****

# Practices for Secure Software Report

Table of Contents

[Practices for Secure Software Report 1](#_Toc128073488)

[Document Revision History 3](#_Toc128073489)

[Client 3](#_Toc128073490)

[Developer 3](#_Toc128073491)

[1. Algorithm Cipher 3](#_Toc128073492)

[History 3](#_Toc128073493)

[2. Certificate Generation 6](#_Toc128073494)

[3. Deploy Cipher 7](#_Toc128073495)

[4. Secure Communications 7](#_Toc128073496)

[5. Secondary Testing 9](#_Toc128073497)

[6. Functional Testing 11](#_Toc128073498)

[7. Summary 12](#_Toc128073499)

[8. Industry Standard Best Practices 12](#_Toc128073500)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **2/22/2023** | **Adam Resemius** |  |

## Client



## Developer

Adam Resemius

## Algorithm Cipher

## History

Encryption algorithms have a long and complex history. The basic concept of encryption involves transforming data into a form that cannot be read by unauthorized parties. This is known to be implemented very early on.   
  
 In fact, One of the earliest forms of encryption was the Caesar cipher, developed by Julius Caesar in ancient Rome. This cipher involved shifting each letter of a message by a certain number of positions in the alphabet, making it unreadable to anyone who did not know the key.

At the end of World War I, a team of codebreakers in England led by developed Arthur Scherbius created a machine called the Enigma, which was used by the German military to encrypt messages. This machine was considered unbreakable until the codebreakers were able to crack it. This lead to further advancements in Cryptography.

In the 1970s, the Data Encryption Standard (DES) was developed by the US government for secure communication. This algorithm used a 56-bit key to encrypt data, and was widely used for several decades. However, as computing power increased, it became easier to crack DES, leading to the development of more advanced encryption methods.

In the 1990s, the Advanced Encryption Standard (AES) was developed as a replacement for DES. This algorithm used a longer key length and more complex encryption methods to provide better security. AES has since become one of the most widely used encryption algorithms.

In today's world of technology, the need to secure sensitive data and information has become problematic. Threats exist more than ever, and while the industry changes, so does the knowledge of hackers. In this section, we will be analyzing the different algorithms that could be used for Artemis Financial, and recommending an appropriate encryption algorithm cipher to deploy, given the security vulnerabilities.

**Overview of the Encryption Algorithm Cipher**

As explained above, an encryption algorithm cipher is a set of mathematical rules that are used to encrypt and decrypt data. The cipher operates by converting the plaintext into ciphertext through the use of an encryption key. The ciphertext can then be decrypted back to the original plaintext using a decryption key. The key is an essential component of the cipher, and the security of the encryption is dependent on the strength of the key.

**Hash Functions and Bit Levels of the Cipher**

Hash functions are used in encryption to generate a unique digital fingerprint of a message or file. This fingerprint is used to verify the integrity of the data, ensuring that it has not been altered in any way. The bit level of the cipher refers to the length of the key used for encryption. A longer key generally provides stronger encryption, but it can also result in slower performance due to the amount of memory being used.

**Use of Random Numbers, Symmetric and Asymmetric Encryption**

Random numbers are used in encryption to generate a random key that is used to encrypt data. Symmetric encryption uses the same key for both encryption and decryption, while asymmetric encryption uses a public key for encryption and a private key for decryption. Asymmetric encryption is more secure than symmetric encryption because the private key is different for both encryption and decryption.

**Blocked Cyphers**

In cryptography, a block cipher is a type of encryption algorithm that operates on fixed-size blocks of data. The block size is a critical parameter that determines the amount of data that the algorithm can process at once. The larger the block size, the more data the algorithm can process in a single operation, but the longer it will take.

In general, a larger block size is preferred for applications that require high security, such as military or government communications, financial transactions, or sensitive data storage. For applications where performance is more critical, such as streaming video or audio, a smaller block size may be ideal.

**What is a Cracked Encryption**

Cracked Encryptions are an algorithm that has been decoded. This means that it has a significant security breach, as unauthorized users are now able to view the data that the algorithm is trying to protect. This makes the encryption useless. When trying to decide our algorithm, we need to ensure that the algorithm is not cracked.

