

# 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** | **6’10/23** | **Joshua Wozny** |  |

## Client



## Developer

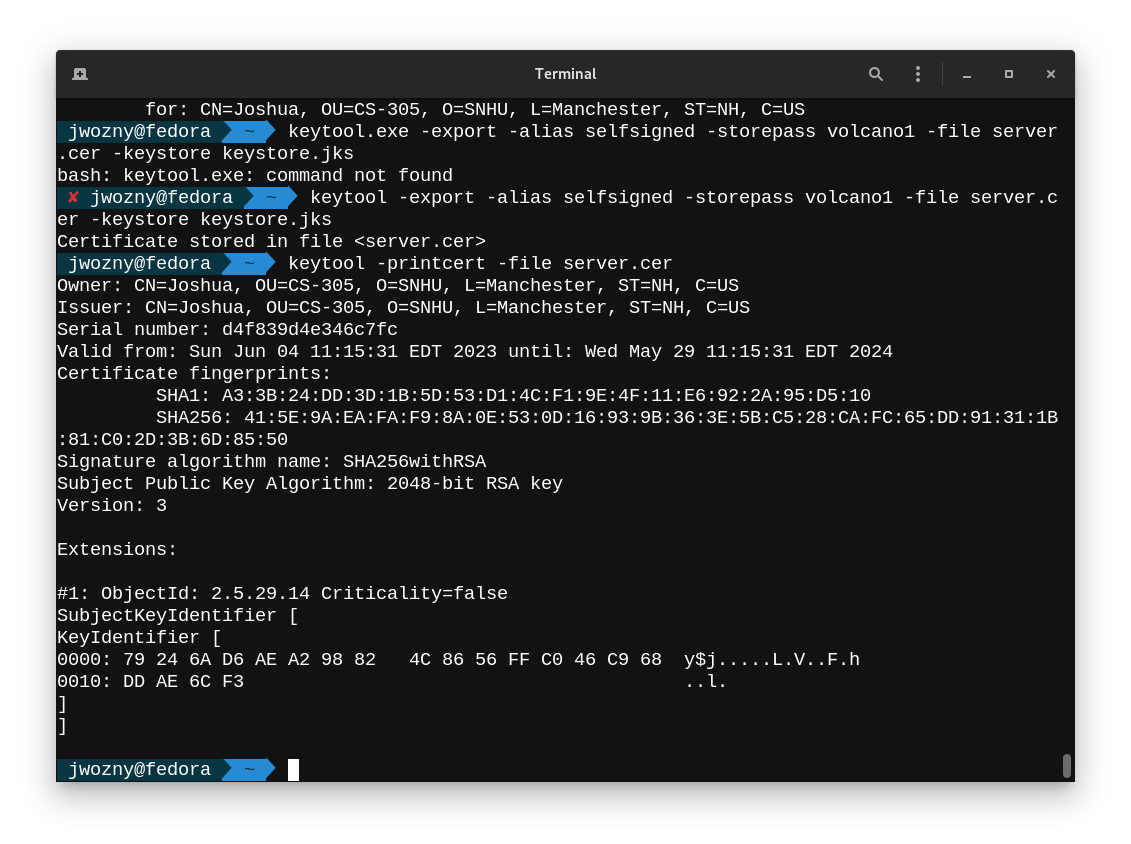
Joshua Wozny

## Algorithm Cipher

To ensure secure communication and data verification using a checksum, I would recommend using a combination of encryption and hash functions using the Advanced Encryption Standard Algorithm (AES), with 256-bits and the Secure Hash Algorithm 256-bit (SHA-256).

