****

# CS 305 Project Two

**Practices for Secure Software Report**

Table of Contents

[Document Revision History 3](#_Toc33111302)

[Client 3](#_Toc33111303)

[Instructions 3](#_Toc33111304)

[Developer 4](#_Toc33111305)

[1. Algorithm Cipher 4](#_Toc33111306)

[2. Certificate Generation 4](#_Toc33111307)

[3. Deploy Cipher 4](#_Toc33111308)

[4. Secure Communications 4](#_Toc33111309)

[5. Secondary Testing 4](#_Toc33111310)

[6. Functional Testing 5](#_Toc33111311)

[7. Summary 5](#_Toc33111312)

[8. cited sources 5](#_Toc33111312)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **04/15/2021** | **Vitalie Cucuta** | **Updated all information for the initial report.** |

## Client



## Instructions

Deliver this completed Practices for Secure Software Report documenting your process for writing secure communications and refactoring code that complies with software security testing protocols.

Respond to the steps outlined below and replace the bracketed text with your findings in your own words. If you choose to include images or supporting materials, be sure to insert them throughout.

## Developer

Vitalie Cucuta

## 1. Algorithm Cipher

* We are choosing the Advanced Encryption Standard (AES) because it is the world’s most accepted encryption standard, a single key is used for encryption and decryption, and the sizes of the keys are considered adequate to secure the classified data to a satisfactory level (Raghavan, 2020).
* Larger key size means it is much more secure against brute force attacks. AES is also not prone to statistical attacks because it is not possible with common techniques to do statistical analysis of ciphertext in AES (Azad & Pathan, 2014). A statistical attack is an exploitation of statistical weaknesses such as floating-point errors or numbers not being truly randomly generated.
* The truly best part about AES is that the algorithms used in it are so simple, but still secure, that they can be easily implemented using cheap processors and a minimum amount of memory but the it does need more processing power and more rounds of communication than the traditional DES ciphers (Azad & Pathan, 2014).
* AES’ advantages really outweigh its disadvantages though and it would be a great fit for what Artemis Financial needs. We could also go for the AESWrap algorithm cipher which provides another layer of security.
* The AES cipher uses hash functions which is an algorithm that takes an arbitrary amount of data input, like a credential, and produces a fixed size output of enciphered text called a has value (Synopsys Editorial Team, 2015). The purpose of the hash function is to verify authenticity of a piece of data, such as a credential.
* We mentioned before that AES using a single key for encryption and decryption and that the size of the key is adequate for security. The size of the key refers to the bit size of the key. AES can encrypt/decrypt data in 128, 192, and 256-bit key lengths. The higher the bit, the more secure the information.
* Now because the AES uses only a single key for encryption and decryption, this makes the AES a symmetric algorithm. The sender and receiver must both know and use the same key. Earlier we mentioned one of the disadvantages to AES was that there were more rounds of communication in AES than DES.
* What this refers to is those processing steps and there are several processing steps that is involved in taking in the plaintext, or readable text, to transform it into ciphertext. For a 128-bit key it takes 10 rounds, for a 192-bit key it takes 12 rounds, and for a 256-bit key it takes 14 rounds, with each round consisting of processing steps such as substitution, transposition, etc. (Cobb, 2020).
* It may seem like these rounds take a lot of time, but it really occurs quite quickly. Compared to TripleDES, it is still six times faster (Raghavan, 2020). All these things are what makes the AES so perfect for Artemis Financial. It is tried and true and a very much trusted cipher across the board.
* If we were to describe the entire history of encryption algorithms we would actually go all the way back to 600 BC where Spartans used a device called a scytale to send secret messages during battle, but for the sake of time we shall look at modern, computer-based cryptography which started in the early 1970s with IBM forming a ‘crypto group’ that created the DES (Thales Group, 2016).
* The DES was a block cipher to protect customer data and, even though it was created in 1973, it remained in use until it was cracked in 1997. It wasn’t until 2000 that the DES was replaced by AES, which was found through a competition open to the public (Thales Group, 2016).
* The AES has still yet to be cracked and some believe it may be another 20 years before it will be cracked. It is even approved for use in classified US government information (Thales Group, 2016). If it’s good enough for the US government then it is good enough for Artemis Financial.
* In 2005, an advanced public-key cryptography scheme that allows shorter encryption keys, known as Elliptic-Curve Cryptography (ECC), was created and these cryptosystems are much more difficult to break than RSA and Diffie-Hellman (Thales Group, 2016). This method of encryption is seen in banking cards, smartphones, IoT devices and even used to protect bitcoins. Even so, AES is still much more widely used and the preferred algorithm when it comes to encryption.

