****

# CS 305 Project Two

**Practices for Secure Software Report**

Table of Contents

[CS 305 Project Two 1](#_Toc85378444)

[Document Revision History 3](#_Toc85378445)

[Client 3](#_Toc85378446)

[Developer 4](#_Toc85378447)

[1. Algorithm Cipher 4](#_Toc85378448)

[2. Certificate Generation 5](#_Toc85378449)

[3. Deploy Cipher 5](#_Toc85378450)

[4. Secure Communications 6](#_Toc85378451)

[5. Secondary Testing 6](#_Toc85378452)

[6. Functional Testing 7](#_Toc85378455)

[7. Summary 7](#_Toc85378456)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| 1.0 | 10/17/2021 | Pomai Ahlo | All sections completed |

## Client



## Developer

Pomai Ahlo

## 1. Algorithm Cipher

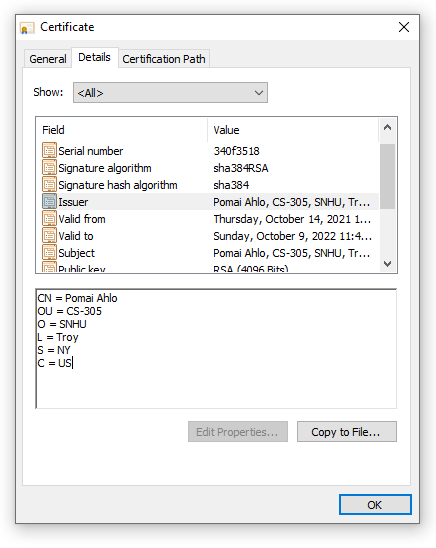
The AES-256 cipher will be used for Artemis Financial. AES is the encryption algorithm of choice for financial institutions around the world. AES is a symmetric cipher, meaning that the same key is used for encrypting and decrypting data. Compare that to asymmetric cryptography, where there is a private key known by the sender and a public key open to anyone.

Cryptography dates back to Egypt in around 1900 BC, where unusual hieroglyphics symbols were used in place of the regular ones. In 100 BC, Julius Caesar is credited for the first use of encryption by using a substitution cipher. During the 16th century, Blaise de Vigenere designed the first cipher that used an encryption key. Fast forward to the 1970s, where IBM designed a cipher called Lucifer. Lucifer was generally accepted and became known as the Data Encryption Standard (DES). DES was eventually broken, and in 2000, a cipher called Rijndael was accepted by National Institute of Standards and Technology and became the Advanced Encryption Standard (AES). While that is one of the most widely used, other encryption algorithms have also been developed, each with their own strengths and weaknesses. A few examples include Blowfish, ECIES, and RSA

Artemis Financial is a financial institution that will be handling a lot of sensitive data, thus security is a more important than speed, making AES a great choice for the company.

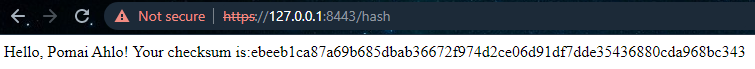
## 2. Certificate Generation

A self-signed certificate was created, and its information is shown in the screenshot below.



## 3. Deploy Cipher

The encryption algorithm cipher used for this application is SHA-256. The code was refactored to add this and encrypt “Pomai Ahlo” when you open https://127.0.0.1:8443/hash.



