****

# 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** |  | **Sean Jette** | **Second Release** |

## Client



## Instructions

Submit this completed practices for secure software report. Replace the bracketed text with the relevant information. You must document your process for writing secure communications and refactoring code that complies with software security testing protocols.

* Respond to the steps outlined below and include your findings.
* Respond using your own words. You may also choose to include images or supporting materials. If you include them, make certain to insert them in all the relevant locations in the document.
* Refer to the Project Two Guidelines and Rubric for more detailed instructions about each section of the template.

## Scenario

You work as a developer for a software engineering company called Global Rain. Global Rain specializes in custom software design and development. The software is for entrepreneurs, businesses, and government agencies around the world. Part of the company’s mission is “Security is everyone’s responsibility.” Global Rain has promoted you to its new agile scrum team.

At Global Rain, you work with a client, Artemis Financial. Artemis Financial is a consulting company that develops individualized financial plans for its customers. The financial plans include savings, retirement, investments, and insurance.

Artemis Financial wants to modernize its operations. As a crucial part of the success of its custom software, the company also wants to use the most current and effective software security. Artemis Financial has a public web interface. The company is seeking Global Rain’s expertise on how to protect its client data and financial information.

Specifically, Artemis Financial wants to add a file verification step to its web application to ensure secure communications. When the web application is used to transfer data, the company will need a data verification step in the form of a checksum. You must take Artemis Financial’s existing software application and add secure communication mechanisms to meet the company’s software security requirements. You’ll deliver a production- quality integrated application that includes secure coding protocols.

**Directions**

You must examine Artemis Financial’s software to address any security vulnerabilities. This examination will require you to refactor the Project Two Code Base linked in the Supporting Materials section to add functionality to meet software security requirements for Artemis Financial’s application. Specifically, you must follow the steps outlined below to facilitate your findings and address and remedy all areas. Use the Project Two Template linked in the What to Submit section to document your work for your practices for secure software report. You will also submit zipped files that contain the refactored code. Review this module’s Resources section to help you with this assignment.

## Developer

Sean Jetté

## Algorithm Cipher

Recommend an appropriate encryption algorithm cipher to deploy, given the security vulnerabilities, and justify your reasoning. Review the scenario and the supporting materials to support your recommendation. In your practices for secure software report, be certain to address the following actions:

1. Provide a brief, high-level overview of the encryption algorithm cipher.
2. Discuss the hash functions and bit levels of the cipher.
3. Explain the use of random numbers, symmetric versus non-symmetric keys, and so on.
4. Describe the history and current state of encryption algorithms.

#### **High-Level Overview**

#### The recommended form of encryption algorithm cipher for Artemis Financial is the Advanced Encryption Standard (AES), which is “a highly trusted encryption algorithm used to secure data by converting it into an unreadable format without the proper key” (GeeksforGeeks, 2024b). Developed by the National Institute of Science and Technology (NIST), the AES algorithm is widely regarded as the gold standard used for securing data due to the nature of its design, widespread usage, and its efficiency concerning securing data at rest (Crawford, 2019). Additionally, this symmetric algorithm cipher operates on fixed block sizes, which consist of 128 bits, and supports a few different key sizes that range from 128, 192, or 256 bits. Its implementation of symmetric encryption standards enables this algorithm cipher to secure data efficiently and is widely used in securing internet communications, protecting sensitive data, and encrypting files; making it a cornerstone of modern cryptography ((GeeksforGeeks, 2024). Moreover, within AES, there are various modes that can be used for block ciphers to encrypt arbitrary amounts of data while also implementing different security and performance characteristics (OWASP, 2024). An option to consider is the implementation of the Galois/Counter Mode (GCM). This could prove useful for its data confidentiality/integrity, high performance, and its widely supported usage of cryptographic libraries (Pake, 2024)—a key factor for securing financial transactions, customer data, and other sensitive information.

#### **Hash Functions and Bit Levels**

AES-128 is an encryption algorithm cipher and not considered a hash function. An example of a hash function would be the Secure Hashing Algorithm or SHA, which is a cryptographic algorithm used to generate a hash from input data (GeeksforGeeks, 2024a). Often, these can be used to verify data integrity but not for encryption. Bit levels for AES in this instance is 128, whereby the 128-bit key is calculated as 2^(128) possible key combinations and as such, it is known to be virtually secure against brute-force attacks; even at today’s computational capabilities (Beal, 2024). Moreover, within the AES standard, there are additional bit sizes at 192 and 256, which offer stronger security for potentially higher-risk applications. In consideration of usage with AES, hashing functions like SHA-256 may be used in conjunction for key derivation, authentication, and additional data integrity requirements (Manico & Detlefsen, 2015).

