****

# 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** | **18 August 2024** | **Michael Devin Moore** |  |

## Client



**Developer**

Michael Devin Moore

## Algorithm Cipher

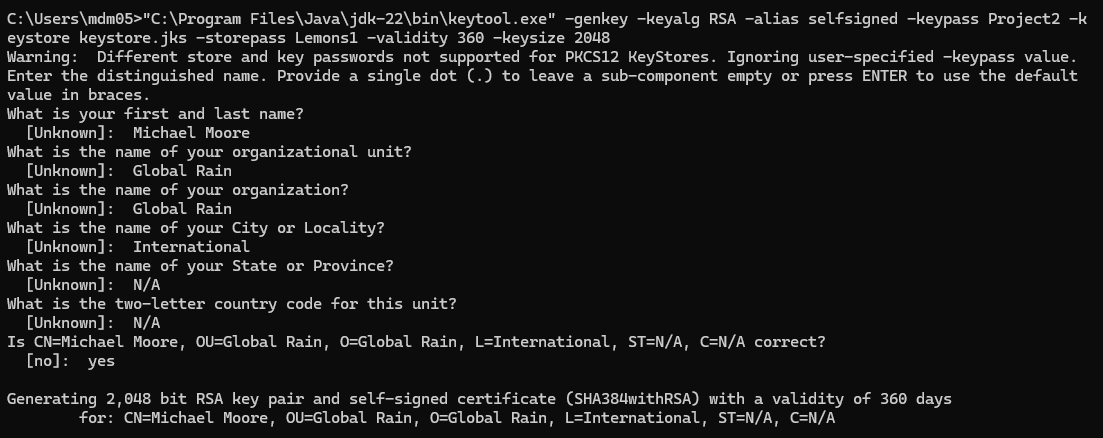
SHA-256 is the hashing algorithm recommended for Artemis Financial and keeping the data it transmits safe. SHA-256 goes through 64 iterations of calculations to manipulate the original text, whatever it may be, into a hashed text, or encrypted string of characters. The hashing process is not just a simple calculation repeated on input data repetitively. It can consist of character swapping, shifting, and other changes alongside mathematical permutations to further scramble data as it gets encrypted. The data sent into the algorithm, no matter the original size, is broken down into 512 bits and is encrypted with a 256-bit key, hence SHA-256. Ciphers that are above 128 bits in length are typically not breakable by hackers and their machines. The immense amount of computer power required to iterate through that many bits quickly is not feasible yet, making 128 bit and above ciphers recommended for use vice earlier ciphers that are less than 128 bits.

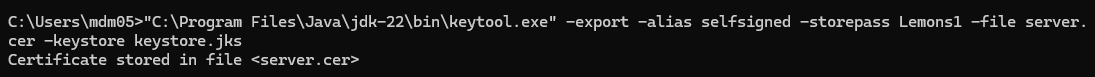
Algorithms can be either symmetric or asymmetric. Symmetric cryptography utilizes a single secret key to encrypt and decrypt data. Asymmetric cryptography utilizes a private key to sign data by a sender and then a public key to decrypt it. SHA-256 can be utilized in either scenario based on requirements of the software it is used in conjunction with. Setting it up as symmetric is easier and typically offers faster performance in the real world, but the asymmetric encryption offers a higher level of security with the drawback of being slower. Asymmetric encryption is more secure by the usage of a public and private key pair. The public key is provided to allow anyone to encrypt data in the SHA-256 method, with the private key being necessary to decrypt and make use of the sensitive information. Therefore, the ability to decrypt lies with the few that have the private key. I recommend an asymmetric setup to allow the highest possible security with this algorithm.