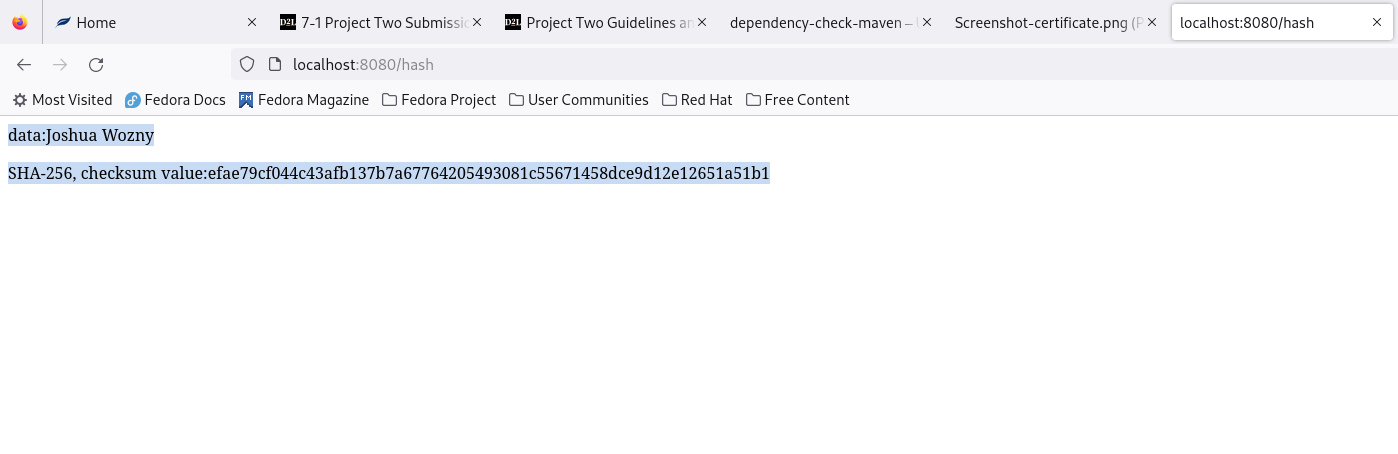
* AES is a widely used symmetric encryption algorithm that provides a high level of security. It supports key sizes of 128, 192, and 256 bits and is considered secure for most applications. You can choose a key size based on your security requirements (Java Security Standard Algorithm Names, 2017) . Block ciphers built as permutations, like AES, ensure that they are collision free. Collissions occur when a has value occurs for two or more different inputs, which would limit accurate decryption and reduce security overall.
* SHA-256 is a cryptographic hash function that generates a fixed-size output of 256 bits. It is widely used for data integrity checks and can provide a high level of assurance that the data has not been tampered with (Ramesh Fadatare, 2020).

