****

# 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** | **06/16/2023** | **Greg Isajewicz** |  |

## 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

Greg Isajewicz

## Algorithm Cipher

Artemis Financial wants to modernize their operations and an important part of this is implementing appropriate security and encryption measures. The encryption algorithm cipher I recommend for implementing a checksum verification is SHA-256 encryption. This cryptographic hash function takes an initial input of an arbitrary length and converts it into a 256-bit hexadecimal hash for an instance of MessageDigest. The large 256-bit output size of a SHA-256 hash makes it highly unlikely that two different inputs will produce the same hash value, thus avoiding collisions. Since the length of the initial input is not determinable from the hash output, the chances of a successful brute force attack are unlikely. Sha-256 can also be seeded with a random character or number string (salt) appended to the initial input which will ensure that even if the exact initial input is guessed, it will still not return the same hash value.

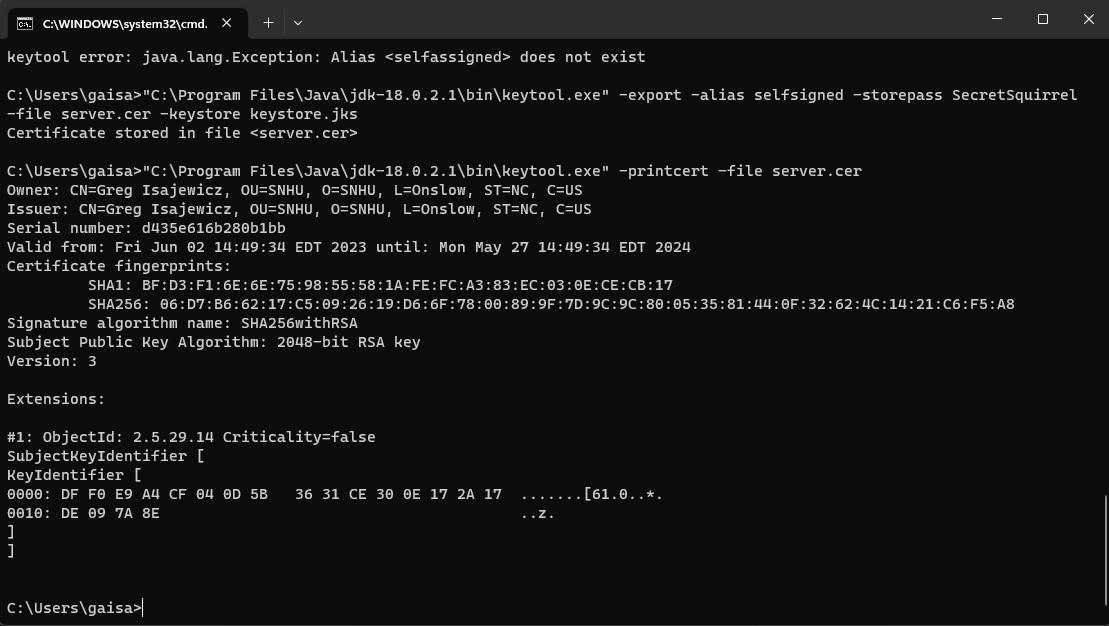
Random numbers are used in encryption algorithms to generate secure cryptographic keys. Understanding that random number generation by a computer program is actually the generation of pseudo-random numbers by algorithm, it is important that the method of pseudorandom number generation be implemented in such a way that the process cannot be easily repeated or guess by a bad actor attempting to crack the key.

Symmetric keys are used in symmetric cryptography, wherein the same encryption key is used for both encryption and decryption. The efficacy of symmetric keys relies on the keys remaining secret. If a bad actor were to get ahold of a key in a symmetric encryption system, they could use it to decrypt and expose sensitive data in transit. Non-symmetric keys on the other hand, use different keys for encryption and decryption, known as asymmetric or public key cryptography.

The history of encryption algorithms dates back to ancient times, where techniques like Caesar ciphers were used to encode messages. The advent of computers saw a major advancement in encryption and the creation of the Data Encryption Standard (DES) by the tech company IBM in 1973. DES remained the national standard in the U.S. until it was cracked in 1997 and was later replace by the Advanced Encryption Standard (AES) in 2000 (Thales, 2023). Encryption algorithms have advanced through time to meet the needs of the day. As technological and mathematical advances are made, the need for more intricate and advanced algorithms continues.

