****

# 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** | **08.13.2022** | **Jessica Megaro** | **Initial creation.** |

## 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

Jessica Megaro

## 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 encryption algorithm used is Elliptic Curve Cryptography (ECC). The hashing algorithm used

is SHA-256 with Elliptic Curve Digital Signature Algorithm (ECDSA). This choice is recommended

because it is stronger than RSA when it comes to key sizes. The routine ECC key of 256 bits is 10,000 times stronger than a 2048-bit RSA key (Thayer, 2014). The hashing functions are used in two places. SHA-256 is being used in the digital certificate, making it more difficult to forge. SHA-256 is also being used in the endpoint of the API, making it more secure with less collisions. ECC is an asymmetric algorithm, using a public and private key. This ensures that the private key is only ever used on the server side and not available for the initial communication by the client. The number of key bits that must be guessed and the computational difficulty it takes to break them in ECC gives it a high cryptographic strength when compared to RSA (The Importance of True Randomness, 2022). Historically speaking, as computational power increases, the need for longer and more random sequences also increases.

## 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

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.

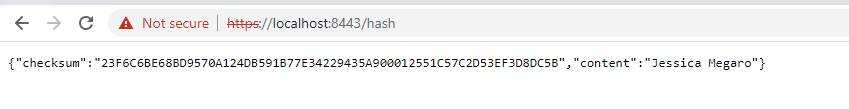
Graphical user interface, text, application

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.



## 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, email

Description automatically generated

Table

Description automatically generated with medium confidence

Graphical user interface, text, application, email

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.

Refer to screenshots of code above shown without error.

## 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 enhance the security of this application, the API has been made to have secure interactions by adding encryption. For cryptography, hashing functions are used. Code quality has been improved by updating to the most current Spring Boot and ensuring that no other code errors are present at time of testing. Adding security will enhance the company and its overall well-being by showing the customer that security is a serious topic. The customer will feel protected and more likely to use the services if they know that their information is being held to a high standard of privacy. The best practices for maintenance are staying up to date with ECC vulnerabilities if any should arise, regular monitoring for abnormal events, and swift resolution should a breech arise. Please see attachments for refactored code included in file named ssl-server\_student.

**References**

Thayer, W. (2014, June 10). *Benefits of elliptic curve cryptography*. PKI Consortium. Retrieved

August 14, 2022, from <https://pkic.org/2014/06/10/benefits-of-elliptic-curve->

cryptography/#:~:text=The%20foremost%20benefit%20of%20ECC,RSA%20keys%20mus

t%20get%20longer.

The Importance of True Randomness in Cryptography. (2022). Retrieved August 14, 2022, from

https://www.design-reuse.com/articles/27050/true-randomness-in-cryptography.html