****

# 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** | **[Date]** | **[Your Name]** |  |

## 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

Mark Alan Leo Rossmiller

## 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.

Artemis financial would like an encryption solution to hide the contents of their archived files. Since we are encrypting large volumes of data at rest, we will be looking for a symmetric-key encryption algorithm rather than a public key cipher. The difference is that symmetric algorithms are faster and more robust at encrypting large volumes of data, whereas public key cryptography is used more for secure key exchange, in order to set up a symmetric encryption session between two parties through a public connection.

While there are many cipher suites available to encrypt data at rest, AES is often touted as the, “gold standard,” of symmetric encryption. It is NIST-certified, and protects much of the United States government’s data. There are alternatives to AES such as Bruce Schneier’s Twofish and Blowfish, 3DES (which AES replaced), Serpent, and countless others, but AES was chosen as a standardized algorithm by the government in 2002 and has since enjoyed widespread popularity.

Since security is not meant to be enacted through obscurity, and as much as on a personal note I like using non-standard algorithms in my daily life, for a corporate usage I recommend keeping with the standard and using AES for our production environment. However, I do recommend abstracting all code using AES to a point where we can easily switch to a new standard if one should come along, or if AES is found to be broken.

Our servers can easily use hard disk encryption in order to keep data private physically, while allowing use on an as-needed basis without resorting to treating the filesystem as a tape feed, and needing to decrypt the entirety of the disk in order to retrieve needed information. Most operating systems support full disk encryption, but since this is going to be run on a server I suggest using a Linux implementation. Linux full-disk encryption is as easy to set up as freshly installing a system, and offers a high degree of functionality. Of course, we want to do a minimal installation, with only the remote file server running if no other functionality is needed from the system.

Ubuntu, the Linux distribution of choice for many IT professionals uses aes-xts-plain64:sha256 with 512-bit keys out of the box for full-disk encryption. XTS is counter-oriented chaining mode. PLAIN64 is an IV generation mechanism. SHA-256 is the hashing algorithm.

The AES standard used for Ubuntu’s full-disk encryption used to use 256-bit keys, but recently it was changed to use 512-bit keys. This offers exponentially harder to crack encryption keys using brute force methods. XTS has replaced CBC, offering several advantages against attacks to the algorithm.[6] The hash function used, SHA-256, has not changed from implementation to implementation in Ubuntu’s full-disk encryption from their previous usage.

When preparing a disk for full-disk encryption, it is recommended to overwrite the disk multiple times with all zeroes, all ones, and then with random data taken from /dev/urandom. This is time consuming but essential to protect the drive if it should the physical disk fall into the hands of an attacker. Of course, physical security perimeter should be given thought as well.

