****

# Practices for Secure Software Report

Ryan Reames

CS-305-J7256

Southern New Hampshire University

Table of Contents

[Practices for Secure Software Report 1](#_Toc120440719)

[Document Revision History 3](#_Toc120440720)

[Client 3](#_Toc120440721)

[Instructions 3](#_Toc120440722)

[Developer 4](#_Toc120440723)

[1. Algorithm Cipher 4](#_Toc120440724)

[2. Certificate Generation 5](#_Toc120440725)

[3. Deploy Cipher 7](#_Toc120440726)

[4. Secure Communications 7](#_Toc120440727)

[5. Secondary Testing 8](#_Toc120440728)

[6. Functional Testing 10](#_Toc120440729)

[7. Summary 10](#_Toc120440730)

[8. Industry Standard Best Practices 10](#_Toc120440731)

[9. References 11](#_Toc120440732)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **11/27/2022** | **Ryan Reames** | **Updated Algorithm Cipher Section** |
| **1.1** | **11/29/2022** | **Ryan Reames** | **Updated Certificate Generation, Deploy Cipher, Secure Communications, and Secondary Testing sections.** |
| **1.2** | **12/7/2022** | **Ryan Reames** | **Ran program and added screenshot under Functional Testing section. Also added potential vulnerabilities under Functional Testing section.** |
| **1.3** | **12/10/2022** | **Ryan Reames** | **Updated Summary and Industry Standard Best Practices sections.** |

## Client



## Instructions

Submit this completed practices for secure software report. Replace the bracketed text with the relevant information. You must document your process for writing secure communications and refactoring code that complies with software security testing protocols.

* Respond to the steps outlined below and include your findings.
* Respond using your own words. You may also choose to include images or supporting materials. If you include them, make certain to insert them in all the relevant locations in the document.
* Refer to the Project Two Guidelines and Rubric for more detailed instructions about each section of the template.

## Developer

Ryan Reames

## Algorithm Cipher

Cryptography has been around in one form or another for thousands of years, with one notable example dating back to ancient Rome with the Caesar Cipher, in which letters were shifted 3 letters to the right (ex. A = D) to create secret messages (Sidhpurwala, 2013). Within the past century, cryptography has been studied scientifically (largely from WWII on) and is used in nearly all digital communication and data storage for governments and large businesses. Encryption, one component of cryptography, is used to prevent others from reading data by scrambling (or encrypting) said data. A key is then used to unscramble (decrypt) this data for use. There are two forms of encryption – asymmetric and symmetric. Asymmetric encryption has both a public and private key. If data is encrypted with the public key, it is then decrypted with the private key, and vice versa. Each user has their own public and private key. With symmetric encryption, one key is used to both encrypt and decrypt data and tends to be faster than asymmetric encryption for data encryption, with the trade-off being if the key is exposed anyone can break the encryption.

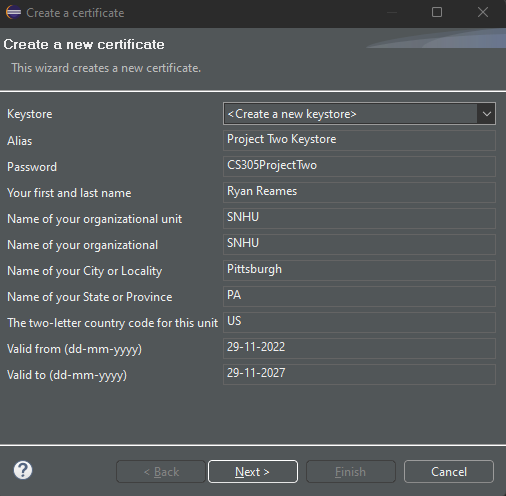
Considering Artemis Financial’s wish to generate a checksum for use in data validation, they should use an asymmetric encryption cipher. This would allow them to generate a certificate using a private key to verify users are using their website. At the same time, users would be able to send encrypted data using a public key which only Artemis Financial would be able to decrypt with their private key. There are multiple encryption algorithms which the company could use, each of which have different properties. For example, each cipher may use a different number of bits, which refer to the length of the encryption key. Of the algorithms listed in the Java Security Standard Names page, the section titled MessageDigest Algorithms contains multiple algorithms that can be used to generate a checksum value such as MD2, MD5, and various SHA and SHA3 algorithms (Oracle, 2017). Of the values listed there, the one I am recommending be used in generating a checksum is SHA-256.

Per FIPS Pub 180-4, SHA-256 is approved by the NIST for use in the generation of digests of messages, which can be used to check if changes were made since data was initially generated (National Institute of Standards and Technology, 2015). Many companies use SHA2 (which includes SHA-256) in signing, as collisions are possible with MD5 and SHA1 (IBM, 2020). In 2008, an MD5 collision was found, and in 2017 Google researchers found a SHA1 collision (Ellis, 2018). Since both have been considered broken and generally not recommended for use. Unlike these two algorithms, SHA-256 is considered collision-resistant, as trying to find a collision would require approximately 40³⁸ attempts which is infeasible with current technology (Ellis, 2018).

