****

# 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 5](#_Toc33111307)

[3. Deploy Cipher 6](#_Toc33111308)

[4. Secure Communications 6](#_Toc33111309)

[5. Secondary Testing 7](#_Toc33111310)

[6. Functional Testing 8](#_Toc33111311)

[7. Summary 8](#_Toc33111312)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **8/13/2022** | **Jonathan Wolanyk** |  |

## 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

Jonathan Wolanyk

## 1. Algorithm Cipher

Determine an appropriate encryption algorithm cipher to deploy given the security vulnerabilities, justifying your reasoning. Be sure to address the following:

The SHA-256 algorithm cipher is widely used in the software engineering sector because it has no known vulnerabilities. Using an algorithm cipher that is currently unbroken is vital to the integrity of encryption since, once a cipher has been broken, it becomes effectively obsolete. SHA stands for “secure hashing algorithm” and 256 points to the number of possible hash values (2^256) that could be generated by the cipher. The cipher utilizes a hash function that takes data in, reformats it, and converts the reformatted data to a 256 bit-long code to send to another entity. Since there are so many possible values for the cipher, it is quite unlikely that collisions would occur or that brute force attempts to crack the cipher would be successful.

Random numbers are quite important to the encryption process. While randomness never truly exists with computers, one can produce pseudorandom numbers through PRNGs, or “Pseudorandom Number Generator’s.” PRNGs generate large, seemingly random prime numbers that are used as keys to encrypt data. While this method of encrypting data is more outdated than techniques used in ciphers such as SHA-256, if the bit-level is high enough, such methods are considered safe.

There are various ways to encrypt data in terms of the way keys are handled by senders and recipients as well. Symmetric and asymmetric encryption deal with the way two parties handle keys during the encryption and decryption processes. Symmetric encryption occurs when a sender and receiver use the same key to transfer information. In the same way that two parties may use a secret code to know whether a door should be opened to an outsider, symmetric encryption ensures that only known parties can access certain data. Asymmetric encryption involves two keys; one key that is public and a second key that is private. While anyone can access the public key to send encrypted data to the receiver, only the receiver is privy to the private key for decrypting the data.

Encryption has been around for thousands of years. Secret codes have been used to send sensitive information from the dawn of civilization. While encryption as we know it in software has been evolving rapidly over the past century, the practice still can be pushed even further to ensure the security of information. Today, encryption is still being pushed further through encryption methods such as elliptic-curve cryptography. Elliptic-curve cryptography is quite new but is quickly becoming the prevailing method to secure data and create short, hard to decrypt keys. As unauthorized users become more capable and intelligent, so must encryption and decryption techniques.

## 2. Certificate Generation

Generate appropriate self-signed certificates using the Java Keytool, which is used through the command line.

Graphical user interface, text, application, email

Description automatically generated

## 3. Deploy Cipher

Refactor the code and use security libraries to deploy and implement the encryption algorithm cipher to the software application. Verify this additional functionality with a checksum.

Text

Description automatically generated

## 

## 4. Secure Communications

Refactor the code to convert HTTP to the HTTPS protocol. Compile and run the refactored code to verify secure communication by typing **https://localhost:8443/hash** in a new browser window to demonstrate that the secure communication works successfully.

Graphical user interface, text, application

Description automatically generated

## 

## 5. Secondary Testing

A screenshot of the refactored code executed without errors

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text

Description automatically generated

A screenshot of the dependency check report

## 6. Functional Testing

Identify syntactical, logical, and security vulnerabilities for the software application by manually reviewing code.

Graphical user interface, text, application, email

Description automatically generated

## 7. Summary

Discuss how the code has been refactored and how it complies with security testing protocols. Be sure to address the following:

Since I had constructed most of this code using REST API, most of my code was secure and efficient before refactoring. However, I did identify a potential vulnerability around initiating the algorithm cipher. The SHA-256 cipher was initially not surrounded by a try-catch block, which opened the code up for a potential error-handling attack; an attacker could have provoked a multitude of errors using this vulnerability to bring down the service. In my refactored code, I surrounded the initialization of the SHA-256 algorithm by a try-catch block.

This program integrates a selfsigned certificate into the program to validate the data. While I encountered quite some trouble with having Google Chrome accept my certificate, the use of certificates in the encryption and decryption process ensures that companies and users are protected from attackers while accesses and transmitting data.

As the code base expands, it will be important to encapsulate data and surround processes with the appropriate error-handling techniques to avoid attacks. Any data that is sent to or received from the client must be encrypted and decrypted using the program included alongside this document.