****

# 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** | **6-23-2024** | **Alex Baires** |  |

## Client



## Developer

Alex Baires

## Algorithm Cipher

As a growing financial institution, Artemis Financial is responsible for safeguarding consumer information and communication. While the industry standard is SHA-256, I recommend future-proofing Artemis Financials’ software using the SHA-3-384 algorithm.

**History and Current State of Encryption Algorithms**

In 2017, SHA-1 was officially cracked through a collision attack by Google researchers, negating the cipher’s security. The NIST had already recommended switching to the newer SHA-2 family, and the attack completed this process. Unfortunately, Grimes (2018) writes that it is only a matter of time before SHA-256 is cracked because several attacks have whittled down SHA-2’s effective protection to 2^37 computational operations (initially, it would take 2^128 operations to find a collision). SHA-2 utilizes an algorithm similar to SHA-1 but with the same cryptographic flaws. Its saving grace is its more considerable hash length (Grimes, 2018). However, an experiment conducted by Huang et al. (2022) shows that a practical collision attack on a 4-round modification of SHA-3-384 using a deduce-and-sieve algorithm is practical and results in a near-collision by only 4 bits. A well-funded adversary with more computing power may be able to find a collision for this particular reduced-round version (Huang et al., 2022). However, this does not lessen the security of the full 24-round SHA-3-384, as the data goes through another 20 rounds of permutation before becoming a hash value.

**Cryptography**

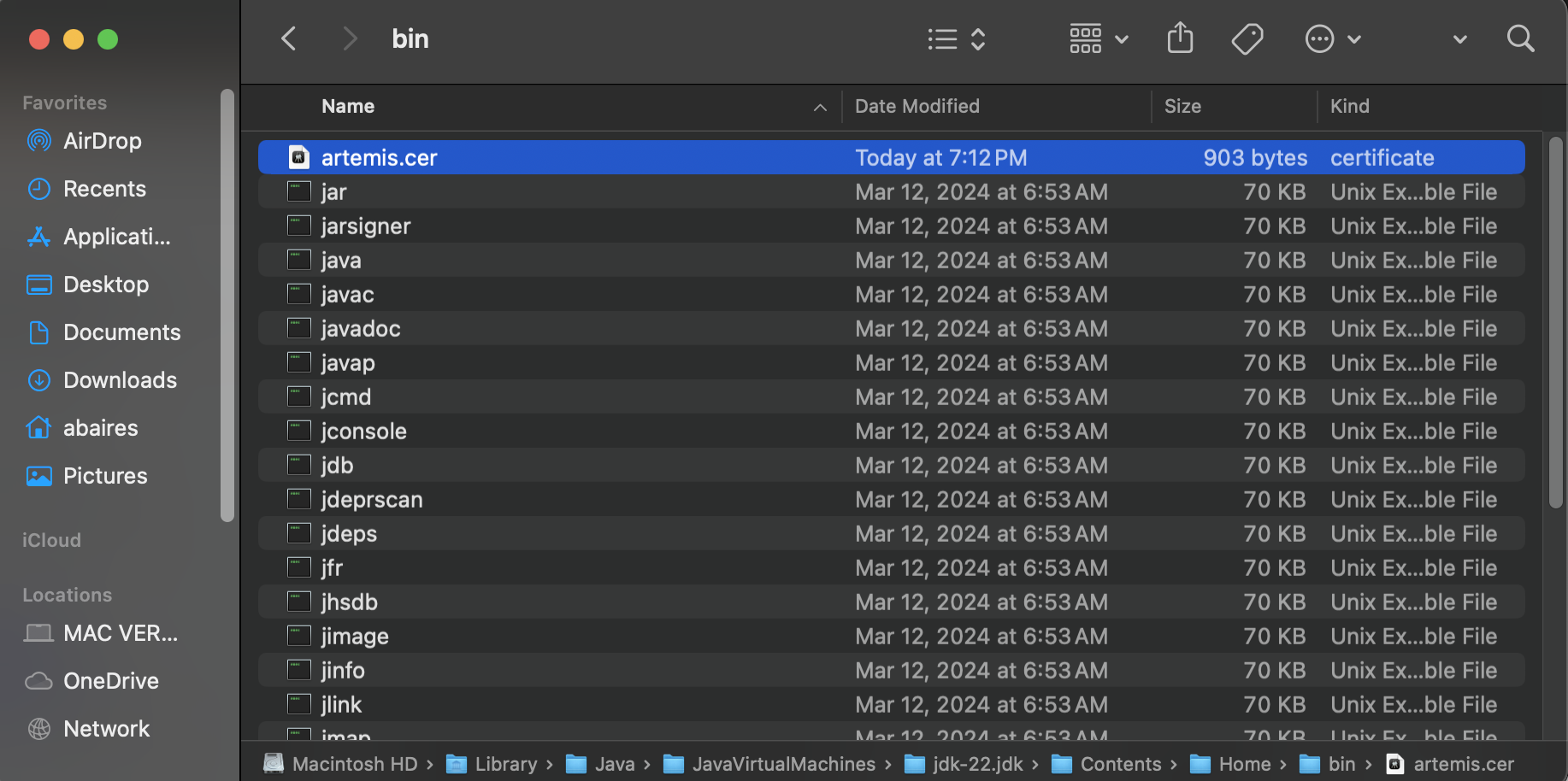
A cipher algorithm converts plain text into illegible cipher text. A key lets one decode the ciphertext into the original plaintext. A symmetric key algorithm utilizes one shared key to encrypt or decrypt the cipher text. An asymmetric key has two keys: a private and a public key. The private key is kept secret, while the public key is shared. Any data encrypted by one key can only be decrypted by the other. The SHA-3-384 algorithm is neither asymmetric nor symmetric. It is considered a one-way hash function that takes a variable-sized input and generates a fixed-length output or digest. The benefit of a one-way hash function is that any modification to the input results in a modified hash ciphertext. If one finds the cipher text, working backward to retrieve the original plaintext data is impossible (*Cryptography Concepts,* 2024).

