****

# 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** | **10/15/2021** | **Jonathan Sussan** |  |

## 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

Jonathan Sussan

## 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 appropriate encryption algorithm cipher to deploy would be the AES (Advanced Encryption Standard) cipher. AES is a symmetric block cipher that is used by the United States government to protect classified data. This is the correct algorithm cipher to deploy because of it is one of the best standards used today and its variety of key sizes.

AES uses either 128-bit, 192-bit, and 256-bit key length to encrypt and decrypt a block of messages. The 256-bit encryption is significantly more difficult for brute-force attacks and is extremely difficult to hack. This however comes at a cost as it requires more processing power and can takes longer to execute than the 128-bit. The correct choice would be to use the 128-bit key length as even with huge amounts of computing power, it is unlikely to be an issue for the foreseeable future and would provide a better experience for users than the 256-bit key length due to it being faster to execute.

Symmetric (secret key) ciphers use the same key for encrypting and decrypting. The sender and receiver must both know and use the same secret key. With symmetric key encryption, the encryption process can be carried out quickly and the length of keys is typically 128-bit or 256-bit, depending on the security level required. Non-symmetric keys use a pair of related keys, a public and private key. The public key is what’s used to encrypt a plain text message before sending it. To be able to decrypt this message, you would need to hold the private key. In non-symmetric encryption, the private key is only shared with the key’s creator since it needs to be secured and maintained. This is a more complicated process than symmetric key encryption and the process is slower. The length of the keys is also much larger, with the recommended RSA key size being 2048 bits.

Encryption is what protects and keeps sensitive data secure when you browse and use the internet. It scrambles data like banking account numbers, passwords, etc. so that hackers or bad actors can’t misuse the information. Encryption dates back all the way to Circa 600 BC, where Spartans use a device called a scytale to send secret messages during war. In the 1970’s, IBM formed a crypto group which designed a block cipher to protect customer data. Soon after, the United States adopted it as a national standard which is known as DES (Data Encryption Standard). Today, encryption involves new concepts and technology which is vital for us to secure our personal information.

## 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.

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, 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

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

Graphical user interface, text, application

Description automatically generated

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.

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application, email

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 have completed refactoring the code and have addressed the following areas of security; For security, I have addressed the API (Application Programming Interface), cryptography, and client/server. I have also addressed and improved code quality. During the process, I created a self-signed certificate, along with generated keys that I was able to use in connecting to a 128-bit AES (Advanced Encryption Standard) to provide a safe and strong connection. By using AES encryption, it allows for secure connection where only the client/server being able to read data. Providing secure connections is vital for a company like Artemis Financial because they deal with highly sensitive data such as bank account numbers. This provides trust between Artemis Financial and their clients.

Maintaining the current security of the software for Artemis Financial is important and an ongoing process to keep their customer’s sensitive data secure and to keep trust high. Best practices for maintaining security include frequently checking for vulnerabilities in the code. It is especially important to check for vulnerabilities if any changes are made to the code, such as adding or altering and functions, and then testing them out before publishing them. If any vulnerabilities are found that at the time have no solution, the development team should determine what impact those vulnerabilities will have on the security of the software and decide how to approach them. They should continue to monitor any fixes that may come out in the future. False positives and vulnerabilities that have no current solution should also be suppressed. It is an ongoing process for the development team to continue monitoring and updating vulnerabilities as solutions become available.

**References**

Bernstein, C., & Cobb, M. (2021, September 24). *What is the Advanced Encryption Standard (AES)? definition from search security*. Search Security. Retrieved October 14, 2021, from <https://searchsecurity.techtarget.com/definition/Advanced-Encryption-Standard>

Mukherjee, L. (2020, June 18). *Symmetric vs asymmetric encryption - 5 differences explained by experts*. InfoSec Insights. Retrieved October 14, 2021, from <https://sectigostore.com/blog/5-differences-between-symmetric-vs-asymmetric-encryption/>

*A brief history of encryption*. Thales Group. (2016, April 18). Retrieved October 14, 2021, from <https://www.thalesgroup.com/en/markets/digital-identity-and-security/magazine/brief-history-encryption>