****

# 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** | **8/8/2022** | **David Anguiano** |  |

## 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

David Anguiano

## 1. Algorithm Cipher

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

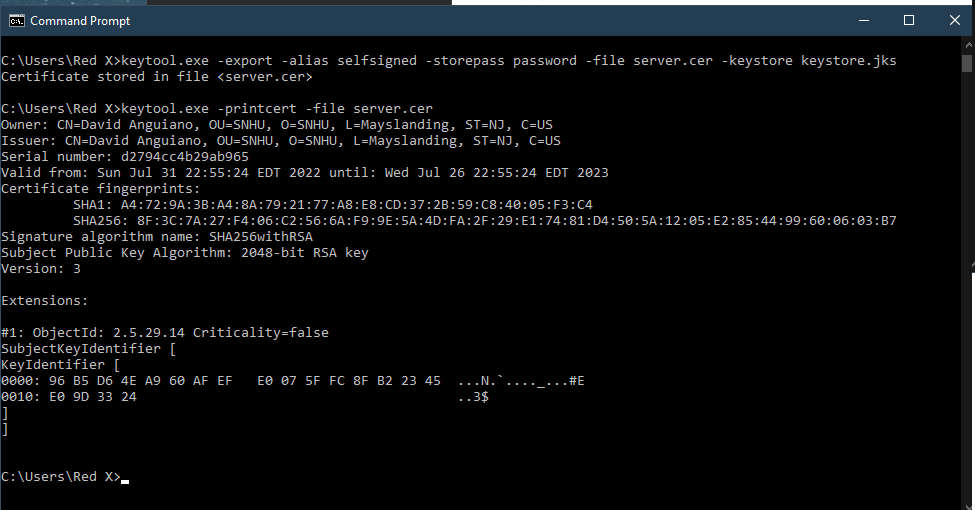
* Provide a brief, high-level overview of the encryption algorithm cipher.
* Discuss the hash functions and bit levels of the cipher.
* Explain the use of random numbers, symmetric vs non-symmetric keys, and so on.
* Describe the history and current state of encryption algorithms.

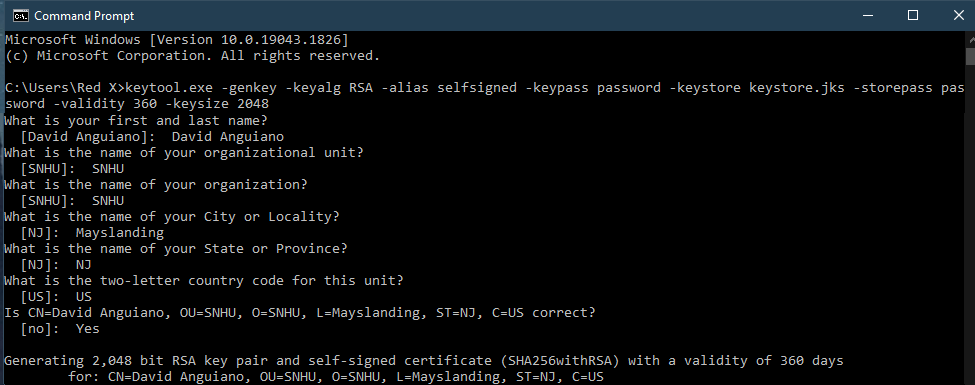
The SHA-256 cipher is trusted by leading public-sector agencies and is widely used by technology leaders. It has the lowest chance of collision with the chances of two different inputs being the same being practically non-existent. All the Hard disks on earth combined couldn’t hold enough 1MB files to achieve 0.01% of collision. Minor changes to a document will completely change the hash value which makes authentication easy since matching files return the same value. SHA-256 has 256-bit key encryption with a vast number of combinations at only 256bits. Asymmetric keys are recommended for secure communications. The use of public keys to encrypt and private keys to decrypt allows for maximum security. Currently, SHA-256 is the standard as it has yet to be cracked.

## 2. Certificate Generation

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

**CER generated, exported certificates (CER file) and CER File Screenshots 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.

