is crucial for maintaining trust and compliance with data protection regulations. By securing data in transit and ensuring its integrity through encryption and hashing, the company can reduce the risk of data breaches and cyberattacks, which can have devastating consequences. Also, adopting secure coding practices shows that the company is committed to cybersecurity, which enhances the company’s reputation with customers, investors, and partners. As cyber threats continue to evolve, maintaining high-security standards helps prevent costly incidents and ensures the long-term success of the business by protecting its digital assets.

References

Griškėnas, S. (2023, June 20). *What is the SHA-256 algorithm?*. NordVPN. https://nordvpn.com/blog/sha-256/

Jena, B. K. (2024, November 16). *A Definitive Guide to Learn The SHA-256 (Secure Hash Algorithms)*. Simplilearn. https://www.simplilearn.com/tutorials/cyber-security-tutorial/sha-256-algorithm

OWASP. (n.d.). *Secure coding practices*. OWASP Secure Coding Practices - Quick Reference Guide . https://owasp.org/www-project-secure-coding-practices-quick-reference-guide/stable-en/02-checklist/05-checklist