## Certificate Generation



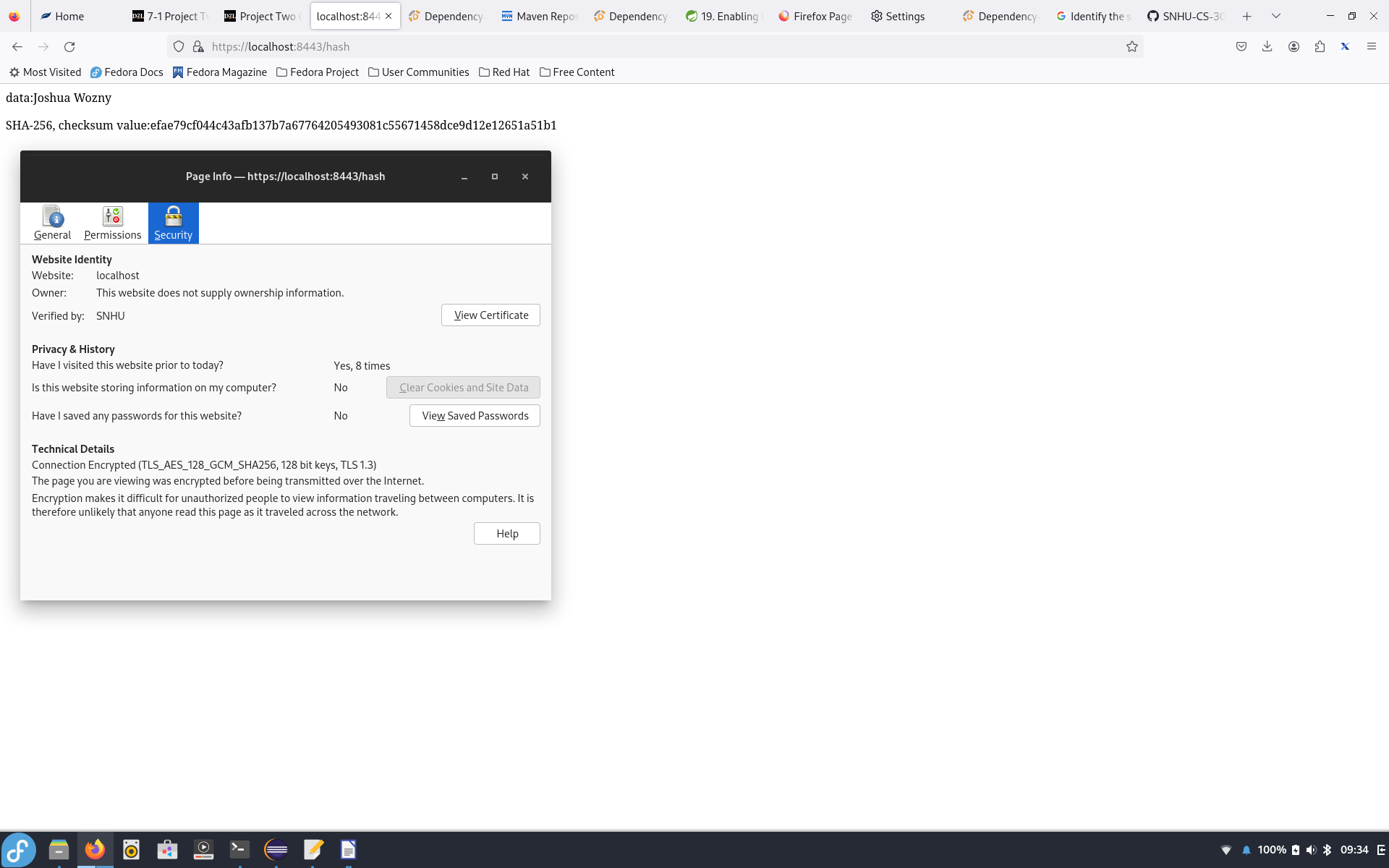
## Deploy Cipher

Insert a screenshot below of the checksum verification.



