****

# Practices for Secure Software Report

**Table of Contents**

[**Document Revision History 3**](#_30j0zll)

[**Client 3**](#_1fob9te)

[**Instructions 3**](#_3znysh7)

[**Developer 4**](#_2et92p0)

[**1. Algorithm Cipher 4**](#_tyjcwt)

[**2. Certificate Generation 4**](#_3dy6vkm)

[**3. Deploy Cipher 4**](#_1t3h5sf)

[**4. Secure Communications 4**](#_4d34og8)

[**5. Secondary Testing 4**](#_2s8eyo1)

[**6. Functional Testing 4**](#_17dp8vu)

[**7. Summary 4**](#_3rdcrjn)

[**8. Industry Standard Best Practices 4**](#_26in1rg)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **12/05/2023** | **William Fishburn** |  |

## 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

William Fishburn

## Algorithm Cipher

Artemis Financial's primary goal is to provide global clients with financial programs. In line with this objective, I recommend opting for the SHA-256 encryption algorithm cipher, ensuring comprehensive protection of all information from external breaches. Recognized as one of the most secure algorithm ciphers, SHA-256 is deemed practically impervious to cracking, requiring years of brute force attempts for a potential breach. In dealings with financial institutions, SHA-256 consistently emerges as the preferred cipher.

The SHA-256 hash function, characterized by its randomness at bit levels, involves compressing input values before generating the hash value, which represents the compressed data. The encryption length is determined by bit levels, and a 256-bit encryption boasts an extensive array of possible combinations, heightening the difficulty for hackers to gain unauthorized access. The infusion of randomness introduces unpredictability, a key factor in enhancing security.

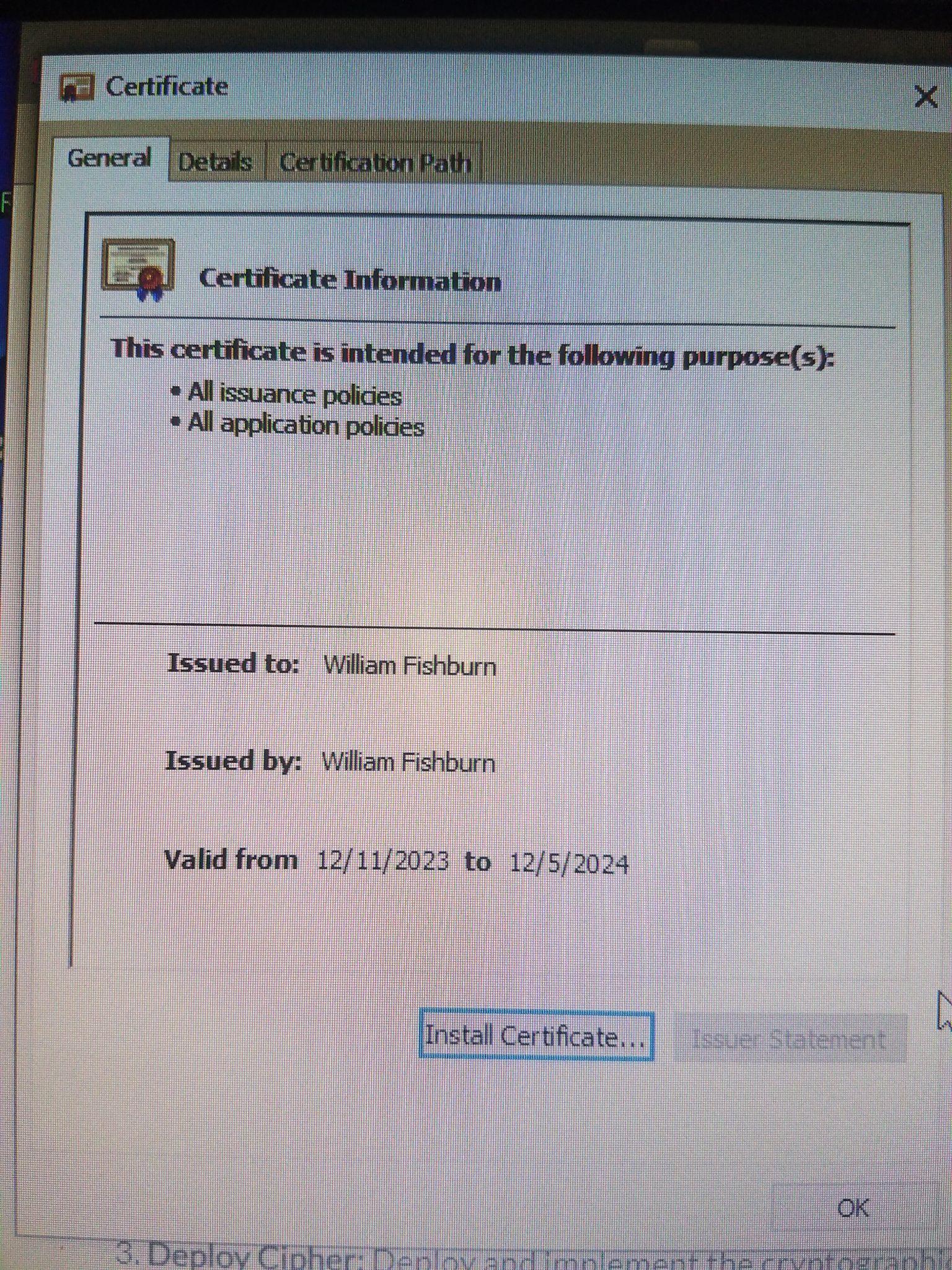
Symmetric keys, requiring only one key, provide a faster execution, which is a significant advantage highlighted in the widely-used AES-256 encryption method for plaintext. Conversely, asymmetric keys, necessitating two keys, are regarded as more secure, making them a popular choice for ensuring the security of internet communications.

