****

# Practices for Secure Software Report

Table of Contents

[Practices for Secure Software Report 1](#_Toc132648303)

[Document Revision History 3](#_Toc132648304)

[Client 3](#_Toc132648305)

[Instructions 3](#_Toc132648306)

[Developer 4](#_Toc132648307)

[1. Algorithm Cipher 4](#_Toc132648308)

[2. Certificate Generation 5](#_Toc132648309)

[3. Deploy Cipher 5](#_Toc132648310)

[4. Secure Communications 6](#_Toc132648311)

[5. Secondary Testing 7](#_Toc132648312)

[6. Functional Testing 8](#_Toc132648313)

[7. Summary 9](#_Toc132648314)

[8. Industry Standard Best Practices 9](#_Toc132648315)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **04-16-2023** | **Charles Adkins** |  |

## 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

Charles Adkins

## Algorithm Cipher

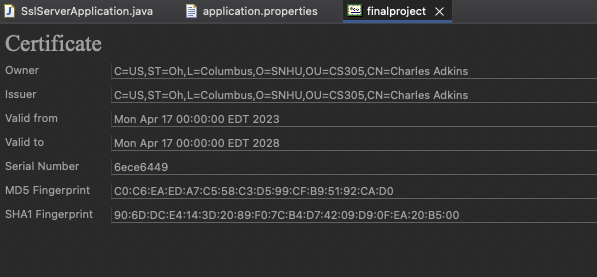
The SHA-256 algorithm is a widely used and secure cryptographic hash function that produces a 256-bit (32-byte) hash value (Jena, 2023). The cipher functions transform the plaintext to unreadable data, so that anyone who accesses the data will not be able to use it without the appropriate key.

The bit levels come into play when it comes to the key size, meaning the strength of the cipher and the duration by which it will take to both encrypt and decrypt the data. The data is broken into blocks of bits, each block containing 16 bytes, and each block is transformed by a sophisticated process and repeated several times, of which is determined by the key length (the longer the key, the more rounds of transformations are performed).

Adding random numbers or data, also known as salt, during the hashing or encryption is a method implemented to strengthen the encryption and to prevent hacking that utilizes dictionary attacks or other targeting methods. Symmetric keys are the same for encryption and decryption, but asymmetric keys involve one key for encryption and another for decryption.

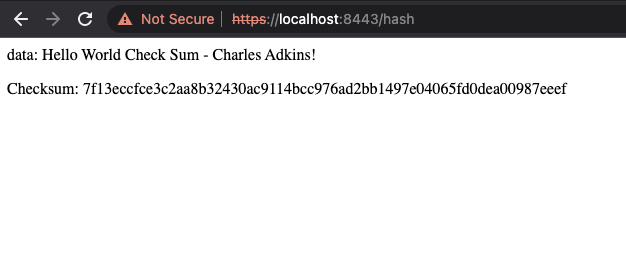
Encryptions date back into ancient Sparta cryptography, where soldiers would send secret messages during battle utilizing engraved letters on a leather strip. The letters meant nothing when the strip was unwrapped, but when wrapped around a specialized wooden rod, the message became legible. Encryption continued through the ages and into World War II, where mathematics was used to develop sophisticated codes. Today, given the ever-changing landscapes of technology, encryption has had to increase its level of complexity to guard against the skills of emerging threats. Tokenization is a technology currently being explored as it involves replacing sensitive data with non-sensitive data equivalence, which eliminates the need to transform the data itself. Tokenization increases security, reduces costs, lowers database requirements and allows for more flexibility in terms of development and innovation.

