****

# 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** | **10/17/2021** | **Eric Boilard** |  |

## 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

Eric Boilard

## 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 recommended encryption cipher is AES/SHA-256 because of its reliable bit size and reputation for never being compromised. Whenever we are hashing a function data is being allocated to certain buckets within memory for storage and whenever this hash function is iterated through it searches each bucket for values its looking for. When we have an instance when more than one value is contained within a bucket this is called a collision. These collisions can be troublesome because when there is more than one value within one of these buckets it takes more to search through the data. To avoid collisions, you can either resize the hash table, which some do this automatically, or you can take advantage of chaining which would allow for multiple keys per bucket but are stored in a linked list within the bucket itself storing key-value pairs. Since SHA-256 is such a large integer the chance of a collision happening is very unlikely. This is because the larger the bit size the more possible combinations there are to use.

AES is the standard for the U.S. Government and has many advantages and very few disadvantages. Even though it is a symmetrical algorithm which means that the same key used by the sender is used by the receiver, it uses 3 different bit levels from 128 to 256. Bit levels make the key even larger and harder to decrypt since the larger a key gets the more computational power it would take to decrypt. Additionally, AES has its own hash mode called AES-Hash. “Hashing generates a unique signature of fixed length for a data set or message. Each specific message has its unique hash, making minor changes to the information easily trackable. Data encrypted with hashing cannot be deciphered or reversed back into its original form. That’s why hashing is used only as a method of verifying data” (Simplilearn, 2021). As mentioned before, AES is symmetric which means the same key is used by both parties making it less secure, but it is faster. Asymmetric encryption uses two keys, a public key and a private key, where one is used to encrypt the data and one is used to decrypt. This is more secure, but it is slower than symmetric encryption. Using random numbers is very important in key creation as you want your key to be as unpredictable as possible. Therefore having 128-bit random numbered key means that the key is going to be of size 128-bit and extremely secure because of the randomness and size combined.

The history of encryption ciphers is very interesting dating back to the end of WW1 and the beginning of WW2 where the Germans used the Enigma machine to generate encrypted messages. Then for a while, cryptography was mainly used for military and government information until the 1970’s. At this point consumers wanted some encrypted support and some of the first block chain encryption was invented. The block chain cipher called lucifer was recognized as the standard and then called DES for data encryption standard. In later years DES was compromised after some exhaustive attacks and the need for a new encryption standard arose. “In 1997, NIST again put out a request for proposal for a new block cipher. It received 50 submissions. In 2000, it accepted Rijndael, and christened it as AES or the Advanced Encryption Standard”(Sidhpurwala, 2019). There are many other encryption algorithm ciphers out there and they are still being created to this day. However, to go much further in encryption technology would take more computing power than we have available. Until then whenever you try to find an encryption cipher you will likely find the same dozen or so that make up the majority of the market like AES, RSA, Blowfish, DES, TripleDES, TwoFish, ARCFOUR, RC2, RC4, RC5, etc. Each algorithm cipher has their own way of handling the encryption and come in a variety of asymmetric and symmetric styles and varying bit lengths that make each one unique. Since the development of many of these AES has remained in the lead as the standard of the U.S. Government and that says a lot about how it is holding up to time and new advancements in attacks.

## 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, application

Description automatically generated

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.

Google Chrome blocks my https request and won't allow a secure connection.

Graphical user interface, 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

Before Refactoring

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application

Description automatically generated

After Refactoring

Graphical user interface, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email, website

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.

Text

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.

The areas of security that are addressed within the refactored code include cryptography and this can be extended to the encryption that was used to generate the checksum and the vulnerabilities that were addressed through the Maven dependency check. Additionally, one can include client/server development under this same code refactoring because the inclusion of refactoring the http protocol to https is among the changes made. This directly affects the security involved in accessing the port used for the checksum itself. Adding layers of security to the software application follows a very methodical process and can be time consuming but when done correctly the result is an optimized and secure application. When starting the process of adding security layers it's always a good idea to run a security check, in this case with maven, to determine what the current status of the system is. Questions should be asked like, does the system run, what types of errors do we get, and how many errors do we get. Once a dependency check is run, we can start to jump into any current issues the system has, and we can fix those quickly. Then we can start adding our new refactored code that adds security benefits keeping in mind the needs of the current system. When the refactored code is in place and ready to implement it should be run to make sure it operates smoothly, and then additional dependency checks should be run to make sure that no new errors were introduced into the system. Once those are run and optimized, we can start thinking about additional layers of security like converting http to https or adding additional encryption, or even adding additional libraries that take advantage of other security benefits. Keeping that all in mind, it is important that we do not introduce code that isn't going to be used or isn't going to suit the company and application well so to avoid this the company and application should be examined to determine the best procedure in adding security layers. Most security benefits run through systems today are not plug and play forever. Once a system is set up it is very important for this system to be kept up to date. Many of the vulnerabilities that came up in the reports were due to versions of something being out of date and by simply updating the newer version these issues were resolved. Security threats change over time and so do security protocols so making sure the keep up of the system is done takes a team of individuals who are up to date with current and emerging technologies. For example, currently with SHA256 we see no way of this being an area of compromise on our hash algorithm, however as times change new ways to break this encryption can come about and be discovered so understanding the newest and most secure algorithms will help keep the system compromised for the least amount of time. Simply put, for best practices to be followed, check versions and updates constantly and make sure current systems are not compromised by outdated tools in addition to continuing looking for areas that can be added to in terms of security.

Resources:

Simplilearn. (2021, September 15). *What is Data Encryption: Algorithms, methods & TECHNIQUES: SIMPLILEARN*. Simplilearn.com. Retrieved September 30, 2021, from <https://www.simplilearn.com/data-encryption-methods-article>.

Sidhpurwala, H. (2019, March 19). *A brief history of cryptography*. Red Hat Customer Portal. Retrieved September 30, 2021, from <https://access.redhat.com/blogs/766093/posts/1976023>.