Despite being symmetric, AES-256 ensures robust security through its key length and advanced encryption techniques. AES-256 employs a 256-bit key, which exponentially increases the number of possible combinations, enhancing resistance against brute-force attacks. Additionally, its intricate substitution-permutation network structure and multiple rounds of encryption contribute to the algorithm's resilience, making it highly secure for encrypting sensitive information. While asymmetric keys are favored for certain scenarios, the formidable security features of AES-256 demonstrate the efficacy of symmetric key cryptography in safeguarding data.

The earliest use of cryptography dates back to around 1900 BC in the tomb of Egyptian nobleman Khnumhotep II. The scribe used unconventional hieroglyphic symbols to alter the text's form, not to hide the message but to create a dignified appearance. This marks the oldest known instance of such text transformation. Over time, encryption techniques have evolved, with modern approaches incorporating advanced cryptographic methods to address contemporary security challenges.

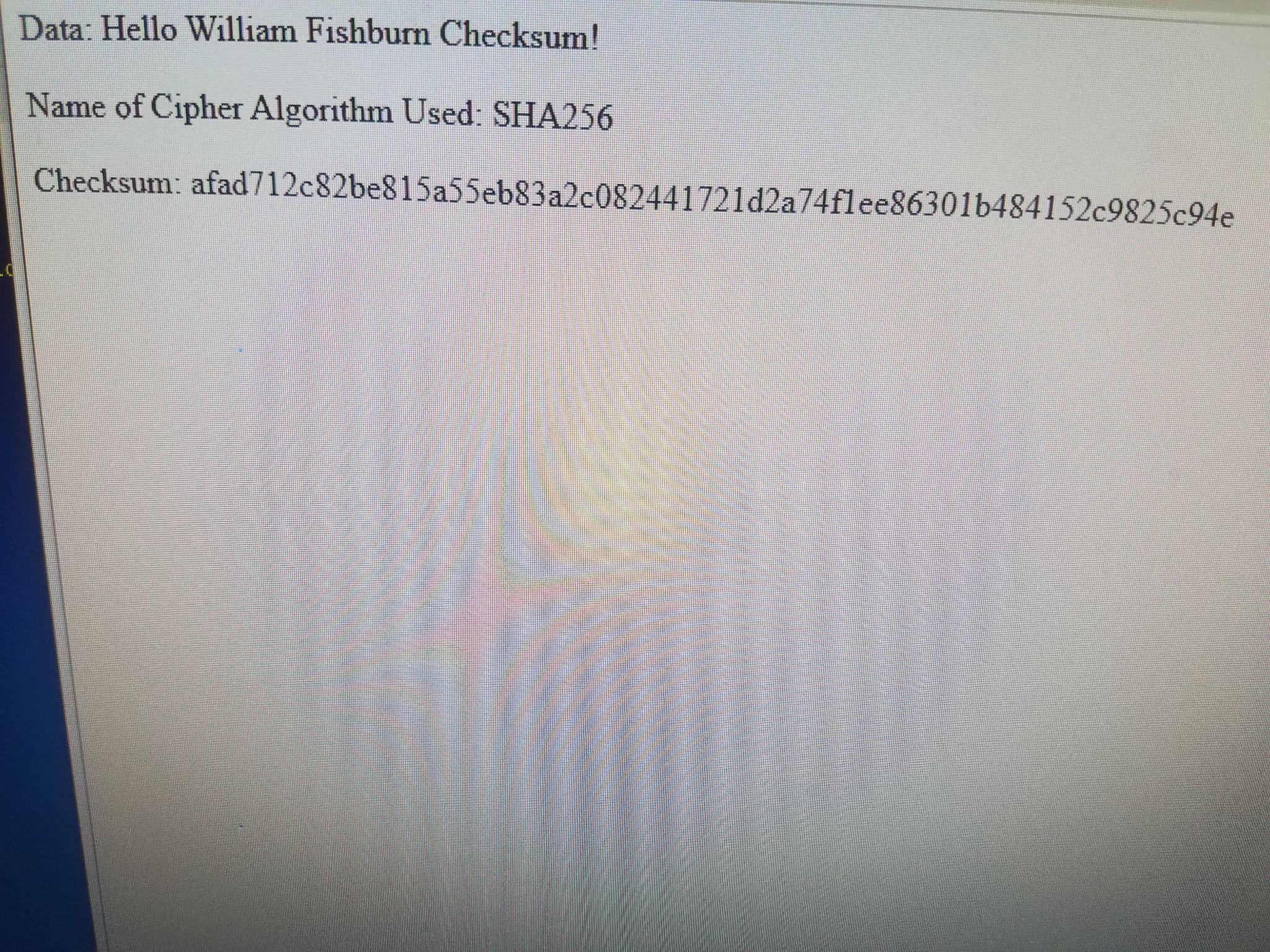
## Certificate Generation

Insert a screenshot below of the CER file.



