****

# Practices for Secure Software Report

Table of Contents

[Document Revision History 3](#_Toc102040754)

[Client 3](#_Toc102040755)

[Instructions 3](#_Toc102040756)

[Developer 4](#_Toc102040757)

[1. Algorithm Cipher 4](#_Toc102040758)

[2. Certificate Generation 4](#_Toc102040759)

[3. Deploy Cipher 4](#_Toc102040760)

[4. Secure Communications 4](#_Toc102040761)

[5. Secondary Testing 4](#_Toc102040762)

[6. Functional Testing 4](#_Toc102040763)

[7. Summary 4](#_Toc102040764)

[8. Industry Standard Best Practices 4](#_Toc102040765)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **10/16/2022** | **Jonathan Carmichael** |  |

## 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

Jonathan Carmichael

## Algorithm Cipher

*Recommend an appropriate encryption algorithm**cipher to deploy, given the security vulnerabilities, and justify your reasoning. Review the scenario and the supporting materials to support your recommendation. In your practices for secure software report, be sure to address the following:*

1. *Provide a brief, high-level overview of the encryption algorithm cipher.*
2. *Discuss the hash functions and bit levels of the cipher.*
3. *Explain the use of random numbers, symmetric versus non-symmetric keys, and so on.*
4. *Describe the history and current state of encryption algorithms.*

Recommended Cipher: **SHA-256**

High-Level Overview:

SHA-256, or Secure Hash Algorithm – 256 bits, is an algorithm designed by the United States National Security Agency. Basically, the algorithm breaks input into multiple fixed-length blocks of random bits. Following this, a compression function is applied to the very first block with an initialization vector. The output of the compression moves into the next compression in line, along with the next message block. This process repeats until all message blocks have been processed thoroughly by repeating a sequence of six distinct compressions. Each of these compressions take input from the previous output and combine it with the next message block. Lastly, this chain of compressions is moved to one final compression that produces a final value, aka the checksum. This hash function is amongst the strongest, most reliable, has functions that are currently available due to advantages of maintaining larger key-sizes (leading to lower hash collision) and the fact that it has yet to be compromised.

Symmetric vs. Asymmetric Keys:

These keys are generated using random numbers, as keys that are generated with non-random numbers have a much higher chance to be cracked. In terms of comparing symmetric keys vs. asymmetric keys, symmetric encryption only utilizes a single key to encrypt/decrypt messages. Whereas, asymmetric uses a public/private key pair to encrypt data. Ultimately, symmetric keys are less secure. Asymmetric keys can only be utilized when combined with a private key, making them more secure.

History of Encryption Algorithms:

Ancient Egypt hieroglyphs are the first known use of cryptography, dating all the way back to 1900 BC. The was a substitution (secret) set of hieroglyphics that was utilized to cipher known hieroglyphs, and this method was used for thousands of years moving forward. Julius Caesar is credited with the next level of advancement, where specific alphabetical letters would be scrambled/swapped with others in order to challenge communications. Next in the 16th century, the development of a cipher key (designed by Vigenere) was created/applied to help translate and look up a specific letter to substitute with the original in any message. Basically, this is where the modern-day alphabet comes from. Moving ahead again to the 19th century, the first electro-mechanical encryption was developed by Edward Hebern. Utilizing what is known as the Hebern Rotor Machine, this device combined mechanical parts of a standard typewriter with the electrical parts of an electric typewriter, connecting the two through a scrambler. This application would become the primary source of encryption throughout WW1 and WW2. The first digital block cipher (48-bits initially) would be known as Lucifer and would be developed in the 1970s by a group of individuals working at IBM. Credited to Horst Feistel, this would become the precursor to the first DES system known as DTD-1. This cipher would ultimately be broken, but in 2000 the NIST accepted an algorithm initially known as “Rijndael” and renamed it AES. Since then, SHA-256 is considered to be the most reliable hash algorithm.

## Certificate Generation

*Generate appropriate self-signed certificates using the Java Keytool in Eclipse.*

1. *To demonstrate that the certificate was correctly generated:*
   1. *Export your certificates (CER file).*
   2. *Submit a screenshot of the CER file in your practices for secure software report.*

Text

Description automatically generated

## Deploy Cipher

Insert a screenshot below of the checksum verification.

I couldn’t get the program to connect to my localhost. Below is my refactored code, as well as what occurs when I run the program. As you can see, the “@RequestMapping(“/hash”)” is located within the class this time, rather than out like I had done in module 5’s assignment. I attempted to look up the error with the Tomcat server, however, I was ultimately unable to resolve the issue. I was not having errors within my code and imported the essential annotations. Furthermore, I’ve attached a third screenshot of my “application.properties” showcasing my certificate information and editing the “????” to have the required information.

Graphical user interface, text, application

Description automatically generated

Text

Description automatically generated

Graphical user interface, text, application

Description automatically generated

## Secure Communications

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

**See information in number 3. I was unable to connect localhost or establish a secure connection.**

## Secondary Testing/Functional Testing

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

Graphical user interface, text, application

Description automatically generated

Graphical user interface, application, Teams

Description automatically generated

Graphical user interface, text, application

Description automatically generated

**COULD NOT COMPLETE SECONDARY-TESTING DUE TO CHECKSUM AND CIPHER ISSUES.**

## Summary

I attempted to create a secure, 256-bit hash algorithm so I could secure our clients’ data. I utilized cryptography techniques as well as creating code for functionality within CheckSum verification. Unfortunately, I was unable to get this portion of the assignment completed due to multiple issues ranging from errors that I did not understand how to resolve in time, as well as simply being unable to connect to my localhost in general. I was able to properly generate an SSL certificate to ensure site security, as well as develop crypted communication security. As for the Sha-256 encryption algorithm, I feel like I developed the code sufficiently, however, due to said issues earlier, or even unknown ones that I do not understand now, was unable to secure that sessions cannot be performed over HTTP, without HTTPS reinforcement. Although I failed to incorporate this, I understand that it adds significant value in securing against additional unforeseen attacks, as well as preventing older browsers from establishing worse connections, leaving them more susceptible to attacks.

Another success was building a dependency check, but without being able to properly run the refactored code, I couldn’t deflect additional vulnerabilities. However, regarding best practices for maintaining security, I think it’s very important to always be scanning for new vulnerabilities. There are new ones being discovered all the time, and regular dependency checks can highlight new threats that are. All new threats will be able to be mitigated using recommendations provided by any sites reporting them. With this, securing code, as well as creating input validation for any/all inputs is crucial, too. This will assist in helping the application operate smoothly, as well as providing better, more involved functionality for said program.