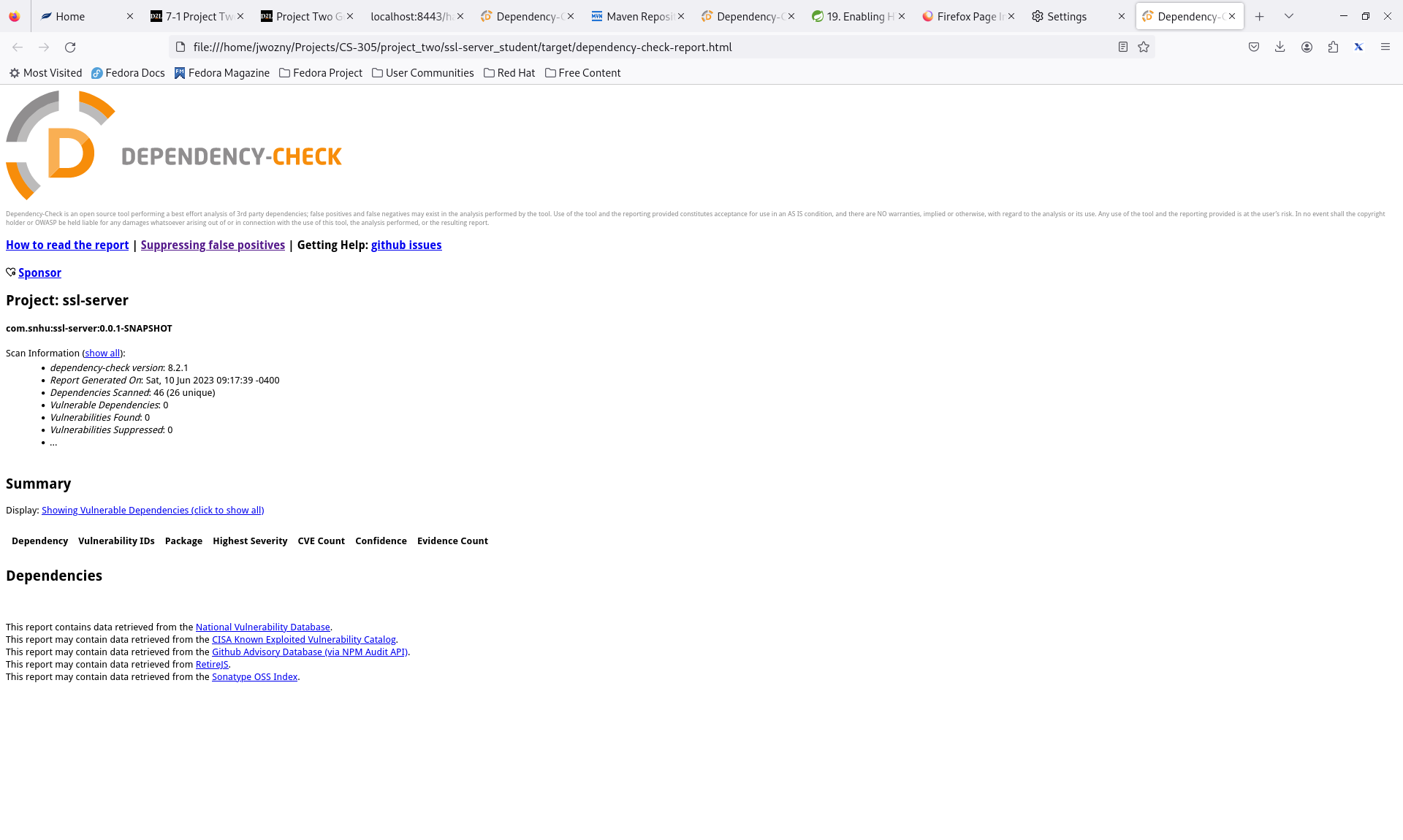
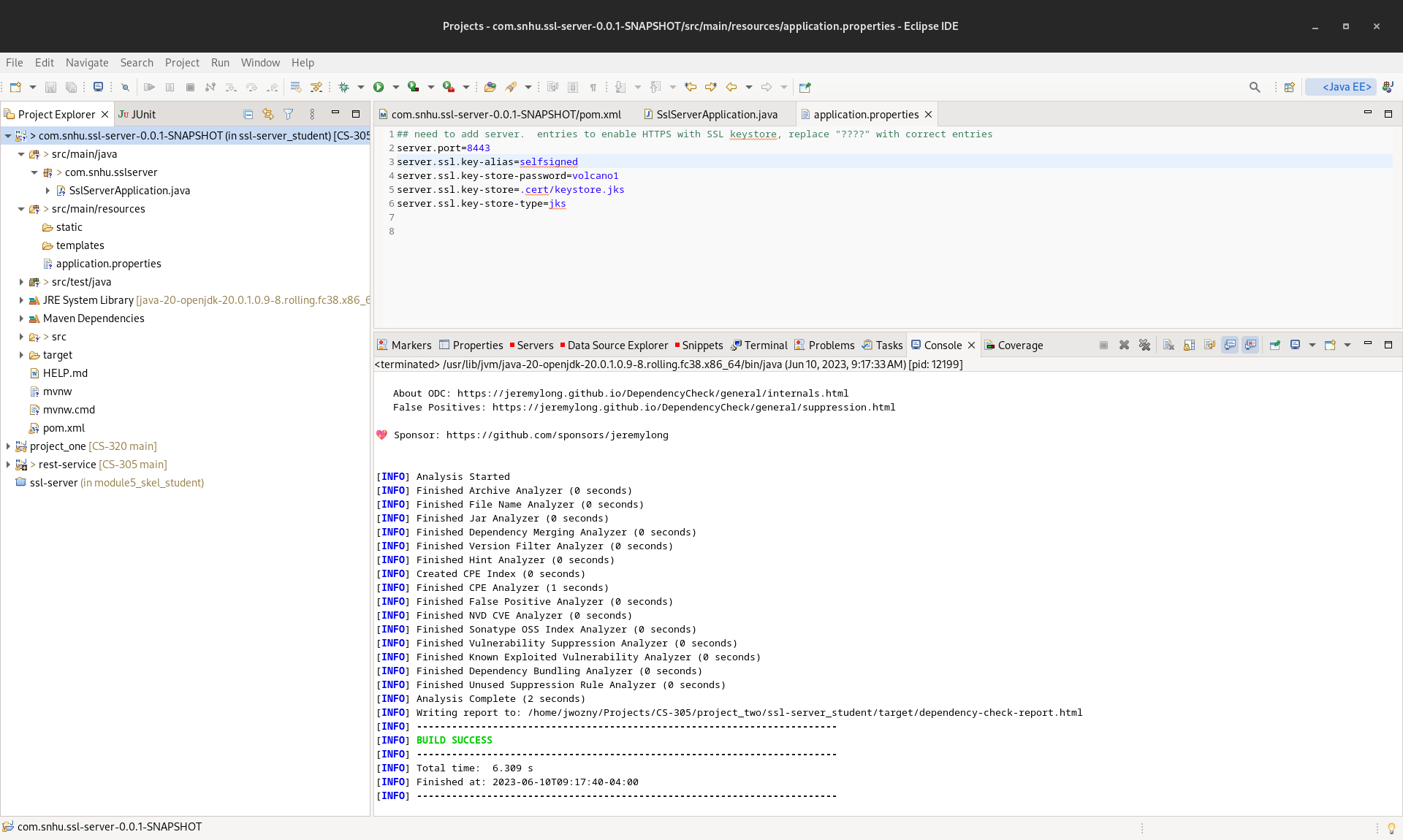
## Secure Communications

