****

# CS 305 Project Two

**Practices for Secure Software Report**

Table of Contents

[CS 305 Project Two 1](#_Toc106447730)

[Document Revision History 3](#_Toc106447731)

[Client 3](#_Toc106447732)

[Developer 4](#_Toc106447733)

[1. Algorithm Cipher 4](#_Toc106447734)

[2. Certificate Generation 4](#_Toc106447735)

[3. Deploy Cipher 6](#_Toc106447736)

[4. Secure Communications 6](#_Toc106447737)

[5. Secondary Testing 9](#_Toc106447738)

[6. Functional Testing 9](#_Toc106447739)

[7. Summary 10](#_Toc106447740)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **June 14, 2022** | **Brandon Hobbs** | **First Release** |

## Client



## Developer

Brandon Hobbs

## 1. Algorithm Cipher

Artemis’ data at rest needs to be protected. As I have conjectured before, the data being financial in nature makes for a more appealing target. To secure the data I would suggest an AES128 encryption utilizing the CTR mode. However, because AES is a symmetric encryption technique the keys need to be secured. RSA should be employed to protect the keys. What is meant by 128-bit?

*128-bit encryption primarily refers to the length of the encryption or decryption key. It is considered secure because it would take massive computation and virtually thousands of years to be cracked. For example, it would take 2128 different combinations to break the encryption key, which is out of reach for even the most powerful computers. (*Techopedia)

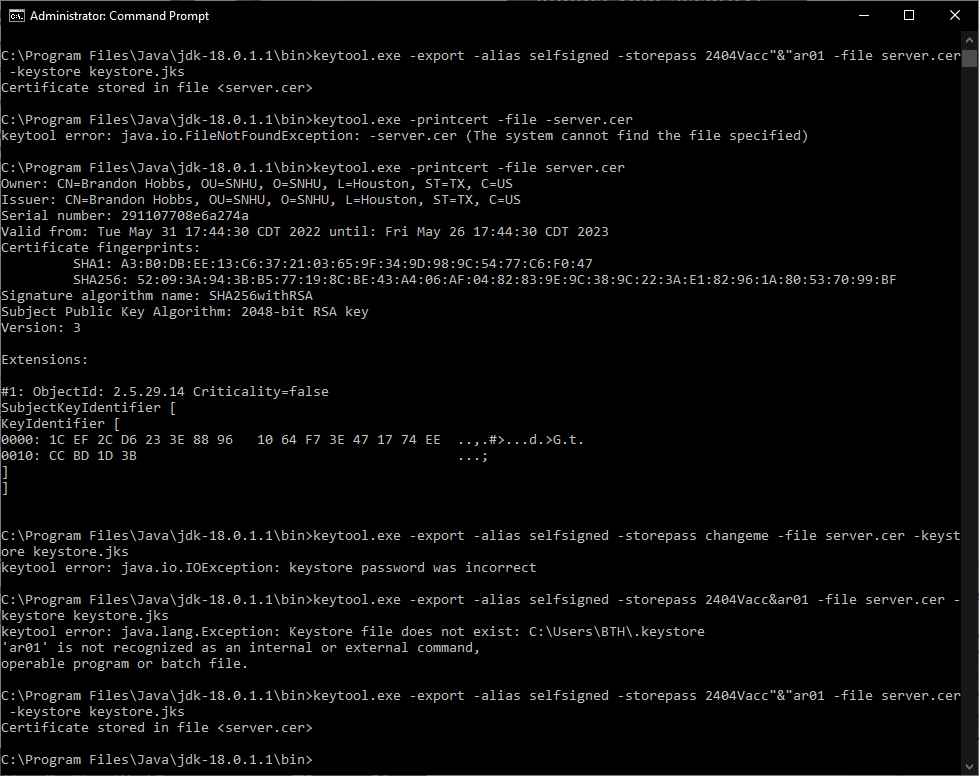
I would choose this scheme because as Franklin (2021) says, “...the bulk of the data…is encrypted by the speedy AES algorithm. To get the secret key required to decrypt that data…[the] sender then uses that public key and RSA to encrypt and transmit to each recipient their own secret AES key…”. This scheme has the advantage of protecting the most vulnerable part of AES (symmetric key) with a more rigid encryption mechanism but without subjecting users to the slower performance of RSA (asymmetric key).