## Certificate Generation



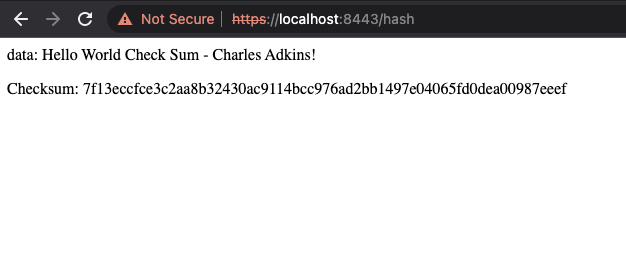
**Figure 1: Certificate generated and displayed in Eclipse IDE.**

## Deploy Cipher

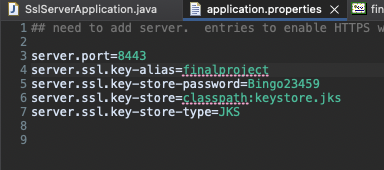


**Figure 2: Cipher Checksum executed successfully.**

## Secure Communications

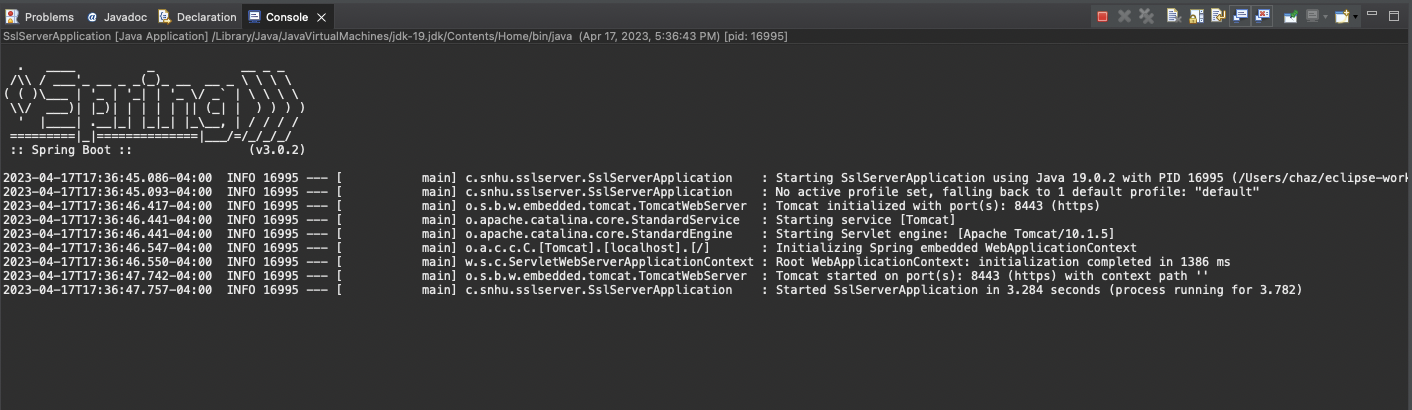


**Figure 3: Circumventing efforts were applied to force the keychain to trust the validity of the self-signed certificate, but it was rejected by the browser.**