**SHA-3-384 Overview**

SHA-3-384 is the next-generation secure hash algorithm released by NIST on August 5, 2015. Its hash function is referred to as a “sponge function” and has absorption and squeeze phases. During the absorption phase, input data is divided into 832-bit blocks and checked, and any small remainder blocks are padded to a size that is the multiple of SHA-3's rate. In our case, this would be a multiple of 832. After this, several permutations occur that generate a message digest of 384 bits. Unlike past SHA algorithms, SHA-3-384 is based on the Keccak algorithm and is mathematically and cryptographically different than previous designs (Akiya et al., 2021; Anand, 2019; Grimes, 2018).

## Certificate Generation

Artemis.cer generated:



Artemis.cer printout:

A screenshot of a computer

Description automatically generated

Artemis.cer in macOS Keychain Access (for verification):

A screenshot of a computer

Description automatically generated

## Deploy Cipher

**Initial Cipher Code Implementation:**

bytesToHex()

**A computer screen shot of text

Description automatically generated**

generateCheckSum()

A computer screen shot of a program

Description automatically generated

**Static Data String with Converted Checksum Value in Browser:**

A screenshot of a computer

Description automatically generated

## Secure Communications

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

A screenshot of a computer

Description automatically generated

## Secondary Testing

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

[Insert screenshots here.]

## Functional Testing

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

[Insert screenshots here.]

## Summary

[Insert text.]

## Industry Standard Best Practices

[Insert text.]

**References**

Akiya, Y., Le, K. T., Luong, M., Wilson, J. C., Eddin, A. S., Formicola, V., & El-Hadedy, M. (2021). *SHA-3-LPHP: Hardware Acceleration of SHA-3 for Low-Power High-Performance Systems*. 2021 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), Software Reliability Engineering Workshops (ISSREW), 2021 IEEE International Symposium on, ISSREW, 393–398. <https://doi-org.ezproxy.snhu.edu/10.1109/ISSREW53611.2021.00107>

Anand, A. (2021, December 12). *Breaking down : SHA-3 Algorithm - InfoSec write-ups.* Medium. <https://infosecwriteups.com/breaking-down-sha-3-algorithm-70fe25e125b6>

*Cryptography Concepts*. (2024, May 7). IBM I. Retrieved June 18, 2024, from https://www.ibm.com/docs/en/i/7.5?topic=cryptography-concepts

Grimes, R. (2018, February 21). *Why aren’t we using SHA-3?* CSO Online. <https://www>.csoonline.com/article/564619/why-arent-we-using-sha3.html

Huang, S., Agmon Ben-Yehuda, O., Dunkelman, O., & Maximov, A. (2022, September 9). *Finding collisions against 4-round SHA3-384 in practical time.* https://eprint.iacr.org/2022/194.pdf.

Manico, J., & Detlefsen, A. (2014). *Iron-Clad Java: Building secure web Applications.* Oracle Press. http://dl.acm.org/citation.cfm?id=2826076