****

# 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** | **12/10/2023** | **Jeyapriya Saravanakumar** | **Initial Version** |

## Client



## Developer

Jeyapriya Saravanakumar

## Algorithm Cipher

I recommend using AES (Advanced Encryption Standard) for client-specific needs for file encryption. AES uses a symmetric algorithm, i.e., the same key for encryption and decryption. It comes with AES-128, AES-192, and AES-256 bit length keys. AES-256 is considered the most secure and is often recommended for use.

AES operates on a fixed-size block, which is 128 bits, suitable for most common applications. Key generation consists of two steps: first, to generate the original secret key, which is used to create a set of round keys, and next, to generate several round keys based on the key bit length.

AES has many advantages, like versatility, efficiency, wide adoption, standardization, and resistance to known attacks, but at the same time, it has some drawbacks as well, like key management, i.e., keeping the secret key securely and limited block size.

Cipher algorithms are used in AES and DES (Data Encryption Standard) for encryption and decryption of data. Hash functions, such as SHA-256 or MD5, are used for data integrity verification and digital signatures. Bit levels refer to the size of the blocks or output in both cases.

Random numbers are unpredictable and used in cipher algorithms to generate secret keys. Symmetric Cryptography uses a single key for encryption and decryption (AES, DES), and Asymmetric Cryptography uses a pair of keys, i.e., private and public (RSA, ECC).

## Certificate Generation

Self-signed certificate generation steps using Java keytool command as shown below.

"C:\Program Files\Java\jdk-17\bin\keytool.exe" -genkey -keyalg RSA -alias selfsigned -keypass changeit -keystore keystore.jks -storepass changeit -validity 360 -keysize 2048

A screenshot of a computer

Description automatically generated

"C:\Program Files\Java\jdk-17\bin\keytool.exe" -export -alias selfsigned -storepass changeit -file server.cer -keystore keystore.jks

A screenshot of a computer

Description automatically generated

"C:\Program Files\Java\jdk-17\bin\keytool.exe" -printcert -file server.cer

A screenshot of a computer

Description automatically generated

"C:\Program Files\Java\jdk-17\bin\keytool.exe" -export -alias selfsigned -storepass changeit -file server.cer -keystore keystore.jks

A screenshot of a computer

Description automatically generated

## Deploy Cipher

Refactored SslServerApplication.java code:

package com.snhu.sslserver;

import java.security.MessageDigest;

import java.security.NoSuchAlgorithmException;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RestController;

@SpringBootApplication

public class SslServerApplication {

public static void main(String[] args) {

SpringApplication.run(SslServerApplication.class, args);

}

}

@RestController

class ServerController{

@RequestMapping("/hash")

public String myHash(){

String input = "Jeyapriya Saravanakumar";

String data = generateChecksum(input);

return "<p>data:"+data;

}

private String generateChecksum(String inputData) {

String output = "";

try {

output += inputData;

//SHA-256 algorithm used

output += "<p> Name of the Cipher Algorithm Used: SHA-256";

//initialize MessageDigest object with SHA-256 algorithm instance

MessageDigest messageDigest = MessageDigest.getInstance("SHA-256");

//get byte value of the input

byte[] hashBytes = messageDigest.digest(inputData.getBytes());

//convert byte array to hexadecimal string

StringBuilder sb = new StringBuilder();

for (byte b : hashBytes) {

String hexValue = Integer.toHexString(0xff & b);

if(hexValue.length() == 1) {

sb.append('0');

}

sb.append(hexValue);

}

//add generated checksum value

output += "<p> CheckSum Value: " + sb.toString();

} catch (NoSuchAlgorithmException e) {

e.printStackTrace();

}

return output;

}

}

Verification:

A screenshot of a computer

Description automatically generated

## Secure Communications

Copied generated keystore.jks file under application resources folder.

Refactored application.properties to added certificates details.

application.properties

server.port=8443

server.ssl.key-alias=selfsigned

server.ssl.key-store-password=changeit

server.ssl.key-store=src\\main\\resources\\keystore.jks

server.ssl.key-store-type=JKS

A screenshot of a computer

Description automatically generated

## Secondary Testing

Updated latest dependency-check-maven plugin version 9.0.4 in pom.xml and executed the “maven verify” command to generate vulnerabilities report.

A screenshot of a computer

Description automatically generated

Before refactoring the code:

A screenshot of a computer

Description automatically generated

After refactoring the code: (No new issues are introduced)

A screenshot of a computer

Description automatically generated

## Functional Testing

As part of the manual review, for error handling, if the checksum value cannot be generated, we should inform the user of a failure. However, the current code is skipping the checksum value information completely without giving any reason—added screen shots of before and after refactoring the code and verification.

Added incorrect Cipher algorithm as SHA-256

A screenshot of a computer

Description automatically generated

The “CheckSum Value:” value section was completely missed without giving any reason.

A screenshot of a computer

Description automatically generated

Refactored the code to provide reason why check sum generation failed.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

## Summary

As part of the manual review, scanned all the application code and addressed the “Code Review – Views” and “Code Review - Controllers” sections from the Vulnerability Assessment Process Flow Diagram.

Highlighted with red color circles.

A screenshot of a computer

Description automatically generated

## Industry Standard Best Practices

The input name should be validated by checking the allowed maximum length to avoid any invalid or script input. ACCEPT and PRODUCE formats can be added with value as JSON to avoid any invalid input/output formats.

The application has only a controller but not an API; the API interface should be defined with API signatures. API should send only defined HTTP codes for each scenario, like for success response 200, invalid input 400 error code, and any system error 500 error code.

A try/catch block should be added, and the error message and HTTP code should be sent to the user. Also, OAuth / JWT token validation can be added for authentication purposes to access only if the token is successfully validated. Scoped roles can be added for each API to implement authorization.