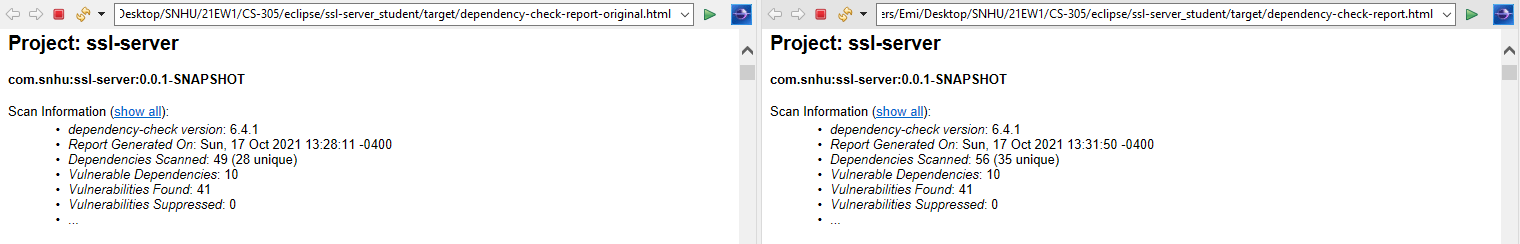
## 4. Secure Communications

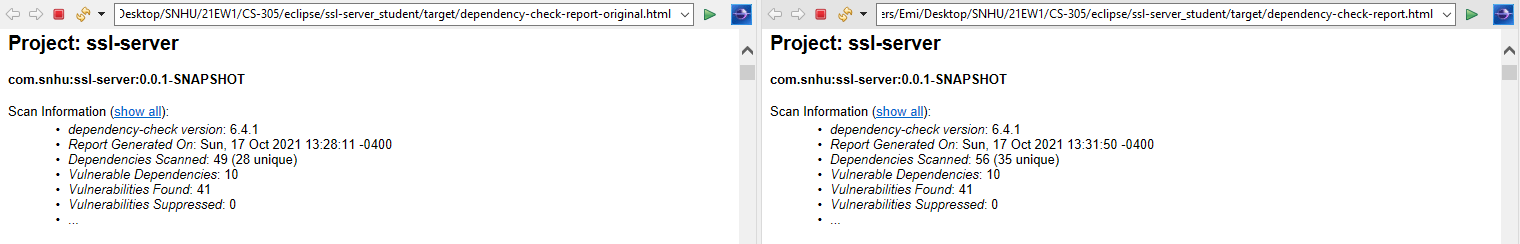
The code was refactored to convert HTTP to HTTPS while working on the previous section. In this screenshot, the certificate was remade, this time with a Subject Alternative Name. This was done in order to get the web browser to recognize the site as secure. Below is a screenshot of the web browser that shows a secure webpage.



## 5. Secondary Testing

The first screenshot shows the original report, while the second screenshot shows the report run with the refactored code. Dependency Google guava was introduced to this code. It originally introduced vulnerabilities, but updating it to its most current version mitigated these vulnerabilities.



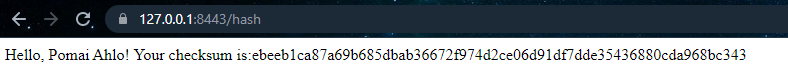


## With the update, the code still works.

## 

## 6. Functional Testing

My code did not have any logical or syntactical errors, and at this point, no more security vulnerabilities were introduced.



## 7. Summary

Refactoring the code addressed the following areas of security:

* Cryptography: Cryptography includes the use of encryption. My name is encrypted in this project, so by definition, the area applies.
* Client/Server: This project is the server side of the Artemis Financial application.
* Code Quality: The refactored code required the most up to date version of the used dependency in order to mitigate introduced vulnerabilities.

Artemis Financial needs to follow best practices for maintaining (and improving) the current security of the software application. The easiest and arguably among the most important is keeping dependencies up to date. If new code is written that introduces a dependency, the dependency should be checked to ensure it doesn’t bring new vulnerabilities. Another practice involves the safe handling of untrusted data. If and when input is needed, it should be validated. This could mean whitelisting special characters and setting a limit to the length of the input. Where possible, the input should be parameterized to prevent injection attacks (Manico and Detlefsen, 2014).

References

Manico, J. & Detlefsen, A. (2014) *Iron-Clad Java: Building Secure Web Applications.* Oracle Press.

https://learning.oreilly.com/library/view/iron-clad-java/9780071835886/

Sidhpurwala, H. (2013). *A Brief History of Cryptography.*

https://access.redhat.com/blogs/766093/posts/1976023