**Encryption Chart:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Algorithm | Key Length | Security | Common Block Size | Cracked |
| RSA | 512 bits – 4096 bits | Moderate to High | N/A | No |
| RC5 | 0 bits – 2040 bits | Moderate | 32bits, 64 bits | No |
| RC4 | 8 bits – 2048 bits | Weak | 8 bits | Yes |
| RC2 | 128 bits | Weak | 64bits | Yes |
| AES | 128 bits, 192 bits, 256 bits | High | 128 bits | No |
| DES | 56 bits | Weak | 64 bits | Yes |
| DESede | 192 bits | Weak | 64 bits | Yes |
| Blowfish | 32 bits – 448 bits | Moderate | 64 bits | No |
| AESwrap | 123 bits, 192 bits, 256 bits | Moderate | N/A | No |
| DESedeWrap | 56 bits | Weak | N/A | Yes |
| ECIES | 128 bits | Strong | N/A | No |

**Recommendation and Justification**

After analyzing the different algorithms available, we recommend AESwrap as the appropriate encryption algorithm cipher to deploy for Artemis Financial. AESwrap is a widely used and secure encryption algorithm that is commonly employed in both government and financial applications. It has a moderate level of security and no known successful attacks.

Encryption algorithms have come a long way from their earliest forms, and are now widely used to protect sensitive information in many industries. The current state of encryption involves a wide variety of algorithms with varying levels of security and complexity, and ongoing research and development is needed to stay ahead of potential attackers.

AESwrap uses authenticated encryption keys to detect tampering with the storage of files, providing an added layer of security for sensitive data. It is also compliant with US government regulations, making it a highly suitable option for organizations that handle sensitive information such as financial institutions or government agencies.

AESwrap uses random numbers within the encryption process. This makes the encryption more secure, as it adds an element of unpredictability to the process. This helps prevent attackers from being able to guess the key or any part of the key.

Another advantage of AESWrap is its efficiency. It allows for multiple keys to be encrypted and decrypted at once, which is particularly beneficial for encrypting long-term archive files, which is what Artemis Financials purposes are for encryption.

AESwrap has yet to be cracked, so we know that it is still secure and is able to be used. The only issue is, that AESwrap has been around for a fair amount of time so hackers may be familiar with this encryption.

A few more things to note is that AES Wrap uses Symmetric keys to encrypt and decrypt data. This can be less secure, as key management and distribution become more important, as they need to be limited between parties who need to communicate with the system.

## Certificate Generation

Insert a screenshot below of the CER file.

**CA Generation:**

For generating this Certificate of Authority I used the keytool.exe command through my terminal in eclipse: “keytool -genkeypair -alias ArtemisFinancialCA -keyalg RSA -keysize 2048 -keystore "C:\Users\adamj\OneDrive\Documents\school\CS 305 Project Two Code Base\ssl-server\_student\keystore.jks" -storepass qwerty -validity 365”, I then filled out the necessary questions regarding the certificate.   
  
Text

Description automatically generated

**CA Verification:**  
To confirm that this certificate was generated correctly, I used “keytool -list -v -alias ArtemisFinancialCA -keystore "C:\Users\adamj\OneDrive\Documents\school\CS 305 Project Two Code Base\ssl-server\_student\keystore.jks" -storepass qwerty” I originally tried to use “keytool -printCert -file “C:\Users\adamj\OneDrive\Documents\school\CS 305 Project Two Code Base\ssl-server\_student\keystore.jks". However, this kept erroring with keytool error: java.lang.Exception: Failed to parse input. Looking further into the matter, this could because I wrote the certificate to the project’s folder instead of creating it then exporting it.   
  
Text

Description automatically generated

## Deploy Cipher

Graphical user interface, text

Description automatically generated

## Secure Communications

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

I could not get a secure webpage in this assignment when trying to export my certificate and importing it as a trusted Certificate Authority. I have tried remaking the certificate but was unable to get this done successfully. However, my encryption seems to be working. This is due to my certificate being self signed.

Text

Description automatically generated

Graphical user interface, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text

Description automatically generated

## Secondary Testing

During this project, I added security measures and a checksum to the current Code base. The refactor code of the SSlServerApplication.java file is shown below:  
Text

Description automatically generated  
  
I have also refactored the application.properties folder, which is shown.

Text

Description automatically generated

I verified that the code was able to run the server correctly as intended after these changes:   
  
Text

Description automatically generated

I then ran a Maven Dependency check to verify that the refactored code does not have any vulnerabilities in the added libraries. As far as what I can see, the current dependencies are not from the libraries that I have added to the codebase, as these are spring boot and maven vulnerabilities.

Graphical user interface, text, email

Description automatically generated

## Functional Testing

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

Text

Description automatically generated

## Summary

I refactored the code by adding a REST controller which handles HTTP requests and responses securely by following the principles of the RESTful architecture. This provides a way for clients to interact with the server using a set of predefined operations. The controller handles a request to hash the provided data string using the SHA-256 algorithm and returns the checksum value to the client.