#### **Symmetric vs. Asymmetric Keys**

AES is a form of symmetric encryption whereby the keys the same key encrypts or scrambles the data and in order to decrypt the information, you use the same key. The key function for this is that the key must remain secret for the integrity of the system (Manico & Detlefsen, 2015). While this might not be as secure of asymmetric encryption standards (Adetunji, 2024), this methodology is more efficient for large datasets. This makes it ideal for scenarios requiring fast and secure processing, such as encrypting bulk financial data or securing communications. Conversely, asymmetric cryptography, which is essential to signing, forms of encryption utilize two keys where one, a public key is known to everyone, and a private key that is only known to the sender (Manico & Detlefsen, 2015). In some instances, the public key can be used for encryption but must be decrypted only by the private key.

#### **Random Numbers**

In relation to the generation of random numbers, this step in the encryption process is critical for the overall algorithm’s security process through the use of key generation, initialization vectors in encryption modes like AES-CBC or AES-GCM, and salts in hashing. Specifically, the essence of random numbers helps to generate two factors: unpredictability and entropy, which is the representation of uncertainty or disorder in the system (Parker, 2023). As a key aspect in cryptography, key generation is applied to both symmetric and asymmetric algorithms. By ensuring the keys are not predictable, the algorithm helps to ensure confidentiality of the data (Parker, 2023). According to Adcyber (2023), the initialization vectors (IVs) used in encryption specific modes are there to ensure the plaintext does not product identical ciphertexts through a random or predetermined sequence of bytes that is added to the encryption process—increasing the overall security of any ciphered message. Random salts within hashing add additional random values to input data before hashing to prevent hash values that are identical (Parker, 2023). While attackers are learning more sophisticated methodologies to crack passwords through usage of programs like Hashcat, this additional measure in the hashing process helps to protect against precomputed attacks like rainbow tables, which is “a large database of hash value pairs linked to their plaintext counterparts” (Wolford, 2024). An example within Java’s functionality of this random number generation is key tool within SecureRandom class. This key tool generates secure cryptographic random numbers and unlike standard random number generation, SecureRandom is designed to ensure high-quality randomness (Manico & Detlefsen, 2015).

#### **History and Current State**

Algorithm ciphers have a long history dating back to ancient times through the seeding of secret messages. Dating back thousands of years, we can see evidence of encryption usage from 1900 BCE in Egypt, to enciphered writings with Mesopotamia, to the legendary Caesar Cipher where each letter in plaintext is replaced by a set of revolving number of letters either going forward or backward within the Latin alphabet (Schneider, 2024). While there has been countless eras where cryptography has been used, such as in WWI and WWII, where the father of modern computing, Alan Turing, the inventor of the Turing machine, developed a sophisticated machine that was used to crack the German Enigma cryptosystem (Schneider, 2024), our modern usage of data encryption standards stems from the early 1970s where the formerly known National Bureau of Standards (NBS), now known as NIST, initiated a program to develop the Data Encryption Standard (DES) to “protect computer data and to allow for large-scale commercial interoperability” (Chen & Scholl, 2022). As such, this newly developed encryption standard, DES, was the first public encryption for use created by the U.S. government (Chen & Scholl, 2022).

In the current state of cryptography, we have methodologies in use like the Ron Rivest, Adi Shamir and Leonard Adleman (RSA) public key cryptosystems—based on the one of the oldest methods to encryption that uses prime number generation for its keys and thus, this asymmetric methodology is prohibitively difficult to factor without the known private and public keys; making it a must-have in our modern cryptographic landscape (Schneider, 2024). Within the world of AES, this type of symmetric algorithm cipher, replaced older algorithms like DES and 3DES due to vulnerabilities and shorter key lengths. It is now the gold standard for encryption, endorsed by the U.S. National Institute of Standards and Technology (NIST) *(Crawford, 2019)*. Additional factors within the encryption standards that are highly necessary are hash functions, such as the Secure Hash Algorithm (SHA), ensuring a high degree of data integrity for digital signatures, blockchain, and password hashing. Moreover, modern encryption supports technologies like TLS/SSL for secure internet communications and is essential to Java web applications (Manico & Detlefsen, 2015).

While these forms of encryption prove ultimately useful, our modern computing landscape is on the brink of the adaptation toward quantum computing breakthroughs that might prove useful for nation states or advanced persistent threat (ATP) actors, such as any adversarial state-sponsored group with access to significant resources. As such, these quantum computing technologies, or post-quantum cryptography, can also prove to be useful where it can have “the potential to be far more secure than previous types of cryptographic algorithms, and, theoretically, even unhackable” (Schneider, 2024).

## Certificate Generation

## Generate appropriate self-signed certificates using the Java Keytool in Eclipse.

## Complete the following steps to demonstrate that the certificate was correctly generated:

## Export your certificates as a CER file.

## Submit a screenshot of the CER file in your practices for secure software report.

Insert a screenshot below of the CER file.

[Insert screenshots here.]

