****

**Practices for Secure Software Report**

Table of Contents

[Document Revision History 3](#_Toc164416099)

[Client 3](#_Toc164416100)

[Developer 4](#_Toc164416101)

[1. Algorithm Cipher 4](#_Toc164416102)

[2. Certificate Generation 4](#_Toc164416103)

[3. Deploy Cipher 6](#_Toc164416104)

[4. Secure Communications 6](#_Toc164416105)

[5. Secondary Testing 9](#_Toc164416106)

[6. Functional Testing 9](#_Toc164416107)

[7. Summary 10](#_Toc164416108)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **19 April 2024** | **Jason Kremhelmer** | **Initial Release** |

## Client



## Developer

Jason Kremhelmer

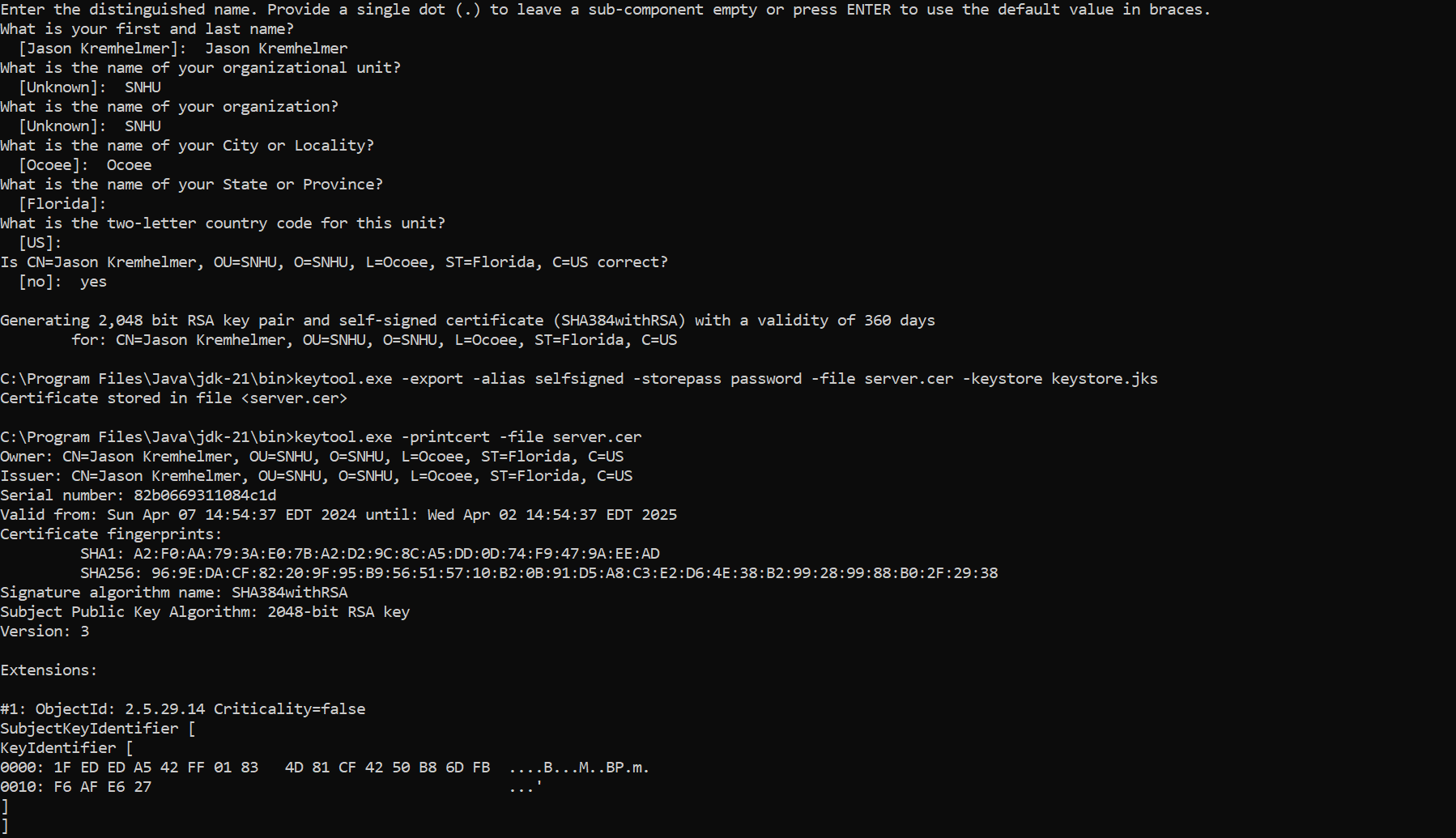
## 1. Algorithm Cipher

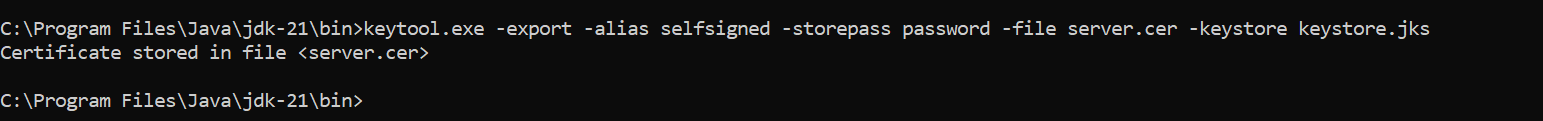
After reviewing the needs of Artemis Financial as well as the system, I have come to the conclusion the ideal encryption cipher would be the Advanced Encryption Standard (AES). In my research, I have found the AES is most commonly used for banking and is a 256-bit encryption. AES-256, is one of the best against hackers as even with quantum computing, the time it would take to brute force the system ifs more than a lifetime. “AES protects data from any third parties” (N-able, 2019). Some of the risks that have been taken into consideration are; making sure the system isn’t leaking information, making sure that access is secured in all forms, firewalls, etc. With security being one of, if not the top, priority choosing the most secure cipher only makes sense. As this is the most common cipher among financial institutions, it would seem that it is the way to go in this industry.

Creating hash functions is done so by taking the input value and converting it to a compressed value. This compressed value is known as the hash, or hash value. The length of the encryption is determined by the Bit levels, for instance, an encryption of 256 indicated there are 256 combinations within the encryption. Using a 256-Bit encryption along with a random number makes hacking into the system all but impossible.AES-256 uses Symmetric keys which are better for sending bulk data. "Because the algorithm behind symmetric encryption is less complex and executes faster, this is the preferred technique when transmitting data in bulk. The plaintext is encrypted using a key, and the same key is used at the