****

# 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** | **21FEB25** | **Alex Hitchens** |  |

## 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

Alex Hitchens

## Algorithm Cipher

The algorithm cipher that I would recommend is the Advanced Encryption Standard (AES) at 256-bit. My justification for this is based on the following. First, AES is widely used throughout the world and is known for its speed and security making it a trusted solution for people’s security needs. Second, It meets the criteria for the GLBA and FFEIC requiring financial institutions to properly secure their clients’ data and transactions. Technically they only require 128-bit encryption, but 256 is considered safer and does not drastically increase running time and is safer. In AES specifically 128-bit encryption goes through 10 ‘rounds’ of data encryption while 256-bit increases that 14 ‘rounds’. These rounds will take the data and use substitution, mixing, column mixing, and sub bytes to make the data unreadable without the key. Lastly AES is meant to protect data at rest, meaning that even if someone physically stole the server hard drives, they would not be able to unencrypt the files without the key, or a quantum level computer.

Hash functions and bit levels of a cypher will tell you a lot about it, and how secure it is. First, AES itself doesn’t have a hash function, instead it is usually paired with one such as the Secure Hash Algorithm (SHA). The Hash function itself uses an algorithm to ‘hash’ a key for the user to use to decrypt the AES encrypted data. The Hash function for the key is whatever length the number of bits is, so 256 vs 512 for example are the length. With AES however, the encryption is always in 128bit blocks, it just goes through different amounts of ‘rounds’ which is the term for sending the data through a round of encryption.

AES is an example of a symmetric key, meaning the same key is used for encryption and decryption. Asymmetric keys however, like Rivest-Shamir-Adleman (RSA) encryption uses a pair of keys, one for encryption and one for decryption. Meaning with Asymmetric cryptography, you would need a different key to read the data than what you would use to write the data. Finally random numbers are important for generating the ‘randomness’ in the encryption algorithm. This is important because the same thing may be encrypted multiple times, but there will appear different because of the randomness, and therefore be more secure.

As long as there are secrets that people wish to keep, there will always be a need to obfuscate information. Though we have come much further technologically then the original Ceaser and Vigenère ciphers back in the day, the concept remains similar in many ways in that we take the information we wish to obscure, use an algorithm to change that information, and have a method to then turn that obscured message back into the information. Probably the most famous example of a mechanical / electric cypher throughout history was the Enigma Machine. Finally cracked by Alen Turing and his team during World War 2, it would send electrical signals sent through a keyboard through a series of scrambling circuits and rotating barrels to output an ever-changing alphabetical message that would make no sense to those reading it. The ‘genius’ behind it was that because it was a 1 to 1 circuit, if you placed the machine in the same state as when the message was created, you could simply type in the scrambled alphabetic message you got out of it, it would map back to the original keys that were pressed, so you could easily get the original message back out. Now, however, the encoding of messages is done through digital means, meaning computers can quickly compute letter substitutions, mixing, sub byte mixing, and various mathematical techniques to obscure the data. Currently the sophistication of the encryption cyphers of today is expected to last for a long time, until computing power gets to the point it can brute force attack an encryption and test every combination in a meaningful amount of time. This is expected to become an issue only after the switch to quantum computing, which will usher in an era when digital security must me radically overhauled. Until then, modern encryption is very reliable.

## Certificate Generation

Insert a screenshot below of the CER file.

A screenshot of a certificate

AI-generated content may be incorrect.

## Deploy Cipher

Insert a screenshot below of the checksum verification.

A black numbers on a white background

AI-generated content may be incorrect.

A screen shot of a computer code

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 screenshot of a computer

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 screen shot of a computer

AI-generated content may be incorrect.

## Summary

The refactoring of the code I did was to utilize my last generate hash function and update to secure hash algorithm (SHA 512). This increase in bit size drastically increased security by making it harder for an attacker to brute force the key solution. I also changed the /hash mapping to test the hash generate hash function by testing it against an already known hash. Though this is good for the assignment and initial testing, it would not need to be in an actual version for Artemis Financial. Following the Vulnerability Assessment Process Flowchart. The first piece is Input Validation. Now there is no input from the user other than going to the /hash location which will only check on static string “Alex Hitchens” so there is no danger at the moment from malicious input. APIs are fine now because the only dependencies directly being utilized are the Spring and Junit APIs which, while they have vulnerabilities, are not a concern for our current setup. With Cryptography, I took into consideration by increasing the bit size of the SHA cipher. Client/Server was not tested because it was not part of the current assignment. Code Error was checked both statically and dynamically and is currently functioning properly. Code Quality I tried to follow the best practices. I don’t like the fact that I had to hard code the hash and the name for the test as required in step three but presumably that will not be present in a final version. Encapsulation was present because the hash function and test are completely their own thing which is seen by the fact that I was able to take most of straight from previous work.

The process of adding layers of security for the software application involved checking each layer as it went along checking to make sure it worked properly then continuing. I added the hash cipher function first, then added the check to see if it matched an existing known hash. Next, I got the certificate working with it. I think the biggest lesson I can take away from this is to ensure pieces work individually, that way when errors occur it is easier to diagnose where they are from.

## Industry Standard Best Practices

The industry’s best practices I used for this were first to upgrade the SHA cipher. I learned I could go higher, so I changed it from the last time I did this assignment to make it more secure by upgrading it to 512. Also, the code was written in such a way that it was modular, and the hash function could be reused elsewhere. Finally, the code was written with error checking in the hash function so if something went wrong it would return to a standard error phrase instead of trying to parse something random. One thing I was not a fan of was how I had to hard code specific data during the test for the 3rd requirement. It is not an industry best practice to do that, but since it was required, It was done.

Following industry’s best standards helps a company over all wellbeing for multiple reasons. First it helps by providing a path that has already been tread before. Why reinvent the wheel and potentially introduce new vulnerabilities when instead you could learn from what others have done and build from that to produce something solid. Next it helps because often the ones maintaining the code are not the ones building the code. By following industry’s best practices, you are not introducing anything out of the ordinary that might confuse people and may cause them to mess up the existing software because you added something that was strange to them. Finally, it makes it easy to stay with in regulation. If you follow the best practices for financial institutions, for example, you will stay within the GLBA and FFEIC requirements for example and there for keep your company safe as you proceed. For all these reasons, when it comes to secure coding, following the best practices that others have laid down already can only help you out.

References:

baeldung. (2024, January 8). *Introduction to HexFormat in Java*. Baeldung. https://www.baeldung.com/java-hexformat

*Cryptography - camellia encryption algorithm*. Tutorialspoint. (n.d.-a). https://www.tutorialspoint.com/cryptography/cryptography\_camellia\_encryption\_algorithm.htm

Manico, J., & Detlefsen, A. (2015). *Iron-clad java: Building secure web applications*. Mc Graw Hill Education.

Probasco, L. (n.d.). *Encryption requirements for Banks & Financial Services*. Encryption Requirements for Banks & Financial Services. https://info.townsendsecurity.com/encryption-requirements-for-banks-financial-services

Sidhpurwala, H. (2023, January 12). *A brief history of cryptography*. Red Hat - We make open source technologies for the enterprise. https://www.redhat.com/en/blog/brief-history-cryptography