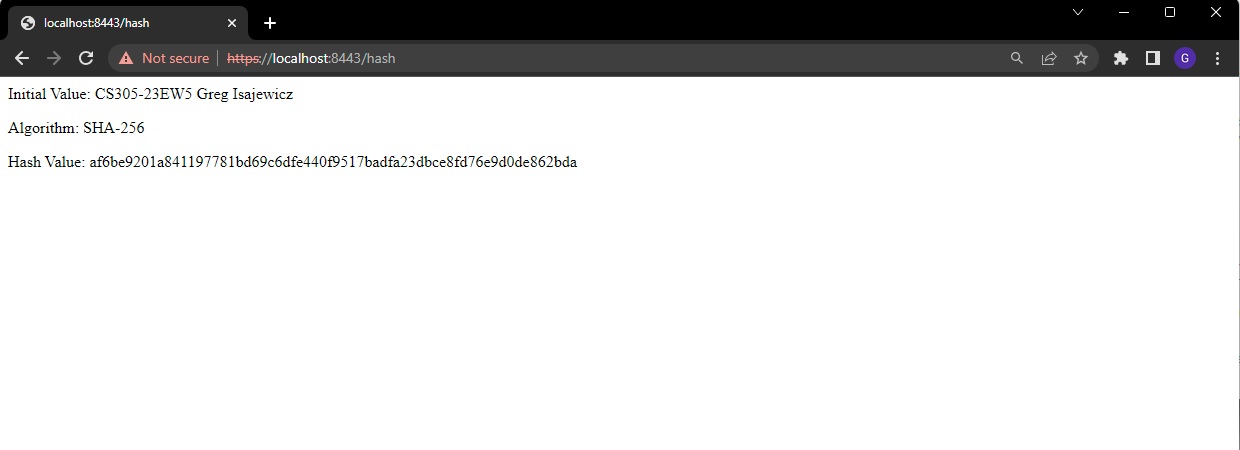
## Certificate Generation

Insert a screenshot below of the CER file.