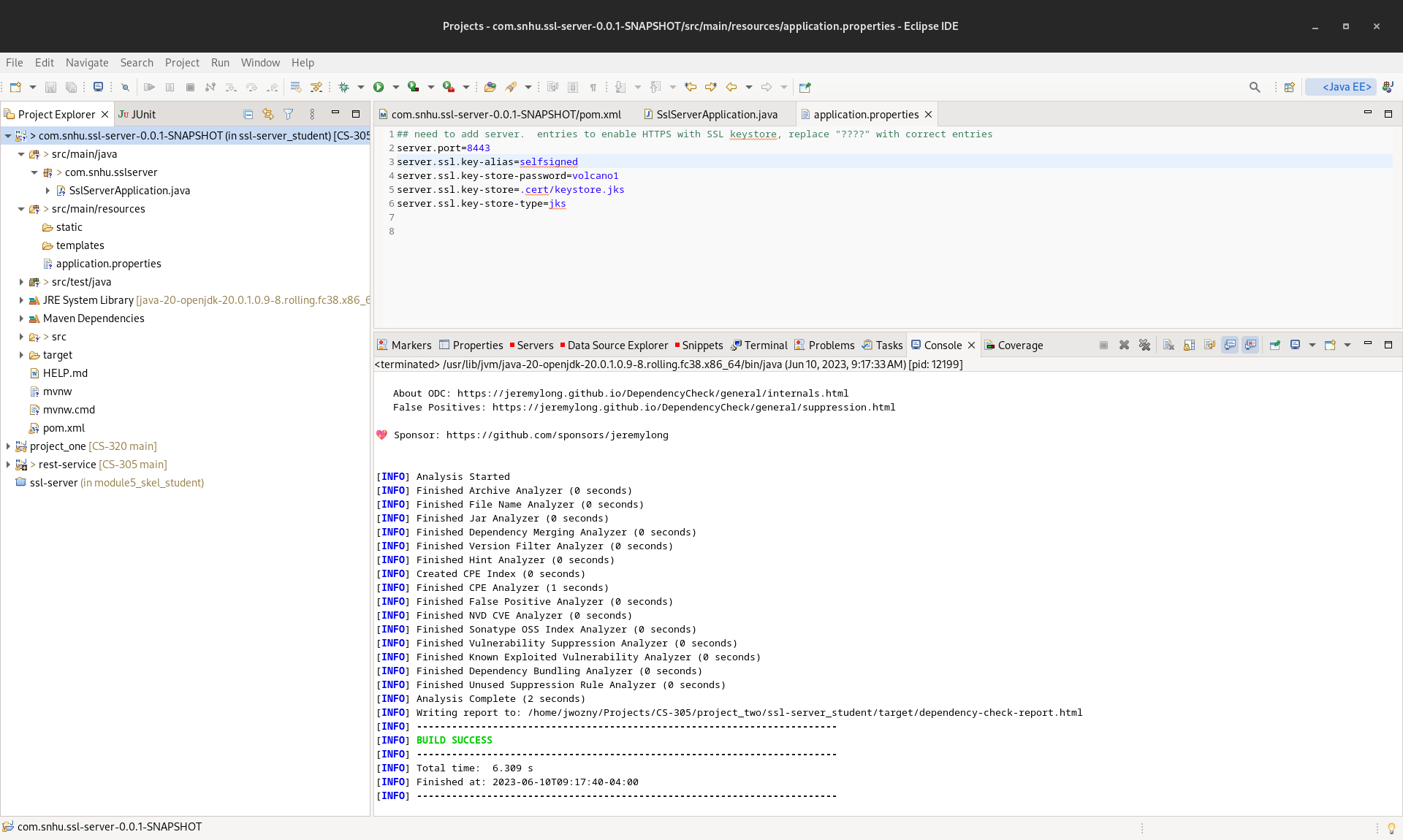
Insert a screenshot below of the web browser that shows a secure webpage.