receiving end to decrypt the received ciphertext. The host in the communication process would have received the key through external means” (Exploring the Differences Between Symmetric and Asymmetric Encryption, 2019). While doing my research It was interesting to learn how far back encryption actually goes. “**Circa 600 BC**: The ancient Spartans used a scytale device to send secret messages during battle. This device consists of a leather strap wrapped around a wooden rod. The letters on the leather strip are meaningless when unwrapped, and the message makes sense only if the recipient has the correctly sized rod.” (A BRIEF HISTORY OF ENCRYPTION (AND CRYPTOGRAPHY), (updated)2023) With the evolution of encryption, it has allowed us to continue protecting our valuable data.

## 2. Certificate Generation

Cas, or Certificate Authorities are used to verify the chain of trust. The CA is informing the users that if in fact the chain is intact, then you can trust this software item that it is in fact what it says it is. This in turn will allow the decryption of the item in question allowing for user to sign in.

CA allows a central authority of checking as opposed to every client needing to verify all issuers independently. They just need to verify that the chain is unbroken, and it ends with a “pre-approved” issuer.



## 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.

***package*** *com.snhu.sslserver;*

***import*** *org.springframework.boot.SpringApplication;*

***import*** *org.springframework.boot.autoconfigure.SpringBootApplication;*

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

***import*** *org.springframework.web.bind.annotation.RequestParam;*

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

***import*** *java.security.MessageDigest;*

*@SpringBootApplication*

***public******class*** *ServerApplication {*

***public******static******void*** *main(String[] args) {*

*SpringApplication.run(ServerApplication.****class****, args);*

*}*

*}*

*@RestController*

***class*** *ServerController{*

*//****FIXME****: Add hash function to return the checksum value for the data string that should contain your name.*

*@RequestMapping("/hash")*

***public*** *String greeting(@RequestParam(value = "name", defaultValue = "Jason Kremhelmer") String name)* ***throws*** *Exception {*