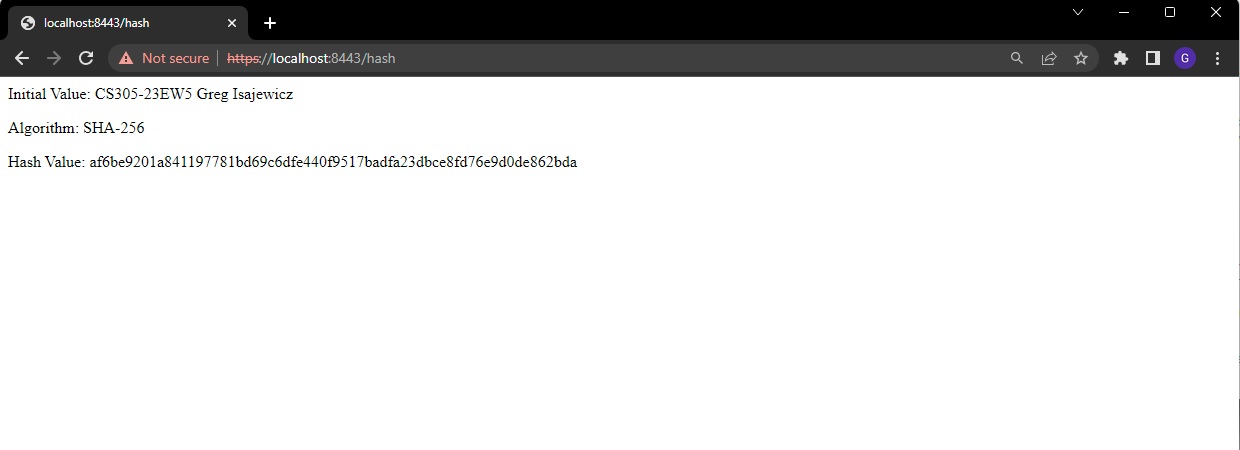
## Deploy Cipher

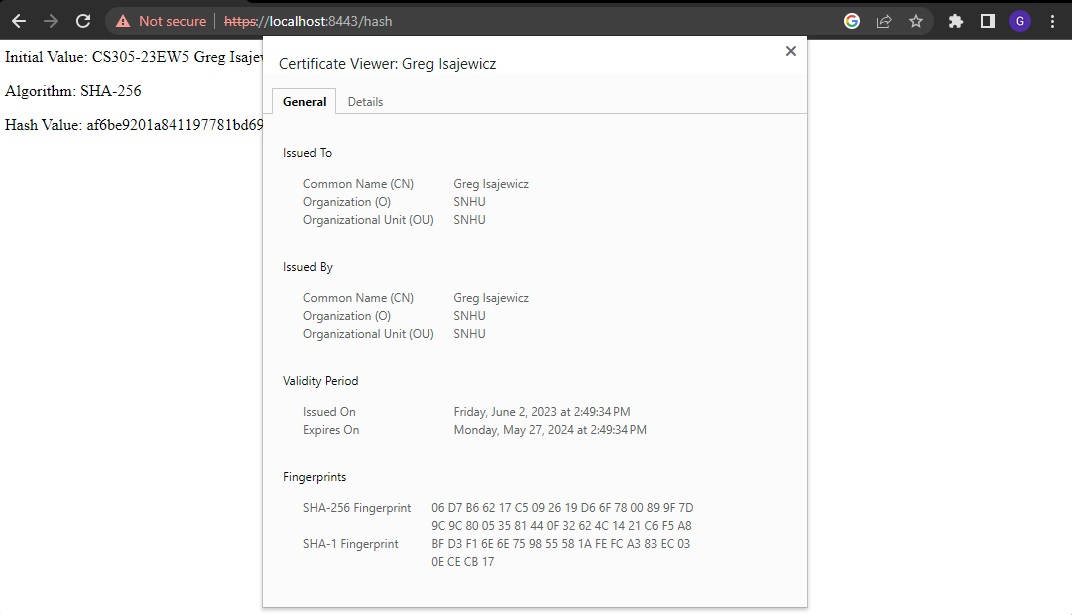
Insert a screenshot below of the checksum verification.



## Secure Communications

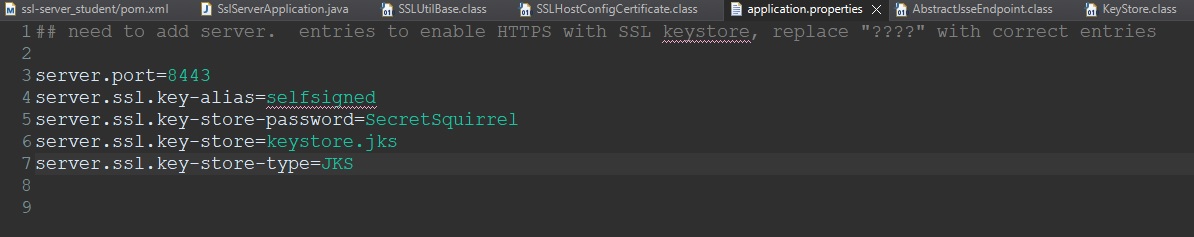
Insert a screenshot below of the web browser that shows a secure webpage.

