****

# 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** |
| --- | --- | --- | --- |
| **2.0** | **02/19/2023** | **James Fowles** | **Updated Proposal** |

## Client



## Developer

James Fowles

## Algorithm Cipher

The algorithm cipher we recommend is the (AES) Advance encryption standard. It was approved by US law and was specifically developed for the US Government by the National Institute of Standards and Technology and has not been broken or compromised. (Cryptographic Standards and Guidelines, n.d.).

It is used by encrypting and decrypting data in blocks of bits up to 256. It uses secret keys that gives each party the same key for encrypting and decrypting.

A ciphers hash function is simply a data structure for authenticating information and because of its different properties such as its 2256 possible hash combinations it is impervious to collisions. Being collision free means that no two hashes will ever map to the same output hash. And since they cannot be hidden means it is impossible to guess the input value. As of today, there has been no known compromise as it would take today’s computers years to brute force a key, which is basically the act of trying every single combination.

Symmetric and non- symmetric keys have a crucial difference which is symmetric uses just one key for both encrypting and decrypting while non- symmetric uses a public key and a private key. Random numbers are essential to modern encryption but the problem is computers cannot generate a true random number but only a pseudo-random number. And as such all algorithms start with a seed which is the basis for all the random numbers the algorithm will generate. Without this process of generating random numbers no encryption would be secure.

Encryption has made everything from phone calls to the internet possible. Primitive forms of encryption have been found as far back as 625 BC as simple forms of encryption were being used by the roman empire. However, encryption really came into play during the world wars when it was used to transmit secret communications. And exploded with the invention of the internet as it is practically used in every application and it’s what makes it possible for banks and companies like Artemis Financial to deploy their application.

## Certificate Generation

Here is a valid certificate Generated that will last until 1/31/2024.

Text

Description automatically generated

## Deploy Cipher

Screenshot of the SHA-256 Check sum being generated.

A picture containing diagram

Description automatically generated

## Secure Communications

Graphical user interface, text

Description automatically generated

The certificate is valid and accepted but since its local and I used “self-assign” chrome does not recognize it as one of its approved CA’s since it’s not in their personal database. However, it is still secure and per the certificate it shows it is valid and the SHA fingerprints used. Per Second screenshot below.

Graphical user interface, text, application

Description automatically generated

## Secondary Testing

Below is six screen shots that include the refactored code and the proof that it runs without error.

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated

Pom.xml

Text

Description automatically generated

Suppression.xml

Text

Description automatically generated

Output before refactoring

Graphical user interface, text, application, email

Description automatically generated

Output after refactoring (this shows the fixes and suppressions applied)

Graphical user interface, text, application, chat or text message

Description automatically generated

## Functional Testing

Here is the output of the application running without errors after refactoring code. Showing the new code and the console output after running the new code proving it runs without errors.

Text

Description automatically generated

Text

Description automatically generated

Shows application runs without error.

A picture containing graphical user interface

Description automatically generated

Summary

To summarize the refactored code we added a GetHash() function that used the MessageDigest library to take our input which is a string and do encrypting so that the DisplayHash() method can output to a server the SHA-256 CheckSum code. These actions satisfies the cryptography element, and by making this application based on a secure RESTful standard we have addressed the API needs. By creating a certificate of authentication and upgrading the application to use HTTPS instead of HTTP we have addressed the Client/Server standard. Finally, by updating the spring boot version and suppressing some vulnerabilities that are not applicable and then implementing a try-catch around fragile functions we have reduced the chance of crashing and as such also addressed the Code quality section of the vulnerability and assessment process flow. The process I used for adding layers of security to this application was to first review what issues the dependency checker found then try to locate a solution such as updating a version in the pom.xml. Then after each update I would re-run the application to make sure this issue was now fixed and that I did not inadvertently create new issues. I would continue this process until I was satisfied all issues were resolved. Additionally, I would review the code for syntax or compile/runtime errors and fix as needed.

## Industry Standard Best Practices

By iterating over the code checking for vulnerabilities and suppressing the ones that we could not directly address we have created a best practice for future developers by making the dependency check list smaller and more manageable. Additionally, we have added certification to the application which is required for financial applications and increases trust with users. There is a lot of value created by applying these industry standards as it makes it much easier to maintain in the future. All the updates we have provided for this application follow a standard that is taught and will be familiar to new developers who take over the project, as keeping to these standards helps cut down on mistakes and common pitfalls.

References

*Cryptographic Standards and Guidelines*. (n.d.). csrc.nist.gov. Retrieved January 29, 2023, from <https://csrc.nist.gov/projects/cryptographic-standards-and-guidelines/archived-crypto-projects/aes-development>

Gramm, Leach, & Bliley. (1999, November 12). *S.900 - Gramm-Leach-Bliley Act*. congress.gov. https://www.congress.gov/bill/106th-congress/senate-bill/900

*Understanding AES 256 Encryption*. (2019, July 29). N-able. Retrieved January 29, 2023, from <https://www.n-able.com/blog/aes-256-encryption-algorithm#:~:text=The%20National%20Institute%20of%20Standards,the%20strongest%20level%20of%20encryption>

*The history of encryption: the roots of modern-day cyber-securit*. (2022, January 14). Tresorit. <https://tresorit.com/blog/the-history-of-encryption-the-roots-of-modern-day-cyber-security/>

*Java Security Standard Algorithm Names*. (n.d.). docs.oracle. https://docs.oracle.com/javase/9/docs/specs/security/standard-names.html#cipher-algorithm-names