****

# 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** | **10/15/2025** | **Phill Nunez** |  |

## 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.

## Developer

Phill Nunez

## Algorithm Cipher

AES-256-GCM-SHA-256

One of the most reliable and efficient encryption ciphers for modern applications is AES-GCM (Advanced Encryption Standard in Galois/Counter Mode). AES-GCM is a symmetric encryption algorithm that provides both confidentiality and integrity, making it a preferred choice for securing sensitive data at rest and in transit. AES specifies a FIPS-approved (Federal Information Processing Standards) cryptographic algorithm that can be used to protect electronic data. The AES algorithm is a symmetric block cipher that can encrypt and decrypt digital information. The AES algorithm can use cryptographic keys of 128, 192, and 256 bits to encrypt and decrypt data in blocks of 128 bits (National Institute of Standards and Technology, 2001).

Secure applications should rely on modern cryptographic hash functions from the SHA-2 or SHA-3 family for message integrity and digital signatures. SHA-256, part of the SHA-2 family, produces a 256-bit hash output and remains the industry standard due to its resistance to cryptanalysis. A hash algorithm is used to map a message of arbitrary length to a fixed-length message digest. Approved hash algorithms for generating a condensed representation of a message are specified in two FIPS: FIPS 180-4 and FIPS 202. FIPS 180-4 specifies seven hash algorithms, including SHA-256, and FIPS 202 specifies the new SHA-3 family of permutation-based functions based on KECCAK (Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology, U.S. Department of Commerce, n.d.). Hash functions are fundamental to ensuring the authenticity of data, verifying digital signatures, and generating checksums.

Cryptographic security depends heavily on high-quality randomness. For cryptographic operations, developers can use SecureRandom, which implements a CSPRNG (Cryptographically Secure Pseudo-Random Number Generator) designed to produce unpredictable values suitable for keys, initialization vectors (IVs), and nonces. In AES-GCM, each encryption operation must use a unique 12-byte IV (nonce) combined with the encryption key. It is recommended to generate IVs with SecureRandom to avoid reusing the same key, which would compromise encryption integrity. This can be done with a Key Management Service or Java Keystore.

AES is a symmetric cipher, which means the same key is used for both encryption and decryption. A symmetric algorithm is faster and better suited for large data volumes, however, they do require a secure key exchange. Because of this, asymmetric encryption algorithms such as RSA or ECC are used to exchange symmetric keys securely and to sign certificates. To meet modern requirements, it is recommended to use RSA keys of at least 3072 bits or ECC keys of 256 bits or higher.

The development of modern cryptography has been influenced by the need for stronger and more efficient algorithms. Early standards such as DES and 3DES have become obsolete and are not recommended to be used due to the growth of computational power. AES was developed through an open competition and replaced DES as the global standard due to its superior performance and cryptographic strength. AES remains secure today, but developers must use it correctly, by implementing modes like GCM, and managing keys properly. While using these methods, it is still important to stay informed about emerging technologies such as post-quantum cryptography, which aims to protect data against future quantum computing threats. Currently, AES-GCM paired with SHA-256 represents the best balance of performance, compatibility, and security for Java applications.

## Certificate Generation

Insert a screenshot below of the CER file.

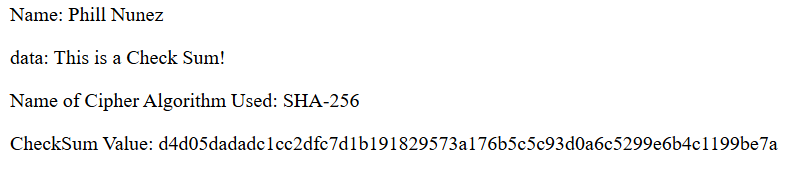
A computer screen with white text

AI-generated content may be incorrect.



## Deploy Cipher

Insert a screenshot below of the checksum verification.



Since I discussed encryption with AES-GCM and decryption, I have included examples of that.

A black text on a white background

AI-generated content may be incorrect.

A close up of a screen

AI-generated content may be incorrect.

## Secure Communications

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



A screenshot of a computer

AI-generated content may be incorrect.