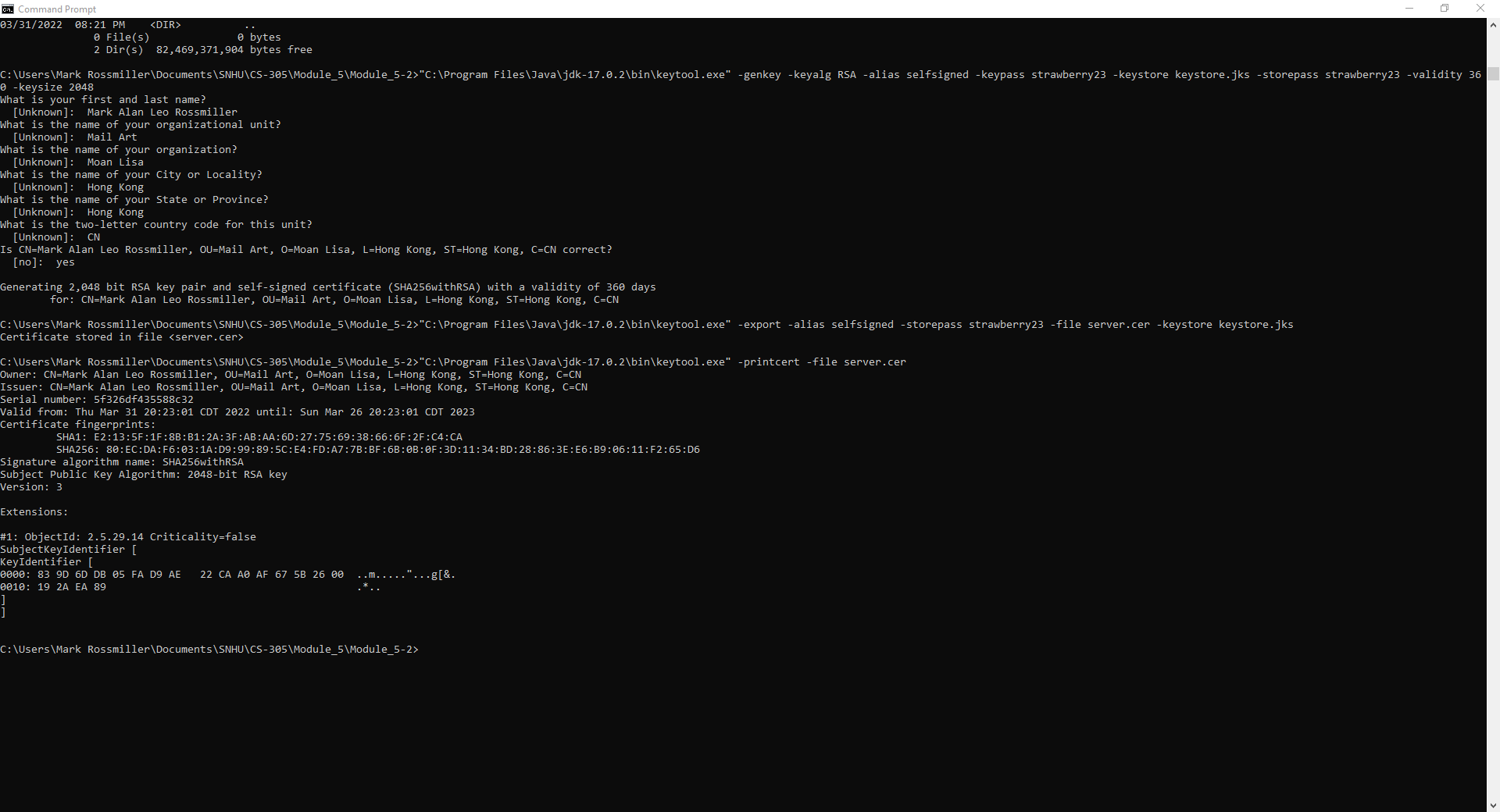
Aside from the physical security of the raw disk, the encryption key used to encrypt and decrypt data on the disk, there must also be put much thought into securing the now-cloud-enabled virtual drive so that attacks which come over the network are repelled.

For data which absolutely must be protected at all costs, I recommend going a step beyond online disk encryption and rather suggest keeping the data offline and only used as needed. This could be done through physically disconnecting the drive, but it also could be used with full-disk encryption, and only unlocking the drive when it (the data on disk) is absolutely needed.

## 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.

Graphical user interface, text, application, email

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.

Graphical user interface, text, application, email

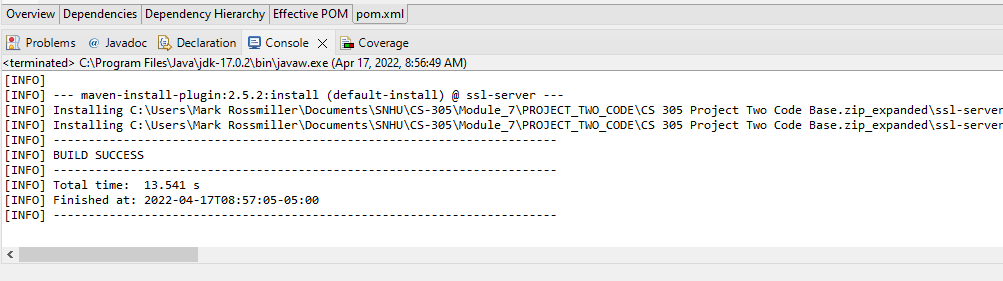
Description automatically generated

## 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

I regret to say that I could not get the code to run due to issues with Eclipse, I did manage to get it to build, which I gave a screenshot below; but the dependency check would not work due to errors at runtime, and running the Java application as a Spring application would not execute with the settings in my Eclipse workstation.



## 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.

I could not get the code to run, but it built successfully.

Graphical user interface, text, 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.

We added HTTPS to the web server, which in and of itself does not make for a completely secure program, but it does help keep out people who might otherwise eavesdrop on our communications. We added a hashing function to be able to verify that a message intended for the recipient has authenticity by its author. During the process, we used the Java cryptography framework, which is built into the Java language. We used the OWASP dependency check software to be able to tell us about known vulnerabilities in our static library code. This project is far from a complete, industrial-strength code base, but it is a start in writing secure code and I am glad to be able to have worked with Artemis Financial during my tenure at SNHU.