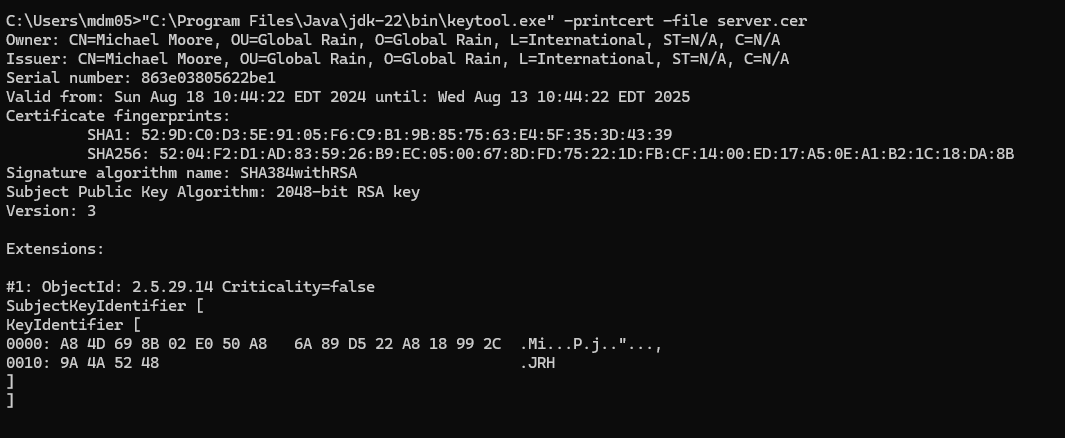
SHA is recommended for use twice in asymmetric form as I used it for the program. I utilized it in the web server portion of the application as SHA-256 to hash data as an example of how transmitted data could be encrypted and handled. SHA was also utilized again as SHA-384 with RSA for generation of the certificate described below.

## Certificate Generation

## Below are three screenshots showing my creation of a selfsigned certificate, it being stored in the certificate file, and then a printout of the certificate from the certificate file. The main takeaway from this is the addition of a certificate for use with to the application, thus web page. The certificate interaction provides an added layer of security by showing that credentials provided by a website are either trusted or expected. The self-generated certificate could be distributed as necessary, or sent to a certificate authority to get it and the website verified.

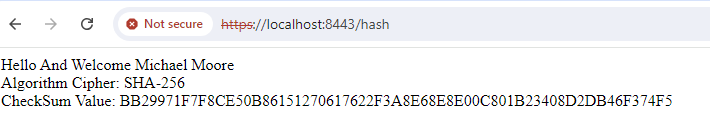






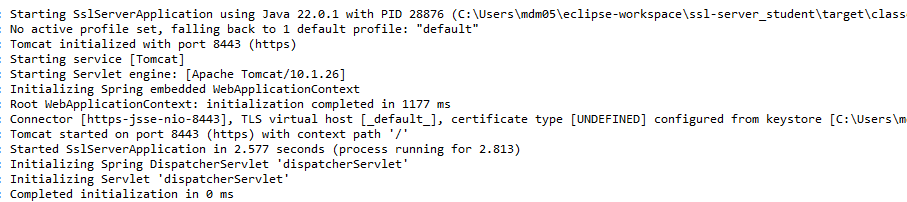
## Deploy Cipher

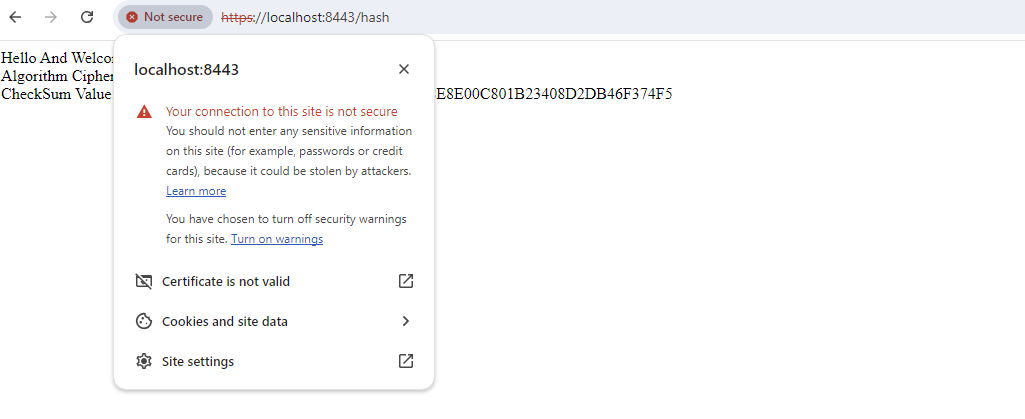
The below screenshot showcases my running of the website before incorporating any changes to ensure that it is a https secure site. The page shows the phrase I encrypted utilizing a SHA-256 algorithm. And then the checksum is the hash value of the encrypted phrase in 512 bits, represented as 64 characters.