## Secondary Testing

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

A computer screen shot of text

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

## Functional Testing

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

A computer screen shot of a program

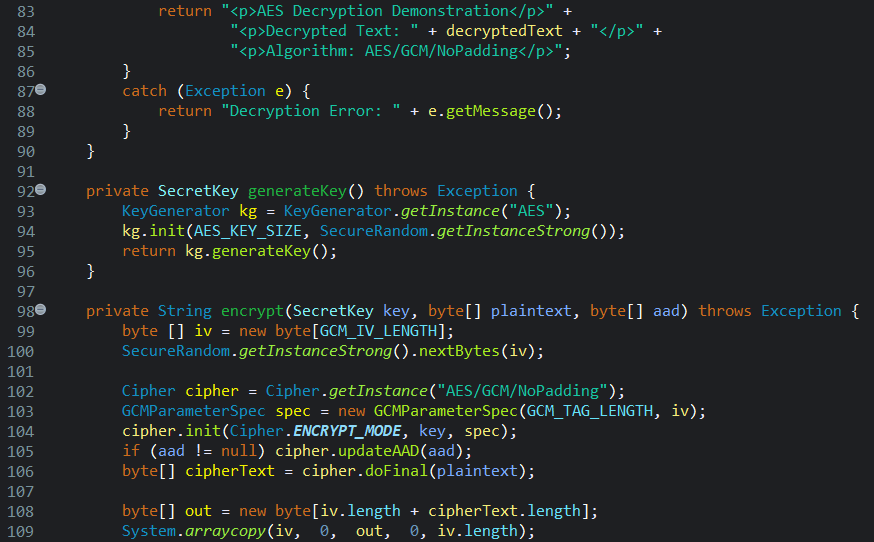
AI-generated content may be incorrect.

A computer screen shot of a program code

AI-generated content may be incorrect.

A computer screen shot of text

AI-generated content may be incorrect.



A screen shot of a computer code

AI-generated content may be incorrect.

A screen shot of a computer program

AI-generated content may be incorrect.

## Summary

Using the Vulnerability Assessment Process Flow, this project went through a structured review to identify and mitigate possible security weaknesses in cryptographic implementation, input validation, and API exposure. The application’s architecture was reviewed to ensure secure encryption was handled with AES-GCM and message hashing using SHA-256. The design ensures that encryption keys are generated securely at runtime and not stored in plaintext. This allows the program to keep confidentiality and integrity policies.

The refactored code focused on secure encryption practices by utilizing AES/GCM/NoPadding, which is a modern, authenticated encryption standard. This avoids using outdated modes for confidentiality and message integrity. Also, SHA-256 hashing was used for checksum verification, ensuring data integrity without storing or transmitting plaintext sensitive data. Secure endpoints (/hash, /encrypt, /decrypt) were created using Spring Boot’s REST controllers. By running HTTPS and implementing proper endpoint separation, the design prevents data from leaking and unauthorized decryption requests. Each controller method includes exception handling and error messages, reducing information exposure. Try/catch block methods prevent stack traces from leaking sensitive implementation details. The cryptographic logic was used in a controller class, AESGCMUtil, which prevents key reuse and separates encryption logic from application routing. Secure data handling ensures random IV generation using SecureRandom.getInstanceStrong(), avoiding predictable patterns.

The application uses HTTPS on port 8443 with a local keystore (keystore.jks) to secure communication channels between client and server. AES-256 with GCM mode ensures data confidentiality and integrity, while SHA-256 provides reliable checksum verification. Static data and parameter validation prevent malicious injection. The application returns structured HTML responses without exposing sensitive stack or key data. Exceptions are caught and returned as generic messages to avoid sensitive error disclosure.

## Industry Standard Best Practices

The refactored code uses secure coding standards defined by OWASP by using things like Strong Encryption Algorithms, which avoid deprecated and weak ciphers, keys are generated per session, which keeps keys from being reused or stored persistently, SHA-256 is used for message verification instead of using outdated and insecure methods. Applying these best practices enhances the company’s security by protecting data integrity and confidentiality, maintaining customer trust, and reducing vulnerability remediation costs. Protecting data integrity and confidentiality provides the ability to reduce the risk of data tampering and unauthorized access. Having secure systems ensures compliance and safeguards client information, which provides customer trust. Early mitigation in development avoids future security breaches and possible incident recovery expenses.

Through this process, the application was refactored to comply with secure coding standards and validated against the vulnerability assessment process. By implementing encryption best practices, HTTPS, error handling, and encapsulation, the final code shows a layered defense model consistent with industry standards for secure software engineering.

Resources

National Institute of Standards and Technology. (2001). FIPS 197. In *Federal Information Processing Standards Publication*. <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197-upd1.pdf>

Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology, U.S. Department of Commerce. (n.d.). *Hash Functions | CSRC | CSRC*. <https://csrc.nist.gov/projects/hash-functions>