****

# 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** | **6/19/22** | **Tyler Pruitt** |  |

## 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

Tyler Pruitt

## 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.

For Artemis Financial’s use in protecting their client data and financial information, I would recommend the SHA-256 Cipher. SHA-256, also known as SHA-2 has no currently known collision attacks and is encouraged for use for any applications of hash functions requiring interoperability according to the NIST of the United States Government website: <https://csrc.nist.gov/Projects/Hash-Functions/NIST-Policy-on-Hash-Functions> . SHA takes an input string, runs it through the SHA function and releases a 160 bit digest. The length of the input string cannot be more than 2^64 bits. The length of the digest is always 160 bits in length. The digest should not re-create the original message when run through the function again. SHA-2 is a hash function used to transform large random sized data into small fixed size data. There does not have to be a key to operate one-way and leaves room for generation of digital certification through signatures.

Use of Random numbers, symmetric vs non-symmetric keys, and so on are all varying ways to increase the complexity of the key. More complex keys are harder to crack and increases the security at the cost of increased encryption complexity. More complex encryption takes longer to decrypt.

In history, encryption has been seen for thousands of years, with possible ciphers being noted back to the ancient Greeks and Spartan military. For example, the Zimmerman note (or Zimmerman Telegram) is one example of an encrypted secret diplomatic communication that was decrypted and lead the United States into World War 1. Encryption algorithms exist until they are cracked, and then are depreciated and new ones are created in their place. When new ones are created, they often have increased bit levels to increase security. DES and 3DES are both examples of this, DES was a 56-bit encryption that was cracked and depreciated in 2005. 3DES was made to replace DES, becoming more secure through running the algorithm three time per data block, however this algorithm also was cracked.

## 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.

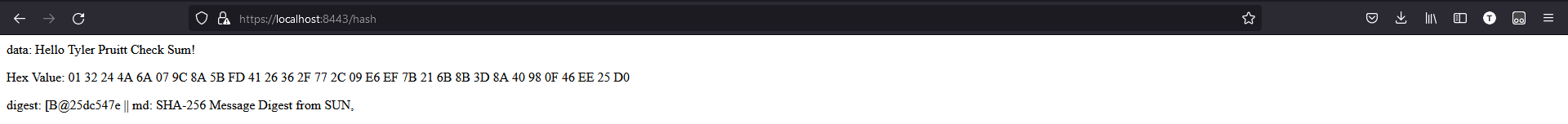
Graphical user interface, text, application, email

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.

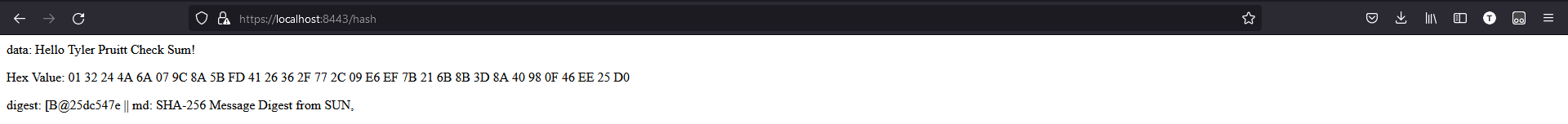
* Insert a screenshot below of the checksum verification. The screenshot must show your name and a unique data string that has been created.



## 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.



## 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

A screenshot of a computer

Description automatically generated

Graphical user interface, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

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.

A screenshot of a computer

Description automatically generated

## 

## Graphical user interface, application 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:

* 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.

While refactoring the code, I addressed the applicable areas of security listed within the Vulnerability Assessment Process Flow Diagram by ensuring first that I had proper use of cryptography. I used the SHA-256 cipher to properly encrypt the data while avoiding collisions. SHA-256 is not known to be cracked so the cryptography will be secure. Next, error handling. To securely implement the hash encryption function, I had to implement a NoSuchAlgorithmException to counteract a common error where the requested encryption algorithm is not supported by the digest function. I used comments and formatting within my refactored ensure my code is neat and secured to fulfill the code quality vulnerability. Finally, I used typings to ensure that each variable and function could only be used by the intended data format such as the myhash function only returns and accepts a string, or the bytesToHex function only accepting byte strings and returning a string as well.