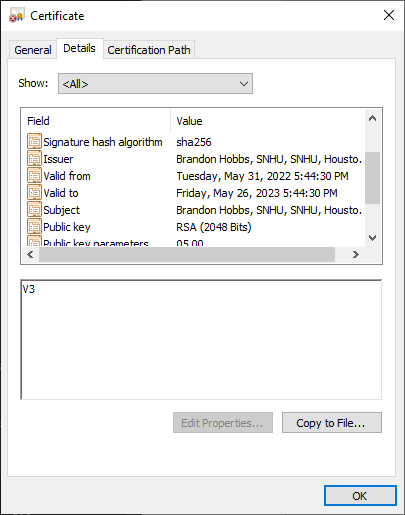
Moreover, I would choose 128-bit AES strength utilizing the CTR method. CTR has, according to Baeldung (2021), the advantage of being able to parallelize encryption and decryption. If there are large volumes of encrypted data, this would aid user satisfaction. CTR is preferred over the older DES and 3DES. ECB should not be used for any data larger than 128 bits due to the repeating patterns becoming obvious.

128-bit AES is suggested over 256-bits for similar performant reasons. “128 bit key is enough security for most of every use case with the exception of quantum computer protection. Also using 128 bit encrypts faster than 256 bit and the key-schedule for 128 bit keys seems to be better protected against related-key attacks (however this is irrelevant to most real-world uses)” (Baeldung, 2021).

## 2. Certificate Generation

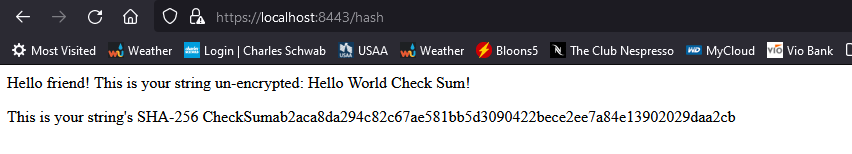
Java’s Keystore tool included in the SDK was used to generate a self-signed certificate.

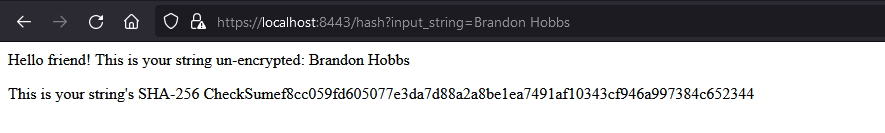


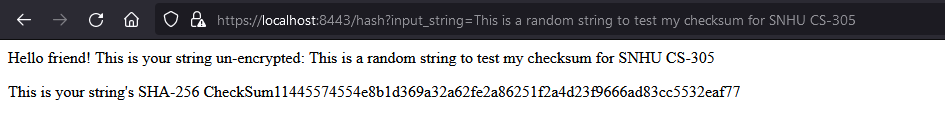


## 3. Deploy Cipher

The RESTful API was built to return a checksum for any sent string. Further work is needed to harden the and parameterize the API for any injection attacks but for now the checksum using SHA-256 works.

