****

# Secure Software Report

Ryan Rieth

Prof. Conlan — CS 305 — SNHU

Table of Contents

[Secure Software Report 1](#_Toc121309310)

[Document Revision History 3](#_Toc121309311)

[Client 3](#_Toc121309312)

[Developer 3](#_Toc121309313)

[Algorithm Cipher 4](#_Toc121309314)

[Certificate Generation 6](#_Toc121309315)

[Deploy Cipher 7](#_Toc121309316)

[Secure Communications 7](#_Toc121309317)

[Secondary Testing 7](#_Toc121309318)

[Functional Testing 8](#_Toc121309319)

[Summary 10](#_Toc121309320)

[Industry Standard Best Practices 12](#_Toc121309321)

[References 13](#_Toc121309322)

## Document Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author | Comments |
| 0.1 | 12/05/2022 | Ryan Rieth | Document inception. |
| 0.2 | 12/07/2022 | *Algorithm Cipher* |
| 0.3 | 12/07/2022 | *Certificate Generation* |
| 0.4 | 12/07/2022 | *Deploy Cipher* |
| 0.5 | 12/07/2022 | *Secure Communications* |
| 0.6 | 12/08/2022 | *Secondary Testing* |
| 0.7 | 12/08/2022 | *Functional Testing* |
| 0.8 | 12/09/2022 | *Summary* |
| 0.9 | 12/09/2022 | *Industry Best Practices* |
| 1.0 | 12/11/2022 | Document finalization. |

## Client



## Developer

Ryan Rieth

## Algorithm Cipher

The cipher recommended for the needs of Artemis Financial — the *Client* — is SHA-256. SHA-256 is highly compatible as it is one of the most widely used algorithm ciphers that exist today; incredibly difficult to crack, even with an inclination to being prone against length extension attacks; and very efficient as it balances security with resource friendliness (Wikimedia Foundation, 2022). SHA-512, perhaps recommended in past documents, is technically stronger and even more difficult to crack with hash value output sizes twice the length of SHA-256, but this takes a heavier toll on computer resources and so is not the currently recommended cipher.

SHA-256 and SHA-512 belong to the SHA-2 family of Secure Hash Algorithms (SHA). This family of cryptographic functions possesses infinitesimal chances of collisions, particularly in SHA-256 (Arias, D. 2019) — another reason to recommend. With 256-bit hash value output sizes, SHA-256 is robust and secure while also efficient and quick. The SHA-2 family is designed by the United States National Security Agency and is the go-to manner of encryption for many entities, including the U.S. Government (Wikimedia Foundation, 2022).

Traces of data encryption go all the way back to the times of Sparta, circa 600 BC (Thales Group, 2022). The Spartans used a “scytale” device to send secret messages between commanders in battle. Julius Caesar around 60 BC invented the first substitution cipher that shifted characters three units over — “A” becomes “D”, for example. In 1553, Giovan Battista Bellaso envisions the first cipher to require a key to decrypt. In 1854, Charles Wheatstone invents the Playfair Cipher that encrypts pairs of letters instead of singular ones. The German Enigma device was invented by Arthur Scherbius in 1918 but was cracked by Polish cryptographer Marian Rejewski in 1932. Modern cryptography, arguably, begins in 1945 with Claude E. Shannon publishing a mathematical theory of cryptography.

Cryptography nowadays is far more advanced than any system previously derived. Encryption algorithms in modern times work to balance resource efficiency with protected security — a “perfect” algorithm is incredibly light and quick to use in a machine and unbreakable from the outside. One of the closest encryption algorithms to this perfect idea is this project’s recommended SHA-256 because it is far from resource intensive and is still difficult to crack based on its 256-bit sized output values and its manner of asymmetric encryption. Asymmetric encryption can be thought of as a “one-way road”; SHA-256 hashes the data, and this hashed data cannot be decrypted, only matched. The standard symmetric encryption manner — the Advanced Encryption Standard (AES) — was introduced by the U.S. Government in 2001 and has been the most popular symmetric encryption method (Bernstein, C., & Cobb, M., 2021). It uses various output bit sizes ranging from 128 to 256. Both the sender and the receiver must know the key, which is used for both encryption and decryption (hence, “symmetric”). If the key is to be discovered, the data is no longer secure. In an asymmetric scenario, should the hashed data be matched — whether by discovering the original data or matching the hash with another form of data, known as a “collision” — the data becomes unsecured.