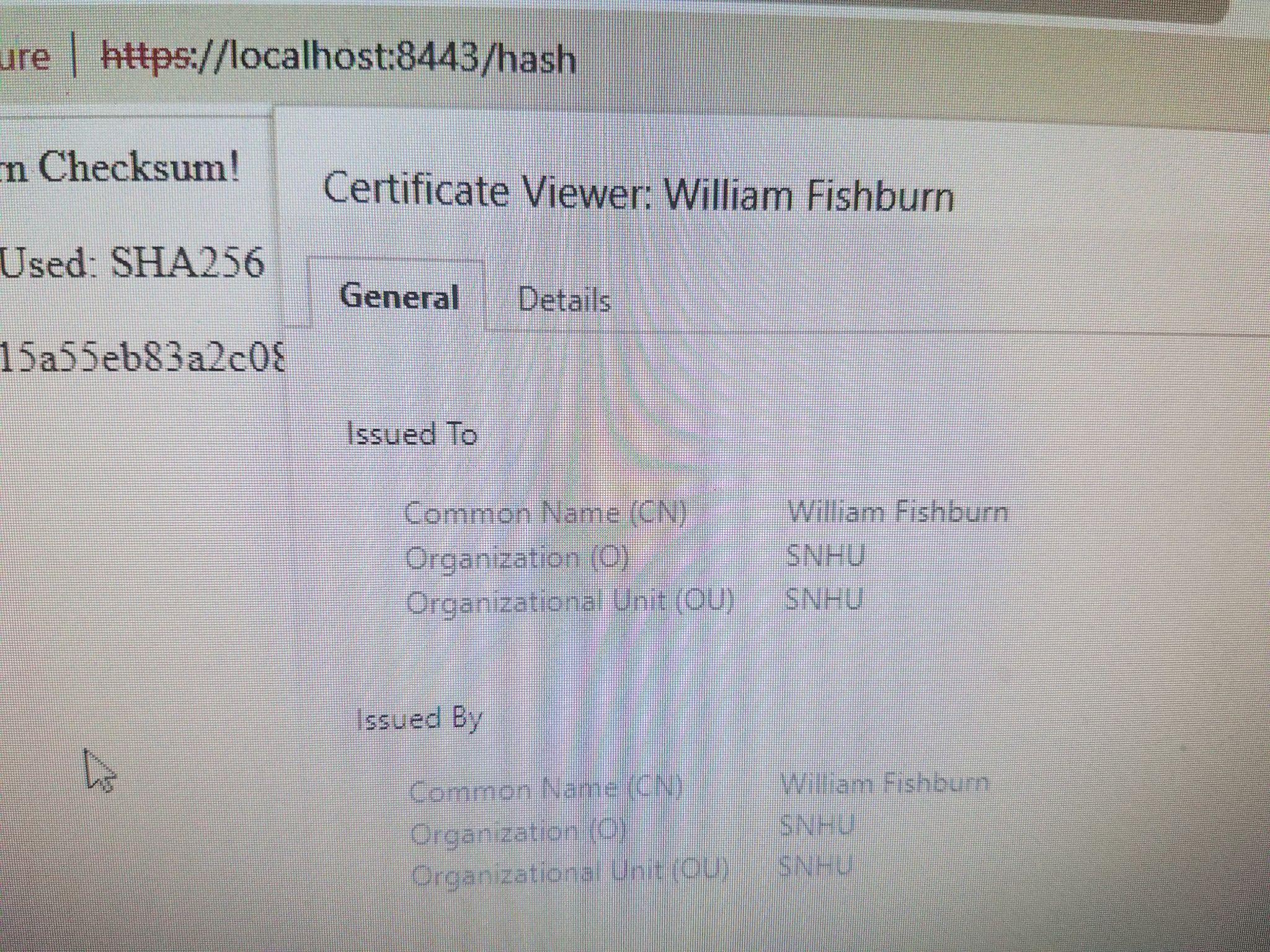
## Deploy Cipher

Insert a screenshot below of the checksum verification.