## Secondary Testing

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

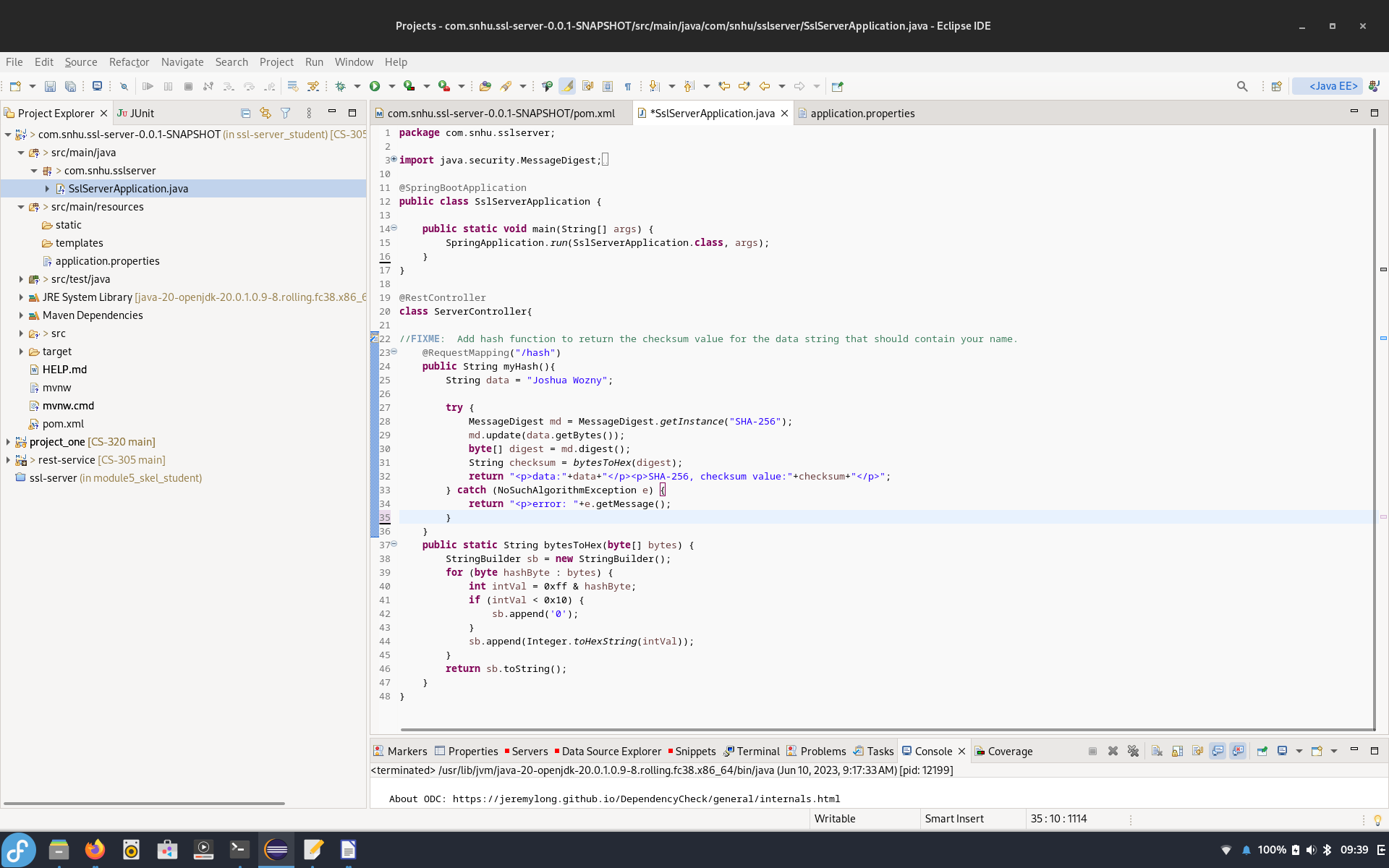




## Functional Testing

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

Added a secure rest controller and enabled https using a self-generated certificate. A manual code review found no significant vulnerabilities.