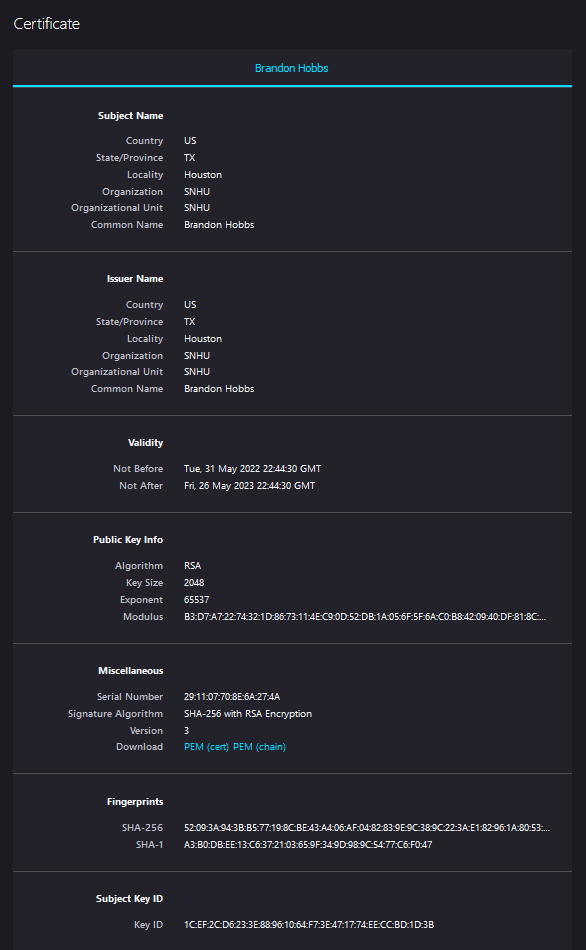
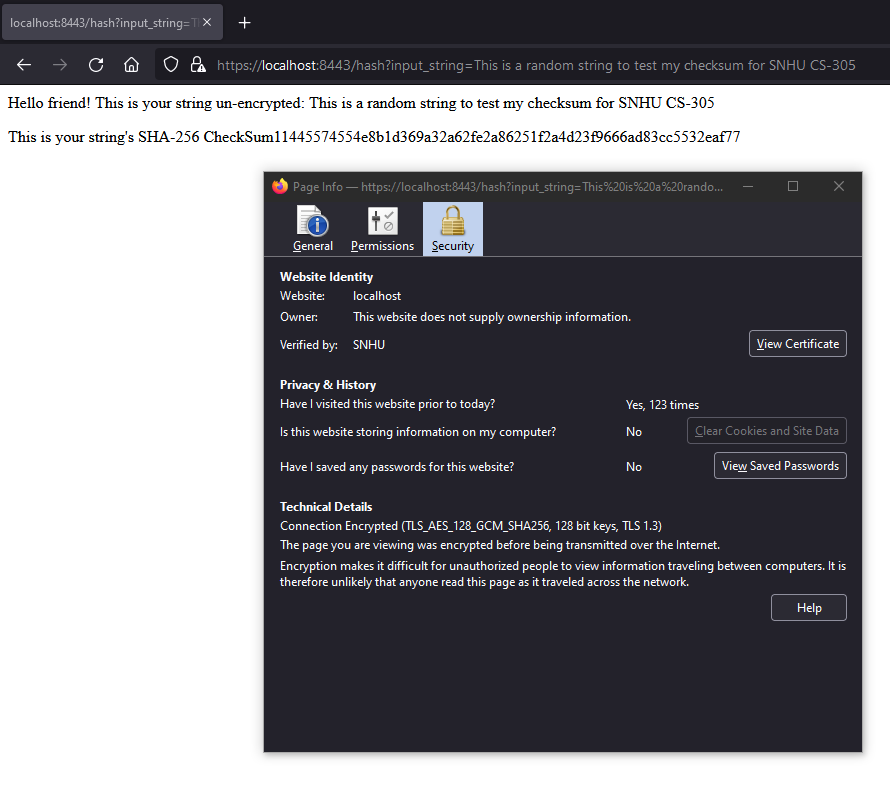