By using a RESTful approach, the controller helps to reduce the risk of security vulnerabilities by separating the data layer from the presentation layer, and by enforcing a clear separation of concerns between the client and server.

Additionally, I made sure to check for any dependency vulnerabilities that may exist in the project. This is important because dependencies can introduce potential security vulnerabilities into an application. To do this, I used a dependency management tool to scan the project's dependencies for any known security vulnerabilities and addressed any issues that were found. This ensures that the application is using secure and up-to-date dependencies, reducing the risk of potential security vulnerabilities. This should be done often to keep up with checking any newly found vulnerabilities.

## Industry Standard Best Practices

In this report, I have applied industry standard best practices for secure coding to mitigate against known security vulnerabilities. One of the key areas where I have used industry standard best practices is in maintaining the software application's current security. To ensure this, I have followed the recommended practices for secure coding, such as using encryption algorithms to protect sensitive data and information.

One way I have applied these practices is through the use of an encryption algorithm cipher. I have analyzed different algorithms that could be used for Artemis Financial and recommended an appropriate encryption algorithm cipher to deploy based on the security vulnerabilities. I have also implemented the SHA256 hash function, bit levels of the cipher, use of random numbers, symmetric and asymmetric encryption, and encryption ciphers like AESWrap. This helps to ensure that the encryption is strong and secure, and that unauthorized users cannot view the data that the algorithm is trying to protect.

Another best practice I have used is the use of certificate generation. This involves generating a unique digital fingerprint of a message or file, which is used to verify the integrity of the data and ensure that it has not been altered in any way. I have also implemented secure communications to ensure that the data transmitted between different systems are secure and protected from unauthorized access or interception.

The value of applying industry standard best practices for secure coding to the company's overall well-being cannot be overstated. By maintaining current security, the company is protected from cyber threats that could lead to data breaches, financial losses, and reputational damage. Additionally, adhering to these best practices helps to build trust with customers and stakeholders who can be assured that their sensitive information is well-protected. Using industry-standard best practices for secure coding is a necessary step towards achieving a secure software application that is critical for the success of the company.

**References:**

Baeldung, W. by: (2022, November 4). Difference between self-signed CA and self-signed certificate. Baeldung on Computer Science. Retrieved February 19, 2023, from https://www.baeldung.com/cs/self-signed-ca-vs-certificate#:~:text=They%20are%20used%20to%20test,owned%20by%20companies%20and%20organizations

Java Security Standard Algorithm Names. Java security standard algorithm names. (n.d.). Retrieved February 15, 2023, from https://docs.oracle.com/javase/9/docs/specs/security/standard-names.html#cipher-algorithm-names

Kaushik, A., & Srivastava, V. K. (2020). Performance Analysis Of AES And DESede On The Sensitive Data Stored In HDFS. 2020 2nd International Conference on Advances in Computing, Communication Control and Networking (ICACCCN), Computing, Communication Control and Networking (ICACCCN), 2020 2nd International Conference On, 319–322. https://doi-org.ezproxy.snhu.edu/10.1109/ICACCCN51052.2020.9362755

Manico, J., & Detlefsen, A. (2015). Proven Methods for Building Secure Java-Based Web Applications. In Iron-clad java: Building secure web applications. essay, McGraw-Hill Education.

N-able. (2021, April 1). SHA-256 algorithm - N-able. N. Retrieved February 20, 2023, from https://www.n-able.com/blog/sha-256-encryption

Nithya, B., Mathew, J. C., G, K., R, A. N., & Kumari, A. (2022). An Analysis on Cryptographic Algorithms for Handling Network Security Threats. 2022 IEEE Delhi Section Conference (DELCON), Delhi Section Conference (DELCON), 2022 IEEE, 1–9. https://doi-org.ezproxy.snhu.edu/10.1109/DELCON54057.2022.9752828

OpenSSL. /news/vulnerabilities.html. (n.d.). Retrieved February 20, 2023, from https://www.openssl.org/news/vulnerabilities.html

Owasp dependency-check. OWASP Dependency-Check | OWASP Foundation. (n.d.). Retrieved February 21, 2023, from https://owasp.org/www-project-dependency-check/

Summerfield, J. (n.d.). Why you need to secure your website with HTTPS and SSL. HSS Web Technologies. Retrieved February 18, 2023, from https://www.hswsolutions.com/resources/why-you-need-to-secure-your-website/

ZenKhifer. (2022, March). Recommend an Appropriate Encryption Algorithm Cipher to Deploy Given Artemis Financial’s Needs. Brogramo. Retrieved February 17, 2023, from https://brogramo.com/recommend-an-appropriate-encryption-algorithm-cipher-to-deploy-given-artemis-financials-needs/