****

# 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** | **October 17, 2021** | **Shannon Donahue** | **Project completed and submitted** |

## 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

Shannon Donahue

## 1. Algorithm Cipher

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

* Provide a brief, high-level overview of the encryption algorithm cipher.
* Discuss the hash functions and bit levels of the cipher.
* Explain the use of random numbers, symmetric vs non-symmetric keys, and so on.
* Describe the history and current state of encryption algorithms.

The history of encryption long predates even computers or algorithms. People have long been using simple encryption techniques with written words to keep messages a secret from others. For example, in ancient Rome Julius Caesar invented a simple cipher that would shift letters over three for example the letter A would become D and so on. This was simple yet effective in hiding the contents of written contact. Today’s modern ciphers are often written in the form of computer algorithms and involve mathematical computations. They have been able to grow more and more complex with the use of technology which is because of the more complex methods available to break through them. As of today, we have AES which stands for advanced encryption standard which has taken the place of the former DES (data encryption standard). AES is like DES in the fact that they are both block ciphers, but AES adds in the use of asymmetric keys which means that both the user and sender must know the same secret key.

I chose to use the AES encryption algorithm cipher. This encryption algorithm has a few different options for block lengths. The block length has to do with the strength of the encryption. The longer the block length the stronger the encryptions, but it also means the encryption will slow down as the block length increases. The three options for block length for AES encryption are 128, 192, and 256 bits in length. I have decided to choose to 256-bit length to put priority into strength over speed as this is a banking application which will have some of the users most sensitive personal information. It is also important to note that the AES encryption is a block cipher and encrypts the data utilizing a secret key block by block rather than just bit by bit. It also uses the process of substitute permutation when encrypting which means it goes through multiple rounds to produce the cipher text. This is important because the chances with running into collisions is lessened by this making the overall encryption more secure. As mentioned above AES is an asymmetric encryption so both user and sender need the secret key. The hash function is then SHA-256 in this application, which means it is what converts the string into a 256 bit signature using a mathematical function. It is not encryption itself but is used with encryption to make a secure secret message.

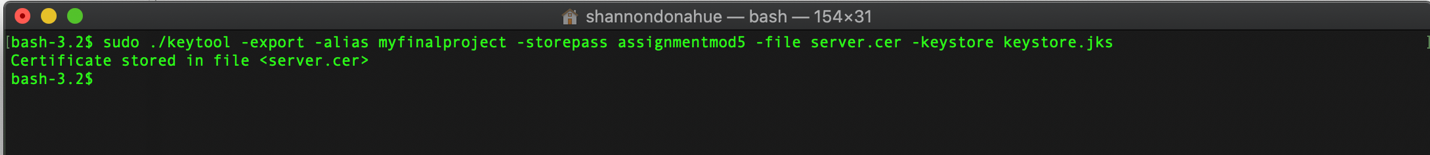
## 2. Certificate Generation

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

* To demonstrate that the keys were effectively generated, export your certificates (CER file) and submit a screenshot of the CER file below.

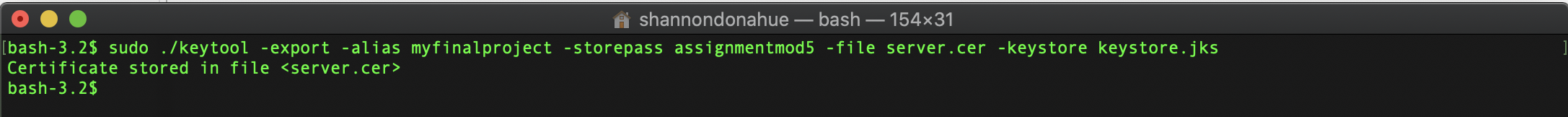
Text

Description automatically generated



A screenshot of a computer

Description automatically generated with medium confidence



## 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.

* Insert a screenshot below of the checksum verification. The screenshot must show your name and a unique data string that has been created.

Text

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated



Graphical user interface, text, application

Description automatically generated

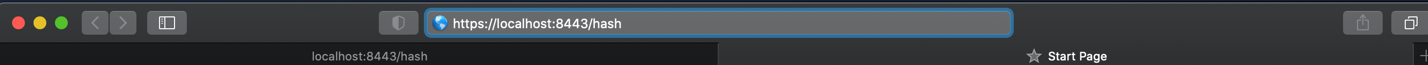
## 4. Secure Communications

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.

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

Graphical user interface, text, application

Description automatically generated



Graphical user interface, application, Teams

Description automatically generated

HTTP no longer works:

Graphical user interface

Description automatically generated with medium confidence

## 5. Secondary Testing

Complete a secondary static testing of the refactored code using the dependency check tool to ensure code complies with software security enhancements. You only need to focus on the code you have added as part of the refactoring. Complete the dependency check and review the output to ensure you did not introduce additional security vulnerabilities.

* Include the following below:
  + A screenshot of the refactored code executed without errors
  + A screenshot of the dependency check report

Text

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Original Dependency Check:

Graphical user interface, text, application, Teams

Description automatically generated

Dependency Check after altered Code no new errors added:  
Graphical user interface, application, Teams

Description automatically generated

## 6. Functional Testing

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

* Complete this functional testing and include a screenshot below of the refactored code executed without errors.

One risk I did see in the code that could potentially cause security risks was that I was utilizing public variables in my class. This could cause a problem due to encapsulation. Later when there are more classes added to the code there would be potential for other classes to access those variables which should be kept private to that class.

Graphical user interface, text, application, email

Description automatically generated

HTTPS:

Graphical user interface, text, application, website

Description automatically generated

HTTP does not work:  
Graphical user interface

Description automatically generated with medium confidence

## 7. Summary

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

* Refer to the Vulnerability Assessment Process Flow Diagram and highlight the areas of security that you addressed by refactoring the code.
* Discuss your process for adding layers of security to the software application and the value that security adds to the company’s overall wellbeing.
* Point out best practices for maintaining the current security of the software application to your customer.

To start I will address a few of the vulnerabilities that were addressed in the code. The first vulnerability that was addressed is cryptography. We encrypted the data that was accessed through the web browser. The data in this example may not have been extremely private but this data could be something as secure as bank account numbers in this application. From here we also incorporated encapsulation by setting our variables to private which will prevent access from other future classes that are added to the application. This will keep these variables secure from outside access. Client/server security was also addressed with the addition of HTTPS protocol. Https encrypts its requests and responses unlike http, which makes it a much more secure browsing experience. Code quality was also an important factor, the code is easy to read and expand upon for future coding. This is important because it makes it easier to find potential future security errors that involve the current code once its built upon. Code error was addressed but could be addressed in more depth as well. For starters each method does only have one functionality that it oversees. There is also errors being raised when unaccounted things happen and errors need to be returned. Code error could have been elaborated on more with unit tests for the functions. This would allow thorough testing of each of the functions which is especially helpful for the future if the code is altered and unit tests for these functions are breaking. In the future of the code, it would be good to add in input validation such as user login to allow the user to make a username and password for an extra layer of security for access to the page. Finally, once the application is built upon it is important to have secure API interactions. This is like the encapsulation portion of security. It is important to not only have private variables but also setter and getter methods to utilize these variables between API’s.