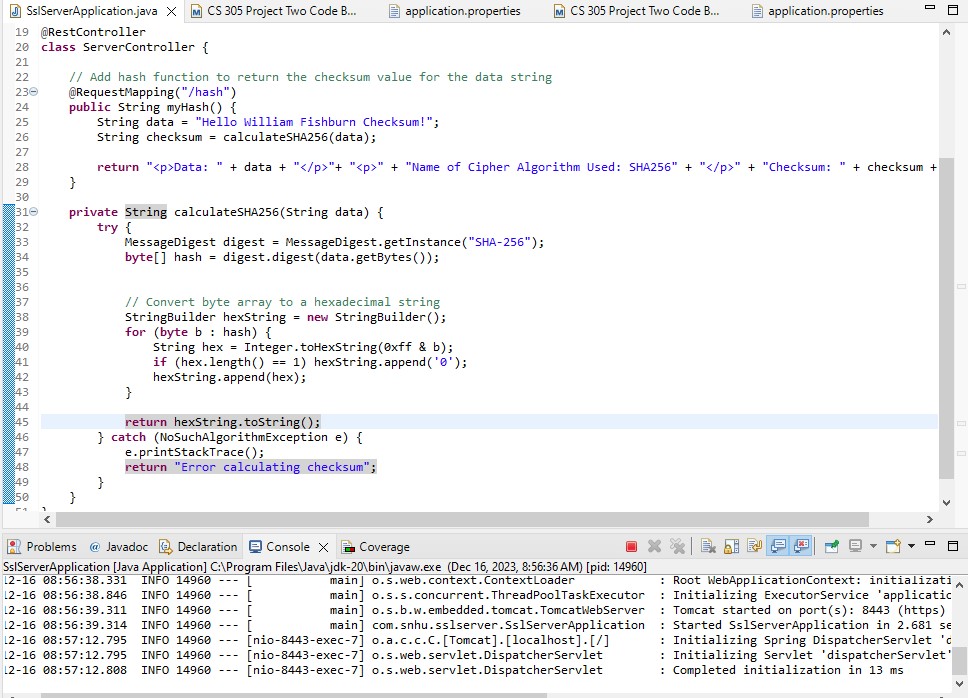


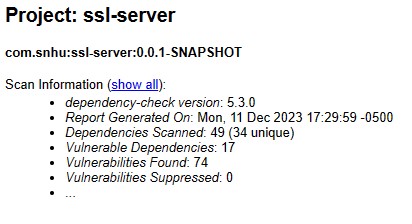
## Secure Communications

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

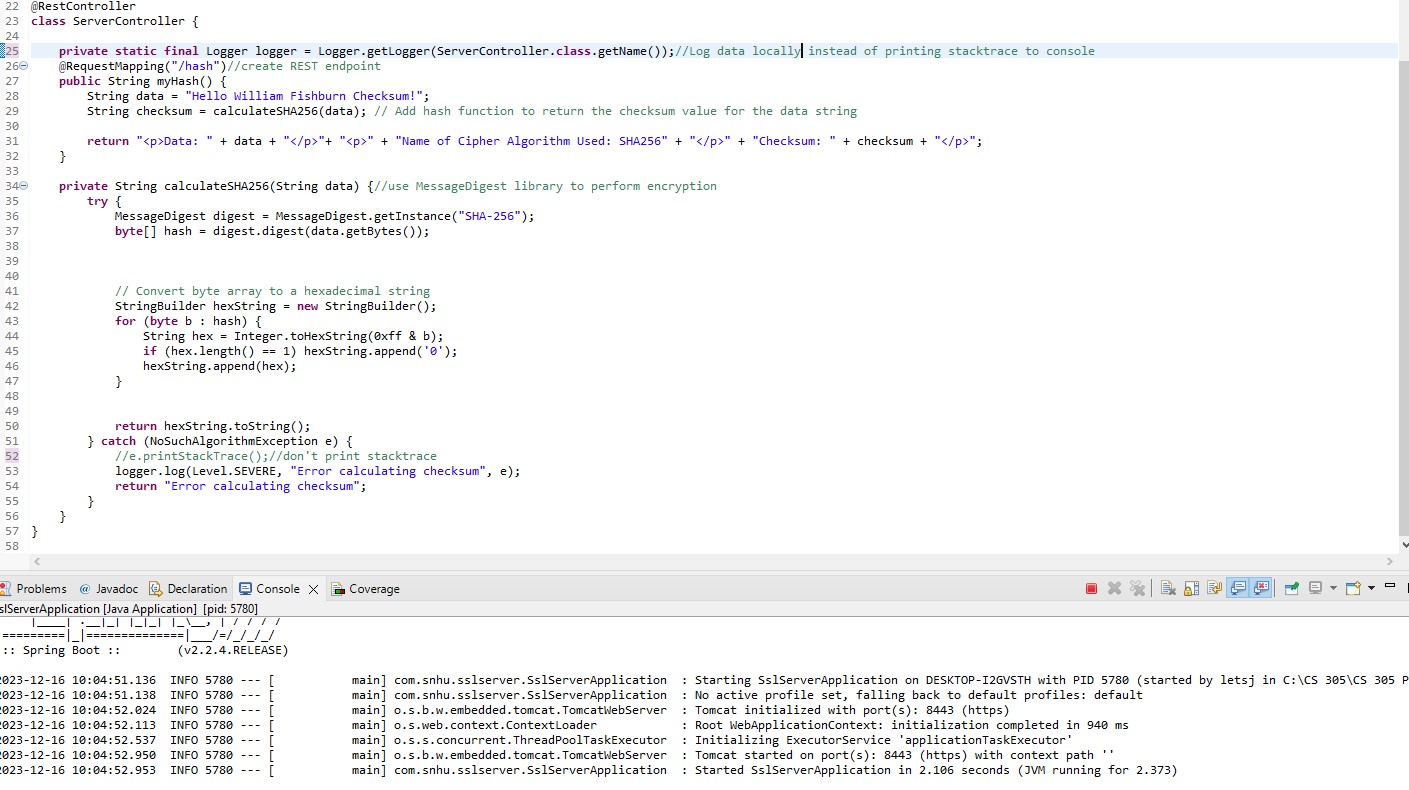


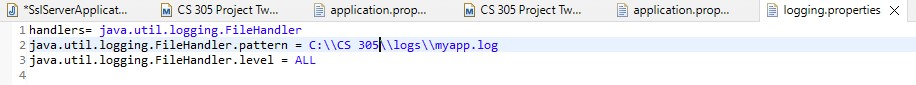
## Secondary Testing

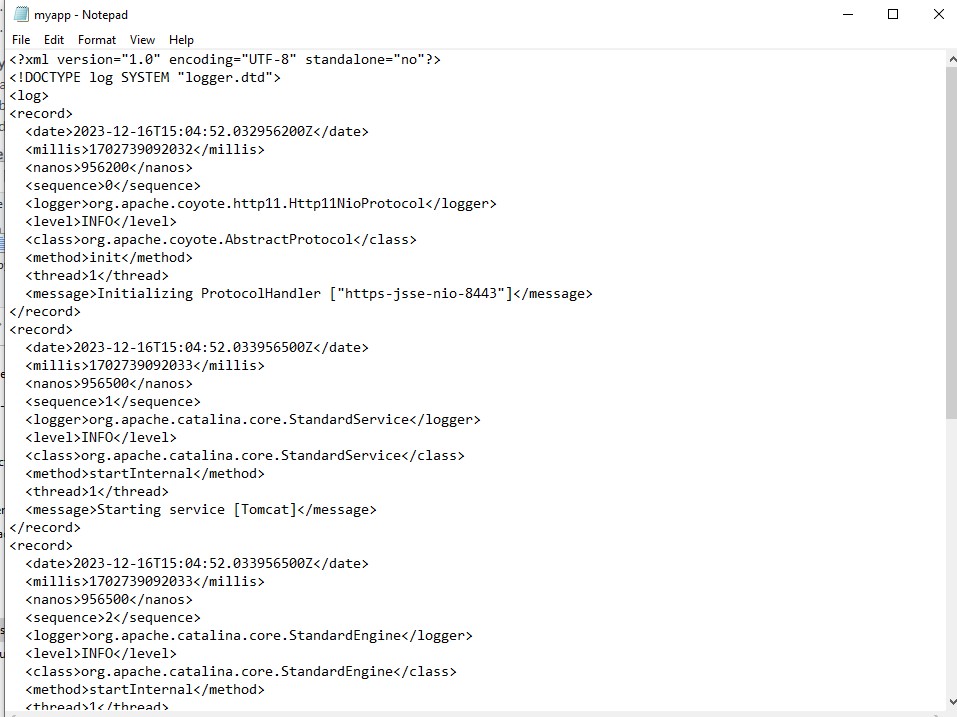
Insert screenshots below of the refactored code executed without errors and the dependency-check report.



## Functional Testing

Insert a screenshot below of the refactored code executed without errors.





## Summary

In the refactored code, security considerations were addressed through the introduction of a Logger instance in the ServerController class, combined with the configuration of a logging.properties file to store logs locally instead of printing the stack trace to the console; this represents a crucial security improvement. By redirecting error information to a log file and avoiding console output, sensitive details about potential vulnerabilities are less likely to be exposed to attackers in a production environment, adding