## 2. Certificate Generation

* ![Text

  Description automatically generated]()See below for selfsigned certificate generation and export through terminal window:

## 3. Deploy Cipher

* See below for successful deployment of cipher with checksum:

Graphical user interface, application

Description automatically generated

## 4. Secure Communications

* See below for successful HTTPS protocol and secure communication (states not secure because certificate is self-signed):

Graphical user interface, application

Description automatically generated

## 5. Secondary Testing

* After running the dependency check tool, no security vulnerabilities occurred.
* The vulnerabilities that did occur had to do with old dependencies. The old dependencies were overridden in the pom file and it removed the vulnerabilities.
* Two vulnerabilities were suppressed as false positives due to no mitigation being available.
* A picture containing text, screenshot, monitor, electronics

  Description automatically generatedGraphical user interface, text, application

  Description automatically generatedThe code will be attached to this report as an attachment, see below for screenshots of successful build and dependency report.

## 6. Functional Testing

* No syntactical, logical and security vulnerabilities for the software application found.
* A picture containing text, screenshot, monitor, electronics

  Description automatically generatedSee screenshot below for successful execution of application:

## 7. Summary

* The code was refactored to incorporate the SHA-256 encryption algorithm.
* We first called for the RestController, builds the REST API, and created the SslSserverController class.
* Within this class we created a hash function with try catch blocks to that any algorithm errors are caught but to also call this function and be able to use different algorithms using the same function, if needed in the future.
* Another function was created to be used to get CheckSum value in hexadecimal value. It changes the format from bytes to hex.
* The use of the preceding functions is an example of how we incorporated encapsulation for our code. Using encapsulation helps to determine where flaws in secure data structures occur and will help to pinpoint where the fix needs to be made.
* Next, we call RequestMapping which maps the web request, and, in this case, it is “/hash”.
* We create another function with a return value and create three string variables for our data, which is what we will encrypt, the algorithm, and format of the output for the data value.
* The data is formatted and printed onto the screen when the application is running. We will then take that data, encrypt it, then print out the CheckSum value, which we used the bytesToHex function to change the output to a hexadecimal value.
* All this is printed out onto the screen and a screenshot of this can be found in sections 3 and 4 above.
* After the dependency check was ran, the code was refactored again, this time in the pom file. We added updated dependency versions and also suppressed false positives so that we get an accurate vulnerability report.
* In our refactored code we can find it meets the standards of our vulnerability assessment.
  + Input validation – using the CheckSum value we determined that the input is secure and encrypted when running it through the cipher.
  + APIs – we are running a RESTful API and it is secured by using HTTPS protocol with the self-signed certificate. When we will be deploying this in real time, the certificate will be validated, and the site will be fully trusted.
  + Cryptography – we are using the SHA-256 encryption algorithm for our system. As stated earlier, it is the recommended algorithm.
  + Client/Server – once again we are using the HTTPS protocol so we will establish a secure connection between the client and the server.
  + Code Error – the refactored code is known to be free of errors and executes as intended so we know that there is no error made.
  + Code Quality – we used secure coding practices by using OOP tactics and creating a secure server.
  + Encapsulation – as previously stated, using encapsulation creates more secure data structures and helps us find where possible errors are but also helps in making it easier to fix errors when you don’t have to fix an entire source code and only a function.
* When dealing with financials of millions, it is important for Artemis Financial to create a feeling of security for the clients.
* It is our job to create these security features that will give your clients that feeling, and we will make sure that any data between a client and your company is secure.
* Any archived data will also be secured and ensured to be free of any threats.
* It is important that security of data is your first and foremost priority because there is always someone that is willing to hack into a client’s account and we must make sure on our end that they are unsuccessful each and every time.
* The success of the company is as successful as its security.
* To maintain the current security of the software application we advise that this be done:
  + Always make sure that all the software and dependencies are constantly updated.
  + Document your security policies.
  + Educate and train users.
  + Monitor user activity.
  + Use security metrics software to constantly measure and improve performance and security.
* You will find the code files attached to this completed report.

## 8. Cited Sources

Azad, S., & Pathan, A. K. (2014). Chapter 6: Advanced Encryption Standard/PROS AND CONS OF AES. In *Practical cryptography: Algorithms and implementations using C++*. Boca Raton: Auerbach

Cobb, M. (2020, April 17). What is aes encryption and how does it work? Retrieved March 27, 2021, from https://searchsecurity.techtarget.com/definition/Advanced-Encryption-Standard#:~:text=AES%2D128%20uses%20a%20128,192%20and%20256%20bits%2C%20respectively.

Raghavan, R. (2020, April 19). Top 8 strongest data encryption algorithms in cryptography. Retrieved March 27, 2021, from https://acodez.in/data-encryption-algorithms/

Synopsys Editorial Team. (2015, December 10). What are cryptographic Hash functions?: Synopsys. Retrieved March 27, 2021, from https://www.synopsys.com/blogs/software-security/cryptographic-hash-functions/#:~:text=A%20cryptographic%20hash%20function%20is,used%20to%20verify%20the%20user.

Thales Group. (2016, April 18). A brief history of encryption. Retrieved March 27, 2021, from https://www.thalesgroup.com/en/markets/digital-identity-and-security/magazine/brief-history-encryption