## Summary

Using the OWASP Dependency-Checker in Maven, several dependencies were identified as having vulnerabilities. All of these vulnerabilities were able to be addressed by upgrading to newer versions. After adjusting the pom file to new versions, the application was run again to ensure that the new versions did not have any breaking changes.

I then updated the application to use secure communication through https using a self-generated certificate. In a production environment we would use a certificate issued by a trusted Certificate Authority (CA). CA issued certificates act as a trusted third party that verifies the identity of the certificate holder. Various methods are used by the CA to ensure that the entity requesting the certificate is legitimate. Users rely on the legitimacy of the CA to rely on the accuracy and reliability of the information contained in the certificate. CAs are organized in a hierarchical structure, where higher-level CAs vouch for the authenticity and reliability of lower-level CAs. This hierarchy allows for a chain of trust, where certificates issued by lower-level CAs are validated and trusted based on the reputation and credibility of the higher-level CAs. This simplifies the task of verification as long as the trust authority remains unbroken (Poettering, B., & Stebila, D., 2017). CAs play a vital role in the following:

* Identity Verification: CAs play an important role in verifying the identity of individuals, organizations, or devices requesting digital certificates. Through domain validation or organizational validation, certificate holder's identities can be verified and trusted.
* Establishing Trust: CAs operate within a hierarchical structure, known as the "certificate chain of trust." This chain of trust allows users to validate the authenticity of a certificate by tracing it back to a trusted root CA, establishing trust in the certificate and the associated entity.
* Encryption and Secure Communications: Digital certificates issued by CAs facilitate secure communication by employing encryption techniques, such as TSL
* Code Signing and Software Integrity: CAs also issue code signing certificates, which are used by software developers to digitally sign their applications or code. This ensures the integrity and authenticity of the software, assuring users that it has not been tampered with or modified by unauthorized parties.

We also created a secure checksum by using the SHA-256 hash function. This will be used by the application to ensure that files generated are not altered while in transit. The dependency checker was used a final time to verify that no new dependencies were added with the refactored code. No new vulnerabilities were discovered. After implementing the changes, the program runs as expected.

## Industry Standard Best Practices

Consider the following best practices:

* Stay informed about the most current government regulations and requirements regarding encryption algorithms, key lengths, and compliance standards. Be aware of regulations in any country where Artemis Financial does business. Failure to comply will result in costly fines.
* Consider employing additional security measures, such as access controls, network security, and monitoring mechanisms, to defend against various types of security attacks.
* Implement secure transmission protocols, such as HTTPS or SFTP, to protect encrypted files during transit.
* Implement Input validation to prevent SQL injection, cross-site scripting (XSS), and similar attacks.
* Use secure authentication and password management, implementing strong password policies and using strong hashing algorithms, like that provided by SHA-256. Use multi-factor authentication (MFA) to provide stronger security when appropriate.
* Update software regularly and confirm no new vulnerabilities have been discovered. Vulnerabilities in dependencies are a major source of security breaches and often some of the easiest to prevent.

REFERENCES

Java Security Standard Algorithm Names. (2017). Oracle.com.

[https://docs.oracle.com/javase/9/docs/specs/security/standard-names.html#cipher-algorithm-names](https://docs.oracle.com/javase/9/docs/specs/security/standard-names.html" \l "cipher-algorithm-names)

Poettering, B., & Stebila, D. (2017). Double-authentication-preventing signatures. *International Journal of Information Security*, *16*(1), 1–22. <https://doi-org.ezproxy.snhu.edu/10.1007/s10207-015-0307-8>

Ramesh Fadatare. (2020, February 25). Java SHA-256 Hash With Salt Example. Javaguides.net; Blogger. https://www.javaguides.net/2020/02/java-sha-256-hash-with-salt-example.html