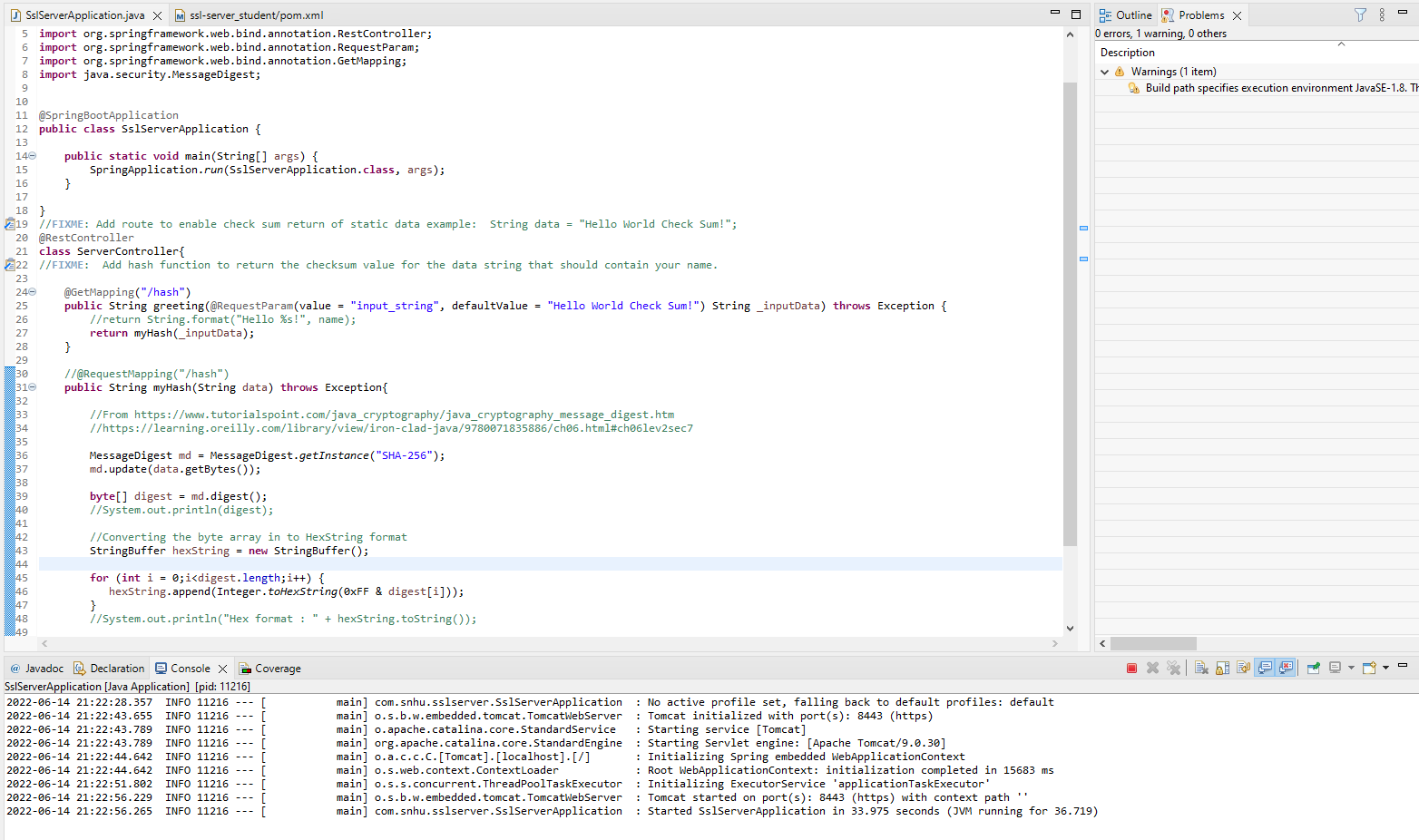


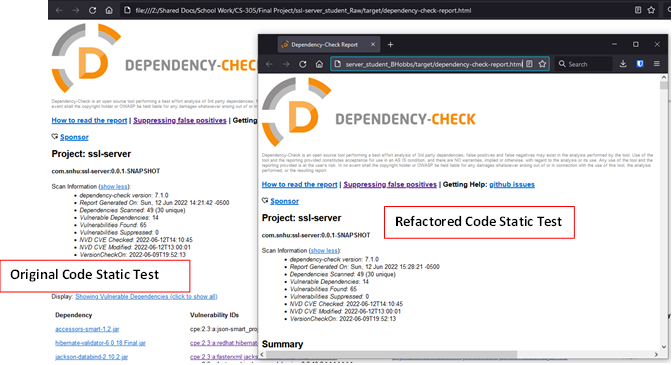
## 4. Secure Communications

The connection to the API is secured by TLS\_AES\_128 and a self-signed certificate (see previous section).