**Checksum verification SHA-256 Hash with name & unique data string.**

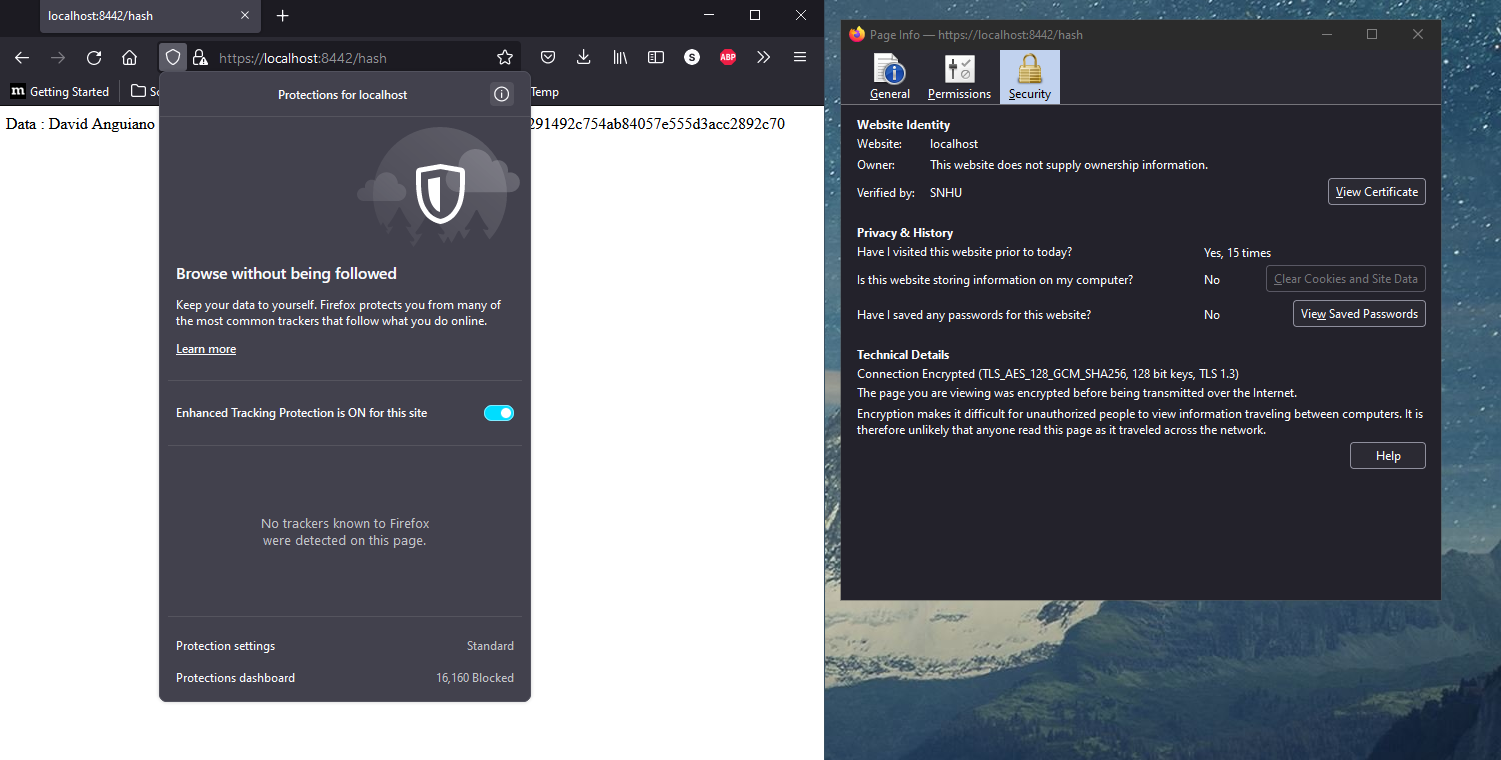
Graphical user interface, text, application

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.

**Screenshots of Webpage Security**



## 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.

**A screenshot of the refactored code executed without errors.**Graphical user interface, text, application

Description automatically generated

**A screenshot of the dependency check report.**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.

Text

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.

I refactored the code to use a RestController to secure connectivity to my programs hash endpoint. The areas of security I was able to address in the Vulnerability Assessment Process Flow Diagram were APIs and Cryptography. By adapting SHA-256 hashing cipher I was able to implement Cryptography into our software. SHA-256 is trusted by leading public-sector agencies and used widely by technology leaders. SHA-256 also has the lowest chance of collision, all the Hard disk on earth couldn’t hold enough 1MB files to achieve 0.01% of collision. I was also able to secure a connection to our application using SSL Certificates and Keystores. I recommend bi-weekly dependency checks be run to keep up with any vulnerabilities. Additionally, I suggest creating a similar schedule to check and update plug-ins.