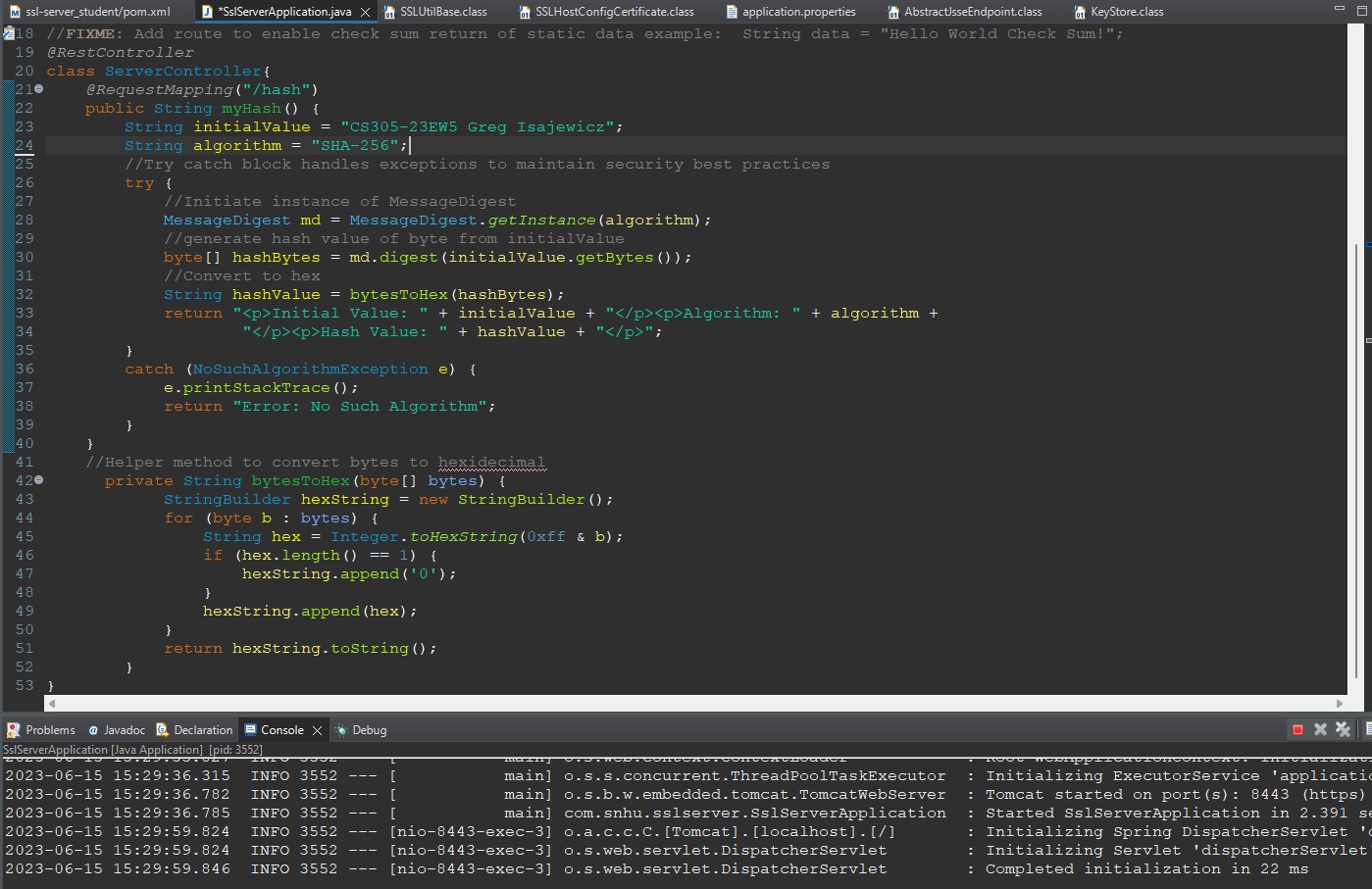


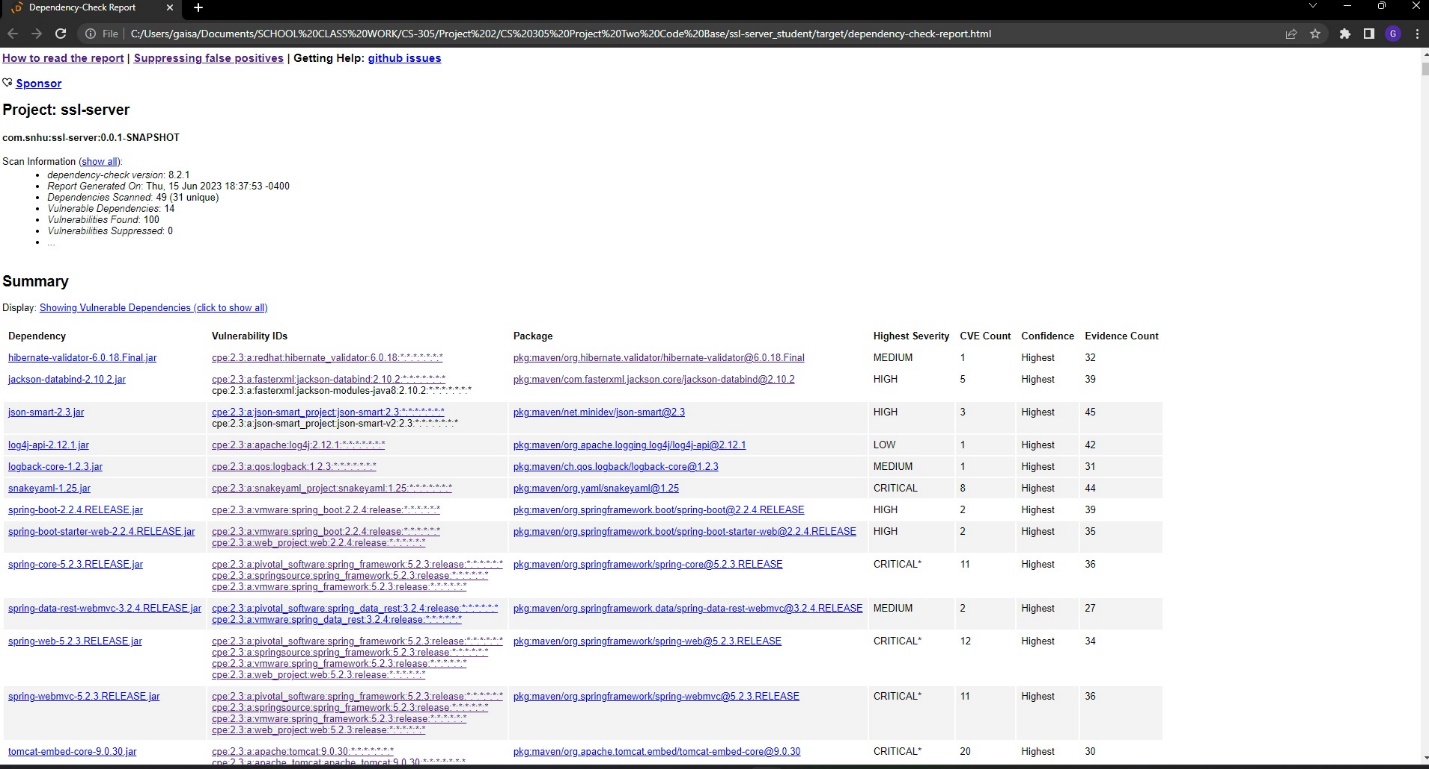


## Secondary Testing

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

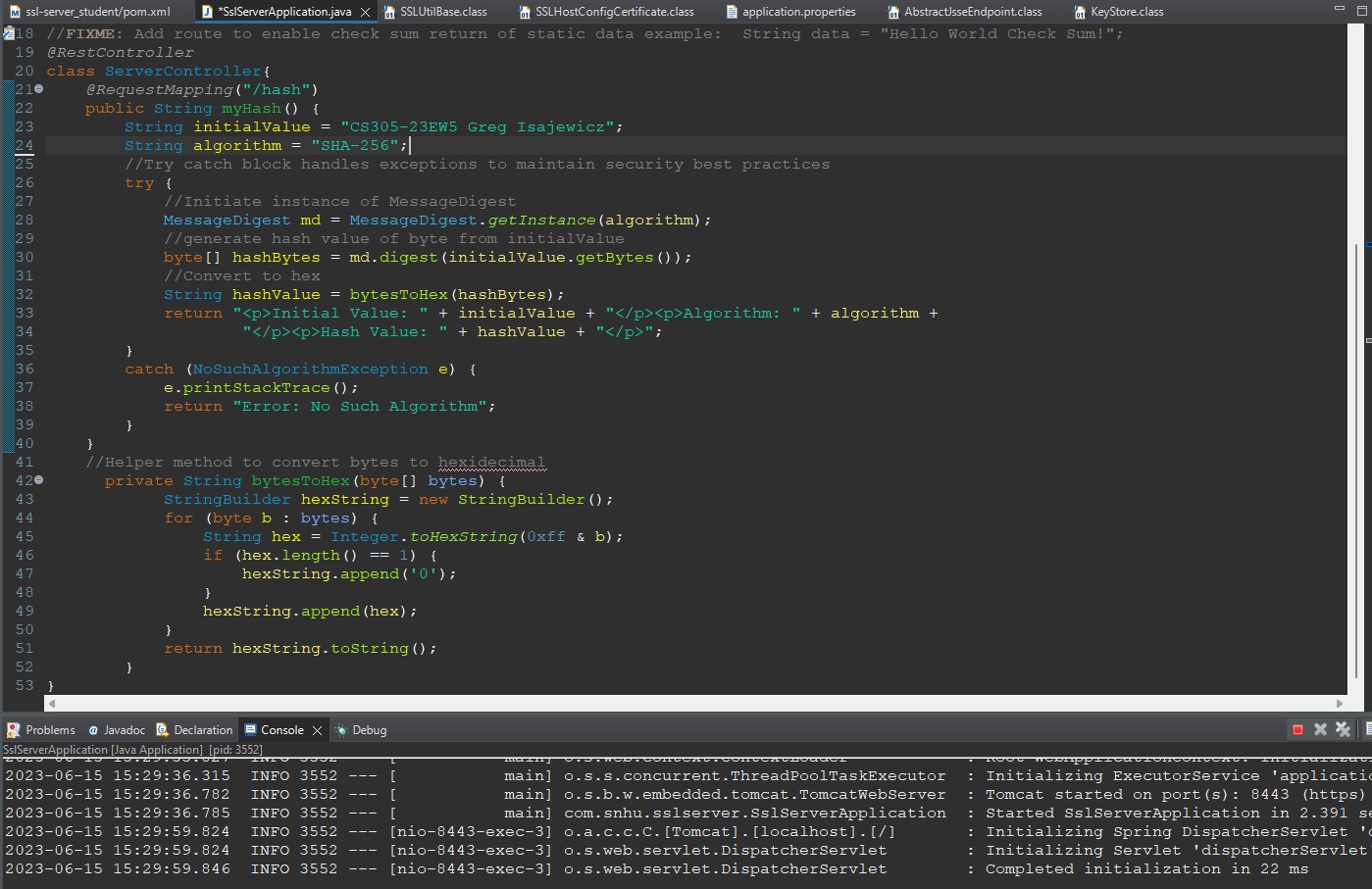






## Functional Testing

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



## Summary

My process for adding layers of security to the software application was to first identify the major threats to consider, then consult the Vulnerability Assessment Process Flow Diagram as I added the relevant security layers. Next, I conducted manual and statis reviews of the code and its dependencies in search of vulnerability that could be mitigated.

One area of the Vulnerability Assessment Process Flow Diagram that I addressed is cryptography. The code has been refactored to incorporate a SHA-256 hash cipher which will help to ensure integrity of the data in the application. While incorporating this particular security layer, I also considered Code Error by integrating a try/catch block to prevent data leakage in the event of an exception. The hash process was incorporated under a RestController to mark the class as a request handler and declare the REST API.