## 5. Secondary Testing

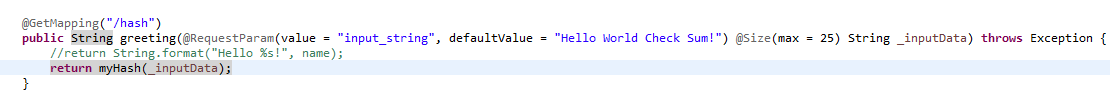
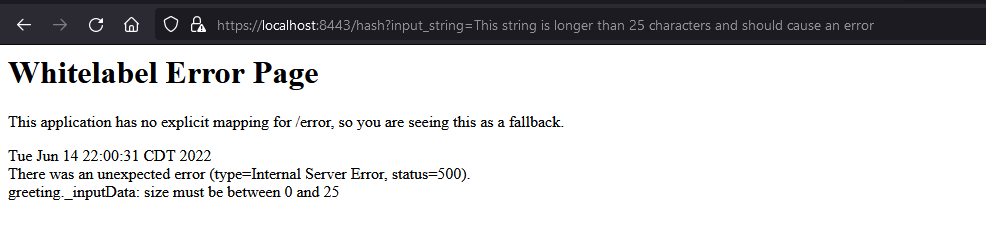




## 6. Functional Testing

User input to the API is not sanitized. For example, within the greeting method the value from the user is passed directly to be hashed. This could help someone exploit any weakness. Multiple API calls would need be slowed so prevent a DOS. The input string also needs to be sanitized to prevent buffer overruns.

An example of preventing DOS attacks can be seen in Srivastava (2020) using the Bucket4J to rate limit the number of API calls.

## 7. Summary

The code was refactored so that the web traffic was sent over HTTPs protocols. This tries to prevent websites from having their information broadcast to anyone spying on the network. This HTTPs communication was secured using an AES 128-bit cipher and signed using a personal certificate – this eventually will need to be changed to a more publicly trusted CA root-type certificate.

The API created was designed to return a checksum of a string. A checksum may be used to check the “fingerprints” of a digital object. The object is pushed through the hashing algorithm and the hash is generated. A well-designed hash has no collisions – meaning two dissimilar digital objects cannot produce the same hash – and so a hash allows someone to validate if the file they are receiving has been manipulated. The checksum was generated using SHA-256 which has a probability of two hashes accidentally colliding of approximately 4.3\*10-60. Other readily available hashing functions, MD5, SHA-0 and -1, all have collisions found.

Collisions would allow someone to potentially spoof the checksum or reverse-engineer the original file (if there was something to be kept secret).

The code was also passed through a static analyzer to check for Common Vulnerabilities and Exposures (CVE). While static testing is good it doesn’t catch all vulnerabilities. A dynamic scanner (errors found at run time) may help to catch other vulnerabilities – memory leaks, HTTPS- redirects, etc.

Finally, the API was parameterized so that input must between a string length of 1- 25 characters to prevent buffer overrun attacks. The final step needed would be to rate-limit the API calls. Rate limit could be achieved using Bucket4j – see Srivastava (2020).

**References**

Baeldung. (2021, Nov. 14). *Java AES encryption and decryption.* <https://www.baeldung.com/java-aes-encryption-decryption>

Crane, C. (2019, June 28). *10 data privacy and encryption laws every business needs to know.* TheSSLStore. <https://www.thesslstore.com/blog/10-data-privacy-and-encryption-laws-every-business-needs-to-know/>

Franklin, R. (2021, Mar. 13). *AES vs. RSA encryption: what are the differences?* Precisely. <https://www.precisely.com/blog/data-security/aes-vs-rsa-encryption-differences>

Srivastava, P. (2020, Sept. 9). *Rate limiting a Spring API using Bucket4j*. Baeldung. <https://www.baeldung.com/spring-bucket4j>

Techopedia. *128-bit encryption*. Retrieved June 14, 2022 from https://www.techopedia.com/definition/29708/128-bit-encryption

Townsend, P. (2019, Mar 25). *RSA vs AES encryption - a primer*. TownsendSecurity. <https://info.townsendsecurity.com/rsa-vs-aes-encryption-a-primer>