## Certificate Generation

Graphical user interface, text, application

Description automatically generated

## Deploy Cipher

## Graphical user interface, text, application Description automatically generatedSecure Communications

## Graphical user interface Description automatically generated

## Secondary Testing

## Graphical user interface, text, application Description automatically generatedGraphical user interface, application Description automatically generated

## Functional Testing

Graphical user interface, text, application

Description automatically generatedGraphical user interface, application

Description automatically generatedGraphical user interface, text, application

Description automatically generated

## Summary

​In this project, and in refactoring its code, multiple areas of security presented in the Vulnerability Assessment Process Flow Diagram were addressed. These include *APIs*, *Cryptography*, *Code Error*, *Code Quality*, and *Encapsulation*. In utilizing HTTPS communication, we implemented secure API interactions between local server and client. Error handling with try-catch statements and thrown exceptions is present in multiple places where necessary. Secure coding practices include simple Java scripting, enough whitespace to clearly portray the code, and only using trusted libraries. The final product is refactored to utilize Eclipse’s environment variables in order to remove the act of hard-coding the keystore password and port information from *application.properties*, further following Java’s secure coding practices.

Layers of security are possessed by the project. Firstly, utilizing HTTPS instead of HTTP. The browser still mentions an unsecure connection, but that is the use of a self-signed certificate for this project’s testing purposes. HTTPS is far more secure than HTTP, so implementing an HTTPS connection is crucial for foundational security. Furthering our layers of security is, perhaps, an overlooked mindset: reviewing the code as its scripted. Making sure what’s being scripted not only works but adheres to standards and requirements is a form of security beyond “common sense” in coding. The final layer — after much research — of security added was implementing the use of environment variables via Eclipse’s in-home environment system found with Run Configurations in to remove plain text hard-coding of sensitive information. Regardless of if this was a school project or professional product, I wanted to remove hard-coded material as best as I could.

Beyond the layers included for security, further security concepts were implemented in the project. For example, hashing data given to a locally connected server via an encryption algorithm. SHA-256 of the Secure Hash Algorithm 2 family was utilized for hashing “sensitive” data — a given string merely to showcase proper hashing technique. This hash was then communicated with an endpoint of a local HTTPS server that was then displayed back to affirm proper encryption and successful communication.

## Industry Standard Best Practices

Various industry standard best practices were implemented in this project. One of them is naming conventions. For example, it is understood that environment variables are referred to in all caps and with underscores, which our environment variables such as KEYSTORE\_PASS and ALIAS were labeled as. Another best practice is to not hard-code sensitive information into a product, which birthed the idea of utilizing environment variables within Eclipse. With these variables, the sensitive information of the keystore password and port number were able to be soft-coded and further security measures.

The value of following best practices for coding is almost immeasurable. Beyond being more professional and proper, it is safer and more secure. For example, following best coding practices can mitigate the inclusion of sensitive information hard coded in the project files. It can also render necessary steps to protect said sensitive information if it is being communicated across servers, or at all, as industry standard encryption methods are vastly safer than experimental or untrusted ones. This applies to a general concept but also a specific company one — this sensitive information may be company-pertinent, and following best practices results in best-practiced security measures.

Modularity also results from following best practices. The idea of updating software or applying it elsewhere in similar concepts only works out if the updated version or adjusted version can be applied to the other entity. If best practices are followed, that means OOP is also best adhered to, and sections of code are, therefore, easier to update or tweak as they are modular and can work separately *and* jointly.

## References

Arias, D. (2019, September 30). *How to Hash Passwords: One-Way Road to Enhanced Security*. Auth0. Retrieved December 7, 2022, from <https://auth0.com/blog/hashing-passwords-one-way-road-to-security/>.

Bernstein, C., & Cobb, M. (2021, September 24). *What is the Advanced Encryption Standard (AES)?* TechTarget. Retrieved December 8, 2022, from <https://www.techtarget.com/searchsecurity/definition/Advanced-Encryption-Standard>.

Thales Group. (2022, November 21). *A Brief History of Encryption (and Cryptography)*. Thales Group. Retrieved December 8, 2022, from <https://www.thalesgroup.com/en/markets/digital-identity-and-security/magazine/brief-history-encryption>.

Wikimedia Foundation. (2022, December 5). *SHA-2*. Wikipedia. Retrieved December 7, 2022, from <https://en.wikipedia.org/wiki/SHA-2>.