Additionally, the code has been refactored to be used with HTTPS protocol. This provides another layer of security to prevent tampering with data in transit. The Refactored code includes the necessary configurations including key alias, key store password, and key store type, to establish and enhance secure communications. Incorporation of a keystore containing SSL certificates also adds a layer of integrity to the application.

## Industry Standard Best Practices

Applying industry standard best practices for secure coding helps to assure the security of the code being developed and the larger application. It can also help to reduce vulnerabilities and save time and cost for the company by addressing security early and continually to minimize the need to go back and make fixes later. An additional higher-level benefit for the company is the general prevention of security vulnerabilities leading to breaches. A good reputation is a valuable commodity for businesses, especially those dealing in people’s sensitive information and suffering a data breach can be detrimental to that reputation. Applying secure coding practices also helps to ensure that a company is in compliance with applicable regulations.

While refactoring this code I attempted to preserve or implement secure coding practices at several points. Firstly, I implemented a try/catch block to handle exceptions. Exceptions pose a risk of leaking sensitive error information if left unhandled, so with a try/catch block we can mitigate this risk by giving the program direct instructions on how to react to such an instance.

Use of the bytesToHex method ensures consistent secure conversion of the bytes string, preventing outlier errors that may occur as a result of custom implementation. Additionally, within this method is the use of StringBuilder to construct the string rather than concatenating them directly. This secure coding practice ensures proper escaping and sanitization of data and in a larger program could help to prevent cross-site scripting.

## Resources

Thales Group. (2023, Feb 01) *A Brief History of Encryption (and Cryptography).* Thales Group.

h[ttps://www.thalesgroup.com/en/markets/digital-identity-and-security/magazine/brief-](ttps://www.thalesgroup.com/en/markets/digital-identity-and-security/magazine/brief-%20) history-encryption