a layer of security by ensuring that log data is managed and stored securely. This coding process included the creation of a logging configuration file within “/src/main/resources”, the addition of required libraries for logging such as “java.util.logging.\*”, the configuration of the logging system in the main method, and specifying the logging configuration as a VM argument in the Run Configuration.

## Industry Standard Best Practices

To keep the software application secure, I followed industry-standard practices in a way that makes sense for our code. I implemented the `java.util.logging` framework and set up a dedicated `Logger` in the `ServerController` class. This helps manage errors securely by directing them to a local log file (`myapp.log`) instead of showing them in the console. Following industry best practices, I also included inline comments that serve as clear documentation for Devs. Using a separate `logging.properties` file for configuration was an additional standard practice that enhances security while keeping code flexible.

Implementing industry-standard secure coding practices significantly benefits the overall health of the company by reducing the risk of security vulnerabilities and potential breaches, addressing critical concerns in today's threat landscape. Aligning with recognized security standards ensures regulatory compliance, protecting the company against legal and industry-specific security requirements. Enhanced maintenance efficiency through modular configuration settings, provides a practical approach for seamless adjustments without unnecessary modifications to the application code. This approach nurtures a security-conscious culture within the development team, encouraging proactive consideration of security aspects throughout the software development lifecycle.

**Additional Details:**  
  
In my submission, I will attach a .zip archive of the completed refactored code which will include the following resources: sever.cer SSL certificate, keystore.jks container, as well as the local logs file that I created using the logging.properties resource.