## Deploy Cipher

## Deploy and implement the cryptographic hash algorithm by refactoring code. Demonstrate functionality with a checksum verification.

## Submit a screenshot of the checksum verification in your practices for secure software report. The screenshot must show your name and a unique data string that has been created.

Insert a screenshot below of the checksum verification.

[Insert screenshots here.]

## Secure Communications

## Verify secure communication. In the application properties file, refactor the code to convert HTTP to the HTTPS protocol. Compile and run the refactored code. Once the server is running, type “https://localhost:8443/hash” in a new browser to demonstrate that the secure communication works.

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

[Insert screenshots here.]

## Secondary Testing

## Run a secondary static testing of the refactored code using the dependency-check tool to make certain the code complies with software security enhancements. You need to focus only on the code you have added as part of the refactoring. Complete the dependency check and review the output to make certain you did not introduce additional security vulnerabilities. Refer to the resources in the module’s Resources section for help on this action. In your practices for secure software report, include the following items:

## A screenshot of the refactored code executed without errors

## A screenshot of the report of the output from the dependency-check static tester

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

[Insert screenshots here.]

## Functional Testing

## Identify the software application's syntactical, logical, and security vulnerabilities by manually reviewing the code.

## Complete this functional testing and include a screenshot of the refactored code, executed without errors, in your practices for secure software report.

## *What if I receive errors or new vulnerabilities? You will need to iterate on your design and refactored code, address vulnerabilities, and retest until no new vulnerabilities are found.*

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

[Insert screenshots here.]

## Summary

Discuss how the code has been refactored and complies with security testing protocols. In the summary of your practices for secure software report, be certain to address the following items:

1. Refer to the vulnerability assessment process flow diagram in the Supporting Materials section. Highlight the areas of security that you addressed by refactoring the code.
2. Discuss your process for adding layers of security to the software application.

[Insert text.]

## Industry Standard Best Practices

Explain how you applied industry standard best practices for secure coding to mitigate known security vulnerabilities. Be sure to address the following items:

1. Explain how you used industry standard best practices to maintain the software application’s existing security.
2. Explain the value of applying industry standard best practices for secure coding to the company’s overall well-being.

[Insert text.]

References:

Adcyber. (2023, June 25). What is the initialization vector in AES encryption? - Cyber Insight. *Cyber Insight*. https://cyberinsight.co/what-is-the-initialization-vector-in-aes/

Adetunji, D. (2024, September 10). *Symmetric and asymmetric key encryption – explained in plain English*. freeCodeCamp.org. https://www.freecodecamp.org/news/encryption-explained-in-plain-english/

Beal, V. (2024, June 17). *128-Bit Encryption*. Techopedia. https://www.techopedia.com/definition/29708/128-bit-encryption

Chen, L., & Scholl, M. (2022, May 26). *The Cornerstone of Cybersecurity – Cryptographic Standards and a 50-Year evolution*. NIST. https://www.nist.gov/blogs/cybersecurity-insights/cornerstone-cybersecurity-cryptographic-standards-and-50-year-evolution

Crawford, D. (2019, February 4). *How does AES encryption work?* ProPrivacy.com. https://proprivacy.com/guides/aes-encryption

GeeksforGeeks. (2024a, June 14). *How Does a Secure Hash Algorithm work in Cryptography?* GeeksforGeeks. https://www.geeksforgeeks.org/how-does-a-secure-hash-algorithm-work-in-cryptography/

GeeksforGeeks. (2024b, July 16). *Advanced Encryption Standard (AES)*. GeeksforGeeks. https://www.geeksforgeeks.org/advanced-encryption-standard-aes/

Manico, J., & Detlefsen, A. (2015). *Iron-Clad Java: Building secure web Applications (Oracle Press)* (Version 1.0) [Webpage]. McGraw-Hill Education. https://learning.oreilly.com/library/view/iron-clad-java/9780071835886/?sso\_link=yes&sso\_link\_from=SNHU

OWASP. (2024). *Cryptographic Storage - OWASP Cheat Sheet Series*. https://cheatsheetseries.owasp.org/cheatsheets/Cryptographic\_Storage\_Cheat\_Sheet.html

Pake, J. (2024). *Galois Counter Mode (GCM) - Complex Security*. Complex Security. https://knowledge.complexsecurity.io/cryptography/gcm/

Parker, J. W. S. (2023, December 30). *What is the importance of random number generation in cryptography?* James Parker. https://www.jamesparker.dev/what-is-the-importance-of-random-number-generation-in-cryptography/

Schneider, J. (2024, August 30). A brief history of cryptography: Sending secret messages throughout time. *IBM Think*. https://www.ibm.com/think/topics/cryptography-history

Wolford, B. (2024, May 31). *What is a rainbow table attack and how to prevent it?* Proton. https://proton.me/blog/what-is-rainbow-table-attack