**Figure 4: The above values were set to correspond with the self-signed certificate generated in the command line.**

## Secondary Testing

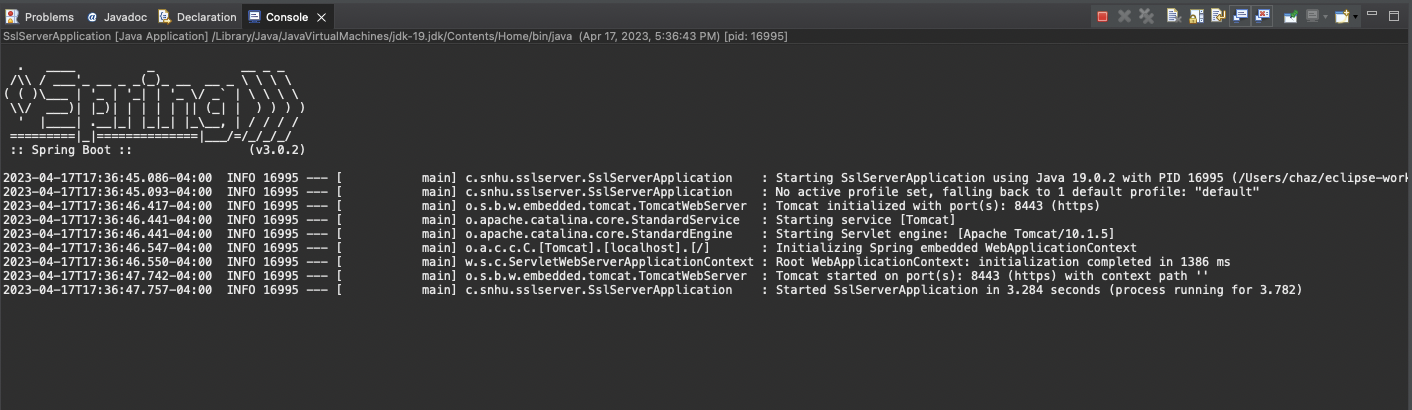


**Figure 5: The application was successfully refactored and executed, as shown by the Spring framework console messages above.**

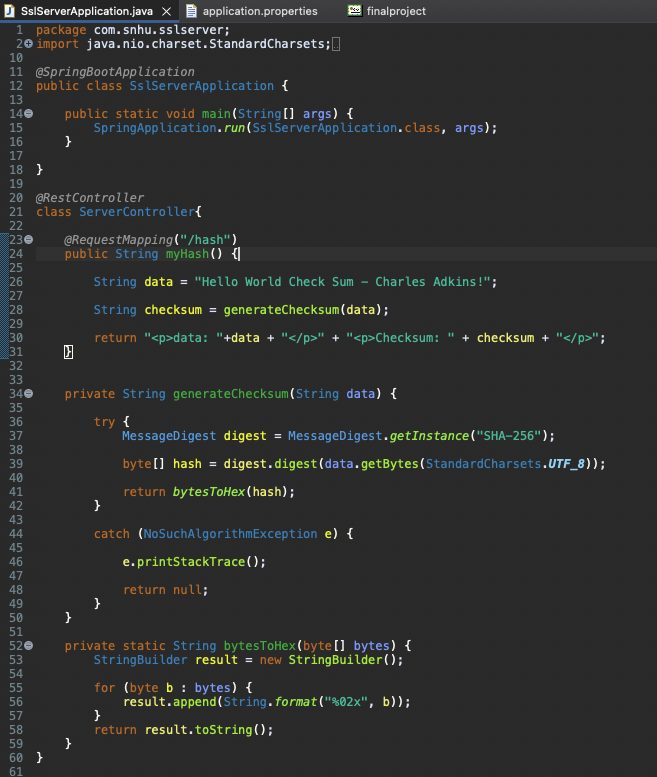


**Figure 6: The OWASP Dependency-Check Report is shown above, with no new vulnerabilities introduced to the application, through the refactoring of the code.**

## Functional Testing



**Figure 7: The Eclipse console window shows the messages of a successful build after refactoring the code.**



**Figure 8: The above showcases the refactored code of the SslServerApplication.java file.**

## Summary

The refactored code takes a simple data string and performs a checksum function on it. The checksum function incorporates a digest and utilizes the SHA-256 algorithm and a hash to cipher the simple data string. This process makes it possible to take data and secure it, obscuring its value from malicious threats. The code utilizes principles such as cryptography, secure error handling, coding practices and data structures.

## Industry Standard Best Practices

I followed industry standard best practices by utilizing well established and trusted data structures, algorithms and error handling code architecture. Evaluating the security of the application through the vulnerability dependency report helped me identify current vulnerabilities and draw conclusions as to how those could jeopardize the security of the application and its data.

**References**

Jena, Baivab Kumar. (2023). A Definitive Guide to Learn The SHA-256 (Secure Hash Algorithms). Retrieved April 14, 2023 from https://www.simplilearn.com/tutorials/cyber-security-tutorial/sha-256-algorithm

Thales. (2023). A Brief History of Encryption (And Cryptography). Retrieved March 25, 2023 from <https://www.thalesgroup.com/en/markets/digital-identity-and-security/magazine/brief-history-encryption>