## Secure Communications

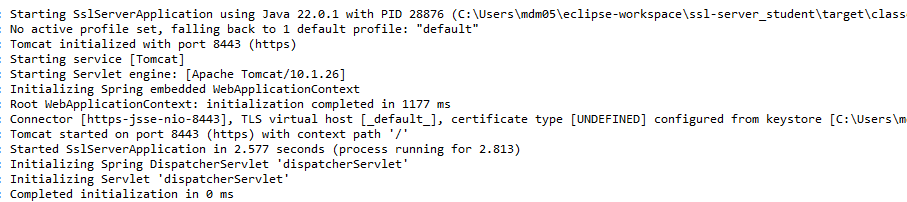
Below are screenshots from Eclipse and the browser where my webpage is running. The eclipse screenshot shows that the application started and is running as a Spring application. It specifically shows that it is running on port 8443 with https active. My browser, however, did not ever display the website as a safe https site. It kept saying the certificate was not valid. I attempted to add the certificate to trusted certificate lists in various locations on my computer, but could not resolve this invalid certificate error.

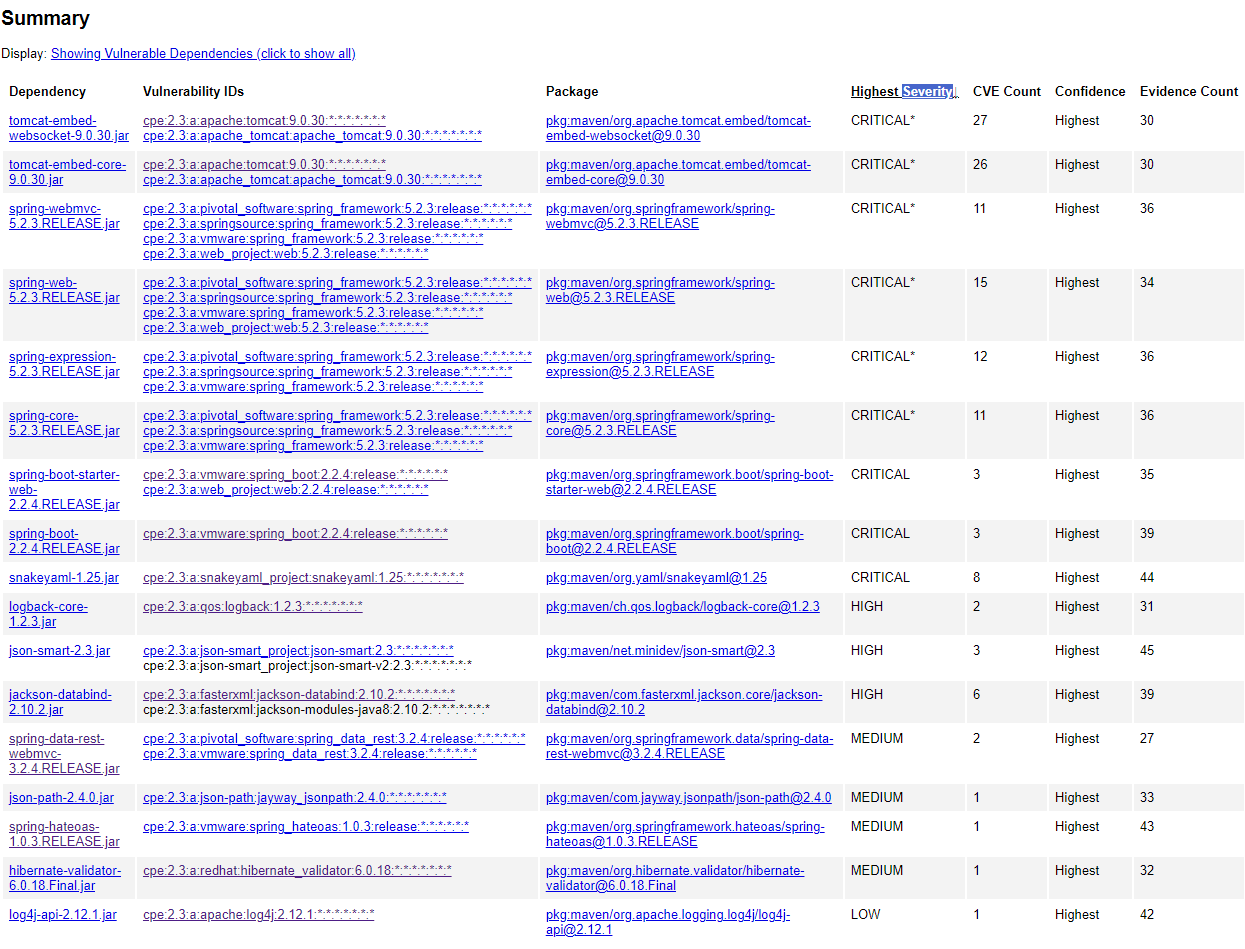




## Secondary Testing

## Below are two screenshots show my code executing with no errors and then a shot of the dependency check results. The report indicates 17 dependencies with vulnerabilities, none of which were caused by the code that I added to provide encryption capabilities or the use of SSL for the HTTPS web page.

