****

# 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** | **11/12/2021** | **Noah Archibald** |  |

## 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

Noah Archibald

## 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.**

Artemis financial will need to implement an algorithm that is both secure and practical for their purposes. More than likely, as a financial institution, Artemis will be involved in sending and receiving sensitive information to and from their customers. As such, I would recommend that Artemis deploy the RSA encryption algorithm. It is an industry trusted and reliable way of encrypting the sensitive data that Artemis will be handling on a regular basis. It relies on the mathematical principle that it is incredibly hard, and nearly impossible given a large enough key size, to factor the product of two large prime numbers. Data can be encrypted with a public key that can be known to anyone but only decrypted with a private key which only the receiver knows. This seems practical in terms of the required functions of the system desired by Artemis as they would be able to securely hold the private decryption key while customers data would be able to be encrypted first with the public key and then sent securely to Artemis’s servers. RSA also allows for messages sent between Artemis and their customers to be digitally signed as a further method of sender/receiver verification.

* **Discuss the hash functions and bit levels of the cipher.**

The RSA algorithm is capable of producing keys with bit sizes of 2048 or 4096 bits. For Artemis’s purposes, it would be more prudent to implement the RSA-2048 algorithm as it’s associated digests take up less space than the RSA-4096 version and it requires less resources to manage. As mentioned in the above paragraph, hash functions can be used to generate check sum values that verify the sent data is the correct received data.

* **Explain the use of random numbers, symmetric vs non-symmetric keys, and so on.**

Generating large and random prime numbers is essential to the security of the entire RSA cryptographic algorithm. If the two primes are known, then it becomes completely trivial to work backwards through the algorithm, break the encryption, and decrypt the intercepted data back into plaintext from its hashed form. RSA is also an asymmetric algorithm as it’s public encryption key is different from its private decryption key. This separation is also vital to the security of any plaintext data being encrypted via RSA as anyone with the private decryption key can view any previously encrypted plaintext.

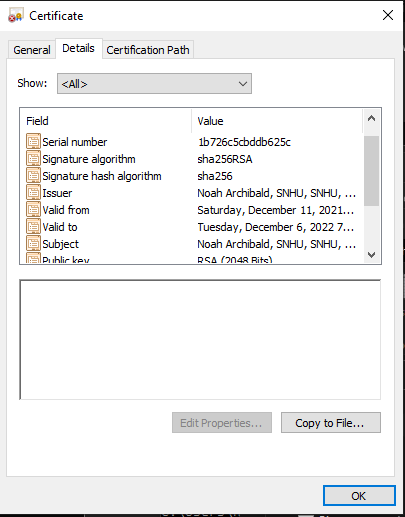
* **Describe the history and current state of encryption algorithms.**

Cryptography itself has been driven to evolve at the same rate as technological advancements. And at its core, cryptography relies upon effective encryption algorithms that can be applied practically to serve as ways of protecting sensitive information. RSA is one of the older encryption algorithm standards that is still being used today. It was developed in the 1970’s and has yet to be broken, given it is deployed in the correct way. Encryption algorithms in their current state are incredibly technically complex and challenging to break. Attacks on these algorithms will need to become more and more sophisticated if they hope to achieve any modicum of success. Subsequently, the algorithms themselves will need to continue to evolve and become more sophisticated themselves so as to anticipate and account for the new technology that is on the horizon in quantum computing.

## 2. Certificate Generation

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

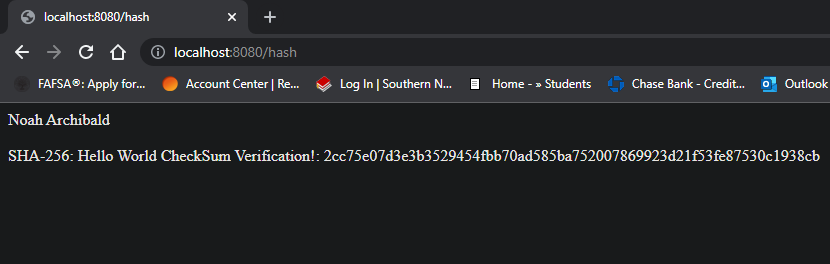
* To demonstrate that the keys were effectively generated, export your certificates (CER file) and submit a screenshot of the CER file below.



## 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.

Unfortunately, I could not for the life of me get the secure connection to work. I ended up never being able to connect to the webpage and got the following error:

A screenshot of a computer

Description automatically generated

## 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

## 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.

## 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.

Some of the best practices for maintaining the security of the software application would be to continually update the dependencies within the program. Many vulnerabilities within programs don’t end up having inherently anything wrong with the way they are coded, but rather there are defects and bugs found within the libraries the program depends on to function. Keeping up to date with the current releases of these libraries would be a good way to prevent issues from arising in the future. Adhering to secure coding practices when working on expanding or implementing new features in the software would also be advisable. Encapsulating data structures properly and practicing good and secure coding practices like requiring input validation and managing access control to the backend of the system to ensure there are no unwanted actors manipulating the system for their own gain.