*//return String.format("Hello %s!", name);*

***return*** *myHash(name);*

*}*

*//@RequestMapping("/hash")*

***public*** *String myHash(String data)* ***throws*** *Exception{*

*//String data = "Jason Kremhelmer";*

*//From https://www.tutorialspoint.com/java\_cryptography/java\_cryptography\_message\_digest.htm*

*//https://learning.oreilly.com/library/view/iron-clad-java/9780071835886/ch06.html#ch06lev2sec7*

*MessageDigest md = MessageDigest.getInstance("SHA-256");*

*md.update(data.getBytes())*

***byte****[] digest = md.digest();*

*//System.out.println(digest);*

*//Converting the byte array in to HexString format*

*StringBuffer hexString =* ***new*** *StringBuffer();*

***for*** *(****int*** *i = 0;i<digest.length;i++) {*

*hexString.append(Integer.toHexString(0xFF & digest[i]));*

*}*

*//System.out.println("Hex format : " + hexString.toString());*

*String returnString = String.format("<p>Hello %s!", data);*

*returnString += "<p>data: This is your First and Last Name";*

*returnString += "<p> SHA-265: CheckSum Value: ";*

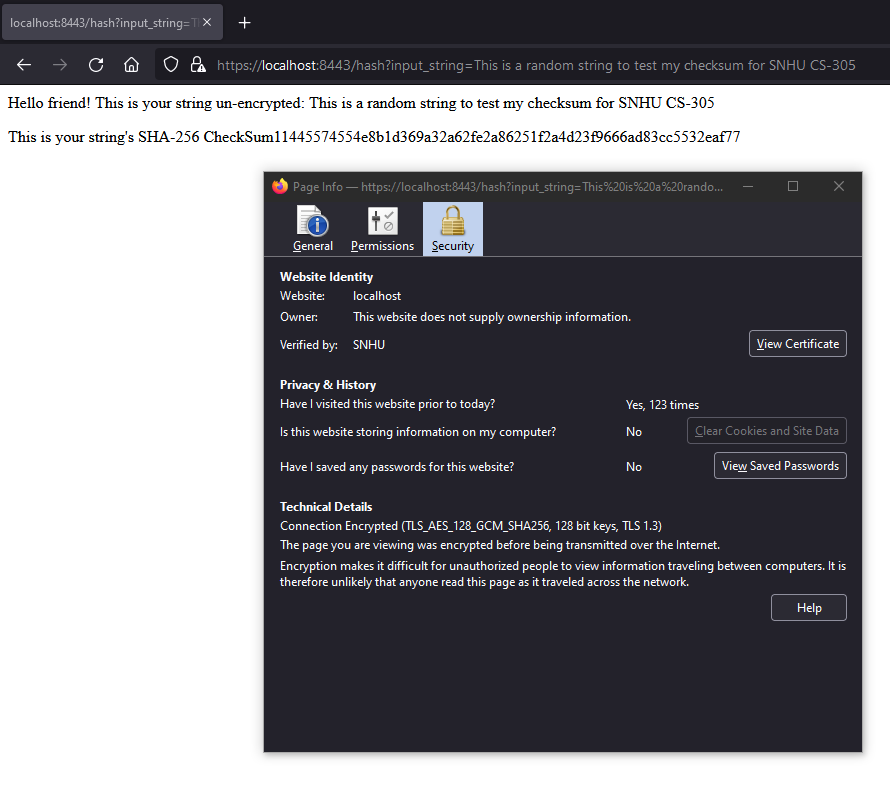
***return*** *returnString + hexString;*