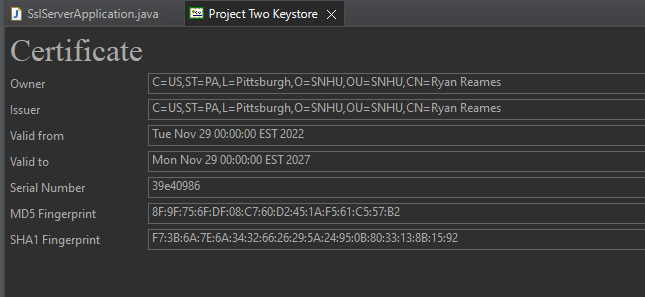
## Certificate Generation

Insert a screenshot below of the CER file.

Creating the certificate in Eclipse:



Created certificate:

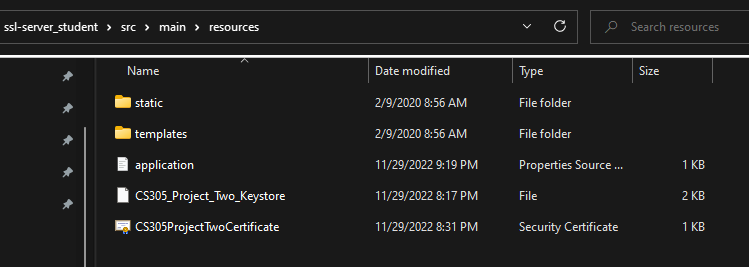


Exporting certificate:

Graphical user interface, text, application

Description automatically generated

Certificate file:



## Deploy Cipher

Insert a screenshot below of the checksum verification.

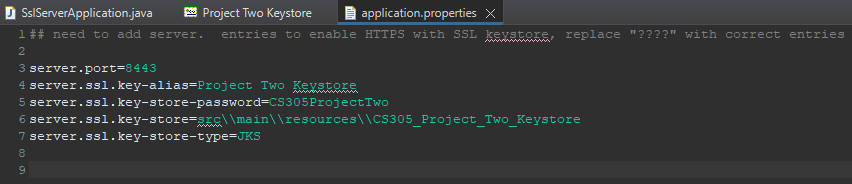
Graphical user interface, text, application

Description automatically generated

## Secure Communications

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

Application properties:



Secure webpage:

Graphical user interface, text, application, email

Description automatically generated

## Secondary Testing

Insert screenshots below of the refactored code executed without errors and the dependency-check report.

Dependency check before refactoring code (with errors from given project file suppressed):

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text

Description automatically generated

Executed code without errors:

Dependency check after refactoring code (errors from given project file still suppressed):

Graphical user interface, text, application, email

Description automatically generated

## Functional Testing

Insert a screenshot below of the refactored code executed without errors.

Graphical user interface, text

Description automatically generated

A few issues with the Artemis Financial program include the following:

* There is no sort of attribute or user-based access control for the site.
* There is no input validation.
* There is no verification of file type to be uploaded by users.
* While the application uses a self-signed certificate, a third-party certificate may be preferable.
* There is no encryption implemented for storing user data.
* Everything is being done out of one class, rather than encapsulating based on requirements.

## Summary

The code provided by Artemis Financial was refactored with the following being implemented:

* A certificate was generated using the Java Keytool to allow for secure API interaction.
* Encryption was used to generate a checksum, verifying data integrity when data is passed to the application. To ensure good code quality, a try catch was included to prevent inaccurate algorithm names being passed in.
* The code was refactored to use HTTPS, which included the certificate generated earlier.
* A dependency check was run on the project file prior to refactoring, after which the pom.xml file was refactored to suppress any preexisting vulnerabilities. The dependency check was run after refactoring to ensure no additional vulnerabilities were introduced.

## Industry Standard Best Practices

To implement industry standard best practices, I researched NIST to determine what standards are currently being followed per the US government. I also researched various well-known sources of information such as IBM or Red Hat to improve my understanding prior to refactoring code. Beyond this, I also ran two dependency checks, one before and one after refactoring, which referenced NIST to see which dependencies in the project were vulnerable, and whether I introduced any new vulnerabilities when refactoring.

## References

References

Ellis, S. (2018, December 13). *The beautiful hash algorithm*. Medium. Retrieved November 22, 2022, from https://steviecellis.medium.com/the-beautiful-hash-algorithm-f18d9d2b84fb

IBM. (2020, October 28). *Why use SHA256 instead of SHA1?* IBM Support. Retrieved November 22, 2022, from https://www.ibm.com/support/pages/why-use-sha256-instead-sha1

National Institute of Standards and Technology. (2015, August). *Secure hash standard (SHS) - NIST*. Retrieved November 22, 2022, from https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf

Oracle. (2017). *Java Security Standard Algorithm Names.* Oracle. Retrieved November 22, 2022, from https://docs.oracle.com/javase/9/docs/specs/security/standard-names.html#cipher-algorithm-names

Sidhpurwala, H. (2013, August 14). *A brief history of cryptography*. Red Hat. Retrieved November 19, 2022, from https://www.redhat.com/en/blog/brief-history-cryptography