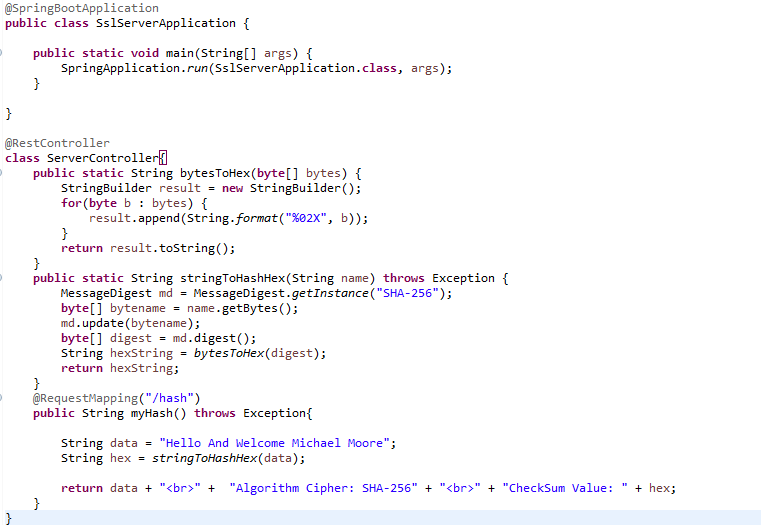




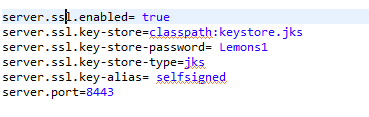
## Functional Testing

## The below screenshots are the additions I made to the program to accomplish the SHA-256 encryption of data that the web page can handle, and then the SSL capabilities for a secure webpage. The code running without errors was shown above.

SslServerApplication.Java



Application properties



## Summary

I encrypted the phrase Hello and Welcome Michael Moore and showcased the SHA-256 encrypted version of this. This allows for information to be encrypted as required for Artemis to conduct business. I added security by creating my own SHA-384 with RSA certificate that would then need to be distributed to users to access and trust the Artemis Financial website. The application properties were configured to utilize the keystore and certificate I generated to establish an SSL secured website, but I did not figure out how to make the browser accept my created self-signed certificate as trusted.

Artemis Financial would benefit the most from having good cryptography and code quality, as I provided in my refactoring of their code. In terms of code quality, the refactored portions of the code introduced no new vulnerabilities but did indeed contribute to the overall security of the program by added cryptography capabilities. The encryption capabilities with the SHA-256 algorithm introduced via my refactoring shows that input data, data in transit, or stored data can be converted into a safe state as a hashed value to be unraveled for use later. The certificate I generated for use in conjunction with a web page allows for users to know they are accessing the correct website via the certificate. The handshake that is certificate authentication and verification allow a user, via their browser, to know their information is safe within the confines of the legitimate webpages of Artemis’s application. This certificate was utilized to create a HTTPS secure web application by adjusting the application to utilize the keystore to recognize the generated certificate.

## Industry Standard Best Practices

Selfsigned certificate algorithm utilized a SHA-384 with RSA encryption. This generated a certificate of 2048 bits that would be impossible to crack in today’s computers to replicate due to the size the algorithm deals with. Information on the Artemis Financial website was shown to utilize a SHA-256 algorithm to convert data into a more secure format that cannot be easily used if stolen. This capability can be translated to any portion of the application that needs to make data secure. Security is a high priority for Artemis, and the higher above 128-bit encryption an algorithm goes, the safer it ultimately becomes. The dependency report shows no new vulnerabilities introduced with the refactoring of the program I performed to introduce data encryption with SHA-256, or the making of the webpage to be safe as a HTTPS page with the handshake performed from certificate to keystore. Vulnerabilities that exist in the program at this point come from dependencies that are simply out of date and many can be rectified by updating to versions that are not as vulnerable, or in many cases the latest versions and then mitigating the effects of the latest vulnerabilities.