****

# CS 305 Project Two

**Practices for Secure Software Report**

Table of Contents

[Document Revision History 3](#_Toc33111302)

[Client 3](#_Toc33111303)

[Instructions 3](#_Toc33111304)

[Developer 4](#_Toc33111305)

[1. Algorithm Cipher 4](#_Toc33111306)

[2. Certificate Generation 4](#_Toc33111307)

[3. Deploy Cipher 4](#_Toc33111308)

[4. Secure Communications 4](#_Toc33111309)

[5. Secondary Testing 4](#_Toc33111310)

[6. Functional Testing 5](#_Toc33111311)

[7. Summary 5](#_Toc33111312)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **02/15/22** | **Joel Gomez Ruiz** |  |

## Client



## Instructions

Deliver this completed Practices for Secure Software Report documenting your process for writing secure communications and refactoring code that complies with software security testing protocols.

Respond to the steps outlined below and replace the bracketed text with your findings in your own words. If you choose to include images or supporting materials, be sure to insert them throughout.

## Developer

Joel Gomez Ruiz

## 1. Algorithm Cipher

Our client requires their application to be complaint with all the necessary security measures to ensure their customers’ personal data and financial information is protected from being exposed. To do that, encryption is the best way to make data unreadable through a variety of steps, so even if someone eavesdropping was able to see it, it would not be possible for them to determine its actual content. With that principle, several algorithm ciphers have been created and some have been proven to be more secure than others. Considering that our client is managing very important and sensitive data, it is highly advisable that they use **AES – 256**, a cipher that uses a 256-bit long symmetric key. A symmetric key means that when one side encrypts the data using a private key, the recipient side uses the same private key to decrypt the data to its original state. On the other hand, an asymmetric key is seen when the one requesting the data sends a public key to the sender, which then encrypts the data with that key to then send the data to the recipient, who can only decrypt it with its private key. The public key determines how the information will be encrypted and produces a result that is impossible to decrypt with the same key, only the private key held by the recipient is able to decrypt it. Using this algorithms can be highly secure but also inefficient if larger amounts of data are required to be sent.

Furthermore, other considerations should be made as to how passing information should be validated. One way to ensure the information received from another system is by validating its hash value. This hash value is the product of executing a hash function, which takes a file of any size and transforms it into a fixed-length string of letters and numbers. This string is unique to any file and almost impossible to be replicated by a different file. Commonly, the best algorithm to produce a hash value is SHA – 256, which produces a string of 256 bits in size. The longer the string the less likelihood of an intruder to find a file that maps to that hash value (256 bits means 2256 possible combinations).

There have been several algorithm ciphers created decades ago, and while those ones were highly secure for their time, nowadays they have turned out to present several vulnerabilities, and easier target to be cracked by hackers by brute force as current technology is capable to iterate very quickly to decipher keys created by older versions of algorithm ciphers.

## 2. Certificate Generation

Graphical user interface

Description automatically generated

## 3. Deploy Cipher

* Cheksum verification:

Graphical user interface, text, application, Word

Description automatically generated

## 4. Secure Communications

* **https://localhost:8443/hash** shows there is a secure connection

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

## 5. Secondary Testing

Graphical user interface, text, application, email

Description automatically generatedRefactored code executed without errors

Graphical user interface, text, application

Description automatically generated

Dependency check report

Graphical user interface, text, application, email

Description automatically generated

A screenshot of a computer

Description automatically generated

## 

## 6. Functional Testing

The original code could be considered to be undeveloped, as only packages were imported without providing any proper functionality. With this in mind, the code added to do a checksum was pretty basic, with an overall functionality that allows the use of SHA - 256 to create a hash as an example of the use of this cipher. Furthermore, in order to overcome some of the vulnerabilities seen in the dependency-check report would consist in using updated versions of the dependencies flagged.

Graphical user interface, text, application

Description automatically generated

## 7. Summary

Considering that a much more developed code would be needed for our client, following the Vulnerability Assessment Process Flow our main concern was cryptography. With this in mind the base code was slightly updated to show the functionality of hash values, as it is very important that verification is enabled without having to disclose sensitive information. The client could enter their credentials from their end, the SHA 256 cipher would create a hash value that would then be compared to the one stored in the server’s database for authentication.

It was also necessary to add extra layers of security in order to ensure safe communication between the client and the server. To do so, a SSL protocol was added, which needed a certificate authority to validate the authenticity of the application. For this example, the certificate was created locally and self-signed to test the appropriate connection was possible through HTTPS.

One of the best practices to ensure minimum vulnerabilities exist is to run a dependency-check to be able to investigate which libraries may possibly cause security issues, which normally could be solved by updating the versions used in the program. Furthermore, it is also important to consider the overall functionality of the program to be developed, for instance, preferring the use of protocols such as TLS to give an extra layer of security and purchasing a trusted certificate authority.