*}*

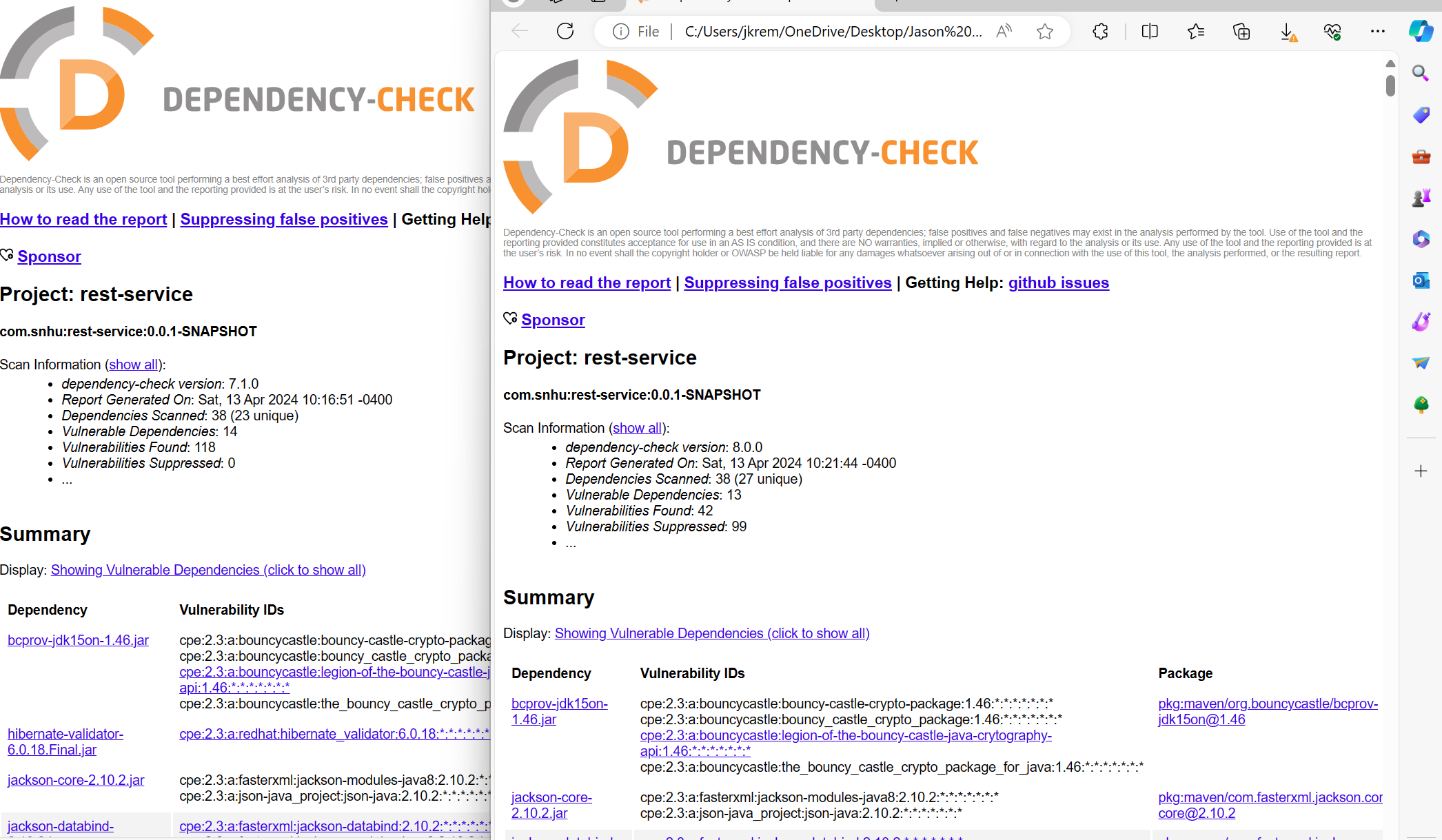
*}*

## 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

API for user input not sanitized. In the greeting method the value from user is passed directly to be hashed. This in turn could be an exploitable weakness. In order to prevent DOS, multiple API calls would be required. In addition, sanitization of the input string is needed in order to prevent buffer overruns.

## 7. Summary

In order to try and prevent websites from having their information broadcast to anyone spying on the network, the code is refactored so web traffic is sent over HTTPs protocols. HTTPs is secured using AES128-bit cipher with a personal certificate used to sign for it.

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.

# References

*A BRIEF HISTORY OF ENCRYPTION (AND CRYPTOGRAPHY)*. ((updated)2023, June 10). Retrieved from Thales: https://www.thalesgroup.com/en/markets/digital-identity-and-security/magazine/brief-history-encryption

*Exploring the Differences Between Symmetric and Asymmetric Encryption*. (2019, November 30). Retrieved from Cyware Social: https://cyware.com/news/exploring-the-differences-between-symmetric-and-asymmetric-encryption-8de86e8a

N-able. (2019, July 29). *Understanding AES 256 Encryption*. Retrieved from N-ABLE: https://www.n-able.com/blog/aes-256-encryption-algorithm